Министерство образования Республики Беларусь
Белорусский государственный университет
Факультет прикладной математики и информатики
Кафедра дискретной математики и алгоритмики

СОГЛАСОВАНО	СОГЛАСОВАНО
Заведующий кафедрой	Декан факультета
_________________ Котов В.М.	_________________ Орлович Ю.Л.
«18» декабря 2023 г.	«19» декабря 2023 г.

Алгоритмы и структуры данных

Электронный учебно-методический комплекс для специальностей:
6-05-0533-09 «Прикладная математика»
6-05-0533-10 «Информатика»
6-05-0533-11 «Прикладная информатика»
1-31 03 04 «Информатика»
направлений специальностей:
1-31 03 03-01 «Прикладная математика (научно-производственная деятельность)»
1-31 03 07-01 «Прикладная информатика (программное обеспечение компьютерных систем)»

В 3 частях. Часть 1

Регистрационный № 2.4.2-24/409

Авторы:
Соболевская Е.П., кандидат физико-математических наук, доцент;
Буславский А.А., старший преподаватель;
Котов В.М., доктор физико-математических наук, профессор;
Сатолина А.В., старший преподаватель.

Рассмотрено и утверждено на заседании Научно-методического совета БГУ 18.01.2024 г., протокол № 5.

Минск, 2023

УДК 004.421(075.8)+004.422.63(075.8)
А 456

Утверждено на заседании Научно-методического совета БГУ
Протокол № 5 от 18.01.2024 г.

Решение о депонировании вынес:
Совет факультета прикладной математики и информатики
Протокол № 4 от 19.12.2023 г.

А в т о р ы:

Соболевская Елена Павловна, кандидат физико-математических наук, доцент кафедры дискретной математики и алгоритмики факультета прикладной математики и информатики БГУ;

Буславский Александр Андреевич, старший преподаватель кафедры дискретной математики и алгоритмики факультета прикладной математики и информатики БГУ;

Котов Владимир Михайлович, доктор физико-математических наук, заведующий кафедрой дискретной математики и алгоритмики факультета прикладной математики и информатики БГУ;

Сатолина Анна Викторовна, старший преподаватель кафедры биомедицинской информатики факультета прикладной математики и информатики БГУ.

Рецензенты:

кафедра программного обеспечения информационных технологий, УО «Белорусский государственный университет информатики и радиоэлектроники» (заведующий кафедрой, кандидат технических наук, доцент Н.В. Лапицкая);

Шлыков В. В., профессор кафедры математики и методики преподавания математики УО «Белорусский государственный педагогический университет имени Максима Танка», кандидат физ.-мат. наук, доктор пед. наук, доцент.

Алгоритмы и структуры данных : электронный учебно-методический комплекс для специальностей: 6-05-0533-09 «Прикладная математика», 6-05-0533-10 «Информатика», 6-05-0533-11 «Прикладная информатика», 1-31 03 04 «Информатика», направлений специальностей 1-31 03 03-01 «Прикладная математика (научно-производственная деятельность)», 1-31 03 07-01 «Прикладная информатика (программное обеспечение компьютерных систем)». В 3 ч. Ч. 1 / Е. П. Соболевская [и др.] ; БГУ, Фак. прикладной математики и информатики, Каф. дискретной математики и алгоритмики. – Минск : БГУ, 2023. – 183 с. : ил. – Библиогр.: с. 182–183.

Электронный учебно-методический комплекс «Алгоритмы и структуры данных. Часть 1» предназначен для студентов специальностей 6-05-0533-09 «Прикладная математика», 6-05-0533-10 «Информатика», 6-05-0533-11 «Прикладная информатика», 1-31 03 04 «Информатика», направлений специальностей 1-31 03 03-01 «Прикладная математика (научно-производственная деятельность)», 1-31 03 07-01 «Прикладная информатика (программное обеспечение компьютерных систем)». Содержание учебного материала включает разделы: анализ сложности алгоритмов, рекуррентные соотношения для алгоритмов поиска и внутренней сортировки, методы разработки эффективных алгоритмов. Изложение соответствует программе учебной дисциплины.

ОГЛАВЛЕНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА **5**
1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ **8**
1.1. Анализ сложности алгоритмов 8
1.1.1. Решение задач в системе автоматического тестирования iRunner 9
1.1.2. Теоретический анализ сложности алгоритма 11
1.1.2.1. Размерность задачи 11
1.1.2.2. Временная и емкостная сложность алгоритма 13
1.1.2.3. Модель абстрактного вычислительного устройства. 14
1.1.2.4. Асимптотики 17
1.1.2.5. Полиномиальные и экспоненциальные алгоритмы 22
1.1.2.6. Примеры получения оценок вычислительной сложности алгоритма 24
1.2. Рекуррентные уравнения для алгоритмов поиска и внутренней сортировки 30
1.2.1. Основные понятия 30
1.2.2. Методы решения рекуррентных уравнений 32
1.2.3. Рекуррентные уравнения для поиска максимального и минимального элементов массива 34
1.2.4. Рекуррентные уравнения для поиска элемента в упорядоченном массиве 37
1.2.5. Рекуррентные уравнения для алгоритмов внутренней сортировки 39
1.2.6. Задача поиска k-ого наименьшего элемента массива 53
1.3. Методы разработки эффективных алгоритмов 56
1.3.1. Метод разделяй и властвуй 57
1.3.2. Динамическое программирование 58
1.3.3. Рекуррентные соотношения и динамическое программирование 60
1.3.3.1. Задача «Факториал» 61
1.3.3.2. Задача «Числа Фибоначчи». 62
1.3.3.3. Задача «Возведение числа 2 в степень n». 64
1.3.3.4. Задача «Ханойские башни». 65
1.3.3.5. Задача «Путь лягушки». 67
1.3.3.6. Задача расстановки единиц. Модульная арифметика. 69
1.3.3.7. Задача оптимального перемножения группы матриц. 73
1.3.3.8. Наибольшая общая подпоследовательность. 77
1.3.3.9. Наибольшая общая подпоследовательность-палиндром. 83
1.3.3.10. Наибольшая строго возрастающая подпоследовательность. 87
1.3.3.11. Наименьшее взвешенное редакционное расстояние Левенштейна. 90

1.3.3.12. Динамическое программирование по профилю. 96
2. ПРАКТИЧЕСКИЙ РАЗДЕЛ 99
2.1. Общие задачи по теме 1.3 99
2.2. Индивидуальные задачи по теме 1.3 104
3. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ 177
3.1. Набор тестовых заданий для проверки теоретических знаний .. 177
3.2. Набор тестов для проверки работоспособности программ 179
3.3. Средства диагностики 180
4. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ 182
4.1. Рекомендуемая литература 182
4.2. Электронные ресурсы 183

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс (ЭУМК) по учебной дисциплине «Алгоритмы и структуры данных. Часть 1» предназначен для студентов специальностей: 6-05-0533-09 «Прикладная математика», 6- 05- 0533- 10 «Информатика», 6-05-0533-11 «Прикладная информатика», 1- 31 03 04 «Информатика», направлений специальностей: 1-31 03 03-01 «Прикладная математика» (научно-производственная деятельность), 1- 31 03 07- 01 «Прикладная информатика» (программное обеспечение компьютерных систем). Комплекс подготовлен в соответствии с требованиями Положения об учебно-методическом комплексе на уровне высшего образования, утвержденного Постановлением министерства образования Республики Беларусь от 26.07.2011 № 167. Содержание разделов ЭУМК соответствует образовательным стандартам, структуре и тематике учебных программ по дисциплине «Алгоритмы и структуры данных» для указанных специальностей и направлений специальностей. Главные цели ЭУМК: помощь студентам в организации самостоятельной работы, повышение качества подготовки специалистов и усиление практико-ориентированности учебного процесса по дисциплине.

ЭУМК состоит из следующих разделов. Теоретический. Включает аннотацию ЭУМК, написанного в соответствии с программами дисциплин. Материал данного комплекса, наряду с конспектом лекций, может быть использован для самостоятельной подготовки студентов к контрольным заданиям и экзамену/зачету. Практический. Содержит набор задач. Данные материалы используются при проведении лабораторных занятий и для самостоятельной работы над учебной дисциплиной. Раздел контроля знаний представлен наборами тестов для проверки теоретических знаний и наборами тестов для проверки работоспособности программ. Описаны формы диагностики и технология определения оценки по дисциплине с учетом текущей успеваемости. Вспомогательный раздел включает рекомендуемую литературу.

Учебная дисциплина «Алгоритмы и структуры данных» знакомит студентов с фундаментальными понятиями, используемыми при разработке алгоритмов и оценке их качества. Для специальности 6-05-0533-09 «Прикладная математика», направления специальности 1-31 03 03-01 «Прикладная математика (научно-производственная деятельность» дисциплина входит в компонент учреждения образования модуля «Дискретная математика и алгоритмика». Для специальностей 6-05-0533-10 «Информатика», 1-31 03 04 «Информатика» дисциплина входит в государственный компонент модуля «Дискретные структуры и алгоритмы». Для специальности 6-05-0533-11 «Прикладная информатика», направления специальности 1-31 03 07-01 «Прикладная информатика (программное обеспечение компьютерных систем» дисциплина входит в государственный компонент модуля «Дискретная математика и алгоритмы». *Цель* преподавания учебной дисциплины состоит в формировании навыков для построения и анализа методов и алгоритмов при решении

модельных задач дискретной оптимизации и их применение на практике. При изложении материала учебной дисциплины целесообразно выделить этап построения математической модели, существенно влияющей на ее адекватность реальной проблеме, а также показать возможность использования аппарата теории алгоритмов для анализа и обоснования выбора наиболее эффективных методов и алгоритмов для решения прикладных задач.

Основные задачи, решаемые при изучении учебной дисциплины «Алгоритмы и структуры данных»: сформировать такие фундаментальные понятий, как размерность задачи и трудоемкость алгоритма; изучить подходы для определения трудоемкости алгоритма посредством составления и решения рекуррентных уравнений; изучить современные структуры данных, уметь обосновывать выбор соответствующей структуры в зависимости от набора базовых операций, используемых в алгоритме;уметь строить графовые модели и применять алгоритмы на графах для решения прикладных задач.

В результате изучения учебной дисциплины, обучающийся должен

знать:

– понятие размерности задачи и трудоемкости алгоритма,

– основные способы решения рекуррентных уравнений,

– основные подходы при разработке эффективных алгоритмов,

– способы организации структур данных и технологию их использования,

– виды поисковых деревьев,

– базовые алгоритмы на графах.

уметь:

– сводить решение исходной задачи к решению подзадач и определять трудоемкость алгоритмов на основе рекуррентных соотношений,

– выбирать подходящие структуры данных при разработке эффективного алгоритма решения задачи,

– реализовывать поисковые деревья,

– строить графовые модели и применять базовые графовые алгоритмы.

владеть:

– основными подходами для разработки эффективных алгоритмов: метод «разделяй и властвуй» и динамическое программирование;

– навыками реализации и использования современных структур данных.

Освоение учебной дисциплины «Алгоритмы и структуры данных» должно обеспечить формирование следующей специализированной компетенции для специальности 6-05-0533-09 «Прикладная математика», направления специальности 1-31 03 03-01 «Прикладная математика (научно-производственная деятельность»:

СК-2. Реализовывать современные структуры данных, строить графовые модели и применять алгоритмы на графах для решения прикладных задач, обосновывать корректность алгоритма и оценивать его асимптотическую сложность.

Освоение учебной дисциплины «Алгоритмы и структуры данных» должно обеспечить формирование следующей базовой профессиональной компетенции для специальностей 6-05-0533-10 «Информатика», 1-31 03 04 «Информатика»:

БПК-6. Выполнять построение математических моделей и проводить их анализ в типовых задачах дискретной математики, интерпретировать получаемые результаты анализа математических моделей и осуществлять выбор структур данных для разработки эффективных алгоритмов решения прикладных задач.

Освоение учебной дисциплины «Алгоритмы и структуры данных» должно обеспечить формирование следующей базовой профессиональной компетенции для специальности 6-05-0533-11 «Прикладная информатика», направления специальности 1-31 03 07-01 «Прикладная информатика (программное обеспечение компьютерных систем»:

БПК-3. Характеризовать предмет и объекты дискретной математики и математической логики, использовать основные приемы разработки эффективных алгоритмов и знания об основных структурах данных при решении прикладных задач.

Для организации самостоятельной работы студентов и самоподготовки по курсу рекомендуется размещение программы курса, списка необходимой основной и дополнительной литературы, презентации лекций, заданий, тестов, методических рекомендаций на доступных сетевых ресурсах факультета и университета. Эффективность самоподготовки студентов целесообразно проверять в виде текущего и итогового контроля знаний в форме компьютерного тестирования как по отдельным темам, так и по разделам курса на образовательной платформе iRunner (https://acm.bsu.by). Тесты, разработанные в iRunner для проверки теоретических знаний, покрывают все разделы учебной дисциплины и генерируются автоматически, что позволяет исключать списывание. В образовательную платформу iRunner интегрирована свободная система управления содержимым (англ. CMS, или content management system) под названием MediaWiki (https://acm.bsu.by/wiki). Вики-документы хорошо адаптированы для просмотра на мобильных устройствах, фрагменты кода на языках программирования отображаются с подсветкой синтаксиса. В образовательную платформу iRunner интегрирован модуль проверки исходного кода на плагиат (Акт о практическом использовании результатов исследования от 27.05.2022 №2.4/148). Модуль активно используется в учебном процессе в качестве диагностического инструментария и позволяет проверять самостоятельность выполнения студентами лабораторных заданий с целью недопущения элементов списывания и использования несанкционированных ресурсов в интернете. При составлении общих и индивидуальных заданий по учебной дисциплине необходимо предусмотреть возрастание их сложности. Рекомендуется делить задания на три модуля: задания, формирующие достаточные знания по изученному учебному материалу на уровне узнавания; задания, формирующие компетенции на уровне воспроизведения; задания, формирующие компетенции на уровне применения полученных знаний.

1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1. Анализ сложности алгоритмов

Алгоритм – конечная последовательность четко определенных, реализуемых компьютером инструкций, предназначенная для решения определенного класса задач. Начиная с начального состояния и начального ввода (возможно, пустого), инструкции описывают вычисление, которое при выполнении проходит через конечное число четко определенных последовательных состояний, в конечном итоге производя вывод и завершаясь в конечном состоянии. Переход от одного состояния к другому не обязательно детерминирован; некоторые алгоритмы рандомизированы. Детерминированный алгоритм: для одних и тех же входных данных все запуски алгоритма одинаковы по поведению. Рандомизированный алгоритм: предполагает в своей работе некоторый случайный выбор и время работы рандомизированного алгоритма зависит от этого выбора (в рамках нашей дисциплины мы будем работать с детерминированными алгоритмами).

Все вычислительные задачи, которые мы будем решать в рамках нашей учебной дисциплины, разрешимы – существует алгоритм, который решает любой частный случай этой задачи [1, 2, 6, 12]. Однако, этого не всегда достаточно, поскольку алгоритму может потребоваться так много времени, что он становится абсолютно бесполезным для практического применения.

Задача коммивояжера

Задан полный граф на n вершинах и матрица расстояний между всеми парами вершин. На рисунке 1.1 приведен пример взвешенного полного графа.

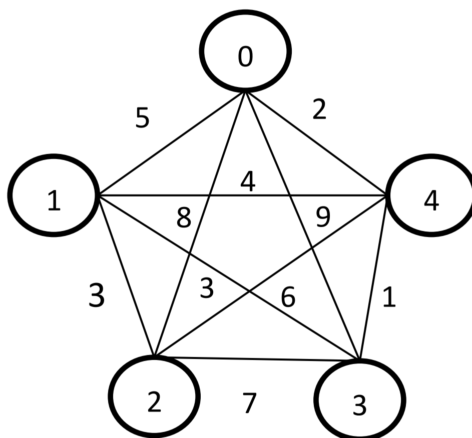


Рисунок 1.1 – Задача коммивояжера.

Необходимо найти замкнутый маршрут минимальной стоимости, проходящий через каждую вершину графа ровно один раз.

Задача разрешима, так как любую индивидуальную задачу можно решить, найдя наилучший обход среди конечного множества обходов. Если алгоритм будет перебирать все обходы, вычислять их длины и выбирать кратчайший обход, то реализация этого алгоритма на вычислительной машине потребует

порядка $(n-1)!/2$ шагов (элементарных команд) и уже для решения задачи среднего размера, например, $n=50$ потребовалось бы многих миллиардов лет при самых оптимистических прогнозах относительно скорости вычислительных машин в будущем, так как $n=49!=6.0828186640342675e+62$ имеет около 63 десятичных знаков.

Минимальное остовное дерево

Заданы натуральное число n и матрица расстояний $\left[d_{i,j}\right]$ между парами вершин, где $d_{i,j}\in Z^{+}$. Необходимо найти остовное дерево на n вершинах (неориентированный, связный, ациклический граф), имеющее наименьшую суммарную длину ребер.

На рисунке 1.2 приведено минимальное остовное дерево для графа, приведенного на рисунке 1.1.

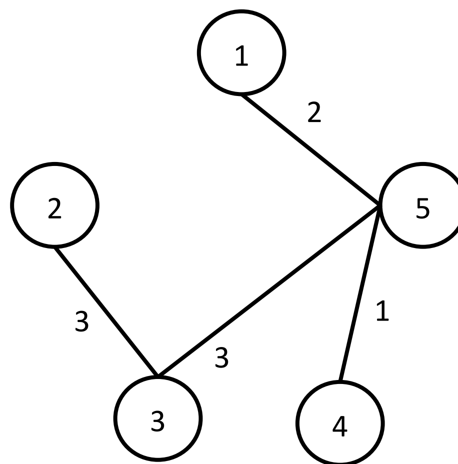


Рисунок 1.2 – Минимальное остовное дерево для графа на рисунке 1.1.

Задача разрешима: можно перебрать все остовные деревья с n вершинами и выбрать наилучшее из них. Поскольку имеется n^{n-2} остовных деревьев с n вершинами, то время, необходимое для работы такого алгоритма перебора, снова неприемлемо: если $n=50$, то $50^{48}=3{,}5527136788005007e+81$.

Для этой задачи существует существенно лучший алгоритм, время работы которого пропорционально n^2 [6].

1.1.1. Решение задач в системе автоматического тестирования iRunner

При решении индивидуальных задач в системе автоматического тестирования iRunner (https://acm.bsu.by) для каждой задачи в условии заданы ограничения на входные данные, а также указаны ограничения по памяти и времени, в которые должна уложиться программа.

Какое время необходимо программе (при ее успешном завершении) при заданных ограничениях на входные данные?

1. Программы, написанные на языках высокого уровня, нужно переводить в машинный код. Это можно делать по-разному. C++ уже на этапе компиляции переводит инструкции программы в хорошо оптимизированный машинный код. Python выполняет преобразования в машинный код на этапе выполнения со

значительными накладными расходами.

2. Реальному ЦП требуется разное количество времени для выполнения различных операций. Например, время выполнения операций на процессоре Intel Core десятого поколения (Ice Lake):

add, sub, and, or, xor, shl, shr…: 1 такт;

mul, imul: 3–4 такта;

div, idiv (32-битный делитель): 12 тактов;

div, idiv (64-битный делитель): 15 тактов.

3. Даже имея готовый ассемблерный код реализации алгоритма, не представляется возможным узнать, какое время потребуется для его выполнения. Для этого необходимо было бы учесть, в частности, следующие моменты.

• Кеширование данных. Процессоры имеют многоуровневую систему кешей (L1, L2, L3), постоянно сохраняющую те или иные ячейки для более быстрого доступа. В зависимости от того, закешировал ли процессор нужную ячейку, время доступа к данным может отличаться в десятки раз.

• Out-of-order execution. Процессор способен выполнять несколько не зависящих друг от друга команд одновременно (например, последовательные mov eax, ecx и add edx, 5). Процессор просматривает программу на сотни инструкций вперед, выискивая те, что может выполнить без очереди.

• Branch prediction и Speculative execution. Большое препятствие для выполнения инструкций наперед – ветвления (в частности, if'ы). Процессор не может заранее знать, в какую ветвь алгоритма ему придется войти, и какой код ему нужно выполнять наперед. Branch prediction модули следят за ходом выполнения программы и пытаются предсказать направления ветвлений (угадывают в $>90\%$ случаев). Штраф за ошибочное предсказание – потеря времени из-за избавления от десятков заранее подготовленных результатов операций и начала вычислений заново.

Таким образом, можно придерживаться следующих рекомендаций.

1. Если вы пишете на C ++ и решаете типичную алгоритмическую задачу, то можете предположить, что за 1 секунду вы сможете выполнить $\sim 10^8$ абстрактных операций.

2. Если вы делаете много делений, если вы обращаетесь к большому количеству памяти в случайном порядке, то вы сможете сделать за 1 секунду гораздо меньше $\sim 10^7$.

3. Если операции простые, а обращения к памяти локальные или последовательные, то вы сможете за 1 секунду выполнить $\sim 10^9$ операций.

Предположим, что вы получили задачу, разработали алгоритм ее решения и обосновали его корректность, выполнили реализацию алгоритма на некотором высокоуровневом языке программирования.

Программа, реализующая разработанный вами алгоритм, успешно прошла все тесты в системе автоматического тестирования iRunner и уложилась в заданные в условии задачи ограчения по времени и памяти.

Можно ли утверждать, что вы разработали эффективный алгоритм решения задачи?

Как следует из данных таблицы 1.1, если число абстрактных операций алгоритма 1 оценивается полиномиальной функцией n^5, а алгоритма 2 – экспоненциальной функции 2^n, то при ограничениях на входные данные: $n \le 20$ быстрее работает алгоритм 2.

Таблица 1.1 – Скорость роста функций.

	$n=10$	$n=20$	$n=30$
n	0,00001с	0,00002с	0,00003с
n^2	0,0001с	0,0004с	0,0009с
n^3	0,001с	0,008с	0,027с
n^5	0,1с	3,2с	24,3с
2^n	0,001с	1с	17,5 мин
3^n	0,059с	58 мин	6,5 лет

Однако, при $n \to +\infty$, алгоритм 2 будет работать существенно дольше, чем алгоритм 1. Более того, если функция, описывающая число абстрактных операций, выполняемых алгоритмом, растет экспоненциально, то при увеличении объема входных данных программа может не завершиться за миллиарды лет, хотя при небольших данных она работает за секунду и успешно проходит по времени все тесты в iRunner. Такой алгоритм решения задачи на практике не применим.

1.1.2. Теоретический анализ сложности алгоритма

Теоретический анализ сложности алгоритма заключается в анализе поведения алгоритма при увеличении объема входных данных. Одна из характеристик для сравнения эффективности алгоритмов – исследование (математическими методами) поведения алгоритма при увеличении размера входных данных, так как именно эти входы определяют границы применимости алгоритма.

1.1.2.1. Размерность задачи

Для характеристики «громоздкости данных» можно ввести некий неотрицательный числовой параметр, характеризующий входные данные для алгоритма, и оценивать трудоемкость алгоритма с помощью функций, зависящих от этого числового параметра. Этот параметр называют *размерностью задачи* (*размер входа*) и обозначают буквой l.

В качестве размерности задачи l принято брать число бит, необходимое для кодировки входных данных задачи.

Для того, чтобы понять, сколько бит нужно для кодировки входных данных, надо определиться с системой кодирования входных данных. Рассмотрим унарную и бинарную системы кодирования.

Унарная система кодирования: длина в битах равна самому числу.

Пусть на вход поступает единственное целое неотрицательное число $n = 5$, тогда в унарной системе кодирования битовая длина такого входа $l = 5$.

1	1	1	1	1
4	3	2	1	0

Бинарная система кодирования: длина в битах равна количеству цифр в его двоичной записи. Будем рассматривать только *компактное представление числа*: старший бит в двоичной записи числа равен 1, т. е. нет незначащих 0.

Пример 1.1. Пусть на вход поступает единственное целое неотрицательное число $n = 6$, тогда в бинарной системе кодирования битовая длина входа $l = 3\ (6 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)$.

1	1	0
2	1	0

Пример 1.2. На вход поступает одно целое неотрицательное число n. Какова битовая длина этого числа в бинарной системе кодирования?

Решение. Предположим, что для компактного задания числа n в двоичной системе счисления $(l+1)$ бита много, а l – достаточно.

1		0	1	0
$l-1$	…	2	1	0

Тогда справедливы неравенства:

$$2^{l-1} \le n < 2^l.$$

Логарифмируем и, учитывая, что число бит является целым числом, получаем:

$$\log_2 n < l \le \log_2 n + 1,$$

$$l = \lfloor \log_2 n + 1 \rfloor.$$

Таким образом, если на вход поступает одно целое число $n \ge 0$, то битовая длина входа в бинарной системе кодирования:

$$l = \begin{cases} 1, n = 0 \\ \lfloor \log_2 n + 1 \rfloor, n > 0. \end{cases}$$

Пример 1.3. На вход поступает натуральное число n и массив чисел $a_1, a_2, \ldots, a_n$. Какова битовая длина входа в бинарной системе кодирования?

Решение.

$$l = \lfloor \log_2 n + 1 \rfloor + \left(\lfloor \log_2 |a_1| + 1 \rfloor + \lfloor \log_2 |a_2| + 1 \rfloor + \cdots + \lfloor \log_2 |a_n| + 1 \rfloor \right).$$

Если предположить, что $\max\limits_{1 \le i \le n} \lfloor \log_2 |a_i| + 1 \rfloor = const$, где, например, $const = 32$, то $l = \lfloor \log_2 n + 1 \rfloor + const \cdot n.$

В данной модели для увеличения объема входных данных будем устремлять $n \to +\infty$, а входными данными задачи размерности l будут только массивы из n элементов.

Выбор, что взять в качестве числовой характеристики возможных входов алгоритма, делается в зависимости от характера задачи.

1) Если на вход поступает массив $a_1, a_2, \ldots, a_n$ чисел и нам нужно их упорядочить или что-то найти в массиве, то в предположении, что $\max\limits_{1 \le i \le n} \lfloor \log_2 |a_i| + 1 \rfloor = const$ в качестве такой числовой характеристики входных данных может выступать число n элементов массива, а такую меру измерения размерности задачи называют *равномерной*; при анализе поведения алгоритма с ростом объема входных данных будем устремлять к бесконечности число n.

2) В задачах на графы в качестве числовой характеристики входных данных задачи берут число вершин $|V|$ и/или ребер $|E|$ (рассуждения будут аналогичны, предыдущему случаю, так как, например, если граф задан матрицей смежности, то элементы матрицы принимают всего два значения 1 или 0 и для кодирования достаточно константной памяти; если граф задан списками смежности, то так как мы работаем с конечными графами, т.е. $|V|$ – конечно, то для кодирования также достаточно константной памяти; при анализе поведения алгоритма с ростом объема входных данных будем устремлять к бесконечности число ребер $|E|$;

3) В арифметических задачах размером входа может быть максимум абсолютных величин входных чисел или количество цифр в его двоичной записи (бинарная система кодирования). Например, если на вход поступает целое число n и необходимо определить, является ли это число простым, то в качестве числовой характеристики входных данных l разумно брать число бит двоичной записи компактного представления числа n; при анализе поведения алгоритма при увеличении объема входных данных будем устремлять к бесконечности битовую длину числа n.

1.1.2.2. Временная и емкостная сложность алгоритма

Определение 1.1. *Временна́я сложность алгоритма* – это функция, которая задаче размерности l ставит в соответствие время $T(l)$, затрачиваемое алгоритмом для ее решения.

Определение 1.2. Если при данной размерности l в качестве меры сложности берется наибольшее из времен (по всем входам этой размерности), то она называется *временной сложностью в худшем случае*.

Определение 1.3. Если при данной размерности l в качестве меры сложности берется среднее время (по всем входам этой размерности), то она

называется *средней сложностью* (среднее время работы алгоритма по всем возможным наборам входных данных задачи размерности l).

Для вычисления средней сложности поступают следующим образом.

1) Все входные данные задачи размерности l разбивают на группы так, чтобы время работы алгоритма для всех данных из одной группы было одним и тем же. Предположим, что у нас m групп.

2) Пусть p_i – вероятность, с которой данные попадают в группу i.

3) Пусть t_i – время работы алгоритма для данных из группы i.

4) Тогда среднее время работы алгоритма по всем возможным набрам входных данных задачи размерности l определяется по следующей формуле:

$$A(l) = \sum_{i=1}^{m} p_i \cdot t_i.$$

Если у нас m групп и входные данные могут оказаться с равной вероятностью в любой из них, то

$$p_i = \frac{1}{m}, \quad \forall i = 1,\ldots,m.$$

В этом случае среднее время работы алгоритма по всем возможным набрам входных данных задачи размерности l равно

$$A(l) = \sum_{i=1}^{m} t_i \Big/ m.$$

Определение 1.4. Емкостная сложность алгоритма – объем памяти, требуемый для хранения данных при реализации алгоритма, как функция от размерности задачи.

Поведение временной (емкостной) сложности при увеличении размерности задачи до бесконечности называется асимптотической временной (емкостной) сложностью.

1.1.2.3. Модель абстрактного вычислительного устройства.

Предположим, что у нас есть некоторый алгоритм решения задачи и мы хотим на некотором исполнителе реализовать его.

Как определить время выполнения и объем памяти программной реализации алгоритма на некотором абстрактном вычислительном устройстве?

В качестве модели такого абстрактного вычислительного устройства возьмем РАМ – одноадресную машину с произвольным доступом к памяти (англ. Random-Access Machine – RAM).

РАМ – универсальная математическая модель вычислений, которая является хорошим приближением к классу обычных вычислительных машин (рисунок 1.3).

РАМ – вычислительная машина с одним сумматором, в котором команды программы не могут изменять сами себя, состоит из:

– процессора с набором команд, все команды – одноадресные;

– входной ленты, с которой она может считывать данные в соответствии с их упорядоченностью (последовательный доступ);

– выходной ленты, на которую она может записывать (в первую свободную клетку);

– оперативной памяти, которая состоит из потенциально бесконечного числа ячеек, в каждой из которых может быть записано произвольное целое число, индексы ячеек могут быть и отрицательными, все ячейки равнодоступны; сумматор – это специальный выделенная тип памяти, в котором выполняются все вычисления.

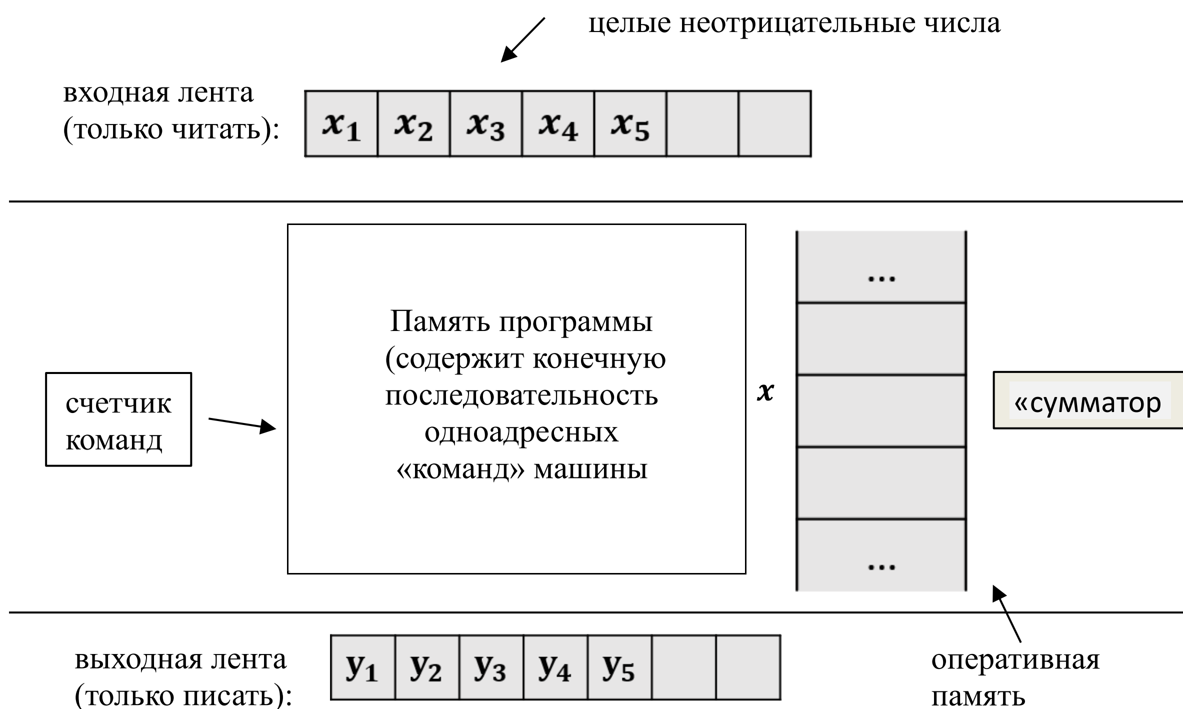


Рисунок 1.3 – РАМ-машина.

РАМ существенно отличается от другой известной абстрактной модели вычислений – машины Тьюринга (МТ) (предложена в качестве абстрактной модели вычислений А. Тьюрингом в 1936 г.).

Для МТ время перехода между ячейками ленты зависит от расстояния между ячейками, а одна ячейка ленты содержит один символ фиксированного конечного алфавита.

Программа для РАМ машины:

– конечная последовательность одноадресных команд, которая изменяет ячейки памяти (команды программы занумерованы числами 1,2, ...);

– есть счетчик команд – целое число;

– программа не записывается в память (т. е. не может менять саму себя);

– команды процессора выполняются последовательно, одновременно выполняемые команды отсутствуют (однопроцессорная машина);

– результат работы программы – набор чисел на выходной ленте.

Формально – любое число вмещается в любую ячейку.

В таблице 1.2 приведен набор команд для РАМ-машины.

Таблица 1.2 – Набор команд для РАМ-машины.

READ x	считывает в ячейку x очередной символ из входной ленты
WRITE x	запись содержимого ячейки x на выходную ленту
LOAD x	загрузить операнд из ячейки x памяти в сумматор
STORE x	отослать содержимое из сумматора в указанную ячейку x памяти
ADD x SUB x	сложить содержимое ячейки x и сумматора вычесть содержимое ячейки x из сумматора
MUL x DIV x	умножить содержимое сумматора и ячейки x разделить число из сумматора на содержимое ячейки x
JGTZ M JZERO M JUMP M HALT WAIT	переход на команду с меткой М, если число в сумматоре > 0 переход на команду с меткой М, если число в сумматоре $= 0$ безусловный переход на команду с меткой М останов (завершение работы программы) ожидание

Операнды могут быть одного из следующих типов:= x – число; i – содержимое i-ой ячейки памяти; $*i$ – содержимое ячейки памяти, номер которой записан в ячейке i (косвенная адресация).

При *равномерной мере сложности* время выполнения каждой команды и размер каждой ячейки для РАМ-машины полагается равным единице. Теперь время работы программы равно общему числу выполненных команд, а объем памяти равен числу ячеек, к которым было обращение.

Пример 1.4. На входной ленте записаны два числа: 2^{100} и 2^{10}. Написать программу вычисления суммы двух чисел и оценить время выполнения программы при равномерной мере сложности.

Решение.

```
READ 1
LOAD 1
READ 2
ADD 2
STORE 3
WRITE 3
```

Время работы программы равно шести (в программе шесть команд). Объем памяти равен трем (к трем ячейкам памяти шло обращение).

При *логарифмической мере сложности* время выполнения каждой команды для РАМ-машины полагается по порядку равным логарифму величины максимального операнда плюс 1, а общее время равно сумме времен выполнения всех выполненных команд. Объем памяти равен сумме максимальных

разрядностей числа в каждой используемой ячейке оперативной памяти плюс максимальная разрядность числа в сумматоре.

Пример 1.5. На входной ленте записаны два числа: 2^{100} и 2^{10}. Написать программу вычисления произведения двух чисел и оценить время выполнения программы и требуемый объем памяти при логарифмической мере сложности.

Решение.

```
READ 1
LOAD 1
READ 2
MUL 2
STORE 3
WRITE 3
```

Время работы программы равно 536 (101+101+11+101+111+111).

Объем памяти равен 334 (101 бит (1-я ячейка) + 11 бит (2-я ячейка) + 111 бит (3-я ячейка) + 111 бит (сумматор)).

1.1.2.4. Асимптотики

Рассмотрим две неотрицательные функции $f(n)$ и $g(n)$.

Будем говорить, что функция $f(n)$ растет *асимптотически медленнее*, чем функция $g(n)$, если

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0.$$

Будем говорить, что функция $f(n)$ растет *асимптотически не быстрее*, чем функция $g(n)$, если для некоторой константы $const$ справедливо

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}\le const.$$

Будем говорить, что функция $g(n)$ растет *асимптотически быстрее*, чем функция $f(n)$, если

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0.$$

Будем говорить, что функция $f(n)$ растет *асимптотически не медленнее*, чем функция $g(n)$, если существует такая константа больше 0, для которой

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}\ge const.$$

Для функции $T(n)$, характеризующей время работы алгоритма в худшем случае, важна скорость ее роста при возрастании объема входных данных. Поэтому, если функция $T(n)$ – сложная, то будем учитывать только ту ее часть, которая с ростом аргумента к бесконечности, растет не медленнее и не быстрее, чем $T(n)$. Эту часть функции $T(n)$ будем называть *порядком роста функции*.

Например, для функции $T(n)=8\cdot n^2+n\cdot\log_2 n+2$ порядок роста можно определить, например, как $const\cdot n^2$ (говорят, что функция растет как n^2).

Покажем, насколько важна скорость роста функций, несмотря на технический прогресс, приводящий к увеличению роста быстродействия ЭВМ. Предположим, что за сутки современная ЭВМ можно выполнять 10^{12} абстрактных операций.

Если алгоритму требуется для решения задачи выполнить $f(n) = n^3$ операций, то за сутки ЭВМ успееет решить задачу для $n = 10^4$ ($n^3 = 10^{12}$, $n = \left(10^{12}\right)^{1/3} = 10^4$). Если предположить, что скорость вычислений ЭВМ за сутки возрастает в 10 раз, то $\tilde{n}^3 = 10 \cdot 10^{12}$ и теперь ЭВМ успевает решить задачу для

$$\tilde{n} = \left(10 \cdot 10^{12}\right)^{1/3} = 10^{1/3} \cdot 10^4 \approx 2{,}15 \cdot 10^4 = 2{,}15 \cdot n .$$

Если алгоритму требуется для решения задачи выполнить $f(n) = \beta^n$ операций, то за сутки ЭВМ успееет решить задачу для $n = \log_\beta 10^{12}$ ($\beta^n = 10^{12}$, $n = \log_\beta 10^{12}$). Если предположить, что скорость вычислений за сутки увеличивается в 10 раз, то $\beta^{\tilde{n}} = 10 \cdot 10^{12}$, и ЭВМ успевает решить задачу для

$$\tilde{n} = \log_\beta \left(10 \cdot 10^{12}\right) = \log_\beta 10 + \log_\beta 10^{12} = \log_\beta 10 + n .$$

В таблице 1.3, показано, как растет размер индивидуальной задачи, решаемой за один день ЭВМ, если скорость вычислений увеличивается в 10 раз:

Таблица 1.3 – Скорость роста функций.

Функция	Размер индивидуальной задачи, решаемой на ЭВМ за 1 день	Размер индивидуальной задачи, решаемой за 1 день на ЭВМ, скорость которой в 10 раз больше
n	10^{12}	$10 \cdot 10^{12}$
$n \cdot \log_2 n$	$0{,}948 \cdot 10^{11}$	$8{,}7 \cdot 10^{11}$
n^2	10^6	$3{,}16 \cdot 10^6$
n^3	10^4	$2{,}15 \cdot 10^4$
$10^8 \cdot n^4$	10	$1{,}8 \cdot 10$
2^n	40	$40 + 3 = 43$
10^n	12	$12 + 1 = 13$
$n^{\log_2 n}$	79	95
$n!$	14	15

В общем случае справедливы следующие утверждения:

1) если время работы алгоритма оценивается функцией $f(n) = \log_2 n$, то увеличение скорости вычислений ЭВМ в v раз позволит решить за такое же время задачу, размер которой в 2^v раз больше;

2) если время работы алгоритма оценивается функцией $f(n) = n^\alpha$, то увеличение скорости вычислений ЭВМ в v раз позволит решить за такое же время задачу, размер которой в $v^{1/\alpha}$ раз больше (мультипликативное увеличение размера задачи, которую алгоритм может решить за фиксированное время);

3) если время работы алгоритма оценивается функцией $f(n) = \beta^n$, то увеличение скорости вычислений в v раз позволит решить за такое же время задачу, размер которой лишь на $\log_\beta v$ больше (аддитивное увеличение размера задачи, которую алгоритм может решить за фиксированное время).

Видно, что использование алгоритов, время которых ограничено функцией вида n^α, позволило кратно увеличить размер решаемых задач при увеличении быстродействия ЭВМ. Таким образом, колоссальный рост скорости вычислений, вызванный появлением нынешнего поколения цифровых вычислительных машин, не уменьшает важность разработки эффективных алгоритмов.

Математические обозначения (*нотации*) $\mathrm{O}, \Omega, \Theta$ введены для функций по аналогии с тем, как для чисел были введены обозначения: $\lceil \ \rceil$ – округление к большему/вверх/англ. *ceiling* (досл. «потолок»); $\lfloor \ \rfloor$ – округление к меньшему/вниз/англ. *floor* (досл. «пол»).

Асимптотики $\mathrm{O}\big(f(n)\big)$, $\Omega\big(f(n)\big)$, $\Theta\big(f(n)\big)$ – способ оценки функции $g(n)$ при $n \to +\infty$ сверху и снизу, чтобы упростить ее внешний вид [1, 5].

Здесь и далее $n \to +\infty$.

Произносится

$g(n) = \mathrm{O}(f(n))$ «g от n равно **о большое** от f от n»;

$g(n) = \Omega(f(n))$ «g от n равно **омега большое** от f от n»;

$g(n) = \Theta(f(n))$ «g от n равно **тетта большое** от f от n».

Асимптотика $\mathrm{O}\big(f(n)\big)$ – это множество функций, которые растут не быстрее, чем функция $f(n)$.

Асимптотика $\Omega\big(f(n)\big)$ – это множество функций, которые растут не медленнее, чем функция $f(n)$.

Асимптотика $\Theta\big(f(n)\big)$ – это множество функций, которые растут не быстрее и не медленнее, чем функция $f(n)$.

Запишем математически определение асимптотик. Пусть $g(n)$ и $f(n)$ – функции, определенные на множестве целых положительных чисел и принимающие положительные действительные значения.

Определение 1.5. Будем писать $g(n) = O(f(n))$, если существуют такие константы $c > 0, n_0 > 0$, что $\forall n \geq n_0$ выполняется $0 \leq g(n) \leq c \cdot f(n)$.

Говорят, что функция $f(n)$ дает асимптотическую верхнюю границу для функции $g(n)$ (рисунок 1.4).

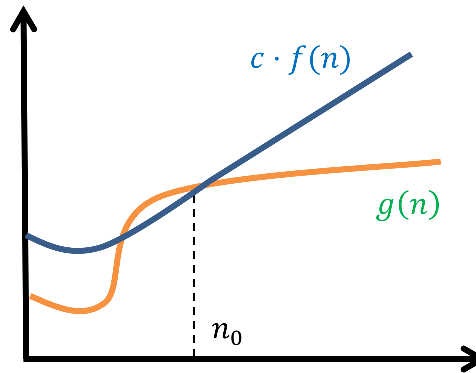


Рисунок 1.4.

Пусть $g(n) = 4 \cdot n^3$, тогда можно записать

$$4 \cdot n^3 = O\left(2^n\right)$$

$$4 \cdot n^3 = O\left(n^3 \cdot \log_2 n\right)$$

$$4 \cdot n^3 = O\left(n^3\right)$$

Определение 1.6. Будем писать $g(n) = o(f(n))$, если $\forall c > 0, \exists n_0 > 0 : \forall n \geq n_0$ выполняется $0 \leq g(n) < c \cdot f(n)$.

Пусть $g(n) = O(f(n))$, тогда $f(n)$ является точной оценкой для функции $g(n)$, только если $g(n) \neq o(f(n))$.

Например, для функции $g(n) = 4 \cdot n^3$ функция $f(n) = n^3$ является точной оценкой, так как $4 \cdot n^3 = O\left(n^3\right), 4 \cdot n^3 \neq o\left(n^3\right)$.

Определение 1.7. Будем писать $g(n) = \Omega(f(n))$, если существуют такие константы $c > 0, n_0 > 0$, что $\forall n \geq n_0$ выполняется $0 \leq c \cdot f(n) \leq g(n)$.

Говорят, что функция $f(n)$ дает асимптотическую нижнюю границу для функции $g(n)$ (рисунок 1.5).

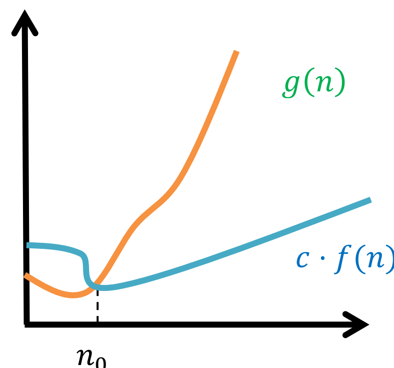


Рисунок 1.5.

Пусть $g(n) = 4 \cdot n^3$, тогда можно записать

$$4 \cdot n^3 = \Omega(n)$$
$$4 \cdot n^3 = \Omega\left(n^2 \cdot \log_2 n\right)$$
$$4 \cdot n^3 = \Omega\left(n^3\right)$$

Определение 1.8. Будем писать $g(n) = \omega(f(n))$, если $\forall c > 0, \exists n_0 > 0 : \forall n \geq n_0$ выполняется $0 \leq c \cdot f(n) < g(n)$.

Пусть $g(n) = \Omega(f(n))$, тогда $f(n)$ является точной оценкой для функции $g(n)$, только если $g(n) \neq \omega(f(n))$.

Например, для функции $g(n) = 4 \cdot n^3$ функция $f(n) = n^3$ является точной оценкой, так как $4 \cdot n^3 = \Omega\left(n^3\right)$, $4 \cdot n^3 \neq \omega\left(n^3\right)$.

Если $g(n) = \Omega(f(n))$, то $f(n) = O(g(n))$.

Определение 1.9. Будем писать $g(n) = \Theta(f(n))$, если существуют такие константы $c_1 > 0, c_2 > 0, n_0 > 0$, что $\forall n \geq n_0$ выполняется $c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)$.

Говорят, что функция $f(n)$ является асимптотически точной оценкой для функции $g(n)$, а про функции $g(n)$ и $f(n)$ говорят, что они имеют одинаковый порядок роста (рисунок 1.6).

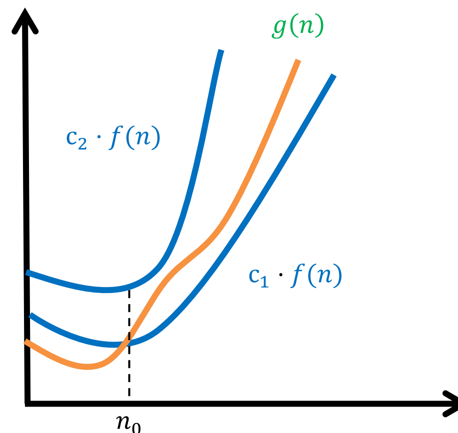


Рисунок 1.6.

Про функции, которые попадают в множество $\Theta\left(n^3\right)$, говорят, что они являются величинами порядка $\Theta\left(n^3\right)$:

$$4 \cdot n^3 = \Theta\left(n^3\right),$$
$$2 \cdot n^3 = \Theta\left(n^3\right),$$
$$n^3 = \Theta\left(n^3\right).$$

Смысл нижней и верхней оценки, полученной для алгоритма с вычислительной сложностью $T_A(l)$:

– если $T_A(l) = \Omega(f_1(l))$, то алгоритм быстрее, чем $f_1(l)$ работать не будет;

– если $T_A(l) = \mathrm{O}(f_2(l))$, то алгоритм медленнее, чем $f_2(l)$ работать не будет.

1.1.2.5. Полиномиальные и экспоненциальные алгоритмы

Алгоритмы, предназначенные для решения определенного класса задач, могут иметь разную временную сложность. В каком случае вычислительную задачу можно считать удовлетворительно решенной? Для сравнения эффективности алгоритмов часто применяют подход, который заключается в различии между *полиномиальными* и *экспоненциальными* алгоритмами.

Напомним, что полиномом степени k от аргумента l называется функция следующего вида:

$$p(l) = \sum_{i=0}^{k} a_i \cdot l^i, a_k \neq 0, k - \text{константа.}$$

Полиномом является асимптотически положительной функцией тогда и только тогда, когда $a_k > 0$.

Полилогарифимческая функция – это функция следующего вида:

$$p(\log l) = \sum_{i=0}^{k} a_i \cdot (\log l)^i, a_k > 0, k - \text{константа.}$$

Полилогарифмическая функция растет медленнее, чем любой положительный показатель степени l:

$$p(\log l) = \mathrm{O}\left(l^{\alpha}\right), \alpha > 1.$$

Функция $f(l)$ *полиномиально ограничена*, если существует такая константа $\alpha > 1$, что $f(l) = \mathrm{O}\left(l^{\alpha}\right)$.

Например, $f(l) = l \cdot \log_2 l$ – полиномиально ограничена, так как $f(l) = l \cdot \log_2 l = \mathrm{O}\left(l^2\right)$.

Определение 1.10. Алгоритм называется *полиномиальным*, если его асимптотическая временная сложность $T(l) = \mathrm{O}(p(l))$, где $p(l)$ – полином или полиномиально ограниченная функция.

Среди специалистов по вычислительным наукам широко распространено мнение, что алгоритм окажется практически полезным для вычислительной

задачи только в том случае, если его сложность растет полиномиально относительно размера входа (в то же время продолжается полемика, в ходе которой выдвигаются серьезные контраргументы утверждению о том, что *полиномиальный* и *практический* – синонимы).

Определение 1.11. Алгоритм называется *экспоненциальным*, если его асимптотическая временная сложность $T(l)=\Omega(\exp(l))$, где $\exp(l)$ – экспоненциальная функция.

Напомним, что *экспоненциальная функция* – это показательная функция следующего вида: $\exp(l)=\beta^l$, $\beta>1$, β – константа.

Например, функции $2^l, 3^l$ являются экспоненциальными функциями (экспонента – это показательная функция e^l, где $e\approx 2{,}718281$ – основание натурального логарифма).

Любая экспоненциальная функция асимптотически растет быстрее полиномиальной функции:

$$\lim_{l\to\infty}\frac{l^\alpha}{\beta^l}=0.$$

Функция $l!$ асимптотически растет быстрее, чем β^l, но медленнее, чем l^l:

$$\lim_{l\to\infty}\frac{\beta^l}{l!}=0,$$

$$\lim_{l\to\infty}\frac{l!}{l^l}=0.$$

Определение 1.12. Алгоритм называется *псевдополиномиальным*, если для любой его индивидуальной задачи I размерности l его асимптотическая временная сложность $T(l)=\Theta\left(p\left(l,\left|\mathrm{Max}(I)\right|\right)\right)$, где p – полином от двух переменных:

1) размерности задачи l (длина в битах входных данных индивидуальной задачи I);

2) $\left|\mathrm{Max}(I)\right|$ – значение наибольшего по абсолютной величине числового параметра индивидуальной задачи I (если у задачи несколько числовых параметров, то иногда берут среднее из них).

Для задач, имеющих числовые параметры, псевдополиномиальные алгоритмы на практике ведут себя как экспоненциальные только при очень больших значениях числовых параметров индивидуальных задач. Во всех случаях, кроме очень больших значений числового параметра (которые могут и не встречаться в реальных задачах), они работают, как полиномиальные.

Различие между полиномиальными и экспоненциальными алгоритмами заметно при решении задач большой размерности, кроме того, как было показано ранее (таблица 1.3.), полиномиальные алгоритмы лучше используют успехи

технологий. На практике при решении большинства задач, как только обнаруживается некоторый полиномиальный алгоритм, так сразу же различные исследователи улучшают идею алгоритма и степень полинома быстро претерпевает ряд уменьшений. Обычно окончательно получается степень роста $O\left(n^3\right)$ или лучше. Экспоненциальные алгоритмы, напротив, требуют на практике столько же времени, сколько и в теории, и от них, как правило, тут же отказываются, как только для той же задачи обнаруживается полиномиальный алгоритм.

Следует отметить, что большинство экспоненциальных алгоритмов – это просто варианты полного перебора [8].

С другой стороны, некоторые экспоненциальные алгоритмы достаточно эффективны на практике, когда размеры решаемых задач невелики. Например, при $n \le 20$ функция $f(n) = 2^n$ ведет себя лучше, чем функция $f(n) = n^5$ (таблица 1.1). Или будет ли алгоритм с оценкой n^{80} иметь практическое применение, если время, необходимое для решения индивидуальной задачи размера $n \le 20$ уже выражается астрономическим числом, и может оказаться, что некоторый экспоненциальный алгоритм работает лучше при всех разумных входных? Кроме того, известны некоторые экспоненциальные алгоритмы (например, симплекс-метод), весьма хорошо зарекомендовавшие себя на практике. Дело в том, что трудоемкость определена как мера поведения алгоритма в наихудшем случае.

Утверждение о том, что алгоритм имеет трудоемкость $f(l) = 2^l$, означает, что решение по крайней мере для одного входа задачи размерности l требуется времени порядка 2^l, но на самом деле может оказаться, что для большинства других задач затраты времени значительно меньше.

1.1.2.6. Примеры получения оценок вычислительной сложности алгоритма

Задача 1.1. *Сумма элементов массива*.

На вход поступает натуральное число n и массив чисел $a_1, a_2, \ldots, a_n$. Необходимо найти сумму: $S = a_1 + a_2 + \ldots + a_n$. Определить асимптотическую сложность алгоритма. Какой это алгоритм: полиномиальный или экспоненциальный?

Решение.

Входные данные: n и $a_1, a_2, \ldots, a_n$. Рассмотрим бинарную систему кодирования входных данных. Размерность задачи:

$$l = \lfloor \log_2 n + 1 \rfloor + C_1 \cdot n, \text{ где}$$

$$C_1 = \max_{1 \le i \le n} \lfloor \log_2 |a_i| + 1 \rfloor - \text{константа.}$$

При $n \to +\infty$ верно, что $\lfloor \log_2 n + 1 \rfloor < n$. Поэтому при $n \to +\infty$ получим, что справедливы неравенства:

$$C_1 \cdot n + 1 \le l < n + C_1 \cdot n = (C_1 + 1) \cdot n$$

$$C_1 \cdot n + 1 \le l < (C_1 + 1) \cdot n$$

$$\frac{1}{C_1 + 1} \cdot l < n \le \frac{l - 1}{C_1} < \frac{1}{C_1} \cdot l$$

$$n = \Theta(l), \quad (l = \Theta(n))$$

Предполагаем, что в результате суммирования не произойдет переполнения: все промежуточные результаты вычислений вмещаются в C_1 бит.

Так как каждое число занимает C_1 бит, то сложение двух чисел будет выполнено:

1) за время равное 1, если для РАМ-машины выбрана равномерная мера измерения времени выполнения операции;

2) за время равное C_1, если для РАМ-машины выбрана логарифмическая мера измерения времени выполнения операции.

Вычислительная сложность алгоритма при равномерной мере:

$$T(l) = 1 \cdot C_2 \cdot n < C_2 \cdot \frac{1}{C_1} \cdot l = \mathrm{O}(l).$$

Вычислительная сложность алгоритма при логарифмической мере:

$$T(l) = C_1 \cdot (C_2 \cdot n) < C_1 \cdot C_2 \cdot \frac{1}{C_1} \cdot l = C_2 \cdot l = \mathrm{O}(l).$$

В обоих случаях (равномерная и логарифмическая меры) при бинарной системе кодирования входных данных алгоритм – полиномиальный.

Задача 1.2. *Факториал числа*.

На вход поступает единственное натуральное число n. Необходимо вычислить: $n! = 1 \cdot 2 \cdot \ldots \cdot n.$ Определить асимптотическую временную сложность алгоритма. Какой это алгоритм: полиномиальный или экспоненциальный?

Решение.

Входные данные: единственное натуральное число n.

1) Рассмотрим сначала унарную систему кодирования входных данных. При унарной системе кодирования размерность задачи $l = n.$

Число мультипликативных операций умножения равно n (оценку выполним по числу операций умножения, а также учтем начальное присваивание и запись результата в память). Так как каждое число занимает l бит, то умножение двух чисел будет выполнено:

– за время равное 1, если для РАМ-машины выбрана равномерная мера измерения времени выполнения операции;

– за время равное максимуму числа бит операндов, если для РАМ-машины выбрана логарифмическая мера измерения времени выполнения операции.

Вычислительная сложность алгоритма при равномерной мере:

$$T(l) = 1 \cdot l = \Theta(l).$$

Вычислительная сложность алгоритма при логарифмической мере:

$$T(l) = \left(\lfloor \log_2 1 + 1 \rfloor + \lfloor \log_2 2 + 1 \rfloor + \lfloor \log_2 3 + 1 \rfloor\right) +$$
$$+\left(\lfloor \log_2 3! + 1 \rfloor + \lfloor \log_2 4! + 1 \rfloor + \cdots + \lfloor \log_2 n! + 1 \rfloor\right) \le$$
$$\le (1 + 2 + 2) + \left(\log_2 3! + \log_2 4! + \cdots + \log_2 n! + (n-3)\right) \le$$
$$\le \sum_{i=3}^{n} \log_2 i! + (n-3) + 5.$$

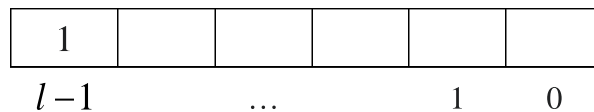
Так как $\log_2 x! = \Theta(x \cdot \log x)$, то

$$T(l) \le (3 + 4 + \cdots + n) \cdot const_1 \cdot \log_2 n + (n-3) + 5 \le$$
$$\le const_2 \cdot n^2 \cdot \log_2 n + (n-3) + 5 = \mathrm{O}\left(l^2 \cdot \log_2 l\right).$$

В обоих случаях (равномерная и логарифмическая меры) при унарной системе кодирования входных данных алгоритм – полиномиальный.

2) Рассмотрим теперь бинарную систему кодирования входных данных. При бинарной системе кодирования входных данных размерность задачи $l = \lfloor \log_2 n + 1 \rfloor$.

Задаче размерности l могут соответствовать несколько входов (это множество натуральных чисел, компактная запись которых в двоичном представлении содержит ровно l бит):



Все возможные входы для задачи размерности l описываются неравенствами:

$$2^{l-1} \le n \le 2^l - 1.$$

Для ассимптотической оценки временной сложности алгоритма в худшем случае необходимо найти такой вход, для которого число мультипликативных операций (умножение) наибольшее. Очевидно, что наибольшее число операций умножения для задачи размерности l будет для входа: $n = 2^l - 1$.

Необходимо также учесть, что длина чисел растет. Например, на последнем шаге мы умножаем число n (в нем столько бит, сколько на входе алгоритма) на число $(n-1)!$, которое гораздо длиннее. Так, значение $21!$ уже не помещается в int64.

Число мультипликативных операций умножения равно $2^l - 1$.

Вычислительная сложность алгоритма при равномерной мере:

$$T(l) = 1 \cdot \left(2^l - 1\right) = \Theta\left(2^l\right).$$

Вычислительная сложность алгоритма при логарифмической мере (с учетом того, что $\log_2 x! = \Theta\left(x \cdot \log x\right)$:

$$T(l) = \left(\lfloor \log_2 1+1 \rfloor + \lfloor \log_2 2+1 \rfloor + \lfloor \log_2 3+1 \rfloor\right) +$$
$$+\left(\lfloor \log_2 3!+1 \rfloor + \lfloor \log_2 4!+1 \rfloor + \cdots + \left\lfloor \log_2 \left(2^l - 1\right)!+1 \right\rfloor\right) >$$
$$> 5 + \left(3 + 4 + \cdots + \left(2^l - 1\right)\right) \cdot \log_2 3 \cdot const_1 =$$
$$= \left(2^{l-1}+1\right) \cdot \left(2^l - 3\right) \cdot const_2 + 5.$$

В обоих случаях (равномерная и логарифмическая меры) при бинарной системе кодирования входных данных алгоритм вычисления факториала – экспоненциальный алгоритм.

Задача 1.3. *Проверить, является ли натуральное число n простым.*

Алгоритм: проверить, есть ли среди чисел $2,3,\ldots,\sqrt[2]{n}$ хотя бы одно, делящее n. Определить асимптотическую сложность алгоритма. Какой это алгоритм: полиномиальный или экспоненциальный?

Решение.

Оценку проведем по числу мультипликативных операций деления.

1) Если в качестве размерности задачи l взять само число n (унарная система кодирования входа), то не получится выписать точную формулу для временной сложности $T(l)$.

В таблице 1.4 приведено число операций деления, которые нужно выполнить алгоритму для проверки на простоту числа l (запись «$l/2-$да» означает, что l делится без остатка на 2).

Таблица 1.4 – Число операций деления при унарной системе кодирования.

l	111	112	113	114	115	116	117	118
$T(l)$	**2** $l/2-$нет $l/3-$да	**1** $l/2-$да	**9** $l/2-$нет $l/3-$нет … $l/10-$нет	**1** $l/2-$да	**4** $l/2-$нет $l/3-$нет $l/4-$нет $l/5-$да	**1** $l/2-$да	**2** $l/2-$нет $l/3-$да	**1** $l/2-$да

Из даных таблицы 1.4 видно, что функция $T(l)$ не является монотонной, поэтому можно лишь показать, что при $l \to +\infty$ для $T(l)$ справедлива оценка сверху: $T(l) \le \sqrt[2]{l} - 1$. Алгоритм определения простоты числа при унарной

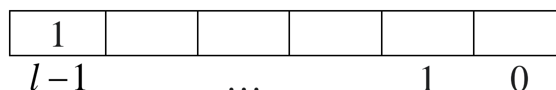
системе кодирования входных данных и равномерной мере для РАМ-машины является полиномиальным.

2) Рассмотрим теперь бинарную систему кодирования входных данных.

При бинарной системе кодирования входных данных размерность задачи

$$l = \lfloor \log_2 n + 1 \rfloor.$$

Задаче размерности l соответствуют несколько входов (это множество натуральных чисел, компактная запись которых в двоичном представлении содержит ровно l бит):



Все возможные входы для задачи размерности l описываются неравенствами:

$$2^{l-1} \le n \le 2^l - 1 < 2^l.$$

Для асимптотической оценки временной сложности алгоритма в худшем случае необходимо найти такой вход, для которого число мультипликативных операций (деления) наибольшее. В таблице 1.5 показано, сколько операций в худшем случае будет выполнено для задачи размерности l при бинарной системе кодирования входа.

Таблица 1.5 – Число операций деления при бинарной системе кодирования.

l бит	**2** _ _	**3** _ _ _	**4** _ _ _ _	**5** _ _ _ _ _	**6** _ _ _ _ _ _	**7** _ _ _ _ _ _ _
диапазон входов для задачи размерности l	2 – 3	4 – 7	8 – 15	16 – 31	32 – 63	64 – 27
число из диапазона, для которого самые большие затраты	3	7	13	31	61	127

$T(l)$	**1**	**1**	**2**	**4**	**6**	**10**

Постулат Бертрана гласит, что при любом целом $N>1$ имеется простое число, принадлежащее интервалу $(N, 2\cdot N)$.

Пусть p_l – самое большое простое число для задачи размерности l, тогда число выполненных делений при проверке этого числа известно и оно равно $\left\lfloor \sqrt[2]{p_l} \right\rfloor - 1$.

Так как справедливы неравенства

$$2^{l-1} \le p_l < 2^l,$$

то верно следующее соотношение

$$\left\lfloor \sqrt[2]{p_l} \right\rfloor - 1 \ge \left\lfloor 2^{\frac{l-1}{2}} \right\rfloor - 1,$$

учитывая которое, получаем следующие неравенства:

$$T(l) \ge \left\lfloor \sqrt[2]{p_l} \right\rfloor - 1 \ge \left\lfloor 2^{\frac{l-1}{2}} \right\rfloor - 1 >$$

$$> 2^{\frac{l-1}{2}} - 2 = 2^{\frac{l}{2}-\frac{1}{2}} - 2 = \frac{2^{\frac{l}{2}}}{\sqrt[2]{2}} - 2 =$$

$$= \left(\frac{1}{\sqrt[2]{2}} - \frac{2}{2^{\frac{l}{2}}} \right) \cdot 2^{\frac{l}{2}} = C_1 \cdot 2^{\frac{l}{2}},$$

где при $4 \le l \le +\infty$, выполняется

$$0 < \left(\frac{1}{\sqrt[2]{2}} - \frac{1}{2} \right) \le C_1 < \frac{1}{\sqrt[2]{2}}.$$

В тоже время для всех входных данных задачи размерности l справедливо:

$$T(l) \le \left\lfloor 2^{\frac{l}{2}} \right\rfloor - 1 \le 2^{\frac{l}{2}} - 1 < 2^{\frac{l}{2}}.$$

Таким образом, при бинарной системе кодирования входных данных задачи получены оценки сверху и снизу для временной сложности алгоритма проверки числа на простоту:

$$C_1 \cdot 2^{\frac{l}{2}} < T(l) < 2^{\frac{l}{2}}, l \ge 4.$$

Откуда можно сделать вывод, что алгоритм при бинарной системе кодирования входных данных и равномерной мере для РАМ-машины является экспоненциальным:

$$T(l)=\Theta\left(\sqrt{2}^{l}\right)$$

или

$$T(l)=\Theta\left(\beta^{l}\right),\ \text{где } \beta=\sqrt{2}$$

Упражнение 1.1. Какие из асимптотик верны для задачи 1.3 при бинарной системе кодирования входных данных: $T(l)=O\left(2^{l}\right), T(l)=\Omega\left(2^{l}\right), T(l)=\Theta\left(2^{l}\right)$?

1.2. Рекуррентные уравнения для алгоритмов поиска и внутренней сортировки

1.2.1. Основные понятия

Определение 1.13. Соотношения, которые связывают одни и те же функции, но с различными значениями аргументов, называются *рекуррентными соотношениями* или *рекуррентными уравнениями*.

Определение 1.14. Рекуррентное уравнение будем называть *правильным*, если значения аргументов у любой из функций в правой части соотношения меньше значения аргументов у любой из функций в левой части соотношения; если аргументов несколько, то достаточно уменьшения одного из них.

Определение 1.15. Правильное рекуррентное уравнение называется *полным*, если оно определено для всех допустимых значений аргументов.

$$\begin{cases} T(n)=2\cdot T\left(\dfrac{n}{2}\right)+100\cdot n^{2},\ n=2^{k},\ k\geq 1; \\ T(1)=7. \end{cases}$$

Определите, какие из следующих уравнений являются полными?

1) $\begin{cases} T(n)=T(n-1)+C_{1}\cdot n,\ n\geq 2 \\ T(0)=C_{2} \end{cases}$

2) $\begin{cases} T(n)=T(n-1)+C_{1}\cdot n,\ n\geq 2 \\ T(1)=C_{2} \end{cases}$

3) $\begin{cases} T(n)=T(n-1)+C_{1}\cdot n,\ n\geq 1 \\ T(0)=C_{2} \end{cases}$

4) $\begin{cases} T(n) = 2 \cdot T(n-2) + C_1 \cdot (n-1), n \geq 2 \\ T(1) = C_2 \\ T(0) = C_3 \end{cases}$

5) $\begin{cases} T(n) = T(n-2) + C_1 \cdot n, n \geq 1 \\ T(0) = C_2 \end{cases}$

Полными являются рекуррентные соотношения 2), 3) и 4).

Рекуррентное соотношение 1) не может быть вычислено для $n = 2$, так как не определено значение $T(1)$, которое требуется для вычисления $T(2)$.

Рекуррентное соотношение 5) не может быть вычислено для нечетных значений n, так как не определено значение $T(1)$, которое требуется для их вычисления.

Рекуррентные соотношения могут задавать правило вычисления некоторой величины и могут включать в себя несколько рекурсивных функций и несколько ограничений (такие задачи сводятся к решению подзадач меньшего размера).

1) Для вычисления $F(n) = n!$, получим следующее рекуррентное соотношение:

$$\begin{cases} F(n) = F(n-1) \cdot n, n \geq 1 - \text{ рекурсивная функция} \\ F(0) = 1 - \text{ ограничение} \end{cases}$$

2) Для вычисления последовательности чисел Фибоначчи, получим следующее рекуррентное соотношение:

$$\begin{cases} F(n) = F(n-1) + F(n-2), n \geq 2 \text{ (рекурсивная функция)} \\ F(0) = 0 \text{ (ограничение 1)} \\ F(1) = 1 \text{ (ограничение 2)} \end{cases}$$

3) Для вычисления $F(n) = 2^n$, получим следующее рекуррентное соотношение:

$$\begin{cases} F(n) = \left(F\left(\frac{n}{2}\right)\right)^2, n - \text{ четно (рекурсивная функция 1)} \\ F(n) = \left(F\left(\frac{n-1}{2}\right)\right)^2 \cdot 2, n - \text{ нечетно (рекурсивная функция 2)} \\ F(1) = 2 - \text{ (ограничение 1)} \\ F(0) = 1 - \text{ (ограничение 2)} \end{cases}$$

4) В задаче Ханойские башни, в которой заданы три стержня, на одном из которых лежат n дисков разных диаметров (диск большего диаметра может лежать только под диском меньшего диаметра), рекуррентное соотношение для числа способов перенести все диски с одного стержня на другой (при переносе диск меньшего диаметра переносится на диск большего диаметра), следующее:

$$\begin{cases} F(n) = 2 \cdot F(n-1) + 1, n \geq 1 - \text{ рекурсивная функция} \\ F(1) = 1 - \text{ ограничение} \end{cases}$$

В последующем мы покажем, что решением этого рекуррентного соотношения является функция $F(n) = 2^n - 1$.

С другой стороны, рекуррентные соотношения мы будем использовать для оценки времени работы алгоритма. Для этого по алгоритму решения задачи выпишем рекуррентное соотношение для числа операций алгоритма в худшем случае, решим его и определим временную сложность алгоритма. Например, если обозначить через $T(n)$ число арифметических операций сортировки «слиянием», то получим следующее соотношение:

$$\begin{cases} T(n) = C_1 + 2 \cdot T\left(\frac{n}{2}\right) + C_2 \cdot n, n = 2^k, k \geq 1; \\ T(1) = C_3. \end{cases}$$

В последующем мы покажем, что решением этого рекуррентного соотношения является функция

$$T(n) = C_2 \cdot n \cdot log_2 n + (C_3 + C_1) \cdot n - C_1,$$

т. е. $T(n) = \mathrm{O}(n \cdot \log n), n = \Theta(l)$ и алгоритм сортировки слиянием – полиномиальный алгоритм.

1.2.2. Методы решения рекуррентных уравнений

Существуют разные методы решения рекуррентных уравнений, например, метод итераций, метод рекурсивных деревьев, метод подстановок [1, 5].

Рассмотрим наиболее распространенный из них – метод итераций.

При решении методом итераций рекуррентное уравнение расписывается через множество других, после чего осуществляется суммирование полученного выражения.

Продемонстрируем метод итераций.

Пример 1.6. Решим следующее рекуррентное уравнение:

$$\begin{cases} T(n) = 2 \cdot T\left(\frac{n}{2}\right) + a \cdot n^2, n = 2^k, k \geq 1; \\ T(1) = b. \end{cases}$$

Решение.

$$T(n) = \underbrace{2 \cdot T\left(\frac{n}{2}\right) + a \cdot n^2}_{1\text{–й шаг}} = \left[T\left(\frac{n}{2}\right) = 2 \cdot T\left(\frac{n}{2^2}\right) + a \cdot \left(\frac{n}{2}\right)^2 \right] =$$

$$= \underbrace{2^2 \cdot T\left(\frac{n}{2^2}\right) + a \cdot \frac{n^2}{2^1} + a \cdot \frac{n^2}{2^0}}_{2\text{–й шаг}} = \ldots =$$

$$= \underbrace{2^m \cdot T\left(\frac{n}{2^m}\right) + a \cdot n^2 \cdot \left(\frac{1}{2^0} + \frac{1}{2^1} + \ldots + \frac{1}{2^{m-1}}\right)}_{m\text{–й шаг}} =$$

$$= 2^m \underbrace{\cdot T\left(\frac{n}{2^m}\right) + a \cdot n^2 \cdot 2 \cdot \left(1 - \frac{1}{2^m}\right)}_{m\text{–й шаг}} = \left[\frac{n}{2^m} = 1, n = 2^m\right]$$

$$= \underbrace{n \cdot T(1) + a \cdot n^2 \cdot 2 \cdot \left(1 - \frac{1}{n}\right)}_{\log_2 n\text{–й шаг}} =$$

$$= b \cdot n + 2 \cdot a \cdot n^2 - 2 \cdot a \cdot n = 2 \cdot a \cdot n^2 - (b - 2 \cdot a) \cdot n$$

Пример 1. 7. Решим рекуррентное сотношение, которое было выписано для задачи Ханойские башни:

$$\begin{cases} F(n) = 2 \cdot F(n-1) + 1, n \geq 1 \\ F(1) = 1 \end{cases}$$

Решение.

$$F(n) = \underbrace{2 \cdot F(n-1) + 1}_{1\text{-й шаг}} = \left[F(n-1) = 2 \cdot F(n-2) + 1\right] = \underbrace{2^2 \cdot F(n-2) + 2^1 + 2^0}_{2\text{-й шаг}} =$$

$$\underbrace{2^3 \cdot F(n-3) + 2^2 + 2^1 + 2^0}_{3\text{-й шаг}} = \cdots = \underbrace{2^m \cdot F(n-m) + \left(2^{m-1} + \cdots + 2^1 + 2^0\right)}_{m\text{-й шаг}} =$$

$$= \underbrace{2^m \cdot F(n-m) + \left(2^m - 1\right)}_{m\text{-й шаг}} = \left[F(n-m) = F(1), n - m = 1, m = n - 1\right] =$$

$$\underbrace{= 2^{n-1} \cdot F(1) + \left(2^{n-1} - 1\right)}_{(n\text{-}1)\text{-й шаг}} = 2 \cdot 2^{n-1} - 1 = 2^n - 1$$

1.2.3. Рекуррентные уравнения для поиска максимального и минимального элементов массива

В дальнейшем будем предполагать (если не оговорено иное), что область определения функции $T(n)$ – это множество неотрицательных целых чисел $\{0, 1, 2, \ldots\}$ и сама функция $T(n)$ принимает только неотрицательные целочисленные значения. Это допущение вызвано тем, что функция $T(n)$ будет нами чаще всего использоваться для описания времени работы алгоритма.

Задан массив из n элементов. Рассмотрим три алгоритма *поиска максимального и минимального элементов массива*, сравним эти алгоритмы по числу выполненных ими операций сравнения.

Алгоритм 1 (последовательный поиск max и min)

1) Первый элемент массива полагаем в качестве max и min.

2) Каждый из оставшихся $n-1$ элементов сравниваем с max и min, и, если надо, то корректируем значения max и min.

```
template <class Iter>
std::pair<Iter, Iter> MinMaxElement2(Iter begin, Iter end) {
    std::pair<Iter, Iter> res = std::make_pair(begin, begin);
    for (Iter it = begin + 1; it != end; ++it) {
        if (*it < *res.first) {
            res.first = it;
        }
        if (*it > *res.second) {
            res.second = it;
        }
    }
    return res;
}
```

Не сложно непосредственно подсчитать число операций сравнения, которое будет выполнено Алгоритмом 1:

$$0 + 2 \cdot (n-1) = 2 \cdot n - 2.$$

Можно также выписать рекуррентное соотношение для числа операций сравнения. Пусть $T(n)$ – число операций сравнения, которые нужно выполнить для того, чтобы Алгоритм 1 нашел максимальный и минимальный элемент среди n элементов массива.

Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = T(n-1) + 2, n \geq 2 \\ T(1) = 0 \end{cases}$$

Решим это рекуррентное уравнение методом итераций:

$$T(n) = \underbrace{T(n-1)+2}_{\text{1-й шаг}} = \left[T(n-1) = T(n-2)+2\right] = \underbrace{T(n-2)+2+2}_{\text{2-й шаг}} =$$

$$\underbrace{T(n-3)+2+2+2}_{\text{3-й шаг}} = \cdots = \underbrace{T(n-m)+2\cdot m}_{m\text{-й шаг}} = \left[T(n-m) = T(1), n-m=1, m=n-1\right] =$$

$$\underbrace{=T(1)+2\cdot(n-1)}_{(n\text{-1})\text{-й шаг}} = 0+2\cdot(n-1) = 2\cdot n-2.$$

В алгоритме последовательного поиска можно получить оценку $2\cdot n-3$, выбирая на начальном этапе за одно сравнение из первых двух элементов массива максимальный и минимальный элемент. Затем оставшиеся $(n-2)$ элемента сравниваются с максимальным и минимальным: $1+2\cdot(n-2) = 2\cdot n-3$.

Алгоритм 2 (поиск max и min деление пополам)

1) Разделим массив на две части (предположим, что $n = 2^k$).

2) В каждой из частей найдем этим же алгоритмом локальные $(\max_1, \min_1)$ и $(\max_2, \min_2)$.

3) Полагаем

$$\max = \text{наибольший}(\max_1, \max_2), \quad \min = \text{наименьший}(\min_1, \min_2).$$

```
template <class Iter>
std::pair<Iter, Iter> MinMaxElement(Iter begin, Iter end) {
    size_t n = end - begin;
    if (n == 1) {
        return std::make_pair(begin, begin);
    } else if (n == 2) {
        Iter first = begin;
        Iter second = begin + 1;
        if (*first < *second) {
            return std::make_pair(first, second);
        } else {
            return std::make_pair(second, first);
        }
    } else {
        Iter mid = begin + n / 2;
        std::pair<Iter, Iter> r1 = MinMaxElement(begin, mid);
        std::pair<Iter, Iter> r2 = MinMaxElement(mid, end);
        return std::make_pair(
            *r1.first < *r2.first ? r1.first : r2.first,
            *r1.second > *r2.second ? r1.second : r2.second
        );
    }
}
```

Пусть $T(n)$ – число операций сравнения для того, чтобы Алгоритм 2 нашел максимальный и минимальный элемент.

Тогда справедливо рекуррентное соотношение:

$$\begin{cases} T(n) = 2\cdot T\left(\dfrac{n}{2}\right) + 2, n = 2^k, k \ge 2 \\ T(2) = 1 \end{cases}$$

Решим это рекуррентное уравнение методом итераций:

$$T(n) = \underbrace{2\cdot T\left(\frac{n}{2}\right) + 2}_{1\text{–й шаг}} = \left[T\left(\frac{n}{2}\right) = 2\cdot T\left(\frac{n}{2^2}\right) + 2\right] =$$

$$= \underbrace{2^2\cdot T\left(\frac{n}{2^2}\right) + 2^2 + 2^1}_{2\text{–й шаг}} = \ldots = \underbrace{2^m\cdot T\left(\frac{n}{2^m}\right) + 2^m + \cdots + 2^2 + 2^1}_{m\text{–й шаг}} =$$

$$= \underbrace{2^m\cdot T\left(\frac{n}{2^m}\right) + \frac{2\cdot\left(2^m - 1\right)}{2-1}}_{m\text{–й шаг}} = \left[\frac{n}{2^m} = 2, \frac{n}{2} = 2^m, m = \log_2 n - 1\right] =$$

$$= \underbrace{\frac{n}{2}\cdot T(2) + 2\cdot\left(\frac{n}{2} - 1\right)}_{(\log_2 n - 1)\text{–й шаг}} = \frac{n}{2}\cdot 1 + 2\cdot\left(\frac{n}{2} - 1\right) = \frac{3}{2}\cdot n - 2$$

Получили, что Алгоритм 2 имеет меньшую константу $(3/2)$ при *n*. Однако несмотря на то, что Алгоритм 2, основанный на принципе «разделяй и властвуй», выполняет меньше сравнений, на практике он может быть медленнее из-за накладных расходов, вызванных рекурсией. Рассмотрим еще один алгоритм, который выполняет $3\cdot n/2$ сравнений, но не является рекурсивным.

Алгоритм 3

(нерекурсивная реализация поиска max и min,
число сравнений $3\cdot n/2$)

1) Среди первых двух элементов массива за одно сравнение находим max и min.

2) Оставшиеся $n-2$ элемента делим последовательно на группы по два элемента.

3) В каждой группе из двух элементов сначала за одно сравнение находим $(\max', \min')$, затем еще за два сравнения корректируем max и min (таким образом, для первых элементов массива, включая эту двойку элементов, задача поиска максимума и минимума будет решена):

$$\max = \text{наибольший}(\max', \max);$$
$$\min = \text{наименьший}(\min', \min).$$

Число выполненных операций сравнения в Алгоритме 3 равно

$$\left\lceil \frac{3\cdot(n-2)}{2} \right\rceil + 1.$$

В C ++ не рекурсивный алгоритм, который выполняет $\sim 3\cdot n/2$ сравнений, реализован как функция `std::minmax_element()` библиотеки STL.

1.2.4. Рекуррентные уравнения для поиска элемента в упорядоченном массиве

Задан упорядоченный массив A из n элементов: $a_0 \le a_1 \le \ldots \le a_{n-1}$. В массиве элементы могут повторяться. Необходимо определить, есть ли среди элементов массива заданный элемент x.

Алгоритм бинарного поиска
(дихотомия / поиск делением пополам)

Определяем границы $[q, r)$ области поиска как $q = 0, r = n$.

1. Определяем индекс центрального элемента области поиска:

$$k = \left\lfloor \frac{q+r}{2} \right\rfloor.$$

2. Сравниваем a_k элемент последовательности и число x.

– Если $x = a_k$, то поиск завершен.

– Если $x < a_k$, то продолжаем аналогичные действия, изменяя правую границу области поиска на k.

– Если $x > a_k$, то продолжаем аналогичные действия, изменяя левую границу области поиска на $k+1$.

3. Алгоритм прекращает работу, как только будет найден требуемый элемент либо станет верным равенство $q = r$ (в этом случае элемента в последовательности нет).

```
def BinarySearch(a, x):
q = 0, r = len(a)
while q < r:
k = (q + r) // 2
if x == a[k]:
return True
else if x < a[k]:
r = k
else: # x > a[k]
q = k + 1
return False
```

Если в упорядоченном массиве требуется найти индекс первого элемента, больше, чем *x*, либо равного ему (**LowerBound**), то в случае `if x == a[k]`, поступаем также, как при движении налево. В случае отсутствия в массиве подходящих элементов договоримся, что возвращаемое значение будет равно *n*.

```
def LowerBound(a, x):
        q = 0, r = len(a)
        while q < r:
                k = (q + r) // 2
                if x ≤ a[k]:
                        r = k
                else: # x > a[k]
                        q = k + 1
return q
```

Если в упорядоченном массиве требуется найти индекс первого элемента, строго больше, чем (**UpperBound**), то в случае `if x == a[k]`, поступаем также, как при движении направо. В случае отсутствия в массиве подходящих элементов договоримся, что возвращаемое значение будет равно *n*.

```
def UpperBound(a, x):
        q = 0, r = len(a)
        while q < r:
                k = (q + r) // 2
                if x < a[k]:
                        r = k
                else: # x ≥ a[k]
                        q = k + 1
return q
```

Пусть $T(n)$ – число операций сравнения для того, чтобы бинарный поиск завершил свою работу.

Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = C_1 + 2 \cdot T\left(\frac{n}{2}\right), n = 2^k, k \ge 2 \\ T(1) = C_2 \end{cases}$$

Решим это рекуррентное уравнение методом итераций:

$$T(n) = \underbrace{C_1 + 2 \cdot T\left(\frac{n}{2}\right)}_{1\text{–й шаг}} = \left[T\left(\frac{n}{2}\right) = C_1 + T\left(\frac{n}{2^2}\right)\right] =$$

$$= \underbrace{C_1 + C_1 + T\left(\frac{n}{2^2}\right)}_{2\text{–й шаг}} = \ldots = \underbrace{m \cdot C_1 + T\left(\frac{n}{2^m}\right)}_{m\text{–й шаг}} = \left[\frac{n}{2^m} = 1, n = 2^m, m = \log_2 n\right] =$$

$$= \underbrace{C_1 \cdot \log_2 n + T(1)}_{(\log_2 n)\text{–й шаг}} = C_1 \cdot \log_2 n + C_2.$$

Вычислительная сложность алгоритма в худшем случае $T(l)=\Theta(\log l), l=\Theta(n)$ – алгоритм полиномиальный.

В стандартной библиотеке языка **С++**

Функция `std::binary_search` выполняет бинарный поиск и возвращает логическое значение (есть элемент или нет). Функции `std::lower_bound` и `std::upper_bound` действуют аналогично рассмотренным и возвращают итераторы.

В языке **Java**

Для классов `Arrays` и `Collections` определен статический метод `binarySearch`, который совмещает в себе описанные выше функции `BinarySearch` и `LowerBound`, однако является менее гибким (при наличии в массиве нескольких элементов, равных искомому, метод может вернуть индекс любого).

В языке **Python**

Бинарный поиск реализован в стандартном модуле *bisect*.

Задача 1.4. Разработайте алгоритм, позволяющий определить, сколько раз в упорядоченном массиве встречается заданный элемент.

Время работы алгоритма должно быть $\mathrm{O}(\log n)$.

1.2.5. Рекуррентные уравнения для алгоритмов внутренней сортировки

Определение 1.16. Пусть задана последовательность $a_0, a_1, \ldots, a_{n-1}$ из n элементов (записей), выбранных из множества, на котором задан линейный порядок.

Каждая запись a_j имеет ключ k_j, который управляет процессом сортировки.

Задача сортировки заключается в поиске перестановки

$$\pi = \pi_0, \pi_1, \ldots, \pi_{n-1}$$

этих n записей, после которой ключи записей расположились бы, например, в неубывающем порядке (будем предполагать порядок сортировки массива по неубыванию, если не оговорено иное):

$$k_{\pi_0} \le k_{\pi_1} \le \cdots \le k_{\pi_{n-1}}.$$

i	0	1	2	3	4	5	6	7	8	9
k_i	7	2	3	5	1	8	11	4	0	6
π_i	8	4	1	2	7	3	9	0	5	6

$$k_8 \le k_4 \le k_1 \le k_2 \le k_7 \le k_3 \le k_9 \le k_0 \le k_5 \le k_6$$

Определение 1.17. Алгоритм сортировки называют устойчивым (стабильным), если в процессе сортировки относительное расположение элементов с одинаковыми ключами не изменяется:

$$\pi_i < \pi_i, \text{ если } k_{\pi_i} \le k_{\pi_j} \text{ и } i < j.$$

i	0	1	2	3	4	5	6	7	8	9
k_i	7	2_1	2023	607	1	2_2	3_1	4	0	3_2
π_i	8	4	1	5	6	9	7	0	3	2

$$k_8 \le k_4 \le k_1 \le k_5 \le k_6 \le k_9 \le k_7 \le k_0 \le k_3 \le k_2$$

$$0 \le 1 \le 2_1 \le 2_2 \le 3_1 \le 3_2 \le 4 \le 7 \le 607 \le 2023$$

Процесс сортировки данных может быть осуществлен различными алгоритмами. Если объем входных данных позволяет обходиться исключительно основной (оперативной) памятью, то говорят об алгоритмах *внутренней сортировки*, в противном случае – об алгоритмах *внешней сортировки* [1].

Большинство алгоритмов внутренней сортировки относятся к классу *сортировок сравнениями* (англ. comparison sort).

Алгоритмы сортировок сравнениями выполняют только операции сравнения элементов и их перемещения (обмены), но никак не используют их внутреннюю структуру.

Для алгоритмов сортировок сравнениями известна нижняя оценка: $\Omega(n \cdot \log n)$, т. е. нельзя выполнить внутреннюю сортировку n записей асимптотически быстрее, чем за время $n \cdot \log n$.

Рассмотрим в качестве примера устойчивый алгоритм сортировки подсчетом (англ. counting sort), который не принадлежит классу алгоритмов сортировки сравнениями.

Устойчивый алгоритм сортировки подсчетом

В сортировке подсчетом предполагается, что все входные числа целые и принадлежат интервалу от 0 до k, где k – некоторая целая константа.

1) Если все числа $0 \le a_i \le r$, то диапазон возможных значений $[0\,..\,r]$.

2) Если все числа целые, неотрицательные и $0 \le l \le a_i \le r$, то диапазон возможных значений $[0\,..\,r-l]$, а при работе с числом мы вычитаем из него число l.

3) Если все числа целые, но среди них могут быть отрицательные и $l \le a_i \le r$, то диапазон возможных значений $\left[0..r+|l|\right]$, а при работе с числом мы добавляем к нему величину $|l|$.

−3	3	5	−2	−0
0	6	8	1	3

$-3 \le a_i \le 5, |l| = 3,$ диапазон новых значений $-[0..8]$

Алгоритм сортировки подсчетом применяется в случае, когда сортируемые элементы можно отобразить в диапазон возможных значений, который достаточно мал, по сравнению с числом сортируемых элементов.

Если $k = O(n)$, то время работы алгоритма сортировки подсчетом $O(n)$.

Предположим, что элементы массива $A = \{a_0, a_1, a_2, a_3, a_4\}$ лежат в диапазоне $[0..8]$:

	0	1	2	3	4
$a[i]$	0	6_1	6_2	8	1

На 1-м этапе создадим массив $C[0..r]$, где $c[i]$ – количество элементов массива, равных i.

	0	1	2	3	4	5	6	7	8
$c[i]$	1	1	0	0	0	0	2	0	1

```
for i = 0 to r
            c[i] = 0;
      for i = 0 to l - 1
            c[a[i]] = c[a[i]] + 1;
```

На 2-м этапе создадим массив $c[i]$ – количество элементов массива A, которые $\le i$.

	0	1	2	3	4	5	6	7	8
$c[i]$	1	2	2	2	2	2	4	4	5

```
for j = 1 to r
            c[j] = c[j] + c[j - 1];
```

На 3-м этапе просматриваем массив A справа налево и заносим элемент $a[i]$ в массив $A^{\text{сорт}}$ по индексу $c[a[i]]-1$ и уменьшаем значение $c[a[i]]$ на единицу.

	0	1	2	3	4
$a[i]$	0	6_1	6_2	8	1
$a^{\text{сорт}}[i]$	0	1	6_1	6_2	8

```
for i = n - 1 to 0
        {aсорт[c[a[i]]-1] = a[i];
         c[a[i]] = c[a[i]] - 1}
```

Время работы: $\Theta(n+r)$.

Требуемая память: $\Theta(n+r)$.

Среди наиболее популярных алгоритмов сортировки сравнениями можно выделить следующие [6].

1. Сортировка выбором (англ. SelectionSort)
2. Сортировка пузырьком (англ. BubbleSort)
3. Шейкерная сортировка (перемешиванием) (англ. CocktailSort)
4. Сортировка вставками (включением) (англ. InsertionSort)

— $\Theta(l^2), l=\Theta(n)$

5. Сортировка слиянием (англ. MergeSort)
6. Быстрая сортировка Ч. Хоара (англ. QuickSort)
7. Сортировка кучей (пирамидальная) (англ. HeapSort) [1]

— $\Theta(l \cdot \log l), l=\Theta(n)$

1. Сортировка выбором

	0	1	2	3	4	5	6
	2	3	7	14	70	0	1
1:	**0**	3	7	14	70	2	1
2:	**0**	**1**	7	14	70	2	3
3:	**0**	**1**	**2**	14	70	7	3
4:	**0**	**1**	**2**	**3**	70	7	14
5:	**0**	**1**	**2**	**3**	**7**	70	14
6:	**0**	**1**	**2**	**3**	**7**	**14**	70

На первой итерации среди n элементов массива нужно найти элемент с минимальным ключом и поменять его с первым элементом. Теперь первый элемент стоит на своем месте. Повторить описанные действия с оставшимися $n-1$ элементом. Процесс завершается через $n-1$ итерацию. *Особенность*: выпоняется только один обмен элементов массива в памяти компьютера на одну итерацию. Пусть $T(n)$ – число операций сравнения для того, чтобы сортировка выбором завершила свою работу. Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = C_1 \cdot n + T(n-1), \ n \geq 2 \\ T(1) = C_2 \end{cases}$$

$$T(n) = \underbrace{C_1 \cdot n + T(n-1)}_{1\text{–й шаг}} = \left[T(n-1) = C_1 \cdot (n-1) + T(n-2)\right] =$$

$$= \underbrace{C_1 \cdot n + C_1 \cdot (n-1) + T(n-2)}_{2\text{–й шаг}} = \ldots$$

$$= \underbrace{C_1 \cdot n + C_1 \cdot (n-1) + \cdots + C_1 \cdot (n-m+1) + T(n-m)}_{m\text{–й шаг}} =$$

$$= \left[n-m=1; m=n-1\right] = \underbrace{C_1 \cdot (2+3+\cdots+n) + T(1)}_{(n-1)\text{–й шаг}} = \frac{n+2}{2} \cdot (n-1) \cdot C_1 + C_2.$$

Решив уравнение методом итераций, получаем, что $T(n) = \Theta\left(l^2\right), l = \Theta(n)$, алгоритм – полиномиальный.

2. *Сортировка пузырьком*

На первой итерации просматриваем массив справа налево и при каждом шаге меньший из двух соседних элементов перемещается к левой позиции (обменами).

Теперь первый элемент стоит на своем месте.

Повторить описанные действия с оставшимися $n-1$ элементом.

Процесс завершается через $n-1$ итерацию.

Особенность: на каждой итерации могут происходить многочисленные обмены элементов массива в памяти компьютера.

Продемонстрируем алгоритм сортировки пузырьком на следующем примере.

0	1	2	3	4	5	6
2	3	7	14	70	0	1
0	2	3	7	14	70	1
0	1	2	3	7	14	70
0	1	2	3	7	14	70
0	1	2	3	7	14	70
0	1	2	3	7	14	70
0	1	2	3	7	14	70

Пусть $T(n)$ – число операций сравнения, которые нужно выполнить для того, чтобы сортировка пузырьком завершила свою работу.

Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = C_1 \cdot n + T(n-1), \; n \geq 2 \\ T(1) = C_2 \end{cases}$$

Получаем, что $T(n) = \Theta\left(l^2\right), l = \Theta(n)$, алгоритм – полиномиальный.

3. *Шейкерная сортировка*

Отличия от пузырьковой сортировки:

1) Чередование направлений просмотра массива: при движении справа налево «всплывает самый легкий», при движении слева направо – «тонет самый тяжелый».

2) Если при некотором проходе нет ни одного обмена, то сортировка досрочно завершается – массив отсортирован.

3) Сужение области просмотра: фиксируется индекс последнего обмена и при движении в противоположную сторону движение начинается с этого индекса.

Продемонстрируем алгоритм шейкерной сортировки на следующем примере.

0	1	2	3	4	5	6
10	**2**	**3**	**4**	**5**	**6** ←	**7**
2	**10** →	**3**	**4**	**5**	**6**	**7**
2	**3**	**4**	**5**	**6** ←	**7**	**10**
2	**3**	**4**	**5**	**6**	**7**	**10**

Пусть $T(n)$ – число операций сравнения для того, чтобы шейкерная сортировка завершила свою работу.

Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = C_1 \cdot n + C_1 \cdot (n-1) + T(n-2), \ n \geq 3 \\ T(2) = C_2 \\ T(1) = C_3 \end{cases}$$

Получаем, что $T(n) = \Theta\left(l^2\right), l = \Theta(n)$, алгоритм – полиномиальный.

4. Сортировка вставками

Пусть элементы $a_0, a_1, \ldots, a_{i-1}$ уже упорядочены на предыдущих итерациях этим же алгоритмом (первоначально в качестве упорядоченной части можно взять первый элемент массива).

На очередной итерации надо взять элемент a_i (первый элемент из еще неупорядоченной части) и включить его в нужное место упорядоченной последовательности $a_0, a_1, \ldots, a_{i-1}$ так, чтобы первые i элементов массива были упорядочены.

Данный процесс называют *просеиванием* (выполняется прямое или двоичное включение).

Прямое включение можно выполнить, например, следующим образом: на очередной итерации просматриваем элементы массива $a_0, a_1, \ldots, a_{i-1}$ справа налево; если текущий элемент $a_j > a_i \ (0 \leq j \leq i-1)$, то перемещаем a_j на одну

позицию вправо; если $a_j \le a_i$, то на позицию $j+1$ ставим элемент a_i и завершаем просеивание.

При двоичном включении сначала, используя дихотомию, находят место элемента a_i в упорядоченном массиве $a_0, a_1, \dots, a_{i-1}$, затем сдвигают все элементы, большие элемента a_i на одну позицию вправо, а затем на свободное место добавляют элемент a_i. В обоих случаях при просеивании одного элемента число операций (сравнения, перемещения элементов) в худшем случае – $O(n)$.

Следует отметить, что алгоритм сортировки вставками является устойчивым, а в случае, когда элементы исходного массива были отсортированы, при прямом включении алгоритм сортировки вставками завершит свою работу за линейное от числа вершин время. Однако, в худшем случае, когда входные данные отсортированы в обратном порядке, авлгоритм будет работать за квадратичное время.

Продемонстрируем алгоритм сортировки вставками на следующем примере.

1	2	3	4	5	6	7
2	**3**	**1**	**14**	**7**	**0**	**4**
2	**3**	**1**	**14**	**7**	**0**	**4**
1	**2**	**3**	**14**	**7**	**0**	**4**
1	**2**	**3**	**14**	**7**	**0**	**4**
1	**2**	**3**	**7**	**14**	**0**	**4**
0	**1**	**2**	**3**	**7**	**14**	4
0	**1**	**2**	**3**	4	**7**	**14**

Пусть $T(n)$ – число операций сравнения для того, чтобы сортировка вставками завершила свою работу. Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = T(n-1) + C_1 \cdot n, \ n \ge 2 \\ T(1) = C_2 \end{cases}$$

Получаем, что $T(n) = \Theta(l^2), l = \Theta(n)$, алгоритм – полиномиальный.

5. Сортировка слиянием

1) Делим последовательность элементов на две части (пусть q и r левая и

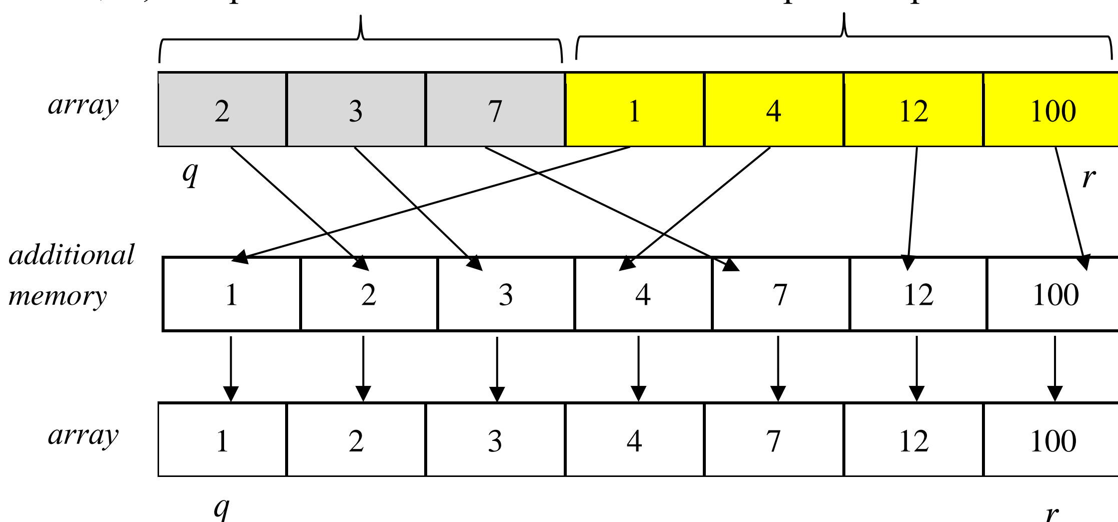
правая границы рассматриваемой последовательности; элементы массива по индексам q и r также включаются в рассмотрение; если сортируемая последовательность состояла из n элементов, то первая часть может содержать $\lfloor n/2 \rfloor$ первых элементов, а вторая часть – оставшиеся; порядок следования элементов в каждой из полученных частей совпадает с их порядком следования в исходной последовательности). Если в последовательности только один элемент, то деление не выполняем.

2) Сортируем отдельно каждую из полученных частей этим же алгоритмом.

3) Производим слияние отсортированных частей так, чтобы сохранилась упорядоченность.

```
def MergeSort (q,r):
  if q ≠ r:
        k = (q + r) // 2
        MergeSort (q,k)
        MergeSort (k+1,r)
        MergeList (q,k,r)
```

Один из недостатков сортировки слиянием – дополнительная память (*additional memory*), размер которой зависит от количества элементов массива и которая требуется при выполнении функции `MergeList (q,k,r)`. При слиянии двух упорядоченных частей, которые в исходном массиве *array* занимают смежные области (начиная с индекса q и заканчивая r), сравниваем наименьшие элементы каждой из отсортированных частей и меньший из них отправляем в список вывода (*additional memory*); повторяем описанные действия до тех пор, пока не исчерпается одна из частей; все оставшиеся элементы другой части пересылаем в *additional memory*. Затем из *additional memory* пересылаем элементы в исходный массив *array*, начиная с индекса q и заканчивая r (т. е. на позиции, которые в массиве занимали элементы рассмотренных частей).



Пусть $T(n)$ – число арифметических операций для того, чтобы сортировка слиянием свою работу. Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = C_1 + 2 \cdot T\left(\frac{n}{2}\right) + C_2 \cdot n, \; n = 2^k, k \ge 1 \\ T(1) = C_3 \end{cases}$$

Решим это рекуррентное уравнение методом итераций:

$$T(n) = \underbrace{C_1 + 2 \cdot T\left(\frac{n}{2}\right) + C_2 \cdot n}_{1\text{–й шаг}} = \left[T\left(\frac{n}{2}\right) = C_1 + T\left(\frac{n}{2^2}\right) + C_2 \cdot \frac{n}{2} \right] =$$

$$= \underbrace{2^0 \cdot C_1 + 2^1 \cdot C_1 + 2^2 \cdot T\left(\frac{n}{2^2}\right) + C_2 \cdot n + C_2 \cdot n}_{2\text{–й шаг}} = \ldots$$

$$= \underbrace{C_1 \cdot \left(2^0 + 2^1 + \cdots + 2^{m-1}\right) + 2^m \cdot T\left(\frac{n}{2^m}\right) + m \cdot C_2 \cdot n}_{m\text{–й шаг}} = \left[\frac{n}{2^m} = 1, n = 2^m, m = \log_2 n \right] =$$

$$= \underbrace{C_1 \cdot (n-1) + C_3 \cdot n + C_2 \cdot n \cdot \log_2 n}_{(\log_2 n)\text{–й шаг}} = C_2 \cdot n \cdot \log_2 n + (C_3 + C_1) \cdot n - C_1$$

Получаем, что $T(n) = \Theta(l \cdot \log l), l = \Theta(n)$, алгоритм – полиномиальный.

6. Сортировка Чарльза Хоара (алгоритм «быстрой сортировки»)

В 1960 году английский ученый Ч. Хоар разработал алгоритм «быстрой сортировки», который является наиболее популярным до настоящего времени.

В алгоритме среди элементов сортируемой области выбирается разделитель x (сепаратор, опорный элемент) (англ. pivot), относительно которого обменами выполняется разделение массива на две части. Разделение может быть выполнено, например, таким образом, что в левой части окажутся элементы с ключами $\le x$ и в правой с ключами $\ge x$ (разделение Ч. Хоара). Затем к каждой из частей применяется этот же алгоритм.

Приведем схему алгоритма «быстрой сортировки», в которой для разделения используется алгоритм Нико Ламуто.

```
def QuickSort(q, r):
    if q < r:
        p=Partition(q, r)
        QuickSort(q, p-1)
        QuickSort(p+1, r)
```

Если $q \ge r$, то `QuickSort(q, r)` завершает работу.

Если $q < r$, то

1. Выбирается разделитель – некоторый элемент x из рассматриваемой области.

Например, в качестве разделителя можно выбрать первый элемент области, т. е. $x = array[q]$.

2. Относительно x массив разделим на три части (алгоритм Н. Ламуто):

I-я часть – элементы строго меньше x (в *array* располагаются по индексам от q до $p-1$);

II-я часть – элемент x (в *array* располагается по индексу p);

III-я часть – элементы больше или равные x (в *array* располагаются по индексам от $p+1$ до *r)*;

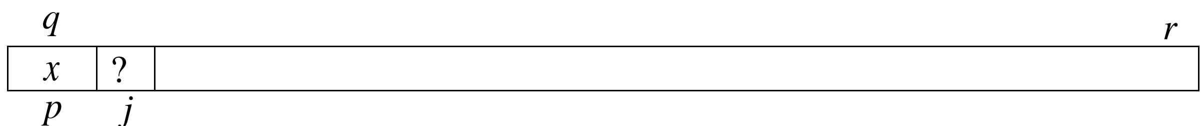
3. Рекурсивно вызываем алгоритм `QuickSort` для первой и третьей части (если они не пустые):

```
QuickSort (q, p-1)
QuickSort (p+1, r)
```

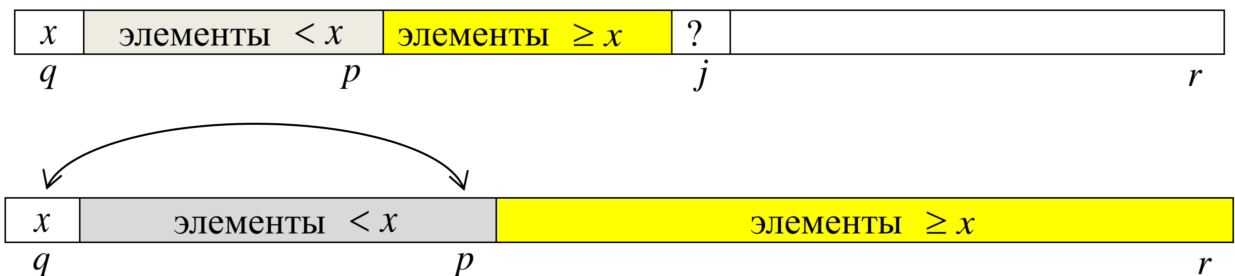
Приведем алгоритм Н. Ламуто, который используется в алгоритме быстрой сортировки для разделения массива на части относительно выбранного разделителя.

Схема разделения Н. Ламуто

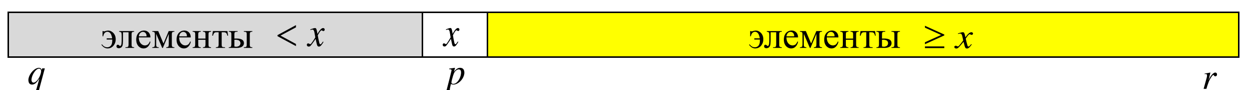
Старт



Итерация



Финиш



```
def Partition (q,r):
 x=array[q]
 p=q
 j=p+1
 while j<=r:
    if array[j]>=x:
          j+=1
     else: # array[j]<x
         p+=1
         array[p]↔array[j]
         j+=1
 array[l]↔array[p]
 return p
```

Худший случай: все данные одинаковы или упорядочены, например, по возрастанию. Сортируемая область в каждом из этих случаев сократится только на один элемент.

Приведем алгоритм разделения, предложеный Ч. Хоаром.

Схема разделения Ч. Хоара

1. Выбирается разделитель. Например, в качестве разделителя можно выбрать первый элемент области, т. е. $x = array[q]$.

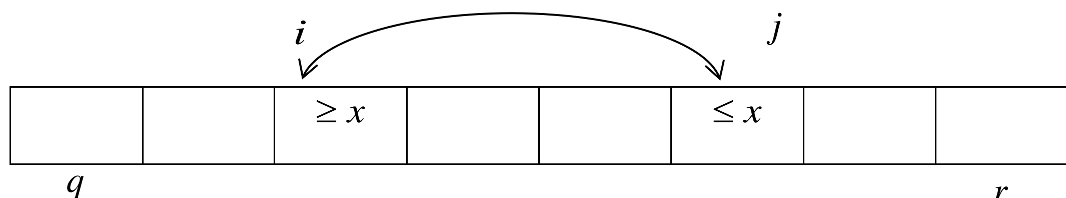
2. Относительно x массив разделим на две части функцией `Hoare_Partition`:

I-я часть – элементы, которые равны или меньше x (в *array* располагаются по индексам от q до j);

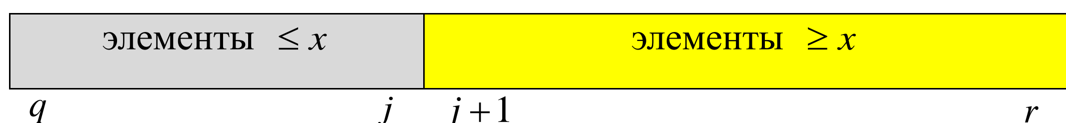
II-я часть – элементы, которые больше или равны x (в *array* располагаются по индексам от $j+1$ до *r)*;

В результате разделения Ч. Хоара будет сформирован индекс j, для которого справедливы неравенства: $q \le j < r$, и каждый элемент подмассива $array[q..j]$ не превышает значений каждого элемента подмассива $array[j+1..r]$.

<u>Итерация</u>



<u>Финиш</u>



```
Hoare_Partition (A,q,r)
x ← array[q]
i ← q − 1
j ← r + 1

while TRUE
    do repeat   j ← j − 1
         untile   array[j] ≤ x
     repeat   i ← i + 1
         untile   array[i] ≥ x
     if i < j
         then array[i] ↔ array[j]
         else return j
```

Худший случай: все данные различны и упорядочены, например, по возрастанию. Тогда в качестве сепаратора на каждом этапе разделения будет выбираться минимальный элемент, и сортируемая область сократится только на один элемент.

Пусть $T(n)$ – число арифметических операций для того, чтобы «быстрая сортировка» упорядочила массив из n элементов. Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} T(n) = C_1 + C_2 \cdot n + T(n-1)+, \ n \geq 2 \\ T(1) = C_3 \end{cases}$$

Получаем, что $T(n) = \Theta(l^2), l = \Theta(n)$, алгоритм – полиномиальный.

Интересен тот факт, что среднее время работы алгоритма «быстрой сортировки» по всем возможным наборам входных данных:

$$A(n) = \mathrm{O}(l \cdot \log l), l = \Theta(n),$$

где деление на классы идет на каждом этапе разделения `Partition`, а класс характеризуется той позицией p, куда будет помещен сепаратор x после того, как будет произведено разделение [7].

Если на каждом этапе разделения в качестве разделителя x выбирать средний по значению элемент (медиану) рассматриваемой области и делать это за линейное от количества элементов время, то время работы алгоритма сортировки `QuickSort` в худшем случае описывается следующим рекуррентным уравнением:

$$\begin{cases} T(n) = C_1 \cdot n + C_2 \cdot n + 2 \cdot T\left(\dfrac{n}{2}\right), \ n = 2^k, k \geq 1 \\ T(1) = C_3 \end{cases}$$

Получаем, что $T(n) = \Theta(l \cdot \log l), l = \Theta(n)$, алгоритм – полиномиальный.

В стандартных библиотеках высокоуровневых языков программирования есть готовые реализации функции сортировки. При этом некоторые из реализаций используют комбинированые подходы.

`C++ std::sort()`

Основой служит алгоритм быстрой сортировки – модифицированный `QuickSort`, он же IntroSort (интроспективная сортировка, подход Дэвида Мюссера), разработанный специально для `stl.`

В качестве опорного элемента выбирается *медиана из трех*: $array[q], array[r], array\left[\frac{q+r}{2}\right]$.

Отличие от `QuickSort` состоит в том, что количество рекурсивных операций не идет до самого конца, как в чистом `QuickSort`. Если количество итераций (процедур разделения массива, т. е. глубина рекурсии) превысило $1{,}5 \cdot \log_2 n$, где n – длина всего массива, то рекурсивные операции прекращаются:

– если количество оставшихся элементов меньше 16, то оставшийся фрагмент сортируется методом простой вставки `InsertionSort` (сортировка вставками работает за время $\mathrm{O}\left(n^2\right)$ и для больших массивов не используется, но на малых длинах эффективна ввиду простоты реализации и устойчивости);

– если количество оставшихся элементов более 32-х, то этот фрагмент сортируется пирамидальным методом `HeapSort` в чистом его виде (пирамидальная сортировка в худшем случае работает за время $\mathrm{O}(n \cdot \log n)$, не устойчива [1, 6]).

`Java java.util.Collections.sort()`

Сортировка реализована на базе сортировки слиянием `MergeSort`, которая выбрана разработчиками из-за ее устойчивости (показывает лучшую производительность по сравнению с другими устойчивыми алгоритмами сортировками, например, таким алгоритмом, как «пузырек»).

`Python sort() и sorted()`

Функции в `Python` реализуют алгоритм `TimSort` (опубликован в 2002 году американским ученым Тимом Петерсом (Tim Peters)), основанный на сортировке слиянием `MergeSort` и сортировке вставкой `InsertionSort`.

Основная идея алгоритма: по специальному алгоритму входной массив разделяется на подмассивы.

Каждый подмассив сортируется сортировкой вставками.

Отсортированные подмассивы собираются в единый массив с помощью модифицированной сортировки слиянием (так называемая «сортировка слиянием галопом»).

1.2.6. Задача поиска k-ого наименьшего элемента массива

Определение 1.18. Элемент, который стоит на k-ом месте в отсортированном по не убыванию массиве, называется *k-ым наименьшим элементом* (k-й порядковой статистикой).

Например, для массива

	1	2	3	4	5	6	8
$a[i]$	2	3	7	1	40	12	100

если $k = 4$, то 4-й наименьший элемент равен 7:

	1		2	3	4	5	6	8
$a^{\text{сорт.}}[i]$	1		2	3	**7**	12	40	100

Определение 1.19. Если n – нечетно, то *медианой* массива из n элементов называется такой его элемент, который стоит на месте $k = \left\lfloor \frac{n+1}{2} \right\rfloor$ в упорядоченном массиве.

	1	2	3	4	5
$a^{\text{сорт.}}[i]$	1	2	**13**	70	90

Если n – четно, то *нижней медианой* массива из n элементов называется такой его элемент, который стоит на месте $k = \left\lfloor \frac{n+1}{2} \right\rfloor$ в упорядоченном массиве.

Если n – четно, то *верхней медианой* массива из n элементов называется такой его элемент, который стоит на месте $k = \left\lceil \frac{n+1}{2} \right\rceil$ в упорядоченном массиве.

	1	2	3	4
$a^{\text{сорт.}}[i]$	1	2	13	70

нижняя медиана — верхняя медиана

Если не оговорено иное, то медианой массива из n элементов будем считать нижнюю медиану, т. е. для любой четности n полагаем $k = \left\lfloor \frac{n+1}{2} \right\rfloor$.

Алгоритм 1.

1) Отсортируем массив, например, сортировкой слиянием.

2) Возьмем в отсортированном массиве элемент по индексу k.

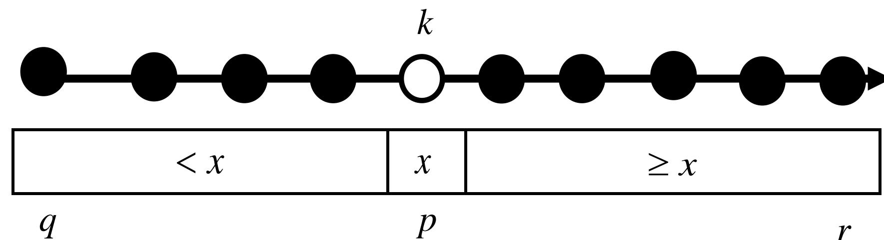
Время работы алгоритма 1 в худшем случае $\Theta(n \cdot \log n)$.

Алгоритм 2.

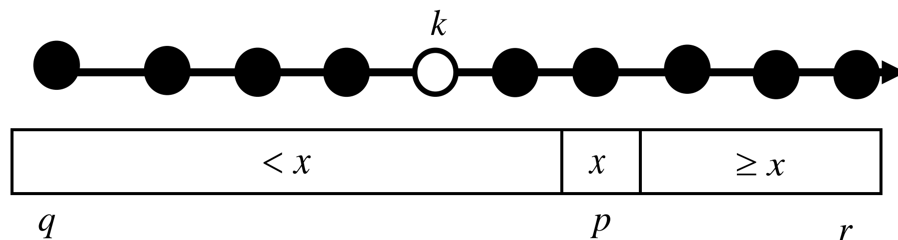
1) Полагаем, $q=1$, $r=n$, в качестве опорного элемента x возьмем первый элемент рассматриваемой области (в рандомизированной версии опорный элемент сначала выбирается случайным образом (*random sampling*) среди элементов с индексами от q до r, затем он меняется с первым элементом рассматриваемой области).

2) Относительно x выполняется разделение массива на отрезке $[q,r]$.

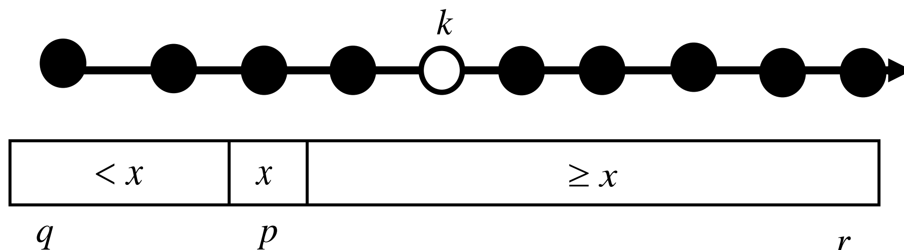
Случай 1. Если $k = p - q + 1$, то опорный элемент x – ответ.



Случай 2. Если $k < p - q + 1$, то продолжаем поиск *k*-ого наименьшего элемента на отрезке $[q, p-1]$.



Случай 3. Если $k > p - q + 1$, то полагаем $k = k - (p - q + 1)$ и продолжаем рекурсивно поиск *k*-ого наименьшего элемента на отрезке $[p+1, r]$.



Время работы детерминированного алгоритма 2 в худшем случае $\Omega(n^2)$ (например, если на каждом шаге в качестве разделителя выбирался максимальный элемент рассматриваемой области, что приводило к тому, что каждый раз рассматриваемая область уменьшалась только на один элемент). Рандомизированный алгоритм в среднем работает за время $O(n)$.

Пусть $k = \lfloor n+1/2 \rfloor$, рассмотрим алгоритм 3, который в худшем случае находит медиану в массиве из n элементов за линейное от числа элементов время.

Алгоритм 3. BFPRT- алгоритм
(1973 г. Manual **B**lum, Robert W. **F**loyd, Vaughan R. **P**ratt,
Ronald L. **R**ivest, Robert Endre **T**arjan)

1) Разбиваем исходный массив *array* из n элементов на $\left\lfloor \frac{n}{5} \right\rfloor$ групп по пять элементов в каждой и еще одну группу, в которой оставшиеся $(n \bmod 5)$ элементов, если число элементов кратно 5, то эта группа – пустая.

Каждую группу сортируем и находим ее медиану. Это требует времени $C_1 \cdot n$.

20	25	15	6	1	7	14	27	
21	26	18	8	2	14	16	29	9
34	28	24	10	3	23	17	30	11
37	38	36	13	4	24	35	31	12
46	39	42	44	5	25	40	41	33

2) Из найденных на шаге 1) медиан строим последовательность M длины $\left\lceil \frac{n}{5} \right\rceil$ и этим же алгоритмом рекурсивно находим ее медиану x. Время – $T\left(\left\lceil \frac{n}{5} \right\rceil\right)$.

3) Элемент x выбираем в качестве разделителя и выполняем процесс разделения исходного массива *array* из n элементов. Это требует времени $C_2 \cdot n$.

Для простоты рассуждений будем считать, что все элементы массива различны.

1	6		14	7	15	25	27	20
2	8	9	16	14	18	26	29	21
3	**10**	**11**	**17**	$x = 23$	**24**	**28**	**30**	**34**
4	13	12	35	24	36	38	31	37
5	44	33	40	25	42	39	41	46

Тогда количество элементов, которые $< x$ (зеленая заливка):

$$\geq 3 \cdot \left(\left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3 \cdot n}{10} - 6.$$

Аналогичные рассуждения приводят к тому, что и в другой области (желтая заливка), имеется

$$\geq \frac{3 \cdot n}{10} - 6$$

элементов, величины которых $> x$.

4) По аналогии с алгоритмом 2, либо завершаем алгоритм 3, либо отбрасываем одну из частей и решаем рекурсивно задачу поиска *k*-ого наименьшего элемента за время, не превышающее величины

$$T\left(\frac{7 \cdot n}{10} + 6 \right).$$

Таким образом рекуррентное уравнение для оценки времени работы алгоритма 3 имеет следующий вид:

$$T(n) \leq C_1 \cdot n + T\left(\left\lceil \frac{n}{5} \right\rceil \right) + C_2 \cdot n + T\left(\frac{7 \cdot n}{10} + 6 \right).$$

Получаем, что в худшем случае время работы алгоритма 3 поиска $k = \lfloor n + 1/2 \rfloor$ наименьшего элемента

$$T(l) = \Theta(l),$$
$$l = \Theta(n).$$

1.3. Методы разработки эффективных алгоритмов

Существует ряд методов разработки алгоритмов, когда решение исходной задачи получается путем комбинирования решений вспомогательных подзадач (подзадача – это задача той же природы, но более простая, например, меньшей размерности).

Как правило, эти методы применяются к задачам, которые имеют рекурсивную структуру.

Рассмотрим два таких метода [6]:

1) метод «разделяй и властвуй»;

2) метод «динамического программирования» (табличный метод).

1.3.1. Метод разделяй и властвуй

Метод «разделяй и властвуй» состоит из следующих трех этапов.

1) «*Разделение*». Задача разбивается на независимые подзадачи, т. е. подзадачи не пересекаются (две задачи назовем независимыми, если они не имеют общих подзадач) (рисунки 1.7, 1.8).

2) «*Покорение*». Каждая подзадача решается отдельно (рекурсивным методом). Когда объем возникающих подзадач достаточно мал, то подзадачи решаются непосредственно.

3) «Комбинирование». Из отдельных решений подзадач строится решение исходной задачи.

Заметим, если бы подзадачи пересекались, т. е. имели общие подзадачи, то метод «разделяй и властвуй» делал бы лишнюю работу, решая некоторые подзадачи по нескольку раз.

Если алгоритм рекурсивно обращается сам к себе, то время его работы

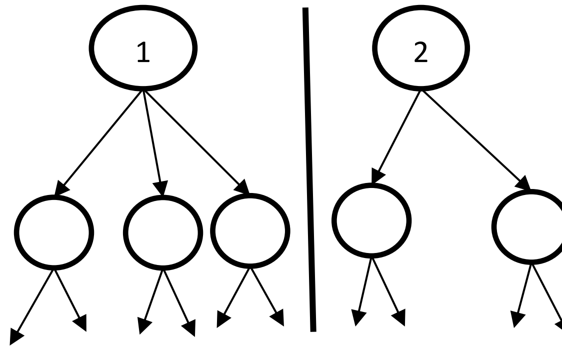


Рисунок 1.7 – Независимые задачи (1) и (2).

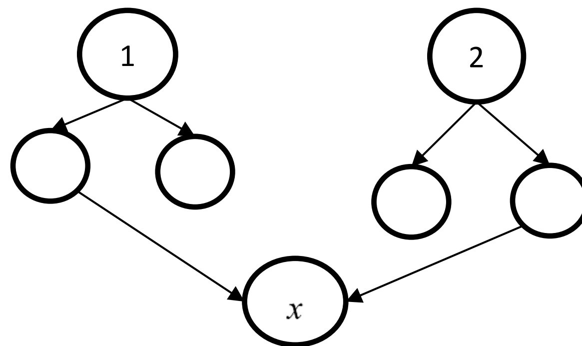


Рисунок 1.8 – Зависимые задачи (1) и (2).

описывается с помощью рекуррентного соотношения, в котором время, требуемое для решения всей задачи, выражается через время, необходимое для решения вспомогательных подзадач. Решая даное рекуррентное соотношение, получим функцию, задающую время работы алгоритма.

Для алгоритмов, основанных на методе «разделяй и властвуй», рекуррентное соотношение часто может быть получено в следующем виде:

$$\begin{cases} T(n) = c_1, n \le c, \\ T(n) = D(n) + a \cdot T\left(\frac{n}{b}\right) + F(n), n > c, \end{cases}$$

где $D(n)$– время, затраченное на этап разделения исходной задачи на a подзадач размера $1/b$ от размера исходной задачи (не всегда $a = b$); $T(n/b)$ – время, необходимое для решения подзадачи; $F(n)$– время, затраченное на этап комбинирования.

Начальное условие говорит о том, что когда размер подзадач $n \le c$, то данная подзадача решается непосредственно за время $\Theta(1)$.

При разбиении задачи на подзадачи полезен *принцип балансировки*, который предполагает, что задача разбивается на подзадачи приблизительно равных размерностей, т. е. идет поддержание равновесия. Обычно такая стратегия приводит к разделению исходной задачи пополам и обработке каждой из его частей тем же способом до тех пор, пока части не станут настолько малыми, что их можно будет обрабатывать непосредственно. Часто такой процесс приводит к логарифмическому множителю в формуле, описывающей трудоемкость алгоритма.

Таким образом, в основе техники рассматриваемого метода лежит процедура разделения. Если разделение удается произвести без слишком больших затрат, то может быть построен эффективный алгоритм.

Данный подход нами был уже применен ранее:

– задача поиска максимального и минимального элементов массива (делением пополам) (раздел 1.2.3);

– алгоритм сортировки слиянием (принцип балансировки выполняется, так как сортируемая область делится всегда на две равные части) (раздел 1.2.5);

– алгоритме быстрой сортировки Ч. Хоара (для выполнения принципа балансировки в качестве опорного элемента необходимо выбирать медиану сортируемой области) (раздел 1.2.5).

1.3.2. Динамическое программирование

Динамическое программирование применяется к задачам, в которых нужно что-то подсчитать или к оптимизационным задачам. Например, в задаче требуется определить число различных способов подняться по ступенькам при заданном способе подъема, или вычислить число способов размещения k единиц в строке длины n, или вычислить F_n-ое число Фибоначчи и т. п.

Напомним, что в оптимизационных задачах существует много решений, каждому из которых поставлено в соответствие некоторое значение, необходимо найти среди всех возможных решений одно с оптимальным (наибольшим или наименьшим) значением. Например, во взвешенном графе между заданной парой вершин существует несколько маршрутов, каждый маршрут

характеризуется своей длиной, и нам необходимо найти маршрут кратчайшей длины.

Процесс разработки алгоритмов с использованием метода динамического программирования можно разделить на следующие этапы.

1) Задача погружается в семейство вспомогательных подзадач той же природы. Возникающие подзадачи могут являться зависимыми и должны удовлетворять следующим двум требованиям.

✓ Подзадача должна быть более простой по отношению к исходной задаче. Задача может быть проще из-за того, что опущены некоторые ограничения. Она может быть проще из-за того, что некоторые ограничения добавлены. Однако, как бы ни была изменена задача, если это изменение приводит к решению более простой задачи, то, возможно, удастся, опираясь на эту более простую, решить и исходную.

✓ Оптимальное решение исходной задачи определяется через оптимальные решения подзадач (в этом случае говорят, что задача обладает свойством оптимальной подструктуры, и это один из аргументов в пользу применения для ее решения метода «динамического программирования»).

2) Каждая вспомогательная подзадача решается (рекурсивно) только один раз. Значения оптимальных решений возникающих подзадач запоминаются в таблице, что позволяет не решать снова встречавшиеся ранее подзадачи.

3) Для исходной задачи строится возвратное соотношение, связывающее значение оптимального решения исходной задачи со значениями оптимальных решений вспомогательных подзадач (т. е. методом восходящего анализа от простого к сложному вычисляем значение оптимального решения исходной задачи).

4) Данный этап выполняется в том случае, когда требуется помимо значения оптимального решения получить и само это решение. Часто для этого требуется некоторая вспомогательная информация, полученная на предыдущих этапах метода.

Таким образом, стратегия метода динамического программирования – попытка свести рассматриваемую задачу к более простым задачам, тогда как стратегия предыдущей техники – «разделяй и властвуй». Так как возникающие подзадачи являются зависимыми, то данная техника находит свое применение, когда все значения оптимальных решений подзадач помещаются в память.

Вычисление идет от малых подзадач к большим, и ответы запоминаются в таблице, что позволяет исключить повторное решение задачи. Одна из клеток таблицы и дает значение оптимального решения исходной задачи.

Метод динамического программирования часто в литературе называют *табличным*.

Реализовать метод динамического программирования (сокр. ДП) можно следующими способами:

– «ДП назад» (при решении задачи x все вспомогательные подзадачи должны быть уже решены; строим решение задачи x через решения вспомогательных подзадач, рисунок 1.9);

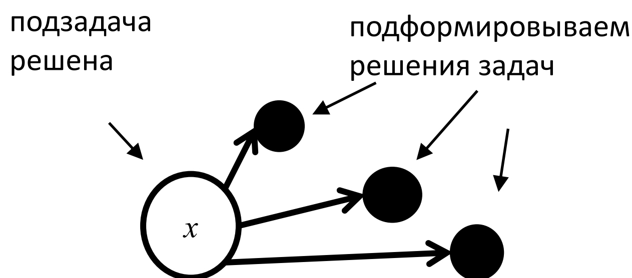
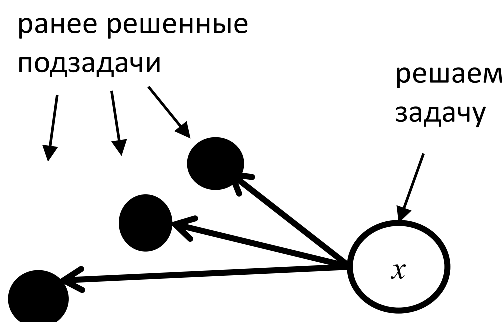


Рисунок 1.9 – ДП назад.

– «ДП вперед» (подзадача x решена, подформировываем решения задач,



которые опираются на решение задачи x, рисунок 1.10);

Рисунок 1.10 – ДП вперед.

– «Рекурсия с мемоизацией» («ленивое ДП») – это прием, который позволяет улучшить рекурсивную версию, устраняя многократные вызовы рекурсии при одних и тех же значениях параметрах. При решении задачи проверяем, была ли решена вспомогательная подзадача: если да, то берем ее решение из таблицы; если нет, то запускаем рекурсивную процедуру (рекурсия углубляется до тех пор, пока не придет к подзадаче, которая уже решена). Как только некоторая подзадача решена, запоминаем вычисленное значение вместе со значение параметра, переданного рекурсии при вызове.

1.3.3. Рекуррентные соотношения и динамическое программирование

Решение задачи через рекуррентное соотношение, это такой метод решения, когда задача разбивается на подзадачи, и из решений этих подзадач может быть получено решение самой задачи.

Рекуррентное соотношение состоит из формул, определяющих, как из подзадач получать решение более общей задачи, и ограничений, задающих решение задачи для подзадач минимального размера.

Реализовать рекуррентное соотношение можно через рекурсию (способ, когда подпрограмма вызывает саму себя, обычно с другими параметрами), или через динамическое программирование (ДП, способ, когда решения подзадач хранятся в какой-то структуре данных, чаще всего массиве). Рассмотрим метод рекуррентного соотношения на нескольких простых задачах.

1.3.3.1. Задача «Факториал».

Факториалом неотрицательного целого числа n называется произведение чисел от 1 до n (записывается $F(n)=n!$).

Для нахождения $n!$ методом рекуррентного соотношения, попытаемся ответить на вопрос, можем ли мы его найти, если нам будем известен $(n-1)!$. Что нужно дописать в правой части выражения $n!=(n-1)!$, чтобы оно стало истинным? Подставим вместо факториалов сами числа:

$$1\cdot 2\cdot 3\cdot \ldots \cdot (n-1)\cdot n = 1\cdot 2\cdot 3\cdot \ldots \cdot (n-1).$$

То есть правую часть для ее равенства левой достаточно умножить на n. Отсюда можно вывести рекурсивную формулу:

$$F(n)=F(n-1)\cdot n.$$

Для того, чтобы вычисление всех возможных подзадач не стало бесконечным процессом, нам нужно ввести ограничение, то есть нужно знать, при каком самом маленьком n можно вычислить $n!$. Математики считают, что $0!=1$. Таким образом, получим следующее рекуррентное соотношение:

$$\begin{cases} F(n)=F(n-1)\cdot n,\ n\geq 1 - \text{ рекурсивная функция} \\ F(0)=1 - \text{ ограничение} \end{cases}$$

Реализовать его можно как с помощью рекурсии:

```
#include <iostream>
using namespace std;
long long Fact(int n){
    if (n == 0)                      /// ограничение
        return 1;
    return Fact(n - 1) * n;         /// рекурсивная функция
}
int main(){
    int n;
    cin >> n;
    cout << Fact(n) << endl;
    return 0;
}
```

так и с помощью метода динамического программирования:

n	0	1	2	3	4	5	6	7	8
$F(n)$	1	1	2	6	24	120	720	5040	40320

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    int n;
    cin >> n;
    vector < long long > a(n + 1);      /// память для подзадач
    a[0] = 1;                          /// ограничение
    for (int i = 1; i <= n; i++)
        a[i] = a[i - 1] * i;           /// рекурсивная функция
    cout << a[n] << endl;
    return 0;
}
```

1.3.3.2. Задача «Числа Фибоначчи».

Числа Фибоначчи равны сумме двух предыдущих чисел, то есть вычисляются по рекурсивной формуле

$$F(n) = F(n-1) + F(n-2).$$

Математики считают, что $F(0) = 0$. Однако обойтись одним ограничением не получится, так как в этом случае

$$F(1) = F(0) + F(-1),$$

а значение $F(-1)$ неизвестно.

Поэтому необходимо ввести дополнительное ограничение, и в результате получим следующее рекуррентное соотношение:

$$\begin{cases} F(n) = F(n-1) + F(n-2), n \geq 2 \text{ (рекурсивная функция)} \\ F(0) = 0 \text{ (ограничение 1)} \\ F(1) = 1 \text{ (ограничение 2)} \end{cases}$$

Реализовать его можно как с помощью рекурсии, так и с помощью метода динамического программирования:

```
#include <iostream>
#include <vector>
using namespace std;

long long Fib(int n){
    if (n == 0)                         /// ограничение 1
        return 0;
    if (n == 1)                         /// ограничение 2
        return 1;
    return Fib(n - 1) + Fib(n - 2); }  /// рекурсивная функция
```

```
int main()
{
    int n;
    cin >> n;

/// динамическое программирование
    vector < long long > a(n + 1);     /// память для подзадач
    a[0] = 0; a[1] = 1;                /// ограничение 1, 2
    for (int i = 2; i <= n; i++)
        a[i] = a[i - 1] + a[i - 2];    /// рекурсивная функция
    cout << a[n] << endl;

/// рекурсия
    cout << Fib(n) << endl;
    return 0;
 }
```

N	0	1	2	3	4	5	6	7	8
$F(n)$	0	1	1	2	3	5	8	13	21

На примере этой задачи можно увидеть разницу между реализацией рекурсией и ДП. У каждого из подходов есть свои достоинства и недостатки.

Так, рекурсия вычисляет только те подзадачи, которые нужны для решения основной задачи, но при этом может вычислять некоторые из подзадач много раз.

Динамическое программирование вычисляет значение каждой подзадачи ровно один раз, но при этом решает все меньшие подзадачи, а не только необходимые.

Разницу в используемой памяти можно не учитывать, так как рекурсия занимает память под каждую копию функции при рекурсивном вызове, а динамическое программирование использует для той же цели массив. Но в целом считается, что динамическое программирование использует больше памяти. Тем не менее, за счет того, что никакая подзадача не вычисляется дважды, в целом ДП быстрее рекурсии.

В качестве примера попробуйте сравнить время работы программы для вычисления 45-го числа Фибоначчи с помощью ДП и рекурсией.

Как мы уже говорили, у рекурсии и ДП есть свои достоинства и недостатки. Можно ли как-то объединить эти подходы, чтобы взять лучшее от обоих подходов? Можно, и такой подход называется *мемоизацией*. В целом идея достаточно простая: давайте использовать рекурсию (чтобы решать только необходимые подзадачи), но при этом сохранять уже решенные подзадачи (чтобы уменьшить время работы).

Давайте реализуем наше рекуррентное соотношение рекурсией с мемоизацией:

```
#include <iostream>
#include <vector>
using namespace std;
vector < long long > b;                         /// память для подзадач

long long FibM(int n){
    if (b[n] == -1)                             /// если не вычисляли
        b[n] = FibM(n - 1) + FibM(n - 2);  /// рекурсивная функция
    return b[n];
}

int main(){
    int n;
    cin >> n;
    b.resize(n + 1, -1);
    b[0] = 0; b[1] = 1;                         /// ограничение 1, 2
    cout << FibM(n) << endl;
    return 0;
}
```

Сравните время работы метода с рекурсией и ДП. Оцените затраты.

1.3.3.3. Задача «Возведение числа 2 в степень n».

Пусть необходимо вычислить 2^n. Решение в лоб – умножить n раз переменную, первоначально равную 1, на число 2.

Можно ли сократить количество умножений? Если n четно, то можно вычислить $2^{\frac{n}{2}}$ и возвести в квадрат (умножить само на себя). Если же n нечетно, то можно вынести одну 2 за скобки, после чего $n-1$становится четным, и его можно вычислить так, как описано выше. Итак, для вычисления $F(n)=2^n$, получим следующее рекуррентное соотношение:

$$\begin{cases} F(n)=\left(F\left(\dfrac{n}{2}\right)\right)^2, n - \text{ четно (рекурсивная функция 1)} \\ F(n)=\left(F\left(\dfrac{n-1}{2}\right)\right)^2 \cdot 2, n - \text{ нечетно (рекурсивная функция 2)} \\ F(1)=2 - \text{ (ограничение 1)} \\ F(0)=1 - \text{ (ограничение 2)} \end{cases}$$

В этом рекуррентном соотношении никакие подзадачи не будут вычисляться дважды, поэтому можно использовать рекурсию (без мемоизации) для реализации.

Так как степень растет очень быстро, то можно не волноваться о возможности превышения максимальной глубины рекурсии.

Разберем рекурсию на примере:

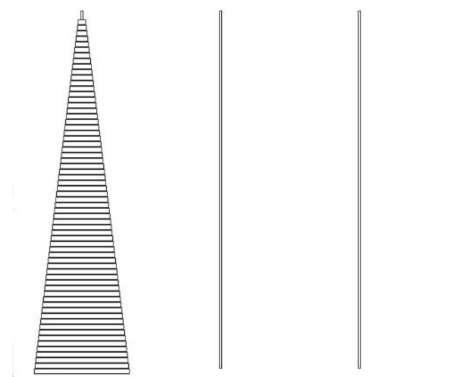
$$F(10)=F(5)^2=\left(F(2)^2\cdot 2\right)^2=\left(\left(F(1)^2\right)^2\cdot 2\right)^2=\left(\left(2^2\right)^2\cdot 2\right)^2=\left((4)^2\cdot 2\right)^2=32^2=1024$$

Итого вместо 10 умножений мы сделали только 4 (однако, выполнено было еще и 3 деления на 2).

Реализуем рекуррентное соотношение через рекурсию, используем побитовые операции для ускорения вычислений (при побитовом сдвиге деление на 2 (как и умножение на 2) выполняется за 1 такт):

```
#include <iostream>
using namespace std;
long long F(int n){
    if (n == 0)                 /// ограничение 2
        return 1;
    if (n == 1)                 /// ограничение 1
        return 2;
    long long t = F(n >> 1);    /// t = F(n / 2);
    if ((n & 1) == 0)           /// n % 2 == 0
        return  t * t;                           /// рекурсивная функция 1
    return (t * t) << 1;        /// t * t * 2    /// рекурсивная функция 2
}
int main(){
    int n;
    cin >> n;
    cout << F(n);
    return 0;
}
```

1.3.3.4. Задача «Ханойские башни».



Есть известная математическая задаче под названием «Ханойские башни». Формулировка: в одном монастыре в Ханое есть монастырь, в котором находятся три стержня. На одном из стержней нанизано 64 диска разного диаметра (внизу самый большой, каждый следующий меньше). Монахи перекладывают диски со стержня, соблюдая два правила: за одно перекладывание можно перенести только один диск; нельзя помещать диск большего диаметра на диск меньшего диаметра. По легенде, когда вся стопка дисков будет с начального стержня перенесена на другой, наступит конец света. Спрашивается, сколько нам осталось до этого события, если монахи круглосуточно безошибочно переносят по 1 диску в секунду?

Давайте формализуем и обобщим задачу. Пусть на начальном стержне n дисков. Сколько перекладываний понадобится, чтобы вся стопка собралась на другом стержне? Легко видеть, что 1 диск мы можем перенести за 1 действие (это будет наше ограничение). С формулой сложнее. Давайте попробуем использовать тот же подход, который мы использовали при нахождении факториала. Можем ли

мы сказать, сколько нужно переносов для n дисков $F(n)$ если нам известно, что $(n-1)$ диск можно перенести за $F(n-1)$?

Пусть начальный стержень имеет номер ***A***, конечный ***B***, а промежуточный ***C***. Тогда для переноса n дисков с ***A*** на ***B*** мы сначала перенесем $(n-1)$ диск с ***A*** на ***C*** за $F(n-1)$, затем самый большой из n дисков перенесем с ***A*** на ***B*** за 1, и, наконец, перенесем $(n-1)$ диск с ***C*** на ***B*** за $F(n-1)$.

Таким образом, рекуррентное соотношение для числа способов перенести все диски с одного стержня на другой, следующее:

$$\begin{cases} F(n) = 2 \cdot F(n-1) + 1, n \geq 1 - \text{ рекурсивная функция} \\ F(1) = 1 - \text{ ограничение} \end{cases}$$

Для $F(3)$ нам понадобится выполнить 7 перекладываний, а для $F(64)$ – 18446744073709551615 перекладываний.

Для того, чтобы выписать явный вид функции, задающей число способов перенести все диски с одного стержня на другой, можно методом итераций решить рекуррентное уравнение. В Разделе 1.2.2 (пример 1.7) было показано, что решением этого рекуррентного соотношения является функция $F(n) = 2^n - 1$.

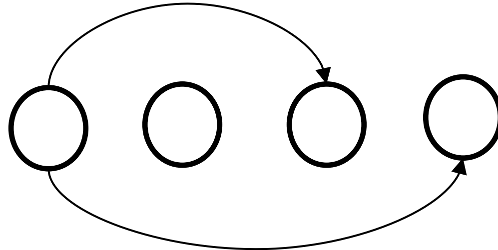
Давайте попробуем вывести инструкцию по переносу n дисков со стержня ***A*** (пусть его номер 1) на стержень ***B*** (пусть его номер 2). Реализация метода, выводящего инструкцию, практически плоностью совпадает с выводом рекуррентного соотношения:

```
#include <iostream>
using namespace std;
unsigned long long P(int n){
    if (n == 1)
        return 1;
    return P(n - 1) * 2 + 1;
}
void H(int n, int a, int b){
    if (n == 0)
        return;
    H(n - 1, a, 6 - a - b);
    cout << a << " -> " << b << endl;
    H(n - 1, 6 - a - b, b);
}
int main(){
    int n;
    cin >> n;
    H(n, 1, 2);
    cout << P(n);
    return 0;
}
```

1.3.3.5. Задача «Путь лягушки».

В одном очень длинном и узком пруду по кувшинкам прыгает лягушка. Кувшинки в пруду расположены в один ряд.

Лягушка начинает прыгать с первой кувшинки ряда и хочет закончить на последней. Но в силу вредности характера лягушка согласна прыгать только вперед через одну или через две кувшинки. Например, с кувшинки номер 1 она может прыгнуть лишь на кувшинки номер 3 и номер 4.



На некоторых кувшинках сидят комарики. А именно, на i-й кувшинке сидят a_i комаров. Когда лягушка приземляется на кувшинку, она съедает всех комариков, сидящих на ней.

Лягушка хочет спланировать свой маршрут так, чтобы съесть как можно больше комаров. Помогите ей: скажите, какие кувшинки она должна посетить на своем пути.

Решение.

Пусть $array[i]$ – число комариков на кувшинке с номером i .

Обозначим через $F[i]$ – максимальное число комариков, которые скушает лягушка, приземлившись на кувшинку с номером i .

Реализуем сначала *одномерное* ДП назад. Справедливо следующее рекуррентное соотношение:

$$\begin{cases} F[1] = array[1] \\ F[2] = -\infty \\ F[i] = max\{F[i-2], F[i-3]\} + array[i],\ i = \overline{3,n} \end{cases}$$

Реализуем теперь *одномерное* ДП вперед. Справедливо следующее рекуррентное соотношение:

$$\begin{cases} F[1] = array[1] \\ F[2] = -\infty \\ \\ i = \overline{1,n}: \\ F[i+2] = max\{F[i+2], F[i] + array[i+2]\}, if\ (i+2) \le n \\ F[i+3] = max\{F[i+3], F[i] + array[i+3]\}, if\ (i+3) \le n \end{cases}$$

Пример 1.8.

i	1	2	3	4	5	6	7
$array[i]$	2	7	3	4	8	12	1
$F[i]$	2	$-\infty$	5	6	13	18	14

Решение задачи – значение последней ячейки массива F (для заданного примера $F[7]=14$). Для восстановления самого решения, можно двигаться обратным ходом по массиву F. Движение начинается из последней ячейки F и заканчивается в первой. Пусть j – индекс текущей ячейки, тогда, если $F[j-3]=F[j]-array[j]$, то переходим в ячейку с номером $j=j-3$, иначе полагаем $j=j-2$.

Время работы алгоритма, основанного на методе динамического программирования, $\mathrm{O}(n)$, где n – число кувшинок в пруду. Требуемый объем дополнительной памяти $M=\mathrm{O}(n)$, где n – число кувшинок в пруду.

Полный перебор всех вариантов описывается n-ым числом Фибоначчи: $\Omega(F_n)$, где $F_n - n$-ое число Фибоначчи. Числа Фибоначчи можно вычислить по формуле Бине, которая, несмотря на иррациональность числа Фидия, дает целые числа:

$$F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}},$$

$$\varphi = \frac{1+\sqrt{5}}{2} = 1{,}61803398874989.... - \text{ число Фидия (золотое сечение)},$$

$$\psi = \frac{1-\sqrt{5}}{2} = -\frac{1}{\varphi} \quad \left(\left| \frac{\psi^n}{\sqrt{5}} \right| < 1, \text{ при любом целом неотрицательном } n \right).$$

Сведения из математики. Золотое сечение – это пропорция, которая получается, если линию разделить на две части так, чтобы длинная часть соотносилась с короткой в такой же пропорции, как вся линия соотносится с длинной. Эта пропорция всегда равняется 1,618…, а это число называют «фи».

Поэтому, зная, что формула Бине дает целые числа, можно утверждать, что $F_n = \left(\varphi^n/\sqrt{5}\right)$, тут скобки $(\ )$ означают округление до ближайшего целого значения (если десятичная часть числа <0.5, число округляется вниз до ближайшего целого значения; если десятичная часть числа ≥ 0.5, число округляется вверх до ближайшего целого значения).

Получаем явное преимущество метода динамического программирования, который дает линейный алгоритм, по сравнению с методом полного перебора вариантов [2].

1.3.3.6. Задача расстановки единиц. Модульная арифметика.

Дано число n. Необходимо определить, сколько есть бинарных строк длины n, в которых ровно k единиц. Сами строки выдавать не надо.

Решение.

Количество способов можно посчитать комбинаторно, так как задача сводится к определению числа способов, которыми можно из n разрядов выбрать k, расставляя в выбранные k разрядов единицы:

$$C_n^k = \frac{n!}{k!\cdot(n-k)!}.$$

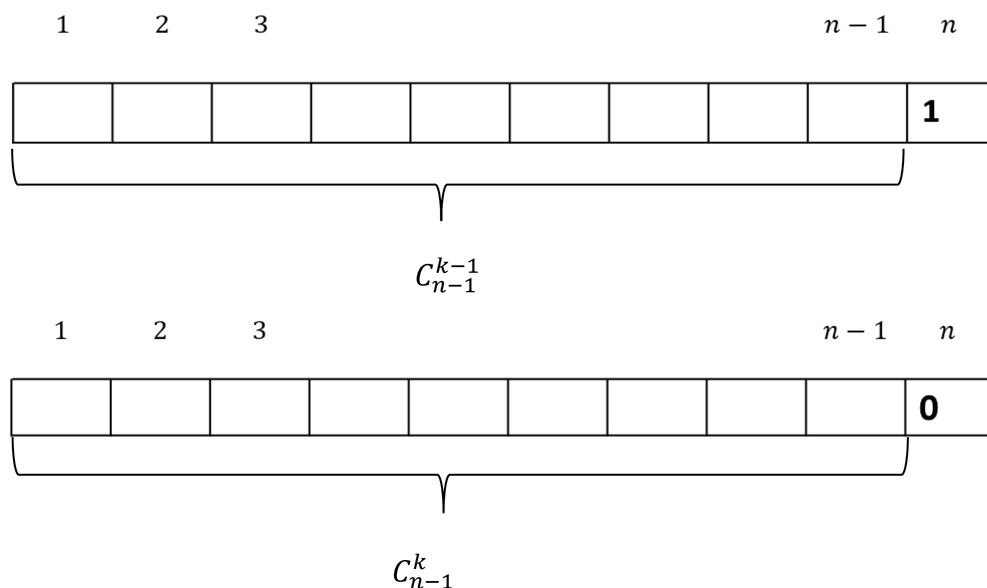
Однако при больших значениях n и k итоговое значение уже может не помещаться в целочисленные типы данных. Например, при подсчете числа сочетаний через факториал при

$$n = 100, k = 1$$

произойдет переполнение, но в тоже время при вычислении с помощью метода ДП не возникнет проблем, так как итоговое значение равно всего лишь 100.

Выпишем рекуррентное соотношение для числа сочетаний (первое слагаемое в выражении соответствует случаю, когда в n-ый разряд ставится единица, а второе слагаемое – не ставится единица):

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k.$$



Обозначим через $F[i, j]$ – число способов, которыми можно в строке длины i расставить ровно j единиц.

Реализуем сначала *двумерное* ДП назад (рисунок 1.11).

Справедливо следующее рекуррентное соотношение:

$$\begin{cases} F[i,i]=1, \quad i=\overline{0,n} \\ F[i,0]=1, \quad i=\overline{1,n} \\ F[i,\ j]=F[i-1,\ j-1]+F[i-1,\ j], \quad i=\overline{1,n},\ j=\overline{1,i-1} \end{cases}$$

	0	1	2	3	4
0	1	X	X	X	X
1	1	1	X	X	X
2	1		1	X	X
3	1			1	X
4	1				1

Рисунок 1.11 – База для ДП назад.

Реализуем теперь *двумерное* ДП вперед (рисунок 1.12).

Справедливо следующее рекуррентное соотношение:

$$\begin{cases} F[0,0]=1, \\ F[i,\ j]=0, \quad i=\overline{1,n},\ j=\overline{0,i} \\ F[i+1,\ j]=F[i+1,\ j]+F[i,\ j], \quad i=\overline{1,n-1},\ j=\overline{0,i} \\ F[i+1,\ j+1]=F[i+1,\ j+1]+F[i,\ j], \quad i=\overline{1,n-1},\ j=\overline{0,i} \end{cases}$$

	0	1	2	3	4
0	1	X	X	X	X
1			X	X	X
2				X	X
3					X
4					

Рисунок 1.12 – База для ДП вперед.

Вычисления элементов матрицы можно организовать, двигаясь, например, по строкам матрицы сверху вниз, а в строке – слева направо.

Пример 1.9.

Дано число $n=4$. Необходимо определить, сколько есть бинарных строк длины n, в которых ровно $k=3$ единиц.

	0	1	2	3	4
0	1	X	X	X	X
1	1	1	X	X	X
2	1	2	1	X	X
3	1	3	3	1	X
4	1	4	6	4	1

Решение задачи – $F[n,k]=F[4,3]=4$.

Время работы алгоритма, основанного на методе динамического программирования,

$$\mathrm{O}(n\cdot k)=\mathrm{O}\left(n^2\right),$$

где n – длина строки, $k\le n$ – число единиц.

Требуемый объем дополнительной памяти $M=\mathrm{O}(n)$, так как, например, при реализации ДП вперед для вычисления элементов строки с номером i требуется информация об элементах строки с номером $i-1$.

На практике, когда результат является достаточно большим числом, в задаче предлагается найти ответ по модулю $(\%\, p)$.

В рассматриваемой задаче можно непосредственно вычислить значение выражения, используя свойства модульной арифметики:

$$\begin{cases} C_n^k=\left(\dfrac{n}{k\cdot(n-k-1)\cdot(n-k)}\cdot C_{n-1}^{k-1}\right) \quad \% \quad p, \\ C_n^0=C_0^k=1. \end{cases}$$

Приведем основные правила модульной арифметики, которые нужно знать при решении задач, в которых предлагается найти ответ по модулю $(\%\, p)$.

Правила модульной арифметики

$$(a+b)\ \%\ p=((a\ \%\ p)+(b\ \%\ p))\ \%\ p$$

$$(a-b)\ \%\ p=((a\ \%\ p)-(b\ \%\ p)+p)\ \%\ p$$

$$(a\cdot b)\ \%\ p=((a\%\ p)\cdot(b\ \%\ p))\ \%\ p$$

Напомним, что в математике существуют следующие эквивалентные формы записи:

$$a \,\%\, p = y,$$
$$a \bmod p = y.$$

Если два числа сравнимы по модулю p, т. е. $a \bmod p = b \bmod p$, то это записывается:

$$a \equiv b \pmod{p}.$$

Малая теорема Ферма утверждает о том, что, если p – простое число, a – целое число, которое не делится на p, то

$$a^{p-1} \equiv 1 \pmod{p}.$$

Следствие из малой теоремы Ферма утверждает о том, что, если p – простое число, a – целое число, то

$$a^{p-2} \equiv \frac{1}{a} \pmod{p},$$

другими словами

$$a^{p-2} \,\%\, p = \frac{1}{a} \,\%\, p.$$

Эта формула означает, что в модульной арифметике деление на некоторое целое число a нужно заменить возведением этого числа a в степень $p-2$ (p – простое число). Для эффективного вычисления значения выражения a^p на практике используют следующий *алгоритм бинарного возведения в степень*:

$$\begin{cases} a^0 = 1, \\ a^p = \begin{cases} \left(a^{\frac{p}{2}}\right)^2, & \text{если } p - \text{ четно;} \\ \left(a^{\frac{p-1}{2}}\right)^2 \cdot a, & \text{если } p - \text{ нечетно.} \end{cases} \end{cases}$$

Данному алгоритму потребуется $O(\log p)$ шагов для возведения целого числа a в степень p.

1.3.3.7. Задача оптимального перемножения группы матриц.

Дана последовательность из s матриц $A_1, A_2, \dots, A_s$. Требуется определить, в каком порядке их следует перемножать, чтобы число атомарных операций умножения было минимальным.

Матрицы предполагаются совместимыми по отношению к матричному умножению (т. е. число столбцов матрицы A_{i-1} совпадает с числом строк матрицы A_i).

Будем считать, что произведение матриц – операция, которая принимает на вход две матрицы размера $k \times m$ и $m \times n$ и возвращает матрицу размера $k \times n$, затратив на это $k \cdot m \cdot n$ атомарных операций умножения.

Базовый тип позволяет хранить любой элемент итоговой и любой возможной промежуточной матрицы, поэтому умножение двух элементов требует одной атомарной операции.

Решение.

Так как перемножение матриц ассоциативно, итоговая матрица не зависит от порядка выполнения операций умножения. Другими словами, нет разницы, в каком порядке расставляются скобки между множителями, результат будет один и тот же. Однако порядок перемножения может существенно повлиять на время работы алгоритма.

Для примера рассмотрим различные порядки, которыми можно перемножить три матрицы, которые согласованы своими размерами:

$$A = A_1 \cdot A_2 \cdot A_3$$
$$A_1 - [20 \times 5]$$
$$A_2 - [5 \times 10]$$
$$A_3 - [10 \times 4]$$

1 порядок:	2 порядок:
$A = (A_1) \cdot (A_2 \cdot A_3)$	$A = (A_1 \cdot A_2) \cdot (A_3)$
$[20 \times 5][5 \times 4]$	$[20 \times 10][10 \times 4]$
Число операций умножения:	Число операций умножения:
$(5 \cdot 10 \cdot 4) + (20 \cdot 5 \cdot 4) = 600$	$(20 \cdot 5 \cdot 10) + (20 \cdot 10 \cdot 4) = 1\,800$

Мы видим, что первый порядок потребует в три раза меньшего числа атомарных операций умножения, чем второй порядок.

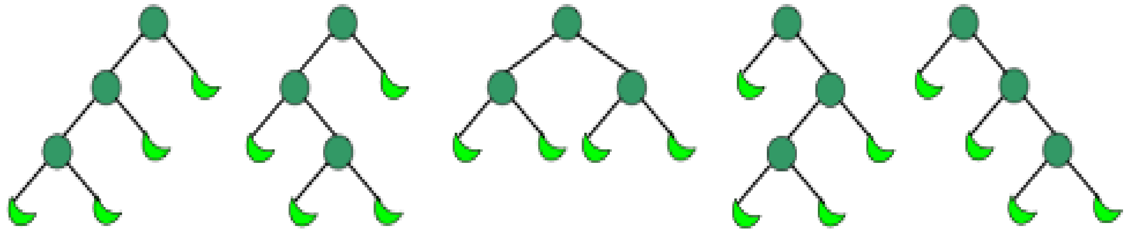
Известно, что количество различных способов перемножения группы матриц $A_1, A_2, \dots, A_s$ задается C_{s-1} числом Каталана.

Напомним, что числа Каталана – это последовательность чисел, названная в честь бельгийского математика Эжен Шарля Каталана.

C_n – обозначение n-ого числа Каталана, которое дает ответ на следующие задачи:

1) Количество способов расстановки скобок в произведении из $(n+1)$ множителя.

2) Количество двоичных корневых деревьев с n листьями, у которых из каждого внутреннего узла выходит ровно 2 узла.



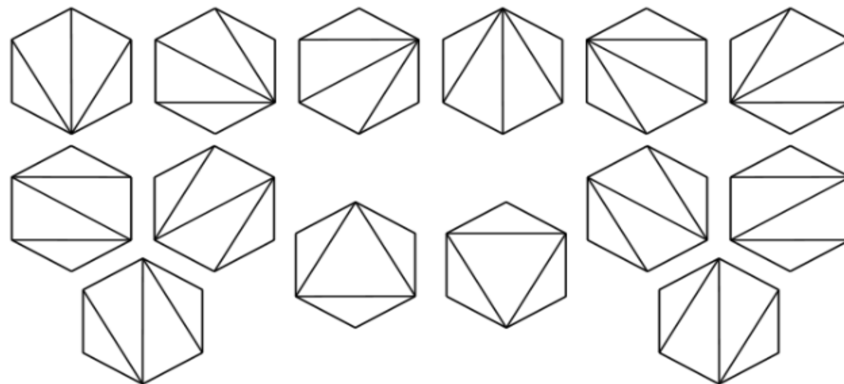
3) Количество правильных скобочных последовательностей длины $2 \cdot n$ (каждая открывающаяся скобка стоит раньше соответствующей ей закрывающейся скобки).

$n = 2$

(( ))

( ) ( )

4) Количество триангуляций выпуклого $(n+2)$-угольника (разбиение на треугольники непересекающимися диагоналями).



Пример последовательности чисел Каталана:

i	0	1	2	3	4	5	6	7	8
C_n	1	1	2	5	14	42	132	429	1 430

Существуют рекуррентные и аналитические формулы для вычисления C_n.

$$C_n = \sum_{k=0}^{n-1} C_k \cdot C_{n-1-k}$$

$$C_n = C_{n-1} \cdot \frac{2 \cdot (2 \cdot n - 1)}{(n+1)}$$

$$C_n = \frac{(2 \cdot n)!}{n! \cdot (n+1)!}$$

$$C_n \approx \frac{4^n}{n^{3/2} \cdot \sqrt{\pi}}$$

Как следует из аналитической формулы, перебор всех возможных способов перемножения матриц потребует экспоненциального времени

$$\approx \frac{4^{s-1}}{n^{3/2} \cdot \sqrt{\pi}}.$$

Покажем, что метод динамического программирования решит задачу оптимального перемножения группы из s матриц за время

$$O\left(s^3\right).$$

Обозначим через $F[i, j]$ – минимальное число операций умножения, чтобы перемножить матрицы с номерами от i до j включительно (т. е. подзадача – это задача перемножения меньшего числа матриц):

$$A_i \cdot A_{i+1} \cdot \ldots \cdot A_k \cdot A_{k+1} \cdot \ldots \cdot A_j$$

$$A_i[n_i \times m_i], m_i = n_{i+1}, \forall i = \overline{1, s-1}$$

Тогда решением исходной задачи будет элемент матрицы $F[1,s]$.

Реализуем *двумерное* ДП назад.

Справедливо следующее рекуррентное соотношение:

$$\begin{cases} F[i,i] = 0, \quad i = \overline{1, s} \\ F[i, i+1] = n_i \cdot m_i \cdot m_{i+1}, \quad i = \overline{1, s-1} \\ F[i, j] = \min_{i \le k < j} \left\{ F[i,k] + F[k+1, j] + n_i \cdot m_i \cdot m_j \right\}, \quad i = \overline{1, s}, \ j > i \end{cases}$$

Матрица F формируется верхним треугольником. Для вычисления элементов матрицы F можно двигаться, например, параллельно главной диагонали, или осуществлять движение по столцам слева направо, а в столбце – снизу вверх.

```
def matrix_chain_multiplication_order (dim):
 n = len(dim)
 m = [[0 for i in range(n)] for j in range(n)]
 for l in range (2, n):
     for i in range (1, n-l+1):
         j = i+l-1
         m[i][j] = float('Inf')
         for k in range (i, j):
             cost = m[i][k] + m[k+1][j] + dim[i-1]* dim[k]* dim[j]
             if cost < m[i][j]:
                m[i][j] = cost
  return m[1][n-1]
```

Пример 1.10. Для заданной группы матриц вычислить минимальное число атомарных операций умножения и указать сам порядок перемножения.

$$A = A_1 \cdot A_2 \cdot A_3 \cdot A_4$$

$$A_1 - [20 \times 5],\ A_2 - [5 \times 35],\ A_3 - [35 \times 4],\ A_3 - [4 \times 25]$$

	1	2	3	4
1	(A_1) 0	$(A_1 \cdot A_2)$ 3 500	$(A_1 \cdot A_2 \cdot A_3)$ 1 100, $k=1$	$(A_1 \cdot A_2 \cdot A_3 \cdot A_4)$ 3100, $k=3$
2	X	(A_2) 0	$(A_2 \cdot A_3)$ 700	$(A_2 \cdot A_3 \cdot A_4)$ 1 200, $k=3$
3	X	X	(A_3) 0	$(A_3 \cdot A_4)$ 3 500
4	X	X	X	(A_4) 0

$F[1,4] = 3100$ – минимальное число выполненных операций умножения для того, чтобы перемножить все матрицы.

Порядок перемножения можно восстановить обратным ходом, двигаясь из клетки $F[1,4]$, используя дополнительную информацию, сохраненную в таблице (параметр k):$\left(\left(A_1\right)\cdot\left(A_2 \cdot A_3\right)\right)\cdot\left(A_4\right)$.

1.3.3.8. Наибольшая общая подпоследовательность.

Даны две последовательности *A* и *B*, каждая имеет длину *n*. Найти наибольшее число *k*, для которого существуют две последовательности индексов

$$0 \le i_1 < i_2 < \ldots < i_k \le n$$

и

$$0 \le j_1 < j_2 < \ldots < j_k \le n$$

такие, что

$$A_{i_1} = B_{j_1}, A_{i_2} = B_{j_2}, \ldots, A_{i_k} = B_{j_k}.$$

Также нужно найти и сами последовательности индексов.

Сформулируем задачу о наибольшей общей подпоследовательности двух конечных последовательносте (англ. longest common subsequence, сокр. **LCS**).

Определение 1.20. Для конечной последовательности $X_n = x_1, x_2, \ldots, x_n$ некоторую ее *подпоследовательность* можно получить, если удалить из X_n некоторое (возможно пустое) множество элементов. Менять элементы последовательности X_n местами нельзя. Удаляемые элементы не обязательно идут подряд.

Например, для последовательности

$$X_5 = x_1, x_2, x_3, x_4, x_5$$

можно получить такую подпоследовательность

$$x_1, \cancel{x_2}, x_3, x_4, \cancel{x_5} = x_1, x_3, x_4.$$

Крайние случаи:

✓ самая короткая – пустая подпоследовательность (удалены все элементы последовательности X_n);

✓ самая длинная – совпадает с самой последовательностью (удалено нулевое множество элементов).

Заданы две конечные последовательности:

$$X_n = x_1, x_2, \ldots, x_n,$$

$$Y_m = y_1, y_2, \ldots, y_m,$$

необходимо найти наибольшую общую подпоследовательность для X_n и Y_m, т. е. общую их подпоследовательность, имеющую максимальную длину (сокр. **НОП**).

$$X_9 = 0, 2, 0, 9, 1, 9, 6, 7, 23$$

$$Y_8 = 2, 5, 0, 1, 1, 9, 5, 5$$

$$\text{НОП}(X_9, Y_8) = 2, 0, 1, 9$$

$$|\text{НОП}(X_9, Y_8) = 4|$$

Задачу нахождения НОП можно решить полным перебором. Для этого для последовательности X_n постороим множество $\overline{X_n}$ всех ее подпоследовательностей. Множество $\overline{X_n}$ содержит 2^n элементов. Теперь для каждой последовательности из множества $\overline{X_n}$ проверяем, является ли она подпоследовательностью для Y_m (в худшем случае для одной проверки понадобится просмотреть всю последовательность Y_m). Таким образом время работы алгоритма нахождения $\text{НОП}(X_n, Y_m)$, основанного на методе полного перебора, $\Omega\left(2^n \cdot m\right)$.

Покажем, что ДП позволит решить задачу $\text{НОП}(X_n, Y_m)$ за время $\text{O}(n \cdot m)$, затраты на дополнительную память $M = \Theta(n \cdot m)$ (если восстанавливать саму НОП не нужно, то затраты памяти можно уменьшить, до $M = \Theta(\min\{n, m\})$.

Обозначим через $F[i, j]$ – длину наибольшей общей подпоследовательности двух префиксов:

$$X_i = x_1, x_2, \ldots, x_i,$$

$$Y_j = y_1, y_2, \ldots, y_j.$$

База ДП: если для подзадачи $\text{НОП}(X_i, Y_j)$ один из префиксов X_i или Y_j пустой, то $F[i, j] = 0$.

		0	1	2	3	4	m
			y_1	y_2	y_3	…	y_m
0		**0**	**0**	**0**	**0**	**0**	**0**
1	x_1	**0**					
2	x_2	**0**					
3	x_3	**0**					
…	…	**0**					
n	x_n	**0**					

Возможны два случая:

1) $x_i = y_j$;

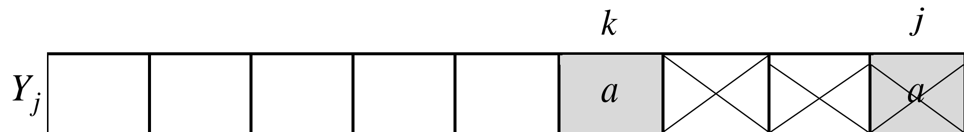
2) $x_i \neq y_j$.

Случай 1. Предположим, что $x_i = y_j$.

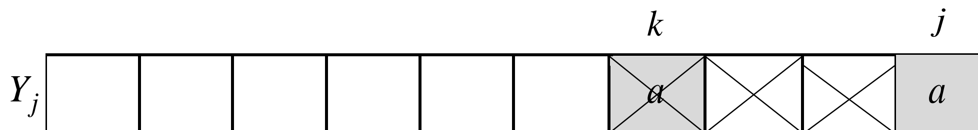
Тогда можно утверждать, что $НОП(X_i, Y_j)$ обязательно будет заканчиваться элементом $x_i = y_j$.

Предположим, что последний элемент $НОП(X_i, Y_j)$ отличен от $x_i = y_j$ (т. е. общая подпоследовательность завершилась элементом, который стоит в префиксах X_i и Y_j не последним). Тогда в конец $НОП(X_i, Y_j)$добавим $x_i = y_j$, получая при этом $ОП(X_i, Y_j)$ большей на 1 длины, чем построенная ранее $НОП(X_i, Y_j)$. Противоречие.

Предположим, что последний элемент $НОП(X_i, Y_j)$ равен x_i, но $\neq y_j$ (т. е. из последовательности Y_j элемент y_j вычеркнули). Тогда в последовательности Y_j есть еще один элемент y_k, $k < j$, который стоит раньше и который $= y_j$, при этом после него ни один из элементов Y_j не вошел в $НОП(X_i, Y_j)$.



Поэтому $НОП(X_i, Y_j)$не изменится, если мы при построении наибольшей общей подпоследовательности вычеркнем из последовательности Y_j элемент y_k, а оставим элемент y_j.



Случай, когда последний элемент $НОП(X_i, Y_j)$ равен y_j, но $\neq x_i$ рассматривается аналогично.

Таким образом мы доказали, что если $x_i = y_j$, то $НОП(X_i, Y_j)$ обязательно будет закачиваться элементом $x_i = y_j$.

Поэтому можно пока забыть про элементы x_i и y_j, решить задачу $НОП(X_{i-1}, Y_{j-1})$, а затем в конец построенной $НОП(X_{i-1}, Y_{j-1})$ приписать элемент x_i $(= y_j)$.

Справедливо следующее рекуррентное уравнение:

$$F[i,\ j] = F[i-1,\ j-1] + 1, \quad \text{если } x_i = y_j.$$

Случай 2. Предположим, что $x_i \neq y_j$.

Тогда можно утверждать, что НОП$\left(X_i, Y_j\right)$ точно не сможет закачиваться на один из этих элементов.

Предположим, что последний элемент НОП$\left(X_i, Y_j\right)$ равен y_j, тогда элемент x_i не войдет в НОП$\left(X_i, Y_j\right)$, поэтому его можно вычеркнуть из X_i и НОП$\left(X_i, Y_j\right) =$ НОП$\left(X_{i-1}, Y_j\right)$.

Предположим, что последний элемент НОП$\left(X_i, Y_j\right)$ равен x_i, тогда элемент y_j не войдет в НОП$\left(X_i, Y_j\right)$, поэтому его можно вычеркнуть из Y_j и НОП$\left(X_i, Y_j\right) =$ НОП$\left(X_i, Y_{j-1}\right)$.

Случай, когда в НОП$\left(X_i, Y_j\right)$ не войдет ни один из элементов x_i, y_j, будет рассмотрен через шаг в подзадачах НОП$\left(X_{i-1}, Y_j\right)$ и НОП$\left(X_i, Y_{j-1}\right)$.

Справедливо следующее рекуррентное уравнение:

$$F[i,\ j] = \max\left\{F[i,\ j-1],\ F[i-1,\ j]\right\}, \quad \text{если } x_i \neq y_j.$$

Объединяя оба случая, получаем следующее рекуррентное соотношение (двумерное ДП назад):

$$\begin{cases} F[0,\ j] = 0, \quad j = \overline{0, m} \\ F[i, 0] = 0, \quad i = \overline{0, n} \\ \\ i = \overline{1, n} \\ j = \overline{1, m} \\ F[i,\ j] = \begin{cases} F[i-1,\ j-1] + 1, & \text{если } x[i] = y[j] \\ \max\left\{F[i-1,\ j], F[i,\ j-1]\right\}, & \text{если } x[i] \neq y[j] \end{cases} \end{cases}$$

Длина наибольшей общей подпоследовательности последовательностей X_n и Y_m будет находиться в нижнем правом углу матрицы – $F\left[n,m\right]$.

Формировать матрицу можно, например, двигаясь по строкам сверху вниз, а в одной строке – слева направо. Либо можно двигаться по столбцам слева направо, а в столбце – сверху вниз.

Так как для вычисления элемента матрицы нужны только элементы этой же строки, стоящие левее, а также элементы предыдущей строки, то, если не

требуется восстановить саму наибольшую общую подпоследовательность, то можно ограничиться дополнительной памятью

$$M = \Theta\left(\min\{n, m\}\right).$$

Однако время, затрачиваемое на вычисление всех элементов, остается равным

$$\Theta\left(n \cdot m\right).$$

Пример 1.11. Для последовательностей:

$$X_5 = \text{м, о, т , м, а}$$
$$Y_6 = \text{м, а, м , и, т, а}$$

найти $\text{НОП}\left(X_5, Y_6\right)$.

Требуется не только найти $\left|\text{НОП}\left(X_5, Y_6\right)\right|$, но и восстановить саму наибольшую общую подпоследовательность.

Решение.

		0	1	2	3	4	5	$m=6$
			м	а	м	и	т	а
0		0	0	0	0	0	0	0
1	м	0	1	1	1	1	1	1
2	о	0	1	1	1	1	1	1
3	т	0	1	1	1	1	2	2
4	м	0	1	1	2	2	2	2
$n=5$	а	0	1	2	2	2	2	3

Для восстановление наибольшей общей подпоследовательности будем использовать «обратный ход»:

1) стартуем из клетки матрицы $F[n,m]$;

2) движение осуществляем до тех пор, пока не придем в нулевую строку или нулевой столбец матрицы (либо пока не придем к нулевому элементу матрицы);

3) предположим, что мы находимся в клетке таблицы $[i,j]$, тогда движение осуществляем по следующему правилу:

– если $x_i = y_j$, то добавляем $x_i\ (= y_j)$ к ответу и переходим к $F[i-1, j-1]$;

– если $x_i \neq y_j$, то переходим к любому из элементов $F[i-1, j], F[i, j-1]$, который равен $F[i,j]$;

4) Для получения $\text{НОП}(X_n, Y_m)$ нужно перевернуть полученный ответ.

Несложно видеть, что время восстановления $\text{НОП}(X_n, Y_m)$ есть $O(n+m)$.

		0	1	2	3	4	5	$m=6$
			м	а	м	и	т	а
0		0	0	0	0	0	0	0
1	м	0	1	1	1	1	1	1
2	о	0	1	1	1	1	1	1
3	т	0	1	1	1	1	2	2
4	м	0	1	1	2	2	2	2
$n=5$	а	0	1	2	2	2	2	3

$X_5 =$ м, ~~о~~, т, ~~м~~, а

$Y_6 =$ ~~м~~, ~~а~~, м, ~~и~~, т, а

$\text{НОП}(X_5, Y_6) =$ м, т, а

1.3.3.9. Наибольшая общая подпоследовательность-палиндром.

Задана непустая строка S длины n, которая состоит только из строчных латинских букв:

$$S_n = s_1, s_2, \dots, s_n.$$

Необходимо удалить из строки S минимальное число символов так, чтобы получился палиндром (строка символов, которая читается слева направо и справа налево одинаково). Например, для строки

$$b\,a\,c\,a\,b\,d\,a\,e\,b\,a$$

наибольшая подпоследовательность-палиндром

$$b\,a\,c\,a\,b\,.$$

Рассмотрим сначала *неявное решение задачи*. Предположим, что у нас задана строка

$$S_n = s_1, s_2, \dots, s_n\,.$$

Перевернем эту строку:

$$\overline{S}_n = s_n, s_{n-1}, \dots, s_1\,.$$

Теперь длина наибольшей подпоследовательности-палиндрома для последовательности S_n равна длине наибольшей общей подпоследовательности двух последовательностей: S_n и $\overline{S}_n$.

Время работы алгоритма, основанного на неявном решении задачи $\Theta(n \cdot m)$. Объем дополнительной памяти $M = \Theta\left(n^2\right)$, если не требуется восстановить сам палиндром, то дополнительная память $M = \Theta(n)$.

Однако, при восстановлении самого палиндрома, может возникнуть ситуация, когда будет восстановлена некоторая $\text{НОП}\left(S_n, \overline{S}_n\right)$, но она не будет палиндромом (т. к. не каждая наибольшая общая подпоследовательность строки и инвертированной строки является палиндромом).

Проиллюстрируем это утверждение на следующем примере:

$S_{10} = b\,a\,c\,a\,b\,d\,a\,e\,b\,a,$

$\overline{S}_{10} = a\,b\,e\,a\,d\,b\,a\,c\,a\,b,$

$\left|\text{НОП}\left(S_{10}, \overline{S}_{10}\right)\right| = 5,$

$\text{НОП}\left(S_{10}, \overline{S}_{10}\right) = b\,a\,c\,a\,b -$ палиндром,

$\text{НОП}\left(S_{10}, \overline{S}_{10}\right) = b\,a\,d\,b\,a -$ не палиндром.

Продемонстрируем сначала «обратный ход», который привел к получению НОП$\left(S_{10}, \overline{S}_{10}\right) = bacab$, которая является палиндромом.

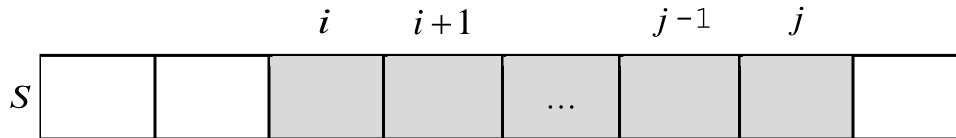
		0	1	2	3	4	5	6	7	8	9	10
			a	b	e	a	d	b	a	c	a	b
0		0	0	0	0	0	0	0	0	0	0	0
1	b	0	0	1	1	1	1	1	1	1	1	1
2	a	0	1	1	1	2	2	2	2	2	2	2
3	c	0	1	1	1	2	2	2	2	3	3	3
4	a	0	1	1	1	2	2	2	3	3	4	4
5	b	0	1	2	2	2	2	3	3	3	4	5
6	d	0	1	2	2	2	3	3	3	3	4	5
7	a	0	1	2	2	3	3	3	4	4	4	5
8	e	0	1	2	3	3	3	3	4	4	4	5
9	b	0	1	2	3	3	3	4	4	4	4	5
10	a	0	1	2	3	4	4	4	5	5	5	5

Продемонстрируем теперь «обратный ход», который привел к получению НОП$\left(S_{10}, \overline{S}_{10}\right) = badba$, которая не является палиндромом.

		0	1	2	3	4	5	6	7	8	9	10
			a	b	e	a	d	b	a	c	a	b
0		0	0	0	0	0	0	0	0	0	0	0
1	b	0	0	1	1	1	1	1	1	1	1	1
2	a	0	1	1	1	2	2	2	2	2	2	2
3	c	0	1	1	1	2	2	2	2	3	3	3
4	a	0	1	1	1	2	2	2	3	3	4	4
5	b	0	1	2	2	2	2	3	3	3	4	5
6	d	0	1	2	2	2	3	3	3	3	4	5
7	a	0	1	2	2	3	3	3	4	4	4	5
8	e	0	1	2	3	3	3	3	4	4	4	5
9	b	0	1	2	3	3	3	4	4	4	4	5
10	a	0	1	2	3	4	4	4	5	5	5	5

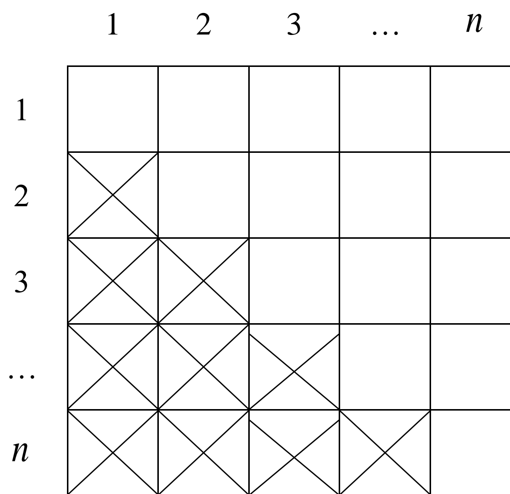
Рассмотрим теперь *явное решение задачи* (двумерное ДП назад).

Обозначим через $F[i, j]$ длину наибольшего палиндрома, который можно получить, если мы рассматриваем элементы строки S от индекса i до индекса j включительно.



Тогда матрица F задается верхним треугольником, так как для того, чтобы подстрока существовала, должно выполняться неравенство $i \le j$.

Решение задачи – элемент матрицы $F[1,n]$.



Справедливо следующее рекуррентное соотношение.

$$\begin{cases} i = \overline{1, n} \\ \quad F[i,i] = 1, \\ i = \overline{1, n-1} \\ \quad F[i,i+1] = \begin{cases} 1, \text{ если } s[i] \ne s[i+1] \\ 2, \text{ если } s[i] = s[i+1] \end{cases} \\ i = \overline{1, n-2}, \quad j > i \\ \quad F[i,\ j] = \begin{cases} F[i+1,\ j-1] + 2, & \text{если } s[i] = s[j] \\ \max\{F[i+1,\ j], F[i,\ j-1]\}, & \text{если } s[i] \ne s[j] \end{cases} \end{cases}$$

Формирование матрицы F может осуществляться по столбцам слева направо, а в столбце – снизу вверх.

Пример 1.12. Для строки $S = a\,s\,d\,f\,s\,l\,a$ построить палиндром наибольшей длины (явное решение). Требуется выдать не только длину наибольшего палиндрома, но и сам палиндром.

Решение.

		1	2	3	4	5	6	7
		a	*s*	*d*	*f*	*s*	*l*	*a*
1	*a*	1	1	1	1	3	3	5
2	*s*		1	1	1	3	3	3
3	*d*			1	1	1	1	1
4	*f*				1	1	1	1
5	*s*					1	1	1
6	*l*						1	1
7	*a*							1

Длина наибольшего палиндома строки $S = a\ s\ d\ f\ s\ l\ a$ равна $F[1,7] = 5$.

Для восстановления самого палиндрома P выполняем «обратный ход» из клетки $[1,n]$. Предположим, что мы находимся в клетке $[i, j]$. Если $s[i] = s[j]$, то добавляем к ответу P символ $s[i]\,(= s[j])$ и переходим в клетку $[i+1, j-1]$. Если $s[i] \neq s[j]$, то переходим в ту из клеток $[i, j-1]$ или $[i+1, j]$, в которой в таблице F стоит то же число, что и в клетке $[i, j]$. Движение осуществляем до тех пор, пока $i \le j$.

Предположим, что в результате обратного хода была сформирована строка P нечетной длины m (в примере $P = p_1, p_2, p_3 = a\,s\,f$, $m = 3$). Дополним строку P до палиндрома, добавляя к ней символы $p_{m-1}, p_{m-2}, \ldots, p_1$ (если m – четно, то добавляем символы $p_m, p_{m-1}, \ldots, p_1$).

Для примера 1.12 получим наибольший палиндром: $P = a\,s\,f\,s\,a$.

Время работы алгоритма – $\Theta(n^2)$, объем дополнительной памяти в случае, когда надо восстановить палиндром, $M = \Theta(n^2)$.

1.3.3.10. Наибольшая строго возрастающая подпоследовательность.

Необходимо из заданной числовой последовательности $X_n = x_1, x_2, \ldots, x_n$, состоящей из n элементов, вычеркнуть минимальное число элементов так, чтобы оставшиеся элементы образовали строго возрастающую подпоследовательность элементов (сокр. НСВП) (или требуется найти наибольшую возрастающую подпоследовательность, сокр. НВП (англ. Longest Increasing Subsequence, LIS)).

Пример 1.13.

$$X_8 = 0, 2, 8, 1, 9, 6, 7, 23;$$

$$\text{НСВП}(X_8) = 0, 2, 8, \not{1}, 9, \not{6}, \not{7}, 23; \left|\text{НСВП}(X_8)\right| = 5.$$

Рассмотрим сначала *неявное решение задачи*.

Предположим, что у нас задана строка $X_n = x_1, x_2, \ldots, x_n$. Отсортируем эту строку по неубыванию, получим последовательность $Y_n = y_1, y_2, \ldots, y_n, y_1 \le y_2 \le \cdots \le y_n$. Сделать это можно, например, используя сортировку слиянием, за время $O(n \cdot \log n)$ [6].

Если требуется, чтобы подпоследовательность строго возрастала, то удалим из Y_n повторяющиеся элементы. Длина наибольшей строго возрастающей подпоследовательности для последовательности X_n равна длине $\text{НОП}(X_n, Y_n)$.

Время работы неявного решения $\Theta(n^2)$.

Требуемый объем дополнительной памяти $M = \Theta(n^2)$.

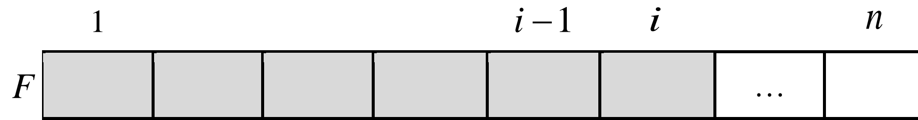
Если не требуется восстанавливать саму строго возрастающую последовательность, то объем дополнительной памяти $M = \Theta(n)$.

Решение (неявное решение задачи для примера 1.13).

		0	1	2	6	7	8	9	23
	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1
2	0	1	1	2	2	2	2	2	2
8	0	1	1	2	2	2	3	3	3
1	0	1	2	2	2	2	3	3	3
9	0	1	2	2	2	2	3	4	4
6	0	1	2	2	3	3	3	4	4
7	0	1	2	2	3	4	4	4	4
23	0	1	2	2	3	4	4	4	5

Рассмотрим теперь *явное* решение задачи (одномерное ДП назад).

Обозначим через $F[i]$ – длину наибольшей строго возрастающей подпоследовательности, которая обязательно заканчивается элементом x_i:



Решение задачи – максимальное значение среди элементов массива F, так как наибольшая строго возрастающая подпоследовательность может заканчиваться на любом элементе последовательности.

Рассмотрим некоторую строго возрастающую подпоследовательность, которая заканчивается элементом x_i. В этой последовательности элемент x_i является последним и остальные элементы этой последовательности строго меньше, чем он. Непосредственно перед x_i могут оказаться те из элементов $x_1, x_2, \ldots, x_{i-1}$, которые меньше, чем x_i.

Для того, чтобы получить последовательность наибольшей длины, заканчивающуюся элементом x_i, нужно, чтобы перед x_i стоял тот из элементов $x_j < x_i \ (j = 1,...,i-1)$, для которого значение $F[j]$ наибольшее.

Справедливо следующее рекуррентное уравнение:

$$\begin{cases} F[i] = 1, i = 1,..., n \\ F[i] = \max_{\substack{x_j < x_i \\ j=1,...,i-1}} F[j] + 1, i = 2,..., n \end{cases}$$

Для $X_8 = 0, 2, 8, 1, 9, 6, 7, 23$ массив F будет иметь следующий вид (восстанавливаем НСВП «обратным ходом»):

	1	2	3	4	5	6	7	8
$X[i]$	0	2	8	1	6	6	7	23
$F[i]$	1	2	3	2	3	4	4	5
	↑			↑		↑	↑	↑

$$|\text{НСВП}(X_8)| = 5;$$

$$\text{НСВП}(X_8) = 0, 1, 6, 7, 23.$$

Время работы алгоритма $\Theta(n^2)$.

Требуемый объем дополнительной памяти $M = \Theta(n)$.

Существует более эффективная реализация приведенного выше алгоритма, которая работает за время $O(n \cdot \log n)$, а требуемая дополнительная память $M = \Theta(n)$.

Среди всех строго возрастающих последовательностей одной длины назовем более *перспективной* ту последовательность, к которой можно присоединить большее число элементов (среди последовательностей одной длины она заканчивается на наименьший элемент).

Например, среди строго возрастающих последовательностей длины 4:

1, 2, 18, 102

1, 4, 19, 105

7, 9, 26, 100

более перспективная: 7, 9, 26, 100.

В процессе работы алгоритма среди последовательностей одной длины будем оставлять только самую перспективную (если к этой последовательности не сможем присоединить элемент, то и ко всем остальным последовательностям такой же длины не сможем его присоединить).

По построению последние элементы перспективных последовательностей будут упорядочены по неубыванию, что позволит для очередного элемента x искать подпоследовательность наибольшей длины, к которой его можно присоединить, используя алгоритм бинарного поиска (UpperBound(x)) [1].

Схематично работу алгоритма можно продемонстрировать на следующем примере. Предположим, что в процессе алгоритма найдены перспективные последовательности длины $1, 2, \ldots, 8$ (последние элементы этих перспективных последовательностей хранятся в массиве F, который строго возрастает).

1	1		1
2	2		2
3	4		4
4	12	$i = \text{UpperBound}(6)$ если $x[i-1] \neq x$ то $F[i] = x$	6
5	17		17
6	19		19
7	23		23
8	77		77
9	–		–

На вход поступает число $x = 6$.

В упорядоченном массиве F находим первый элемент > 6, это элемент массива F по индексу $i = 4$.

Так как $x[i-1] \neq x[i]$, то это означает, что ко всем строго возрастающим последовательностям длины 1, 2, 3 элемент $x = 6$ можно присоединить, а к строго возрастающим последовательностям большей длины – нет.

Так как необходимо, чтобы последовательность была наибольшей длины, то присоединяем $x = 6$ к последовательности длины 3, получаем при этом последовательность длины 4, которая более перспективна, чем та, которая была построена ранее $(6 < 12)$, поэтому изменяем значение 4 элемента массива $F[4] = 6$.

Отметим, что, если $x[i-1] = x[i]$, то в этом случае при решении задачи НСВП ничего делать не надо, т. е. массив *F* не изменится. При решении задачи НВП всегда выполняем присваивание: $F[\text{UpperBound}(x)] = x$.

1.3.3.11. Наименьшее взвешенное редакционное расстояние Левенштейна.

Предположим, что у нас есть две символьные строки одинаковой длины:

$$X = x_1, x_2, \ldots, x_n$$

$$Y = y_1, y_2, \ldots, y_n.$$

Р.У. Хэмминг (1915 – 1998), США

Тогда для определения близости двух строк можно использовать следующую метрику (расстояние Хэминга):

– число позиций, в которых соответствующие символы двух строк одинаковой длины отличаются.

Если расстояние Хэмминга равно 1, то говорят, что строки являются «соседними».

Например, для строк *X* и *Y*, приведенных в таблице,

X	а	л	г	о	р	и	т	м	и	к	а
Y	о	л	г	о	р	р	т	м	и	к	а

расстояние Хэмминга $d(X,Y) = 2$.

Если строки имеют разную длину:

$$X = x_1, x_2, \ldots, x_n$$

$$Y = y_1, y_2, \ldots, y_m$$

то для определения близости строк можно использовать следующую метрику: расстояние Левенштейна.

В.И. Левенштейн
(1935-2017)
СССР

Расстояние Левенштейна для двух строк равно минимальному числу односимвольных «редакторских правок»:

– замена символа в строке $X = x_1, x_2, \ldots, x_n$ (**R** – replace);
– удаление символа из строки $X = x_1, x_2, \ldots, x_n$ (**D** – delete);
– вставка символа в строку $X = x_1, x_2, \ldots, x_n$ (**I** – insert);

Например, для строк *X* и *Y*, приведенных в таблице, расстояние Левенштейна $d(X,Y) = 4$.

X		л	г	о	р	р	и	т	м	и	к		к	к
Y	а	л	г	о	р		и	т	м	и	к			а
	I					**D**							**D**	**R**

Для решения задачи применим метод динамического программирования. Обозначим через $F[i, j]$ расстояние Левенштейна для *i*-ого префикса строки *X* и *j*-ого префикса строки *Y*:

$$X_i = x_1, x_2, \ldots, x_i,$$
$$Y_j = y_1, y_2, \ldots, y_j.$$

Тогда база ДП (граничные условия, когда один из префиксов – пустая строка):

$$\begin{cases} F[0, j] = j, \quad j = \overline{0, m} \quad (I - \text{операции вставки символа}) \\ F[i, 0] = i, \quad i = \overline{0, n} \quad (D - \text{операции удаления символа}) \end{cases}$$

Решение задачи – $F[n, m]$.

		0	1	2	3	4	m
			y_1	y_2	y_3	…	y_m
0		**0**	**1**	**2**	**3**	**…**	$\boldsymbol{m}$
1	x_1	**1**					
2	x_2	**2**					
3	x_3	**3**					
…	…	**…**					
n	x_n	$\boldsymbol{n}$					

1. Предположим, что x_i был удален.

Тогда, чтобы получить Y_j, нужно преобразовать X_{i-1} в Y_j, а потом удалить x_i:

$$X_i = \left(x_1, x_2, \dots, x_{i-1}\right), x_i \xrightarrow{F(i-1, j)} \left(y_1, y_2, \dots, y_j\right), x_i \xrightarrow{D(x_i)} y_1, y_2, \dots, y_j = Y_j$$

2. Предположим, что y_j был добавлен.

Тогда, чтобы получить Y_j, нужно преобразовать X_i в Y_{j-1}, а потом добавить y_j:

$$X_i = x_1, x_2, \dots, x_i \xrightarrow{F(i, j-1)} y_1, y_2, \dots, y_{j-1} \xrightarrow{I(y_j)} \left(y_1, y_2, \dots, y_{j-1}\right), y_j = Y_j$$

3. Предположим, что x_i не удаляли, а y_j не добавляли.

Если $x_i = y_j$, то преобразуем X_{i-1} в Y_{j-1}, а элемент $y_j \left(= x_i\right)$ уже стоит на своем месте:

$$X_i = x_1, x_2, \dots, x_i = \left(x_1, x_2, \dots, x_{i-1}\right), y_j \xrightarrow{F(i-1, j-1)} \left(y_1, y_2, \dots, y_{j-1}\right), y_j = Y_j$$

Если $x_i \neq y_j$, то преобразуем X_{i-1} в Y_{j-1}, а элемент x_i заменим на y_j:

$$X_i = x_1, x_2, \dots, x_i \xrightarrow{F(i-1, j-1)} \left(y_1, y_2, \dots, y_{j-1}\right), x_i \xrightarrow{R(x_i, y_j)} y_1, y_2, \dots, y_j = Y_j$$

Обозначим

$$\delta\left(x_i, y_j\right) = \begin{cases} 1, & \text{если} \quad x_i \neq y_j; \\ 0, & \text{если} \quad x_i = y_j. \end{cases}$$

Тогда справедливо следующее рекуррентное соотношение:

$$\begin{cases} F[0, j] = j, \quad j = \overline{0, m} \\ F[i, 0] = i, \quad i = \overline{0, n} \\ F[i, j] = \min\left\{F[i-1, j] + 1, F[i, j-1] + 1, F[i-1, j-1] + \delta\left(x_i, y_j\right)\right\} \end{cases}$$

Предположим, что одиночные операции вставки, удаления и замены имеют разную стоимость:

$p(\boldsymbol{D})$ – стоимость удаления одного символа;

$p(\boldsymbol{I})$ – стоимость вставки одного символа;

$p(\boldsymbol{R})$ – стоимость замены одного символа на другой.

Рассуждая аналогичным образом, получаем рекуррентное соотношения для нахождения минимального *взвешенного рассстояния Левенштейна* близости двух строк:

$$\begin{cases} F[0,j] = p(I)\cdot j, \quad j=\overline{0,m} \\ F[i,0] = p(D)\cdot i, \quad i=\overline{0,n} \\ F[i,j] = \min\begin{Bmatrix} F[i-1,j]+p(D), F[i,j-1]+p(I), \\ F[i-1,j-1]+\delta(x_i,y_j)\cdot\min(p(R), p(D)+p(I)) \end{Bmatrix} \end{cases}$$

Заметим, что из формулы следуют следующие неравенства:

$$F[i,j] \le F[i-1,j]+p(D) \quad (1)$$

$$F[i,j] \le F[i,j-1]+p(I) \quad (2)$$

Из неравенств (1) и (2) следует, что всегда будет выполняться неравенство:

$$\begin{aligned} & F[i,j] \le (\text{н-во } 1) \le F[i-1,j]+p(D) \le (\text{н-во } 2) \\ & \le (F[i-1,j-1]+p(I)) + p(D) = \\ & = F[i-1,j-1]+p(I)+p(D). \end{aligned}$$

Поэтому случай замены x_i на y_j через две операции:

удаления x_i за $p(D)$

добавления y_j за $p(I)$

можно не рассматривать отдельно, так как он уже будет учтен.

Следовательно, справедлива более простая формула для вычисления наименьшего взвешенного расстояния Левенштейна близости двух строк:

$$\begin{cases} F[0,\,j] = p(I)\cdot j, \quad j=\overline{0,m} \\ F[i,0] = p(D)\cdot i, \quad i=\overline{0,n} \\ F[i,j] = \min\left\{ f[i-1,j] + p(D), f[i,j-1] + p(I), f[i-1,j-1] + \delta\left(x_i, y_j\right)\cdot p(R)\right\} \end{cases}$$

Формировать матрицу F можно по строкам сверху вниз, а в одной строке – слева направо.

		0	1	2	3	4	m
			y_1	y_2	y_3	…	y_m
0		**0**	**1**	**2**	**3**	**…**	***m***
1	x_1	**1**	→	→	→		
2	x_2	**2**	→	→	→		
3	x_3	**3**	→	→	→		
…	…	**…**	→	→	→		
n	x_n	***n***	→	→	→		

Время работы алгоритма $\Theta(n\cdot m)$.

Дополнительная память $M = \Theta(n\cdot m)$.

Пример 1.14.
Рассмотрим две строки:

$X =$ лгорритмиккк,

$Y =$ алгоритмика.

Найдем наименьшее расстояние Левенштейна близости двух строк, если

$$p(I) = p(D) = p(R) = 1\,.$$

Решение.

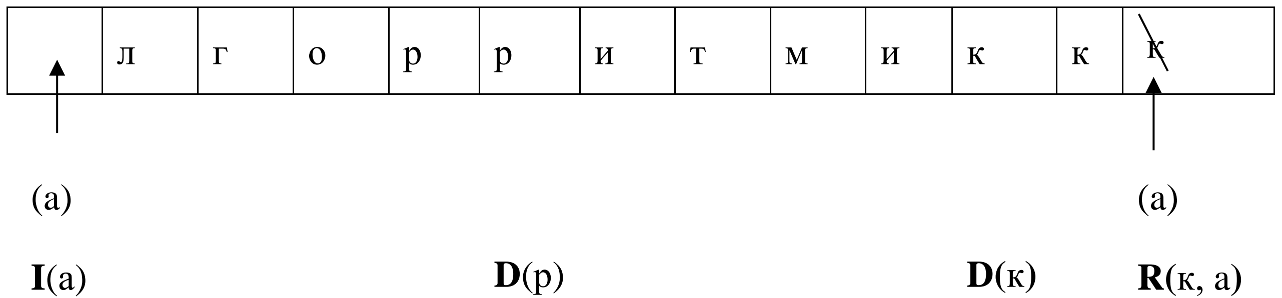
		0	1	2	3	4	5	6	7	8	9	10	11
			а	л	г	о	р	и	т	м	и	к	а
0		0	1	2	3	4	5	6	7	8	9	10	11
1	л	1	1	1	2	3	4	5	6	7	8	9	10
2	г	2	2	2	1	2	3	4	5	6	7	8	9
3	о	3	3	3	2	1	2	3	4	5	6	7	8
4	р	4	4	4	3	2	1	2	3	4	5	6	7
5	р	5	5	5	4	3	2	2	3	4	5	6	7
6	и	6	6	6	5	4	3	2	3	4	4	5	6
7	т	7	7	7	6	5	4	3	2	3	4	5	6
8	м	8	8	8	7	6	5	4	3	2	3	4	5
9	и	9	9	9	8	7	6	5	4	3	2	3	4
10	к	10	10	10	9	8	7	6	5	4	3	2	3
11	к	11	11	11	10	9	8	7	6	5	4	3	3
12	к	12	12	12	11	10	9	8	7	6	5	4	**4**

Решение задачи находится в нижнем правом углу матрицы:

$$F[12,11]=4,$$

значит за 4 операции можно из строки $X=$ лгорритмиккк получить строку $Y=$ алгоритмика.

Обратным ходом по матрице F можно восстановить порядок выполненных действий, который необходим для преобразования строк:



1.3.3.12. Динамическое программирование по профилю.

Для начала давайте определимся, что такое профиль. В жизни вы встречались с профилями, когда обозначали отличия одного объекта от другого. Так, например, класс физико-математического профиля отличает повышенное количество часов изучения физики и математики, профиль человеческого лица отличается, например, формой носа, профиль города отличается формой зданий и т. д. Для *динамического программирования по профилю* будем понимать набор признаков, по которым различаются подзадачи.

Рассмотрим задачу, которая называется «Симпатичные узоры» (https://acmp.ru/asp/do/index.asp?main=task&id_course=2&id_section=15&id_topic=18&id_problem=96).

Суть задачи состоит в том, что нужно посчитать количество симпатичных узоров, которые можно выложить белыми и черными квадратными плитками 1×1 площадку размером $n\times m$.

Узор назовем симпатичным, если в нем не встречается квадрата 2×2, выложенного плитками одинакового цвета.

Так, три узора слева на рисунке 1.13 симпатичные, а три справа – нет.

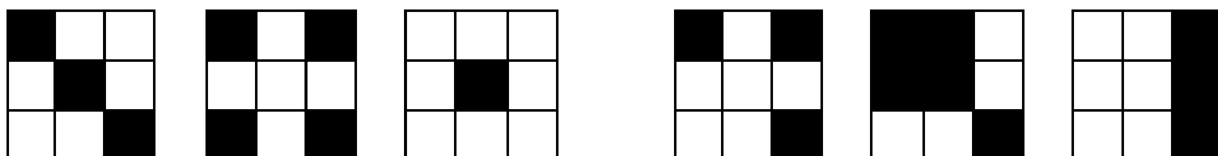


Рисунок 1.13.

Проверить узор на симпатичность можно достаточно просто – перебираем плитки, и смотрим на их соседей справа, снизу и справа/снизу. Если плитка совпадает по цвету с соседями, то узор не симпатичный.

Сложность проверки $\mathrm{O}(n\cdot m)$.

Можно было бы перебрать все возможные узоры, и каждый из них проверить на симпатичность. Версия задачи на acmp.ru имеет меньшие ограничения, и вариантов узоров будет $2^{n\cdot m}\le2^{30}$. Но даже в этом случае узоров будет слишком много, чтобы за 1 секунду проверить их все, а в iRunner у задачи еще более жесткие ограничения. Поэтому задачу нельзя решить перебором.

Первое, что мы сделаем, это отсортируем m и n по неубыванию (решения задач одинаковые по значению, а нам проще работать с меньшим по значению параметром).

Оценим, в каком диапазоне может быть меньшее ограничение:

$$1\times30, 2\times15, 3\times10, 5\times6,$$

то есть $1\le m\le5$.

Давайте для примера разберем решение задачи при $m = 3$. То есть узор представляет собой полоску шириной в 3 плитки. Нам нужно выяснить, каким количеством симпатичных узоров можно ее выложить.

Рассмотрим какой-то из узоров:

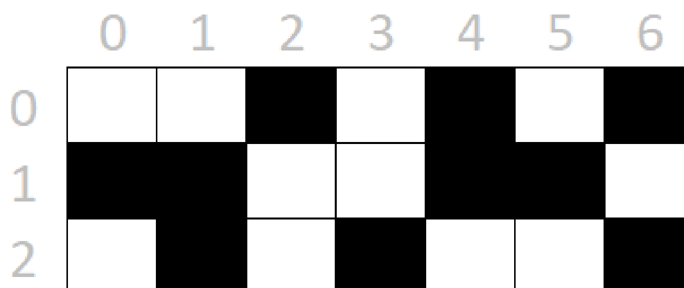


Рисунок 1.14.

Попробуем применить метод рекуррентного соотношения.

Будем разбивать задачу на подзадачи. Так как у нас есть полоска длины n, попробуем решить задачу, зная, сколько симпатичных узоров можно выложить на полоске длиной $n-1$.

В отличие от рассмотренных ранее простых задач (например, факториал или числа Фибоначчи) мы не можем взять любой столбик из 3-х плиток и продолжить им предыдущую полоску.

В некоторых случаях это может привести к тому, что узор станет несимпатичным (например, нельзя продублировать столбики 1, 2, 3 или 4 с рисунка 1.14).

Тогда введем понятие *профиля столбика*, то есть его цветовую раскладку, и понятие совместимости столбиков по профилю, то есть можно ли добавить столбик с профилем X после столбика с профилем Y, чтобы не нарушилась симпатичность узора.

Проверять можно каждый раз заново, но скорее всего будет выгодно предварительно создать таблицу совместимостей.

Давайте пронумеруем профили, рассматривая в качестве номера двоичное число, где 1 в очередном разряде будет обозначать черную плитку на соответствующей клетке столбика (для столбика из 3 клеток будет $2^3 = 8$ профилей, рисунок 1.15):

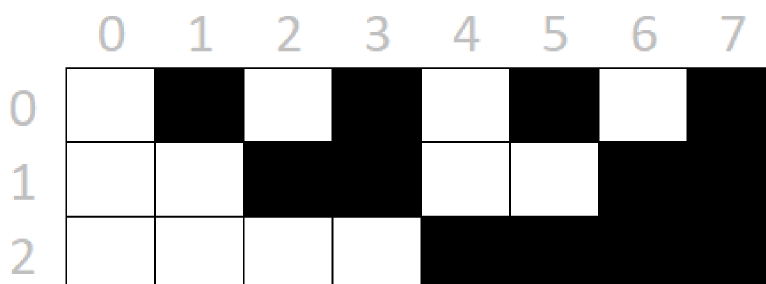


Рисунок 1.15.

Тогда построим таблицу, можно ли продолжить профилем *j* профиль *i*:

	0	1	2	3	4	5	6	7
0	0	0	1	1	0	1	1	1
1	0	0	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	0	1	1	1	0
4	0	1	1	1	0	1	1	1
5	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	0	0
7	1	1	1	0	1	1	0	0

То есть, например, после профиля 0 могут идти только профили 2, 3, 5, 6 и 7, а после профиля 2 может идти любой.

Оценим затраты на построение справочника. Памяти понадобится

$$M\left(k^2\right),$$

где k – количество профилей (в нашем конкретном случае $k = 2^3 = 8$).

На проверку совместимости профилей тратим m операций, где m – высота столбика, значит, сложность алгоритма заполнения справочника

$$\mathrm{O}\left(k^2 \cdot m\right) = \mathrm{O}\left(\left(2^m\right)^2 \cdot m\right).$$

Теперь можно построить двумерное ДП, в котором одним параметром будет номер профиля, а другим – длина полоски. То есть в ячейке *i*, *j* будет храниться количество полосок длиной *j*, образующих симпатичных узор, заканчивающийся профилем *i*.

Ограничением нашего рекуррентного соотношения будет 1-й столбец, заполненный единицами (существует ровно один способ заполнить столбик $M \times 1$ каждым из профилей).

Остальные ячейки матрицы заполняются по формуле:

$$F\left[i, j\right] = \sum_{k=0}^{2^m-1} F[k, j-1] \cdot A\left[i, k\right],$$

где A – справочник.

Ответом на задачу будет сумма элементов в столбце n основной матрицы.

Давайте оценим затраты на построение основной таблицы. Памяти понадобится $M\left(2^m \cdot n\right)$. Вычислительная сложность алгоритма $\mathrm{O}\left(\left(2^m\right)^2 \cdot n \cdot m\right)$ (если нет справочника A), $\mathrm{O}\left(\left(2^m\right)^2 \cdot n\right)$ (если есть справочник A).

2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

2.1. Общие задачи по теме 1.3

Задача 0.1. Путь лягушки.

В одном очень длинном и узком пруду по кувшинкам прыгает лягушка. Кувшинки в пруду расположены в один ряд. Лягушка начинает прыгать с первой кувшинки ряда и хочет закончить на последней. Но в силу вредности характера лягушка согласна прыгать только вперед через одну или через две кувшинки. Например, с кувшинки номер 1 она может прыгнуть лишь на кувшинки номер 3 и номер 4.

На некоторых кувшинках сидят комарики. А именно, на i-й кувшинке сидят a_i комаров. Когда лягушка приземляется на кувшинку, она съедает всех комариков, сидящих на ней.

Лягушка хочет спланировать свой маршрут так, чтобы съесть как можно больше комаров. Помогите ей: скажите, какие кувшинки она должна посетить на своем пути.

Формат входных данных

Первая строка входного файла содержит число n – число кувшинок в пруду $(1 \le n \le 100\,000)$.

Вторая строка содержит n чисел, которые разделены пробелами: i-ое число сообщает, сколько комаров сидит на i-ой кувшинке $(1 \le i \le n)$.

Все числа целые, неотрицательные и не превосходят $1\,000$.

Формат выходных данных

В первой строке выведите одно число – максимальное число комаров, которые может съесть лягушка.

Во второй строке выведите последовательность чисел – номера тех кувшинок, на которых должна побывать лягушка, в возрастающем порядке.

Если решений несколько, выведите любое.

Если лягушка не может добраться до последней кувшинки, то выведите одно число -1.

входной файл	*выходной файл*
6 1 100 3 4 1000 0	5 1 4 6
2 8 9	-1

Задача 0.2. Единицы (часть 1).

Дано число N. Необходимо определить, сколько есть бинарных строк длины N, в которых ровно K единиц.

Формат входных данных

В строке входного файла записаны два целых неотрицательных числа N и K $(0 \le K \le N \le 1\,000)$.

Формат выходных данных

Выведите одно число – ответ на задачу. Так как ответможет быть очень большим, необходимо его вывести по модулю $10^9 + 7$.

входной файл	*выходной файл*
3 2	3
4 0	1
5 4	5
6 4	15
7 2	21
8 0	1

Задача 0.3. Единицы (часть 2).

Дано число N.

Необходимо определить, сколько есть бинарных строк длины N, в которых ровно K единиц.

Формат входных данных

В строке входного файла записаны два целых неотрицательных числа N и K $\left(0 \le K \le N \le 10^6\right)$.

Формат выходных данных

Выведите одно число – ответ на задачу.

Так как ответможет быть очень большим, необходимо его вывести по модулю $10^9 + 7$.

входной файл	*выходной файл*
3 2	3
4 0	1
5 4	5
6 4	15
7 2	21
8 0	1

Задача 0.4. Порядок перемножения матриц.

Дана последовательность из s матриц $A_1, A_2, \ldots, A_s$. Требуется определить, в каком порядке их следует перемножать, чтобы число атомарных операций умножения было минимальным. Матрицы предполагаются совместимыми по отношению к матричному умножению (т. е. число столбцов матрицы A_{i-1} совпадает с числом строк матрицы A_i).

Будем считать, что произведение матриц – операция, которая принимает на вход две матрицы размера $k \times m$ и $m \times n$ и возвращает матрицу размера $k \times n$, затратив на это $k \cdot m \cdot n$ атомарных операций умножения. Базовый тип позволяет хранить любой элемент итоговой и любой возможной промежуточной матрицы, поэтому умножение двух элементов требует одной атомарной операции.

Так как перемножение матриц ассоциативно, итоговая матрица не зависит от порядка выполнения операций умножения. Другими словами, нет разницы, в каком порядке расставляются скобки между множителями, результат будет один и тот же.

Формат входных данных

В первой строке входного задано число матриц s $(2 \le s \le 100)$.

В последующих s строках заданы размеры матриц: $i+1$ содержит через пробел число n_i строк и число m_i столбцов матрицы A_i $(1 \le n_i, m_i \le 100)$. Гарантируется, что m_i совпадает с n_{i+1} для всех индексов от 1 до $s-1$.

Формат выходных данных

Выведите минимальное число атомарных операций умножения, необходимое для перемножения s матриц.

входной файл	*выходной файл*
3 2 3 3 5 5 10	130
4 20 5 5 35 35 4 4 25	3100

Замечание.

В первом примере можно умножать двумя способами:

$\left(A_1 \cdot \left(A_2 \cdot A_3\right)\right)$: требуется $3 \times 5 \times 10 + 2 \times 3 \times 10 = 150 + 60 = 210$ операций;

$\left(\left(A_1 \cdot A_2\right) \cdot A_3\right)$: требуется $2 \times 3 \times 5 + 2 \times 5 \times 10 = 30 + 100 = 130$ операций.

Второй способ эффективнее.

Задача 0.5. Наибольшая общая подпоследовательность.

Даны две последовательности A и B, каждая имеет длину n. Найти наибольшее k, для которого существуют две последовательности индексов $0 \le i_1 < i_2 < \ldots < i_k < n$ и $0 \le j_1 < j_2 < \ldots < j_k < n$ такие, что $A_{i_1} = B_{j_1}, A_{i_2} = B_{j_2}, \ldots, A_{i_k} = B_{j_k}$.

Также нужно найти и сами последовательности индексов.

Формат входных данных

В первой строке записано число n $(1 \le n \le 1000)$, длина последовательностей A и B. Во второй строке содержится n целых чисел a_i $(1 \le i \le 1000)$ – элементы последовательности A. В третьей строке содержится n целых чисел b_j $(1 \le j \le 1000)$ – элементы последовательности B.

Формат выходных данных

В первой строке выведите число k. Во второй строке выведите индексы $i_1, i_2, \ldots, i_k$. В третьей строке выведите индексы $j_1, j_2, \ldots, j_k$. Если подходящий последовательностей индексов несколько, выведите любые из них.

входной файл	*выходной файл*
2 1 2 1 2	2 0 1 0 1
5 1 2 3 4 5 1 3 2 4 4	3 0 1 3 0 2 4
6 1 2 3 3 4 6 1 6 3 3 2 4	4 0 2 3 4 0 2 3 5

Задача 0.6. Палиндром.

Вводится непустая строка S, которая имеет длину не более 7 000 символов и состоит только из строчных латинских букв.

Необходимо удалить из строки минимальное число символов так, чтобы получился палиндром (строка символов, которая читается слева направо и справа налево одинаково).

Формат входных данных

В первой строке входного файла записана исходная строка S.

Формат выходных данных

Выведите в первой строке длину получившегося палиндрома, а во второй строке сам палиндром (если палиндромов несколько, то выведите только один из них).

входной файл	*выходной файл*
asddfsa	6 asddsa

Задача 0.7. Строго возрастающая без разрывов последовательность.

Необходимо из заданной числовой последовательности A, состоящей из n элементов, вычеркнуть минимальное число элементов так, чтобы оставшиеся элементы образовали строго возрастающую подпоследовательность элементов.

Построенный алгоритм должен иметь трудоемкость $O(n \cdot \log n)$.

Формат входных данных

Первая строка входного файла содержит число n $(1 \le n \le 700\,000)$.

Следующая строка содержит n элементов последовательности A, которые разделены пробелами (элементы последовательности – целые числа, не превосходящие по модую $1\,000\,000\,000$).

Формат выходных данных

Выведите одно число – длину строго возрастающей подпоследовательности элементов.

входной файл	*выходной файл*
6 1 2 3 4 7 6	5

Задача 0.8. Преобразование строк.

На вход подаются две символьные последовательности A и B, каждая последовательность непустая, состоит из маленьких латинских букв и имеет длину не более 1 000 символов.

Необходимо преобразовать последовательность A в последовательность B с минимальным суммарным штрафом, который определяется следующим образом:

- удаление символа из строки A равно x баллов;
- вставка символа в строку A равна y баллов;
- замена символа в строке A на любой другой символ равна z баллов.

Формат входных данных

В первых трех строках входного файла находятся числа x, y, z соответственно. Все числа целые положительные и не превосходят 1 000 000.

В следующих двух строках находятся символьные последовательности A и B (тип элементов последовательности `string`).

Формат выходных данных

Выведите минимальный суммарный штраф.

входной файл	*выходной файл*
2 3 1 abcd bce	3

2.2. Индивидуальные задачи по теме 1.3

Задача 1. Максимальная стоимость товара

Покупатель имеет n купюр достоинством $a_1, a_2, \ldots, a_n$, а продавец – m купюр достоинством $b_1, b_2, \ldots, b_m$.

Необходимо найти максимальную стоимость p товара, которую покупатель не сможет оплатить, потому что нет возможности точно рассчитаться за этот товар с продавцом, хотя денег на покупку у него достаточно.

Формат входных данных

Первая строка входного файла содержит целые числа n и m ($1 \le n, m \le 100$).

Следующие две строки содержат разделенные пробелом номиналы купюр покупателя и продавца соответственно (все числа натуральные, не превосходящие 100).

Формат выходных данных

Если покупатель может купить товар любой стоимости, то выведите единственную строку `YES`.

В противном случае в первой строке выведите `NO`, а во второй – единственное число p.

входной файл	*выходной файл*

`1 3`
`25`
`1 2 4`
	`NO`
	`17`

Задача 2. Максимальная стоимость товара

Покупатель имеет n купюр достоинством $a_1, a_2, \ldots, a_n$, а продавец – m купюр достоинством $b_1, b_2, \ldots, b_m$.

Может ли покупатель приобрести вещь стоимости s так, чтобы у продавца нашлась точная сдача (если она необходима)?

Формат входных данных

Первая строка входного файла содержит целые числа n, m и s, разделенных пробелом$\left(0 \le n \le 100,\ 0 \le m \le 100, 1 \le s \le 2\,000\,000\right)$).

Следующие две строки содержат разделенные пробелом номиналы купюр покупателя и продавца соответственно (все числа натуральные, не превосходящие 100).

Формат выходных данных

Единственная строка должна содержать ответ `Yes` или `No`.

входной файл	*выходной файл*

`3 3 36`
`12 22 23`
`12 21 1`
	`Yes`

Задача 3. Проход из 1-й строки в *n*-ю.

Задана матрица A натуральных чисел из n строк и m столбцов. За каждый проход через клетку с индексами (i, j) взимается штраф $A(i, j)$. Необходимо с минимальным штрафом пройти из какой-либо клетки 1-й строки в n-ю строку, при этом из текущей клетки (i, j) можно перейти в любую из трех соседних, стоящих в строке с номером, на единицу большим: $(i+1, j-1)$, $(i+1, j)$, $(i+1, j+1)$ (если они существуют).

Формат входных данных

Первая строка входного файла содержит целые числа n и m $(2 \le n \le 200, 1 \le m \le 1\,000)$.

Следующие n строк содержат элементы матрицы A штрафов по строкам, в строке числа разделяются пробелами (все числа натуральные и не превосходят 1 000 000).

Формат выходных данных

Выведите одно число — минимальный штраф.

входной файл	*выходной файл*
2 2 1 1 2 3	3

Задача 4. Триангуляция n-угольника.

Выпуклый n-угольник задается координатами своих вершин в порядке обхода по контуру. Необходимо разбить n-угольник на треугольники $n-3$ диагоналями, не пересекающимися кроме как по концам, чтобы сумма их длин была минимальной.

Формат входных данных

Первая строка входного файла содержит целое числ n $(3 \le n \le 100)$.

Следующие n строк содержат координаты n-угольника в порядке обхода по контуру; в строке координаты разделены пробелом (все числа целые и по модулю не превосходят 1 000).

Формат выходных данных

Выведите одно действительное число – минимальную суммарную длину диагоналей разбиения.

Абсолютная погрешность вашего ответа не должна превышать 0,01.

входной файл	*выходной файл*
5 0 0 0 10 5 15 10 10 10 0	24.14

Задача 5. Триангуляция *n* угольника с максимальной диагональю.

Выпуклый *n*-угольник задается координатами своих вершин в порядке обхода по контуру.

Необходимо разбить *n*-угольник на треугольники $n-3$ диагоналями, не пересекающимися, кроме как по концам, таким образом, чтобы максимальная из диагоналей имела наименьшую длину.

Формат входных данных

Первая строка входного файла содержит целое числ n $(3 \le n \le 100)$.

Следующие n строк содержат координаты 222*n*-угольника в порядке обхода по контуру; в строке координаты разделены пробелом (все числа целые и по модулю не превосходят 1 000).

Формат выходных данных

Выведите длину самой длинной диагонали с абсолютной погрешность не более 0,01.

входной файл	*выходной файл*
5 0 0 0 10 5 15 10 10 10 0	14.14

Задача 6. Ящер-Отшельник, дом и камень.

Ящер-Отшельник хочет поселиться на острове и выделить на нем территорию для своего дома. Остров представляет собой прямоугольник $n \times m$ клеток. Территория для дома также должна быть прямоугольником (возможно, даже пустым).

Но не все так просто. Ящер-Отшельник знает, что в каждый квадрат острова придет по одному герою.

Если герой придет в клетку (i, j) и она будет находиться внутри территории дома Ящера-Отшельника, он потребует у Ящера-Отшельника $A_{i,j}$ камней.

При этом, если $A_{i,j}$ отрицательно, он подарит ему $|A_{i,j}|$ камней.

Ящер хочет выбрать подпрямоугольник под свой дом так, чтобы сумма $A_{i,j}$ была максимальна – ему очень нравится отказывать героям в просьбе дать камень.

Помогите ему, или он не даст вам не только камень, но и рычаг.

Формат входных данных

На вход в первой строке подаются два целых положительных числа $N \le 600$ и $M \le 600$.

Далее в N строках записано по M целых чисел $|A_i| < 10^9$.

Формат выходных данных

Выведите одно число – ответ на задачу.

входной файл	*выходной файл*
2 3 5 0 9 1 2 7	24
4 5 -7 8 -1 0 -2 2 -9 2 4 -6 -7 0 6 8 1 4 -8 -1 0 -6	20

Задача 7. Строго возрастающая почти всюду последовательность.

Необходимо из заданной числовой последовательности A, состоящей из n элементов, вычеркнуть минимальное число элементов так, чтобы оставшиеся элементы образовали строго возрастающую почти всюду подпоследовательность, то есть содержащую не более одного разрыва – пары (a_i, a_{i+1}) подряд идущих элементов, второй из которых не больше первого.

Построенный алгоритм должен иметь трудоемкость

$$\mathrm{O}(n \cdot \log n).$$

Формат входных данных

Первая строка входного файла содержит число n $(1 \le n \le 300\,000)$.

Следующая строка содержит n элементов последовательности A, которые разделены пробелами (элементы последовательности – целые положительные числа, не превосходящие $1\,000\,000\,000$).

Формат выходных данных

Выведите одно число – длину строго возрастающей подпоследовательности элементов.

входной файл	выходной файл
9 1 2 12 7 3 8 14 13 9	6
13 2 12 3 7 11 8 16 23 17 5 12 66 100	10
1 1000000000	1

Задача 8. Строго возрастающая с m разрывами последовательность.

Необходимо из заданной числовой последовательности A, состоящей из n элементов, вычеркнуть минимальное число элементов так, чтобы в оставшейся последовательности каждый последующий элемент был строго больше предыдущего, кроме не более чем m пар соседних элементов (разрывов).

Формат входных данных

Первая строка входного файла содержит числа n и m $(0 \le m < n \le 30\,000, m \le 100)$.

Следующая строка содержит n элементов последовательности A, которые разделены пробелами (элементы последовательности – целые числа, по модулю не превосходящие $1\,000\,000\,000$).

Формат выходных данных

Выведите одно число – длину наибольшей последовательности элементов.

входной файл	*выходной файл*
9 2 1 2 12 7 3 8 14 13 9	7
9 3 1 2 12 7 3 8 14 13 9	8

Задача 9. Выбрать поднабор.

По заданному набору A из n целых чисел необходимо построить наибольший поднабор, включающийся в A, элементы которого можно расположить в порядке, при котором каждый последующий элемент делится нацело на предыдущий.

Формат входных данных

Первая строка входного файла содержит число n $(0 \le n \le 11\,000)$. Следующая строка содержит n элементов набора A, которые разделены пробелом (все числа целые, по модулю не превосходящие $1\,000\,000\,000$).

Формат выходных данных

Выведите наибольшую длину последовательности, состоящей из элементов набора A, в которой каждый последующий элемент делится нацело на предыдущий элемент.

входной файл	*выходной файл*
8 1 2 4 8 16 32 64 100	7

Задача 10. Параллелепипеды.

В n-мерном пространстве m параллелепипедов заданы своими размерами $\left(a_{i,1}, a_{i,2}, \ldots, a_{i,n}\right)$, где $i = 1, \ldots, m$. Параллелепипед может располагаться в пространстве любым из способов, при котором его ребра параллельны осям координат.

Необходимо построить максимальную по длине последовательность вкладываемых друг в друга параллелепипедов. Будем считать, что параллелепипед с размерами $\left(a_{i,1}, a_{i,2}, \ldots, a_{i,n}\right)$ помещается в параллелепипед с размерами $\left(a_{j,1}, a_{j,2}, \ldots, a_{j,n}\right)$, если $a_{i,k} \le a_{j,k}$ для всех $k = 1, \ldots, n$.

Формат входных данных

Первая строка входного файла содержит целые числа n и m $\left(2 \le n \le 100,\ 1 \le m \le 1\,000\right)$.

Следующие m строк содержат размеры параллелепипедов, все числа не превосходят $1\,000\,000$).

Формат выходных данных

Выведите максимальное число вкладываемых друг в друга параллелепипедов.

входной файл	*выходной файл*
2 5 1 1 50 50 10 10 100 10 10 100	4

Задача 11. Разложение числа.

Заданы три числа a, b и c. Необходимо определить, можно ли представить число a в виде $a = x_1 \cdot x_2 \cdot \ldots \cdot x_k$, где $k \ge 1, a \le x_i \le b$ и x_i, a, b, c – целые числа. Кроме того, требуется минимизировать число множителей.

Также следует предусмотреть вариант, когда такого представления не существует.

Формат входных данных

Единственная строка входного файла содержит числа a, b, c $\left(2 \le a \le 1\,000\,000\,000, 1 \le b \le c \le 1\,000\,000\right)$, которые разделены пробелом.

Формат выходных данных

Если разложения не существует, выведите единственное число -1, в противном случае – наименьшее число множителей в разложении.

входной файл	*выходной файл*
200 8 100	2
100 2 3	–1

Задача 12. Бинарные последовательности.

Пусть x и y – две бинарные последовательности длины n и m соответственно, состоящие из нулей и единиц. Сами последовательности x и y можно рассматривать как запись в двоичной форме некоторых двух положительных целых чисел.

Необходимо найти максимальное число z, двоичную запись которого можно получить как из x, так и из y вычеркиванием цифр. Ответ выдайте в виде бинарной последовательности.

Формат входных данных

Первая строка входного файла содержит числа n и m $(1 \le n, m \le 3\,000)$. Следующие две строки содержат элементы последовательностей x и y соответственно. Гарантируется, что у заданных строк есть хотя бы один общий символ.

Формат выходных данных

В первой строке выведите число цифр максимального числа z. В следующей строке выведите саму двоичную запись числа z. Число z не должно содержать незначащих ведущих нулей.

входной файл	*выходной файл*
11 9 11000100100 001011001	6 111001

Задача 13. Пробирки.

Заданы три пробирки по 100 литров каждая. Две пробирки могут иметь деления, причем деления у первой и второй пробирки совпадают. Третья пробирка делений не имеет. Деления – это отметки, i-я из которых обозначает число d_i литров. Имеется 100 литров воды, которая разлита в три пробирки. Известно, какой объем воды находится в каждой из двух пробирок с делениями в начальный момент времени, оставшаяся жидкость находится в третьей пробирке. Необходимо определить, можно ли отмерить в третью пробирку ровно x литров воды. Если можно, то за какое минимальное число переливаний.

Разрешается выливание (в другую пробирку) из пробирки до меток или доливание (из другой пробирки) в пробирку до меток. Разрешается также выливать (в другую пробирку) всю воду из пробирки.

Формат входных данных

Первая строка входного файла содержит целое число x литров воды, которое надо отмерить в третью пробирку $(0 \le x \le 100)$.

Вторая и третья строки содержат целые числа k и l литров воды в первой и второй пробирках соответственно в начальный момент времени $(0 \le k + l \le 100)$.

Четвертая строка содержит деления пробирок – целые числа d_i $(1 \le d_i \le 99)$, записанные через пробел в порядке возрастания.

Строка завершается нулем.

Формат выходных данных

Выведите минимальное число переливаний.

Если отмерить x литров воды в третью пробирку нельзя, то выдайте сообщение `No solution`.

входной файл	*выходной файл*
24 100 0 6 0	8
37 100 0 2 96 0	No solution

Задача 14. Игра.

Задается натуральное число n. Двое играющих называют по очереди числа, меньшие 10^7, по следующим правилам.

Начиная с числа n, каждое новое число должно увеличивать одну из цифр предыдущего числа (возможно, незначащий нуль) на 1, 2 или 3 (после увеличения цифра не может превосходить 9).

Проигравшим считается тот, кто называет число 9 999 999.

Для заданного n необходимо определить, может ли выиграть игрок, делающий первый ход, при оптимальной игре противника.

Вывести сообщение `The first wins` или `The second wins`.

В случае возможности выигрыша первым игроком требуется вывести все его возможные выигрышные первые ходы.

Формат входных данных

Первая строка входного файла содержит единственно число n $\left(1 \le n \le 9\,999\,998\right)$.

Формат выходных данных

Выведите сообщение `The first wins` или `The second wins`.

В случае возможности выигрыша первым игроком во второй строке через пробел выводятся все его выигрышные первые ходы в порядке возрастания.

входной файл	*выходной файл*
16	The first wins 19 36 216 2016 20016 200016 2000016
40	The secod wins

Задача 15. Полигон.

Существует игра для одного игрока, которая начинается с задания Полигона с n вершинами. Пример графического представления Полигона показан на рисунке 2.1, где $n = 4$.

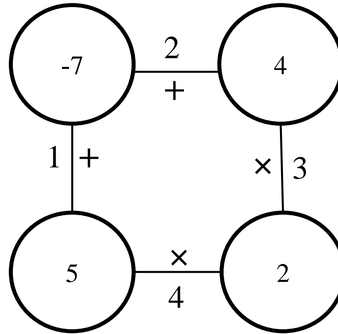


Рисунок 2.1 – Графическое представление Полигона.

Для каждой вершины Полигона задается значение – целое число, а для каждого ребра – метка операции + (сложение) либо × (умножение).

Ребра Полигона пронумерованы целыми числами от 1 до n.

Первым ходом в игре удаляется одно из ребер.

Каждый последующий ход состоит из следующих шагов:

– выбирается ребро e и две вершины v_1 и v_2, которые соединены ребром e;

– ребро e стягивается, то есть вершины v_1 и v_2 удаляются и заменяются новой вершиной v со значением, равным результату выполнения операции, определенной меткой ребра e, над значениями вершин v_1 и v_2. В ребрах, отличных от e и инцидентных вершинам v_1 и v_2, эти две вершины заменяются на вершину v.

Игра заканчивается, когда больше нет ни одного ребра. Результат игры – это число, равное значению оставшейся вершины.

Рассмотрим Полигон на рисунке 2.1. Игрок начал игру с удаления ребра 3 (рисунок 2.2).

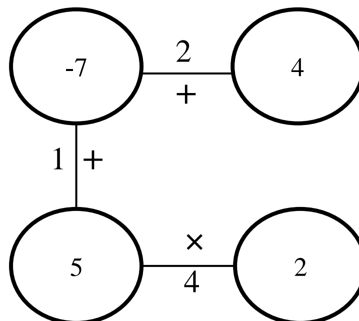


Рисунок 2.2 – Удаление ребра 3.

После этого, игрок выбирает ребро 1 (рисунок 2.3).

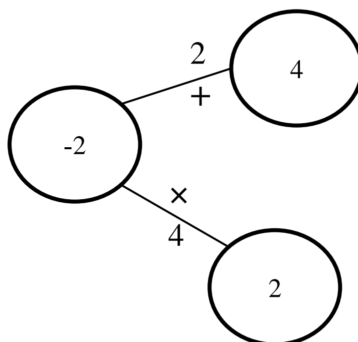


Рисунок 2.3– Результат выбора ребра 1.

Затем, игрок выбирает ребро 4 (рисунок 2.4).

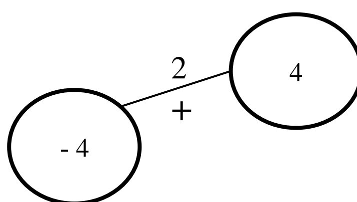


Рисунок 2.4 – Результат выбора ребра 4.

Наконец, игрок выбирает ребро 2 (рисунок 2.5).
Результатом будет число 0.

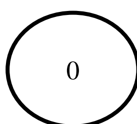


Рисунок 2.5– Результат выбора ребра 2.

Напишите программу, которая по заданному Полигону, вычисляет максимальное значение оставшейся вершины и выводит список всех тех ребер, удаление которых на первом ходу игры позволяет получить это значение.

Формат входных данных

В первой строке входного файла записано число n $(3 \le n \le 50)$.

Вторая строка содержит метки ребер с номерами $1, 2, \dots, n$, между которыми записаны через пробел значения вершин (первое из них соответствует вершине, инцидентной ребрам 1 и 2, следующее – инцидентной ребрам 2 и 3, и так далее, последнее – инцидентной ребрам n и 1).

Метка ребра – это буква t, соответствующая операции +, или буква x, соответствующая операции ×.

Гарантируется, что при любой последовательности ходов значения вершин находятся в пределах от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

Формат выходных данных

В первой строке выведите максимальное значение оставшейся вершины, которое может быть достигнуто для заданного Полигона.

Во второй строке – список всех тех ребер, удаление которых на первом ходу, позволяет получить это значение.

Номера ребер должны быть записаны в возрастающем порядке и отделяться друг от друга одним пробелом.

входной файл	*выходной файл*
4 t -7 t 4 x 2 x 5	33 1 2

Задача 16. Эпидемия.

В связи с эпидемией гриппа в больницу направляется *a* больных гриппом *A* и *b* больных гриппом *B*.

Больных гриппом *A* нельзя помещать в одну палату с больными гриппом *B*. Имеется информация об общем числе *p* палат в больнице, пронумерованных от 1 до *p*, и о распределении уже имеющихся там больных.

Необходимо определить максимальное число *m* больных, которых больница в состоянии принять. При размещении новых больных не разрешается переселять уже имеющихся больных из палаты в палату.

Формат входных данных

В первой строке входного файла находится целое число a $(0 \le a \le 20\,000)$.

Во второй строке – целое число b $(0 \le b \le 20\,000)$.

В третьей строке – целое число p $(1 \le p \le 1\,000)$.

В каждой из последующих *p* строк находится по 3 числа n_i, a_i и b_i, разделенных пробелом, где n_i – вместимость палаты, a_i – число уже имеющихся в палате больных гриппом *A*, b_i – число уже имеющихся в палате больных гриппом *B*.

Информация о вместимости палат вводится последовательно для палат с номерами $1, 2, \ldots, p$.

Числа n_i, a_i и b_i – целые неотрицательные, не превосходящие 20 000.

Формат выходных данных

В первой строке выведите число *m*.

Если все поступившие больные размещены, то во второй строке выведите в порядке возрастания номера палат, разделенные пробелом, куда помещаются больные гриппом *A*.

входной файл	*выходной файл*
10 7 3 5 2 0 4 0 1 8 0 0	13
21 52 8 1 0 0 4 0 0 7 0 0 11 0 0 13 0 0 15 0 0 22 0 0 0 0 0	73 1 3 5

Задача 17. Торговые скидки.

В магазине каждый товар имеет цену. Например, цена одного цветка равна $2, а цена одной вазы равна $5. Чтобы привлечь покупателей, магазин ввел скидки. Скидка заключается в том, чтобы продавать набор одинаковых или разных товаров по пониженной цене. Примеры: три цветка за $5 вместо $6, или две вазы вместе с одним цветком за $10 вместо $12.

Необходимо вычислить наименьшую цену, которую покупатель должен заплатить за заданные заранее выбранные покупки. Любой товар или набор товаров по скидке в магазине можно приобрести несколько раз.

Однако покупателю нельзя приобретать лишние товары: набор товаров, который требуется купить, нельзя дополнять ничем, даже если бы это снизило общую стоимость набора.

Для описанных выше цен и скидок наименьшая цена за три цветка и две вазы равна $14: две вазы и один цветок продаются по сниженной цене за $10 и два цветка – по обычной цене за $4.

Формат входных данных

Вначале файла описываются покупки («корзина с покупками»).

Первая строка содержит число b $\left(0 \le b \le 5\right)$ различных видов товара в корзине.

Каждая из следующих b строк содержит три целых числа x, k и p, где x – уникальный код товара $\left(1 \le x \le 999\right)$; k – число единиц товара с кодом x, которое должно находится в корзине$\left(1 \le k \le 5\right)$; p – стоимость единицы товара с кодом x $\left(1 \le p \le 9\,999\right)$. Обратите внимание, что общее число товаров в корзине может быть не более $5 \times 5 = 25$ единиц. Затем описываются скидки, которые действуют в магазине. В отдельной строке записано число s скидок $\left(0 \le s \le 99\right)$.

Каждая из следующих s строк описывает одну скидку, определяя набор товаров и общую стоимость набора. Первое число n в такой строке определяет число различных видов товара в наборе $(1 \le n \le b)$. Следующие n пар (x, k) чисел указывают, что k единиц товара с кодом x включены в набор для скидки $(1 \le k \le 5,\ 1 \le x \le 999)$. Гарантируется, что уникальный код x товара ранее встречался в описании товаров, которые нужно купить. Последнее число p в строке определяет сниженную стоимость набора $(1 \le p \le 9\,999)$. Стоимость набора меньше суммарной стоимости отдельных единиц товаров в наборе.

Формат выходных данных

Выведите одно число – наименьшую стоимость, за которую можно приобрести требуемое число товаров.

входной файл	*выходной файл*
2 7 3 2 8 2 5 2 1 7 3 5 2 7 1 8 2 10	14

Задача 18. Больше не запишешь.

В файловой системе персонального компьютера файлы организованы в каталоги. В компьютере полное имя файла является строкой, состоящей из имен каталогов и имени файла, разделенных символом \, причем этот символ не может идти два раза подряд, также быть первым или последним. Имя файла (каталога) может быть произвольной длины, но полное имя файла не может содержать более n символов.

В качестве символов, допустимых к употреблению в именах файлов (каталогов), могут использоваться символы из алфавита, состоящего из k букв (символ \ не входит в их число).

Необходимо для данных k и n определить максимальное число l файлов, которое можно записать на данный компьютер.

Формат входных данных

В единственной строке входного файла содержатся числа k и n $(1 \le k \le 49,\quad 1 \le n \le 44)$, разделенные пробелом.

Формат выходных данных

Выведите максимальное число l файлов, гарантируется, что $l \le 2\,147\,483\,647$.

входной файл	*выходной файл*
3 5	696

Задача 19. Концерт.

Во время трансляции концерта предприниматель решил сделать бизнес на производстве кассет. Он имеет m кассет с длительностью d звучания каждая и хочет записать на них максимальное число песен. Эти песни (их общее число n) передаются в порядке $1, 2, \ldots, n$ и имеют заранее известные ему длительности звучания $l_1, l_2, \ldots, l_n$. Предприниматель, прослушивая песни по порядку, может выполнять одно из следующих действий: если песня на текущую кассету помещается, то он может записать ее на кассету или пропустить песню; если песня на кассету не помещается, то он может пропустить песню или начать ее записывать на новую кассету (при этом старая кассета откладывается и туда уже ничего не может быть записано).

Необходимо определить максимальное число песен, которые предприниматель может записать на кассеты.

Формат входных данных

В первой строке входного файла находятся числа n, m и d $\left(1 \le n \le 200,\quad 1 \le m \le 50,\quad 1 \le d \le 1\,000\right)$. Во второй строке находятся целые числа $l_1, l_2, \ldots, l_n$, разделенные пробелом $\left(1 \le l_i \le 1\,000\right)$.

Формат выходных данных

Выведите максимальное число песен, которые предприниматель может записать на кассеты.

входной файл	*выходной файл*
3 2 4 4 1	2

Задача 20. Палиндромы.

Дана строка, состоящая из строчных букв латинского алфавита, без пробелов. Палиндромом называется текст, одинаково читаемый слева-направо и справа-налево. Необходимо найти палиндром максимальной длины, который можно получить из исходной строки вычеркиванием символов (менять порядок символов нельзя). Так как ответов может быть несколько, то найти минимальный и максимальный лексикографически (как в словаре).

Формат входных данных

В первой строке входного файла записана исходная строка из строчных букв латинского алфавита, без пробелов длиной от 1 до 10 000 символов.

Формат выходных данных

В первой строке вывести лексикографически минимальный палиндром максимальной длины, во второй – максимальный. Если они совпадают, вывесли оба, каждый в своей строке.

входной файл	*выходной файл*
arbat	aba ara

Задача 21. Юбилей.

В связи с открытием олимпиады-1998 по информатике в Могилеве n участников решили устроить вечеринку.

Для проведения вечеринки достаточно купить хотя бы m_F бутылок фанты, m_B бананов и m_C тортов.

Товары можно закупать в виде наборов, каждый из которых описывается количествами бутылок фанты, бананов и тортов и стоимостью.

Всего доступно m вариантов наборов, разрешается покупать любые наборы сколько угодно раз, но делить наборы нельзя.

Определите минимальный взнос участника вечеринки, если все участники вносят равную долю в сумму.

Гарантируется, что из предлагаемых наборов можно сформировать нужный заказ.

Формат входных данных

В первой строке входного файла находятся два целых числа n и m – количество людей и количество доступных наборов $\left(1 \le n \le 10, 0 \le m \le 100\,000\right)$.

В каждой из следующих m строк записаны четыре целых числа f, b, c, s – количество бутылок фанты, штук бананов и тортов в наборе и стоимость набора $\left(0 \le f, b, c \le 1\,000\right)$.

В последней строке находятся три целых числа m_F, m_B и m_C $\left(0 \le m_F, m_B, m_F \le 9\right)$.

Формат выходных данных

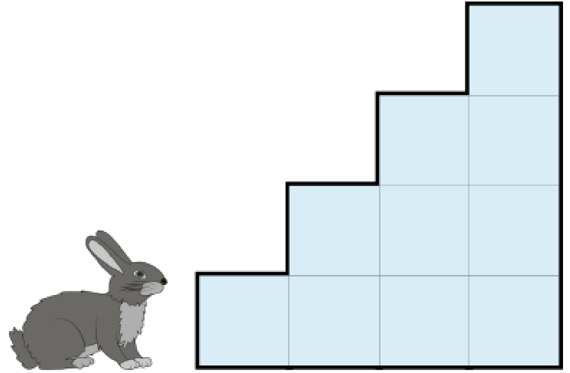
Выведите минимальный взнос участника вечеринки для оплаты заказа.

Ответ округлите вверх до ближайшего целого числа.

входной файл	*выходной файл*
6 8 4 4 4 4 1 1 1 1 1 2 1 4 2 2 3 8 3 2 3 9 2 2 1 2 1 1 2 2 2 2 2 6 6 6 6	1

Задача 22. Заяц.

Заяц любит прыгать по лестнице вверх. Лестница имеет n ступенек, изначально заяц находится рядом с лестницей. За один прыжок заяц может прыгнуть на следующую ступеньку или перепрыгнуть через одну ступеньку. Определите, сколькими способами заяц может добраться до вершины лестницы. Способы считаются различными, если множества ступенек, на которых побывал заяц, отличаются.



Формат входных данных

В первой строке входного файла задано число n ступенек $\left(1 \leq n \leq 10\,000\,000\right)$.

Формат выходных данных

Выведите число различных способов добраться от основания к вершине лестницы по модулю 1 000 000 007.

входной файл	*выходной файл*
1	1
2	2
4	5

Задача 23. Канадские авиалинии.

Вы победили в соревновании, организованном Канадскими авиалиниями. Приз – бесплатное путешествие по Канаде.

Путешествие начинается с самого западного города, в который летают самолеты, проходит с запада на восток, пока не достигнет самого восточного города, в который летают самолеты. Затем путешествие продолжается обратно с востока на запад, пока не достигнет начального города.

Ни один из городов нельзя посещать более одного раза, за исключением начального города, который надо посетить ровно дважды (в начале и конце путешествия). Вам также нельзя пользоваться авиалиниями других компаний или другими способами передвижения.

Необходимо для данного списка городов и списка прямых рейсов между парами городов найти маршрут, включающий максимальное число городов и удовлетворяющий выше названным условиям.

Формат входных данных

В первой строке входного файла записано число n городов $(2 \le n \le 500)$ и число m прямых рейсов (положительное целое число).

В каждой из следующих n строк – название города, в который летают самолеты. Названия упорядочены с запада на восток, то есть i-й по порядку город находится восточнее j-го тогда и только тогда, когда $i > j$ (не существует городов на одном меридиане).

Название каждого города – строка, состоящая не более чем из 15 цифр и/или латинских букв, например: AGR34 или BELA.

В каждой из следующих m строк – названия двух различных городов из списка городов, разделенные пробелом.

Если пара (ГОРОД$_1$, ГОРОД$_2$) содержится в строке, то это означает, что есть прямой рейс из ГОРОД$_1$ в ГОРОД$_2$, а также прямой рейс из ГОРОД$_2$ в ГОРОД$_1$.

Формат выходных данных

Если маршрута не существует, то выведите строку `No solution`.

Если же маршрут существует, то выведите число различных городов, посещаемых в маршруте.

входной файл	*выходной файл*
2 1 C1 C2 C1 C2	2
5 3 g1 g2 g3 g4 g5 g1 g2 g2 g3 g3 g5	No solution

Задача 24. Блок из единиц.

В таблице размера $n \times m\,(1 \le n, m \le 1\,000)$, состоящей из нулей и единиц необходимо найти квадратный блок максимального размера, состоящий из одних единиц.

Формат входных данных

В первых двух строках входного файла находятся размеры матрицы n и m.

Третья строка пустая.

Следующие n строк содержат элементы матрицы – числа 0 или 1.

Каждой строке входных данных соответствует одна строка матрицы.

Числа в строке разделены пробелами.

Формат выходных данных

В первой строке выведите размер наибольшего блока.

В следующих двух строках выведите координаты (столбец и строку) левого верхнего угла наибольшего блока.

Столбцы и строки нумеруются с единицы.

Если блоков максимального размера несколько, то необходимо вывести координаты самого левого из имеющихся блоков.

В случае наличия одинаковых блоков с одинаковым номером самого левого столбца необходимо вывести координаты верхнего из этих блоков.

входной файл	*выходной файл*
10 6 1 0 0 1 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1	2 5 1

Задача 25. Подпоследовательности.

Дана последовательность a из n чисел и число m.

Вам нужно определить для всех пар (s, p) $0 \le s, p < m$, сколько подпоследовательностей последовательности a имеют сумму элементов, имеющую остаток s по модулю m, и произведение элементов, имеющее остаток p по модулю m. Достаточно посчитать все количества подпоследовательностей по модулю m.

Напомним, что подпоследовательностью называется любой набор элементов исходной последовательности, который может быть получен удалением некоторых элементов (при этом возможно, что ни один элемент не удален или удалены все).

Подпоследовательности называются разными, если хотя бы для одного индекса i исходной последовательности i-й элемент удален в одной подпоследовательности и не удален в другой.

Сумма и произведение чисел пустой подпоследовательности по определению равны 0 и 1, соответственно.

Формат входных данных

В первой строке входных данных записано два целых числа n и m $\left(1 \le n \le 10^5, 2 \le m \le 300, n \cdot m^2 \le 10^8\right)$ – длина последовательности a и модуль.

В следующей строке записано n целых чисел $a_1, a_2, \ldots, a_n \left(0 \le a_i < m\right)$ – элементы исходной последовательности.

Формат выходных данных

Выведите m строк, по m чисел в каждой строке. j-е число в i-ой строке (в 1 - индексации) должно быть равно остатку от деления на m количества различных подпоследовательностей, сумма и произведение которых имеют остатки $\left(i-1\right)$ и $\left(j-1\right)$ от деления на m, соответственно.

входной файл	*выходной файл*
4 3 0 0 0 0	0 1 0 0 0 0 0 0 0
9 10 9 9 8 2 4 4 3 5 3	4 1 5 0 0 2 9 0 0 0 7 0 0 2 2 1 2 0 8 0 6 0 1 0 0 0 0 4 0 0 4 0 0 2 0 1 0 0 4 0 4 0 1 0 9 2 0 0 4 1 4 2 6 0 0 1 8 0 0 0 2 0 0 0 1 2 2 0 4 1 1 0 6 0 0 4 0 0 0 0 4 1 1 0 0 2 9 0 4 0 4 0 0 0 8 1 0 0 6 2

Задача 26. Гвозди.

Имеется деревянная планка, параллельная оси OX, в которую вбито n гвоздей.

Известны координаты x_i этих гвоздей $1 \le i \le n$, причем $x_i < x_{i+1}$.

К гвоздям требуется привязать веревочки таким образом, чтобы:

– каждая веревочка связывала ровно два гвоздя;

– к каждому гвоздю была привязана хотя бы одна веревочка;

– суммарная длина веревочек была бы минимальна.

Формат входных данных

В первой строке записано целое число n $\left(2 \le n \le 100\right)$. В следующих строках (по одному числу в строке) указаны координаты гвоздей в порядке возрастания. Координаты являются вещественными числами от 0 до 1000 включительно и задаются не более чем с двумя знаками после десятичной точки.

Формат выходных данных

Выведите минимальную суммарную длину веревочек (с двумя знаками после десятичной точки).

входной файл	*выходной файл*
5 11 12 13 16 17	3.00
4 6.34 6.82 15.89 24.58	9.17

Задача 27. Снегоуборочная машина.

Главная улица города представляет собой прямоугольник с вершинами в точках $(0, 0)$, $(0, 2)$, $(n, 2)$, $(n, 0)$. Ночью в городе был сильный снегопад и теперь на некоторых единичных квадратиках улицы лежит снег. Снегоуборочная машина представляет собой отрезок длины 1, который первоначально расположен так, что его концы имеют координаты $(k, 0)$ и $(k, 1)$. Снегоуборочная машина может за 1 секунду переместиться в горизонтальном направлении на 1, при этом ее отрезок «заметает» клетку. Если на этой клетке был снег, то он исчезает. Кроме того, машина может мгновенно переместиться на 1 вверх или вниз (при этом отрезок перемещается по прямой, на которой он лежит).

Необходимо как можно быстрее убрать снег. Выясните, за какое время это можно сделать. Выводить последовательность действий машины, которая позволяет убрать снег за это время, не нужно.

Формат входных данных

Первая строка входного файла содержит n и k $\left(1 \le n \le 500\,000, 0 \le k \le n\right)$.

Следующие n строк содержат по два числа каждая, первое число $(i + 1)$-й строки равно 1, если на клетке $(i, 0)$ лежит снег, и 0 в противном случае. Второе число показывает, есть ли снег в клетке $(i, 1)$.

Формат выходных данных

В единственной строке выведите минимально возможное число секунд, которое потребуется, чтобы убрать снег.

входной файл	*выходной файл*
4 0 0 0 1 1 0 0 0 1	6

Задача 28. Сотовый телефон.

Современные сотовые телефоны поддерживают набор SMS, то есть коротких текстовых сообщений. К сожалению, клавиатура кнопочного сотового телефона имеет только 10 клавиш, а в русском алфавите 33 буквы. (Ауниверсальный алфавит, содержащий символы разных письменностей, имеет еще больший размер.) Поэтому на одной клавише телефона изображают сразу несколько букв, например, на цифре 1 – АБВ, на 2 – ГДЕЕ, и так далее. Тогда, чтобы набрать букву, которая написана на клавише первой, надо нажать на эту клавишу 1 раз, чтобы набрать вторую букву – 2 раза и т. д., чтобы набрать букву, написанную *k*-й по счету, надо нажать на клавишу *k* раз.

Ученые подсчитали, что человек набирает за время жизни букву *А* в среднем z_A раз, $Б - z_Б$ раз и так далее. Чтобы человек не путался, буквы должны быть расположены на клавишах в алфавитном порядке. Однако с целью уменьшения среднего общего числа нажатий на клавиши, требуется так расположить буквы на клавишах, чтобы минимизировать сумму

$$s = z_А \cdot c_А + z_Б \cdot c_Б + \ldots + z_Я \cdot c_Я,$$

где, например, за $c_У$ обозначен номер буквы *У* на клавише, на которой она находится. Так, если бы у нашего телефона было 2 клавиши, а в нашем алфавите было бы только 3 буквы *АБВ*, то возможно было бы 2 способа разместить буквы на клавиатуре:

$$1 - АБ,\ 2 - В,\ s = z_А + 2 \cdot z_Б + z_В;$$

$$1 - А,\ 2 - БВ,\ s = z_А + z_Б + 2 \cdot z_В.$$

В зависимости от значений величин $z_А$, $z_Б$ и $z_В$ один из этих двух вариантов предпочтительнее.

Найдите размещение *m* букв универсального алфавита на клавиатуре с *n* клавишами, минимизирующее значение *s*.

Формат входных данных

В первой строке входного файла записано число *n* клавиш на клавиатуре телефона $(1 \le n \le 100)$.Во второй строке – число *m* букв алфавита $(1 \le m \le 1000)$.

Каждая из следующих *m* строк содержит целые неотрицательные числа $z_А, z_Б, \ldots$ каждое из которых не превосходит 100 000.

Формат выходных данных

В единственной строке выведите число *s*.

входной файл	*выходной файл*
2 3 1 2 3	8

Задача 29. Диски.

Имеется n дисков одинаковой толщины с радиусами $r_1, r_1, \ldots, r_n$. Эти диски упаковываются в коробку таким образом, что каждый из них стоит ребром на дне коробки и все диски находятся в одной плоскости. Необходимо найти длину коробки, в которую все диски могут быть упакованы, при условии, что диски можно упаковывать только в порядке следования их радиусов на входе.

Формат входных данных

Первая строка содержит число n дисков $(n \le 100)$.

В следующих n строках следуют n чисел $r_1, r_1, \ldots, r_n$ $(0.001 \le r_i \le 1000)$ – радиусы дисков, в том порядке, в котором они должны быть упакованы. Радиусы даны не более, чем с тремя знаками после точки.

Формат выходных данных

Выведите единственное число – длину коробки, в которую все диски могут быть упакованы, при условии, что порядок упаковки строго задан на входе. Длина коробки выводится с точностью до 5 знаков после запятой (если длина коробки целое число, то выводится это целое число и 5 нулей после десятичной точки, например, если длина коробки равна 15, то в выходном файле – 15.00000).

входной файл	*выходной файл*
3 2.0 1.0 2.0	9.65685

Задача 30. Паркет.

Коридор имеет размер $3 \times n$. Для заданного n необходимо определить число различных способов уложить паркет в этом коридоре плитками размером 1×2 и 2×2 (сами варианты укладки паркета определять не надо).

Формат входных данных

Единственная строка содержит целое число n $(1 \le n \le 10\,000\,000)$.

Формат выходных данных

Выведите остаток от деления числа различных вариантов укладки паркета на 1 000 000 007.

входной файл	*выходной файл*
2	5

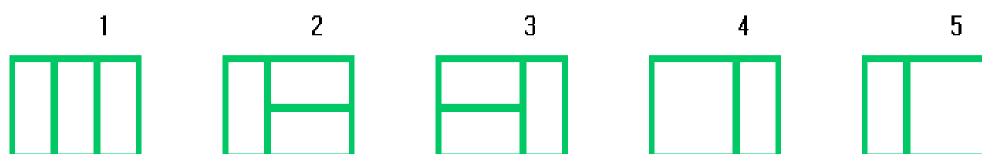


Рисунок 2.6– Укладки при $n = 2$.

Задача 31. Валюта.

Квадратная таблица A размера $n \times n$ заполнена неотрицательными действительными числами. Число $a_{i,j}$ определяет курс обмена валюты i на валюту j. Так, например, если $a_{i,j} = 2{,}5$, то это значит, что за 1 единицу валюты i дают 2,5 единицы валюты j. Если $a_{i,j} = 0$, то считаем, что курс обмена валюты i на валюту j прямо не установлен.

Необходимо определить, можно ли, имея некоторую сумму денег в одной из валют, получить бо́льшую сумму денег в той же валюте, совершив несколько обменов.

Формат входных данных

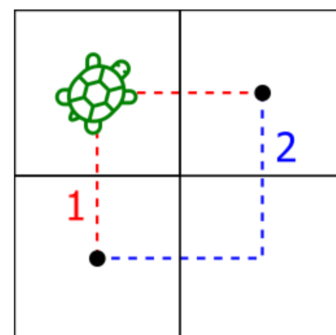
В первой строке входного файла записано число n $(3 \le n \le 100)$, а следующие n строк соответствуют строкам матрицы, в строке элементы разделены пробелами.

Формат выходных данных

Выведите сообщение `yes` или `no`.

входной файл	*выходной файл*
3 1 2 3 0.4 1 2 0.5 0.5 1	yes
3 1 1 1 1 1 1 1 1 1	no

Задача 32. Черепашка.



Черепашка гуляет по прямоугольному полю. Сейчас она находится в левом нижнем углу, но мечтает попасть в правый верхний угол. Черепашка очень любознательна, но слаба в точных науках, поэтому просит вас посчитать, сколько способов существует попасть в столь желанный ею угол, если она может перемещаться только вверх и вправо.

Формат входных данных

В первой строке входного файла записаны два числа: n – число строк поля, m – число столбцов поля $(1 \le n, m \le 10\,000)$.

Формат выходных данных

Выведите число способов добраться из квадрата с координатами $(1,1)$ до квадрата с координатами (n, m) по модулю 1 000 000 007.

входной файл	*выходной файл*
2 2	2

Задача 33. Пещера.

В Байтландии есть много пещер. Перед вами карта одной из них.

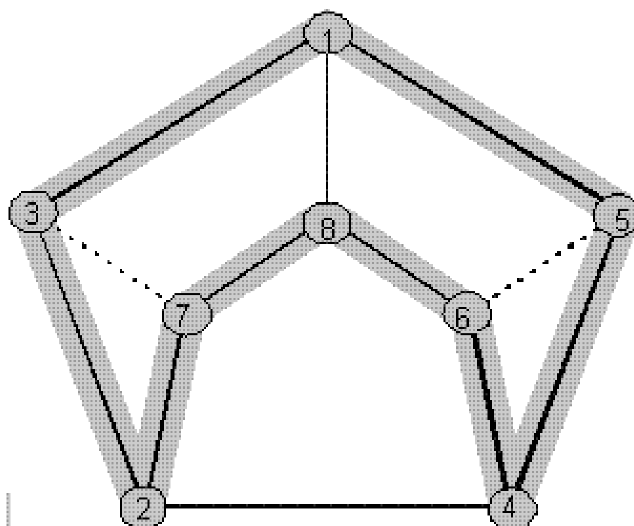


Рисунок 2.7 – Карта пещеры.

Все пещеры в Байтландии удовлетворяют следующим свойствам:

– все гроты и переходы расположены на одном уровне;

– переходы не пересекаются;

– некоторые из гротов расположены на внешнем кольце и называются внешними гротами;

– все остальные гроты расположены внутри внешнего кольца и называются внутренними гротами;

– имеется вход в пещеру, ведущий к одному из внешних гротов;

– существует ровно три прохода, ведущие от каждого грота к трем другим гротам;

– если грот является внешним, то два перехода ведут к двум соседним гротам внешнего кольца и точно один переход ведет к внутреннему гроту;

– переходы, связывающие внешние гроты, называются внешними переходами, а оставшиеся переходы называются внутренними;

– всегда можно пройти от одного грота к другому, используя только внутренние переходы; при этом существует только один такой маршрут, если разрешается использовать каждый внутренний переход не более одного раза;

– не все переходы равнозначны по сложности преодоления, они делятся на две группы: легкие и тяжелые.

Было решено открыть все пещеры для посещения. Чтобы обеспечить легкое и безопасное движение по пещере, посетители должны следовать по выбранному заранее маршруту и посетить каждый из гротов ровно один раз. Исключением является входной грот, где каждый тур начинается и заканчивается. Этот грот посещается дважды. Маршрут должен содержать как можно меньше опасных переходов.

Необходимо найти маршрут через пещеру, который начинается и заканчивается входным гротом, позволяет посетителям увидеть все другие гроты ровно один раз и содержит как можно меньше тяжелых переходов.

Формат входных данных

В первой строке записаны разделенные пробелом два целых числа: число n всех гротов и число k внешних гротов $(3 \le k < n \le 500)$.

Гроты нумеруются от 1 до n. Грот 1 является входным. Гроты $1, 2, \ldots, k$ являются внешними. Во внешнем кольце они не обязательно располагаются по порядку.

Следующие $3 \cdot n/2$ строк содержат описание переходов.

Каждое описание состоит из трех целых чисел a, b и c, разделенных пробелом.

Числа a и b – номера гротов на концах перехода.

Число c принимает значение 0 или 1, где 0 – легкий переход, а 1 – тяжелый переход.

Формат выходных данных

В единственной строке выведите набор целых чисел: первое число есть 1 (входной грот), а далее последовательно номера гротов в построенном маршруте.

Если маршрутов, оптимальных по числу тяжелых переходов, несколько, выведите любой из них.

входной файл	*выходной файл*
8 5 1 3 0 3 2 0 7 3 1 7 2 0 8 7 0 1 8 0 6 8 0 6 4 0 6 5 1 5 4 0 2 4 0 5 1 0	1 5 4 6 8 7 2 3

Замечание.

Приведенный пример изображен на рисунке 2.7.

Входной грот имеет номер 1.

Тяжелые переходы обозначены пунктиром.

Маршрут, проходящий по гротам в порядке

1, 5, 4, 6, 8, 7, 2, 3,

не содержит тяжелых переходов.

Последний грот (с номером 1) предполагается и не перечислен.

Задача 34. Доминошки.

Доминошка – это прямоугольная плитка, лицевая сторона которой разделена на два квадрата, каждый из которых содержит от 0 до 6 точек.

Ряд доминошек выложен на столе:

6	1	1	1
1	5	3	2

Число точек в верхней строке равно $6+1+1+1=9$, а в нижней строке равно $1+5+3+2=11$. Разница между нижней и верхней строкой составляет $|11-9|=2$. Разница – это абсолютная величина разности двух сумм. Каждая доминошка может быть повернута на 180°, меняя местами верхний и нижний квадраты.

Необходимо определить, какое минимальное число поворотов необходимо для минимизации разницы.

Вприведенном примере нужно повернуть последнюю доминошку для того, чтобы уменьшить разницу до нуля. В этом случае ответом будет 1.

Формат входных данных

В первой строке входного файла записано число n доминошекккк, лежащих на столе $(1 \le n \le 250\,000)$. Каждая из следующих n строк содержит по два целых числа a и b, разделенных пробелом $(0 \le a, b \le 6)$. Числа a и b, записанные в $(i+1)$-й строке определяют число точек на i-й доминошке в верхнем и нижнем квадратах соответственно.

Формат выходных данных

Выведите наименьшее число поворотов, необходимых для минимизации разности.

входной файл	*выходной файл*
4 6 1 1 5 1 3 1 2	1
10 6 0 5 1 6 3 6 4 3 2 2 1 0 6 1 6 0 4 0 4	2

Задача 35. Симпатичные узоры.

Компания планирует заняться выкладыванием во дворах у состоятельных клиентов узора из черных и белых плиток, каждая из которых имеет размер 1×1 метр. Известно, что дворы у всех состоятельных людей имеют наиболее модную на сегодня форму прямоугольника $m \times n$ метров. Однако при составлении финансового плана у директора этой организации появились целые две серьезные проблемы.

Во-первых, каждый новый клиент очевидно захочет, чтобы узор, выложенный у него во дворе, отличался от узоров всех остальных клиентов этой фирмы.

Во-вторых, узор каждого клиента должен быть симпатичным. Как показало исследование, узор является симпатичным, если в нем нигде не встречается квадрата 2×2 метра, полностью покрытого плитками одного цвета.

Так, симпатичными являются узоры:

1	0	0
0	1	0
0	0	1

1	0	1
0	0	0
1	0	1

0	0	0
0	1	0
0	0	1

Несимпатичными являются узоры:

1	0	1
0	0	0
0	0	1

1	1	0
1	1	0
0	0	1

0	0	1
0	0	1
0	0	1

Для составления финансового плана директору необходимо узнать, сколько клиентов он сможет обслужить, прежде чем симпатичные узоры данного размера закончатся. Помогите ему!

Формат входных данных

В первой строке входного файла находится целое положительное число m, а во второй строке – целое положительное число n $(1 \le m \le 10, 1 \le n \le 30)$.

Формат выходных данных

Выведите число различных симпатичных узоров, которое можно выложить во дворе размера $m \times n$.

Узоры, получающиеся друг из друга сдвигом, поворотом или отражением, считаются различными.

Обратите внимание, что ответ может не поместиться в стандартные целочисленные типы данных.

входной файл	*выходной файл*
2 2	14

Задача 36. Покупка билетов.

В очередь за билетами на футбольный матч Аргентина – Ямайка выстроились n болельщиков, при этом каждый хотел приобрести один билет. В связи с кризисом на стадионе работала единственная касса, поэтому процесс продажи билетов продвигался очень медленно. Некоторые фанаты заметили, что на продажу билетов разным людям тратится разное количество времени: один выплачивает наличными строго нужную сумму, другому выдается сдача, третий совершает операцию с помощью пластиковой карточки. Часто несколько билетов в одни руки кассир продает быстрее, чем когда эти же билеты продаются по одному. Несколько подряд стоящих болельщиков подумали, что могли бы отдавать деньги первому из них, чтобы он купил билеты на всех. Однако, следуя нормам закона «больше трех не собираться», кассир продавала не более трех билетов в одни руки, поэтому совместно купить билет таким образом могли лишь два или три подряд стоящих человека. Известно, что на продажу i-му человеку из очереди одного билета кассир тратит a_i секунд, на продажу двух билетов – b_i секунд, трех билетов – c_i секунд. Определите минимальное время, за которое могут быть обслужены все болельщики в очереди. Обратите внимание, что билеты на группу объединившихся людей всегда покупает первый из них. Также никто в целях ускорения не покупает лишних билетов (то есть билетов, которые никому не нужны).

Формат входных данных

В первой строке входного файла записано число n покупателей в очереди $(1 \le n \le 100\,000)$. В последующих n строках записаны n троек (a_i, b_i, c_i) натуральных чисел. Каждое из этих чисел не превышает 3 600. Люди в очереди нумеруются, начиная от кассы.

Формат выходных данных

Выведите одно число – минимальное время в секундах, за которое могли быть обслужены все болельщики.

входной файл	*выходной файл*
5 5 10 15 2 10 15 5 5 5 20 20 1 20 1 1	12
2 3 4 5 1 1 1	4

Замечание.

В первом примере первый и второй болельщики покупают по одному билету, а третий покупает билеты на троих.

Задача 37. Почта.

Вдоль большой широкой дороги располагаются деревни. Дорога представляет собой ось с целочисленными координатами, а позиции каждой деревни соответствует ее координата (единственное целое число).

Известно, что все деревни расположены в разных местах (нет двух деревень с одинаковой координатой). Расстояние между двумя координатами вычисляется как абсолютная разность их величин. Было решено построить почтовые отделения в деревнях, но не обязательно во всех. Будем считать, что координаты деревни и почтового отделения, построенного в этой деревне, совпадают.

Для построения почты в деревнях, координаты почтовых отделений должны быть выбраны таким образом, чтобы сумма расстояний от каждой деревни до ближайшего почтового отделения была минимальна.

Необходимо по данным координатам деревень и количеству почтовых отделений вычислить минимальную сумму из всех возможных сумм расстояний между деревнями и их ближайшими почтовыми отделениями.

Формат входных данных

В первой строке входного файла записаны два числа: v – число деревень $\left(1 \le v \le 2\,000\right)$ и число p почтовых отделений, $\left(1 \le p \le 30,\ p \le v\right)$.

Вторая строка содержит v целых чисел x_i в возрастающем порядке, которые соответствуют координатам деревень $\left(|x_i| \le 10\,000\right)$.

Формат выходных данных

Выведите минимальную сумму расстояний.

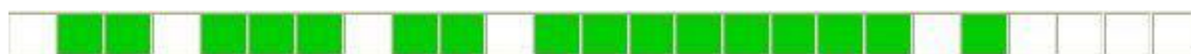
входной файл	*выходной файл*
10 5 1 2 3 6 7 9 11 22 44 50	9

Задача 38. Японская полоса.

На клетчатой полоске бумаги высотой в одну клеточку и длиной n клеточек некоторые клетки раскрашены в зеленый и белый цвета. Дальше по этой полоске строится ее код, которым является последовательность чисел, определяющих число подряд идущих зеленых клеток слева направо. Например, для полоски кодом будет последовательность 2, 3, 2, 8, 1.

При этом число белых клеток, которыми разделяются группы зеленых клеток, нигде не учитывается (главное, что две группы зеленых клеток, разделяются по крайней мере одной белой клеткой). Поэтому одному и тому же коду могут соответствовать несколько полосок.

Например, указанному выше коду может соответствовать также полоса:



Необходимо найти число полосок длины n, которые соответствуют заданному коду.

Формат входных данных

В первой строке входного файла записаны число n – длина полоски $(1 \le n \le 200)$.

Вторая строка содержит k чисел в коде $(0 \le k \le (n+1)/2)$.

Третья строка пустая.

Далее следуют k чисел, задющих код, по одному в каждой строке.

Формат выходных данных

Выведите число полосок, соответствующих заданному коду.

входной файл	*выходной файл*
6 2 2 2	3

Задача 39. Телефонные номера.

В окружающем мире вы часто встречаете много телефонных номеров, и они становятся все длиннее. Вам надо запомнить некоторые из телефонных номеров. Один из методов, как это легче сделать, – это сопоставить буквам цифры, как это показано в следующей таблице:

1	I J
2	A B C
3	D E F
4	G H
5	K L
6	M N
7	P R S
8	T U V
9	W X Y
0	O Q Z

Таким образом, каждые слово или группа слов могут быть сопоставлены одному номеру, и вы можете запоминать слова вместо номеров. Особенно приятно, если есть возможность найти какую-то простую связь между словом и самим человеком. Вы легко можете запомнить номер вашего друга по шахматной игре:

9	4	1	8	3	7	2	9	6
W	H	I	T	E	P	A	W	N

И номер вашего любимого учителя:

2	8	5	5	3	0	4
B	U	L	L	D	O	G

Необходимо написать программу нахождения самой короткой последовательности слов из заданного словаря (т. е. последовательности, содержащей минимально возможное число слов), которая соответствует данному номеру. Соответствие цифр и букв приведено выше в таблице.

Формат входных данных

Первая строка содержит телефонный номер, транскрипцию которого вам нужно найти. Номер содержит не более 100 цифр. Цифры номера не разделяются пробелами. Вторая строка содержит общее число слов в словаре (максимум 50 000). Каждая из последующих строк содержит одно слово, которое состоит не более чем из 50 символов – больших букв английского алфавита и цифр.

Общая длина входного файла не превышает 1 500KB.

Формат выходных данных

В первой строке выведите число слов в самой короткой последовательности, а во второй – найденную самую короткую последовательность слов. Слова отделяются одним пробелом. Если нет решения, в первой строке выведите `No solution`. Если есть более одного решения, имеющего минимум слов, вы можете выбрать любое из них.

входной файл	*выходной файл*
42 4 A E G BE	2 G A
2831123 14 BLABLA BLABALAAASD YOYOYOYYO AE TP HIJ STEREO ATEI K SHOP BY IBZ 7TH 8TH	No solution

Задача 40. Ребенок.

Ребенок нарисовал *n* пронумерованных окружностей разных цветов. Он соединил эти окружности цветными ориентированными отрезками линий.

Каждая пара окружностей может иметь любое число ориентированных отрезков любых цветов, соединяющих эти окружности. Каждому цвету (окружности или отрезка) назначен свой собственный уникальный положительный номер. Начиная игру, ребенок прежде всего выбирает три различных целых числа l, k и q (все три от 1 до n). Затем он ставит одну пешку в окружность с номером l, а другую пешку – в окружность с номером k. Далее он начинает двигать их, используя следующие правила: пешка может быть передвинута только вдоль отрезка, который имеет один цвет с окружностью, в которой находится другая пешка; пешка может передвигаться только в направлении отрезка (все отрезки ориентированные); две пешки никогда не могут находиться в одной и той же окружности в одно и то же время; порядок ходов пешек произвольный (т. е. нет необходимости двигать пешки поочередно); игра заканчивается, когда одна из пешек (любая из двух) достигает окружности с номером q. Необходимо найти кратчайшее (т. е. содержащее минимальное число ходов) решение этой игры, если оно существует.

Формат входных данных

Первая строка входного файла содержит целые числа n, l, k и q, разделенные пробелами $\left(1 \le l, k, q \le n \le 100\right)$. Вторая строка содержит n целых чисел $c_1, c_2, \ldots, c_n$, разделенных пробелами, в заданном порядке, где c_i – цвет окружности, имеющей номер i. Третья строка состоит из целого числа m отрезков $\left(0 \le m \le 10000\right)$. Затем следуют m строк, каждая из которых содержит описание одного ориентированного отрезка. Каждый отрезок задается тремя целыми числами a_j, b_j, c_j, разделенных пробелами, где a_j и b_j – номера окружностей, соединенных j-м отрезком (в направлении от a_j к b_j), а c_j – цвет этого отрезка $\left(1 \le a_j, b_j \le n, a_j \ne b_j, 1 \le c_j \le 100\right)$.

Формат выходных данных

В первой строке выведите слово Yes, если решение существует (игра пришла к концу), или No в противном случае. Если ответ Yes, то во второй строке выведите число минимальное число шагов, которое ребенок должен сделать, чтобы закончить игру.

входной файл	*выходной файл*
5 3 4 1 2 3 2 1 4 8 2 1 2 4 1 5 4 5 2 5 1 3 3 2 2 3 2 4 5 3 1 3 5 1	Yes 3

Задача 41. Елочные игрушки.

Наташа занимается изготовлением елочных игрушек и любит дарить их своим друзьям перед Новым годом. Самым близким друзьям она дарит коробки с наборами игрушек, на которые она потратила больше всего времени. Каждый набор состоит из d_i игрушек, связанных определенной темой, поэтому Наташа не хочет дарить игрушки из одного набора разным людям. Для каждого набора известно время t_i, потраченное на его создание. Сейчас перед Наташей стоит проблема собрать коробку игрушек для самого близкого ей человека. Коробка для хранения игрушек состоит из $n \times m$ ячеек, по одной игрушке в каждой. Часть ячеек могут остаться пустыми, если не удалось найти подходящий набор. Игрушки из одного набора можно разместить в произвольных ячейках коробки. Помогите Наташе выбрать k наборов игрушек для упаковки в одну коробку так, чтобы общее время их изготовления было наибольшим.

Формат входных данных

Первая строка содержит три целых положительных числа: s – количество наборов, n и m – размеры коробки. Гарантируется, что s не превосходит 10^5, а произведение $s \cdot n \cdot m$ не превосходит 10^7.

Каждая из следующих s строк содержит по два целых числа d_i и t_i $\left(1 \le d_i \le 10^9,\quad 1 \le t_i \le 10^9\right)$число игрушек в наборе и время его изготовления.

Формат выходных данных

В первой строке выведите одно число k $\left(0 \le k \le s\right)$ – количество наборов, которые нужно упаковать в коробку.

Во второй строке выведите k различных целых чисел от 1 до s – номера выбранных наборов.

Наборы пронумерованы в порядке следования во входных данных.

Так как оптимальных решений может быть несколько, можете вывести любое из них.

входной файл	*выходной файл*
5 2 7 5 3 10 5 2 1 6 4 5 2	3 1 3 4
1 1 1 2 1000	0
2 1 3 2 100 3 100	1 2

Задача 42. Лента.

Задана лента шириной в одну клетку и длиной в n клеток. На каждой клетке написано некоторое целое число. Играют два игрока, которые ходят поочередно. За один ход игрок отрезает от ленты одну из крайних клеток и забирает ее. Игра останавливается, когда лента заканчивается. Выигрыш игрока равен сумме чисел на находящихся у него клетках ленты.

Необходимо определить величину выигрыша, которую может себе гарантировать игрок, начинающий игру первым. (Это тот размер выигрыша, который первый игрок обеспечит себе при любой игре второго игрока.)

Формат входных данных

Первая строка содержит число n $\left(1 \le n \le 1\,000\right)$.

Следующая строка содержит числа ленты (числа в строке разделены пробелом).

Формат выходных данных

Единственная строка должна содержать целое число – гарантированную величину выигрыша игрока, который начинает игру первым.

входной файл	*выходной файл*
`3` `1 3 4`	`5`
`1 1 1` `2 1000`	`0`

`2 1 3`
`2 100`
`3 100`
	`1`
	`2`

Задача 43. Цифровой экран.

Для проведения чемпионата Европы по футболу в Португалии было решено соорудить k-цветный цифровой экран $n \times m$ пикселей (n по вертикали, m по горизонтали). Но в связи с тем, что экран большой, работать можно только с прямоугольной областью экрана $n_1 \times m_1$ пикселей (n_1 по вертикали, m_1 по горизонтали). Экран был сконструирован таким образом, что при его включении цвета пикселей присваиваются случайным образом, а необходимо, чтобы все пиксели были черного (0-го) цвета.

Вам необходимо определить минимальное число операций, за которое можно сделать из исходного экрана экран черного (0-го) цвета, или указать, что решения не существует.

За одну операцию можно: взять прямоугольную область экрана размера $n_1 \times m_1$ пикселей (n_1 по вертикали, m_1 по горизонтали), полностью принадлежащую экрану, и все номера цветов этой области увеличить на единицу по модулю k (т. е. если номер цвета равен $k-1$, то после увеличения станет 0).

Замечание.

Прямоугольная область берется ровно из n_1 пикселей по вертикали и m_1 по горизонтали, (т. е. нельзя взять m_1 пикселей по вертикали и n_1 по горизонтали (повернуть)).

Формат входных данных

В первой строке находятся числа n и m, разделенные одним пробелом $(1 \le n, m \le 1000)$.

Во второй строке – числа n_1 и m_1, также разделенные одним пробелом $(1 \le n_1 \le n, 1 \le m_1 \le m)$.

В третьей строке – одно число k $(2 \le k \le 1000)$.

Далее идут n строк по m чисел (разделенных одним пробелом) от 0 до $k-1$ в каждой – номера цветов пикселей.

Формат выходных данных

Если нет решения, выведите строку `impossible`, иначе одно число – минимальное число операций.

входной файл	*выходной файл*
1 1 1 1 1000 1	999

Задача 44. Солдаты.

Из n солдат, выстроенных в шеренгу, требуется отобрать нескольких в разведку.

Для того, чтобы сделать это, выполняется следующая операция: если солдат в шеренге больше 3, то удаляются все солдаты, стоящие на четных позициях, или все солдаты, стоящие на нечетных позициях.

Эта процедура повторяется до тех пор, пока в шеренге не останется 3 или менее солдат. Их отсылают в разведку.

Посчитайте, сколькими способами могут быть сформированы таким образом группы разведчиков из любого числа человек (т. е.1, 2 или 3).

Формат входных данных

В единственной строке входного файла содержится число n $(1 \le n \le 10\,000\,000)$.

Формат выходных данных

Выведите число вариантов.

входной файл	*выходной файл*
31	15

Задача 45. Балансировка камней.

Вам даны рычажные весы. На левой чаше весов лежит один камень с весом $\left(w \le 2^n\right)$.

У вас есть множество камней с весами $2^0, 2^1, 2^2, \ldots, 2^{n-1}$.

Определите, сколькими способами можно расположить некоторые из этих камней на чашах весов так, чтобы уравновесить их.

Выведите это значение по модулю d.

Формат входных данных

В первой строке входного файла записаны три числа: n, k и d $\left(1 \le k, n \le 5\,000\,000, 1 \le d \le 1\,000\right)$.

Далее во второй строке ровно k символов – двоичная запись числа w.

Формат выходных данных

Выведите искомое число способов, взятое по модулю d.

входной файл	*выходной файл*
6 4 6 1000	3
6 6 100 100110	5

Задача 46. Гамма язык.

Как стало известно, гамма-язык отличается от европейских языков наличием очень большого числа иероглифов.

В недавно изданном справочнике по гамма-иероглифам содержатся 2 000 000 000 различных иероглифов, и каждому присвоен порядковый номер от 1 до 2 000 000 000.

Правила приличия не позволяют использовать в одном связном гамма-тексте два раза один и тот же иероглиф.

Текст X (возможно, пустой) называется общим подтекстом текстов A и B, если X может быть получен вычеркиванием некоторых иероглифов, как из A, так и из B.

Наибольшим общим подтекстом двух текстов называется их общий подтекст, имеющий максимально возможную длину.

Считается, что чем больше длина наибольшего общего подтекста, тем больше шансов того, что два текста написаны одним и тем же автором.

Заданы два текста A и B, содержащие соответственно n и m иероглифов. Тексты являются приличными, то есть каждый из них не содержит два раза один и тот же иероглиф.

Каждый иероглиф задается своим номером, который является натуральным числом, не превышающим 2 000 000 000.

Вам требуется определить длину наибольшего общего подтекста этих текстов.

Формат входных данных

В первой строке входного файла записаны два целых числа n и m, разделенных одним пробелом $(1 \le n, m \le 100\,000)$.

Вторая строка описывает текст A. Она состоит из n натуральных чисел, разделенных одиночными пробелами. Эти числа представляют собой номера иероглифов в том порядке, как они следуют в тексте.

Третья строка содержит m чисел, которые описывают текст B в аналогичном формате.

Формат выходных данных

Выведите длину наибольшего общего подтекста текстов A и B.

входной файл	*выходной файл*
10 15 1 2 3 4 5 6 7 8 9 10 2000000000 5 3 13 2 1 11 4 9 8 7 10 19 20 1000000000	4
1 1 2000000000 1000000000	0

Задача 47. Делимость и сумма.

Назовем число счастливым, если оно делится на k и сумма его цифр лежит в интервале $[p, q]$.

Подсчитайте, сколько счастливых чисел лежат в интервале $[a, b]$.

Формат входных данных

Первая строка входного файла содержит десятичную запись числа k $(1 \le k \le 100)$

Вторая строка – числа a и b $\left(1 \le a, b < 10^{19}\right)$.

Третья – десятичную запись чисел p и q $(1 \le p, q < 162)$.

Формат выходных данных

Выведите найденное число.

входной файл	*выходной файл*
11 11 40 3 6	2

Задача 48. Дорожный контроль.

В одной стране под названием Инфолэнд есть n городов, связанных между собой двусторонними дорогами. Цепью между городами a и b называется маршрут между этими городами, который использует каждую дорогу не более одного раза. Дороги в этой стране построены так, что для любых городов a и b существует ровно одна цепь, связывающая эти два города. Правительство этой страны приняло указ о контроле дорог своего государства.

Было решено в некоторых городах создать комитеты по контролю всех дорог, начинающихся или заканчивающихся в этом городе. В целях экономии необходимо создать наименьшее возможное число таких комитетов, но, чтобы любая дорога находилась под контролем хотя бы одного комитета.

Формат входных данных

В первой строке – целое число n $(1 \le n \le 100\,000)$.

Далее идут n строк, описывающих для каждого города все города, соединенные прямой дорогой с этим городом: сначала число k_i таких городов, затем номера этих городов.

Формат выходных данных

Выведите минимальное число городов, в которых могут размещаются комитеты контроля.

входной файл	*выходной файл*
6 1 3 1 3 5 1 2 4 5 6 1 3 1 3 1 3	1

Задача 49. Единицы (часть 3).

Определите, сколько на интервале $[a, b]$ содержится натуральных чисел, в двоичной записи которых ровно k единиц.

Формат входных данных

Единственная строка входного файла записаны три целых числа a, b и k $\left(1 \le a \le b \le 10^{18}, 0 \le k \le 62\right)$.

Формат выходных данных

Выведите одно число – ответ на задачу.

Так как ответможет быть очень большим, необходимо его вывести по модулю $10^9 + 7$.

входной файл	*выходной файл*
1 1 1	1
735627 67249026 20	230230

Задача 50. Лестница.

Методист по информатике О. Г. живет на *n*-м этаже 99-этажного дома с лифтом, который может останавливаться на каждом этаже. Между соседними этажами дома имеется лестница из двух пролетов, разделенных площадкой, по k ступенек в каждом пролете.

Сколькими способами О. Г. может подняться на свой этаж, если, поднимаясь по лестнице, можно становиться на следующую ступеньку или через одну ступеньку, но при этом нельзя перейти мгновенно с одного пролета на другой без посещения площадки между ними?

Методист может проехать любое число этажей на лифте (но он не может спускаться вниз ни на лифте, ни по лестнице).

Поскольку ответ может быть достаточно большим, вычислите его по модулю 1 000 000 009. Можете считать, что вход в подъезд расположен на площадке первого этажа.

Формат входных данных

Единственная строка входного файла содержит два натуральных числа *n* и *k* $\left(1 \le n \le 99, 1 \le k \le 100\right)$.

Формат выходных данных

Выведите число спосбов подняться на *n*-ый этаж, вычисленное по модулю 10^9+9.

входной файл	*выходной файл*
5 2	625
1 1	1

Задача 51. Пасьянс.

Как известно, одной из важнейших программ в составе любой операционной системы является пасьянс. В состав операционной системы PisiPuck входит следующая разновидность пасьянса. Колода карт из *n* карт, пронумерованных от 1 до *n*.

На столе имеется *n* ячеек, расположенных в один ряд. Все ячейки также пронумерованы от 1 до *n*. Каждая ячейка в начале игры содержит ровно одну карту.

Необходимо сложить все карты в колоду на произвольной ячейке таким образом, чтобы внизу лежала карта *n*, на ней – карта с номером $n-1$, а на самом верху колоды – карта с номером 1. Таким образом, вся колода карт окажется сложенной в одной ячейке, а все остальные ячейки будут пустыми.

При раскладывании пасьянса разрешается только одна операция: можно перенести карту с номером *m* и лежащие на ней карты (если они имеются) на карту с номером $m+1$. За каждую такую операцию переноса игроку начисляется штраф, равный расстоянию, на которое переносится стопка карт, т. е.разности в номерах мест ячеек.

Вам необходимо определить минимальный суммарный штраф, который может быть получен при складывании пасьянса из заданной конфигурации.

Формат входных данных

В первой строке задается число n карт пасьянса $(1 \le n \le 500)$. В следующей строке n чисел – номера карт, выложенных в ячейках, в начале игры.

Карты задаются в порядке возрастания номеров ячеек, в которых они выложены.

Формат выходных данных

Выведите одно число – минимально возможную суммарную величину штрафа.

входной файл	*выходной файл*
2 2 1	1
73 1 24 57 62 68 36 2 72 31 8 37 48 9 14 23 16 43 64 38 35 27 3 70 56 15 71 22 13 29 30 41 39 26 34 11 6 20 18 59 45 12 42 40 44 32 25 46 17 49 50 61 4 53 54 55 19 21 58 51 5 47 63 52 7 65 73 10 60 69 66 33 67 28	880

Задача 52. Вика и квадраты.

Вике подарили матрицу, состоящую из n строк и m столбцов, каждый элемент которой равен `0` или `1`. Помогите ей найти количество таких квадратных подматриц, которые полностью состоят из единиц.

Формат входных данных

Первая строка входного файла содержит два натуральных числа n и m – размеры подаренной Вике матрицы $(1 \le n, m \le 1000)$.

Каждая из следующих n строк сосотоит из m чисел, каждое из которых равно `0` или `1`.

Формат выходных данных

Выведите одно число – ответ на задачу.

входной файл	*выходной файл*
1 5 1 1 1 1 1	5
3 4 1 1 1 1 1 1 1 1 1 1 0 1	15

Задача 53. Проход в матрице из (1, 1) в (n, m).

Задана матрица A натуральных чисел размера $n \times m$. За каждый проход через элемент взимается штраф $a_{i,\,j}$.

Необходимо минимизировать штраф и пройти из элемента $(1,1)$ в (n,m), при этом из текущего элемента можно перейти в любой из трех соседних, стоящих в строке с номером, на 1 большим (т. е. из элемента $(i,\ j)$ можно перейти в один из соседних элементов

$$(i+1,\ j-1),(i+1,\ j),(i+1,\ j+1),$$

если таковые существуют).

Формат входных данных

Первая строка входного файла содержит два числа, где n – число строк, а m – число столбцов матрицы A $(2 \le n \le 200, 1 \le m \le 1\,000)$.

Следующие n строк задают саму матрицу A: каждая строка содержит по m целых чисел, разделенных одиночными пробелами.

Формат выходных данных

Выведите одно целое число – минимальную стоимость пути или −1 (минус единицу), если пути нет.

входной файл	*выходной файл*
6 3 112 213 1 123 1 123 1 324 343 1 546 644 978 1 999 123 123 1	117

Задача 54. Строка.

Дана строка S, состоящая из n маленьких латинских букв $(1 \le n \le 300)$.

За один ход Вам разрешается удалить один или несколько подряд идущих одинаковых символов.

Необходимо удалить все символы из строки S за минимальное число ходов.

Формат входных данных

Единственная строка входного файла содержит исходную строку S.

Формат выходных данных

Выведите одно целое число – минимальное число ходов.

входной файл	*выходной файл*
acdcbbc	117

Задача 55. Треугольник.

				7				
			3		8			
		8		1		0		
	2		7		4		4	
4		5		2		6		5

Рисунок 2.8– Числовой треугольник.

Необходимо определить максимальную сумму чисел, расположенных на пути, который начинается с верхнего числа и заканчивается на каком-нибудь числе в основании треугольника (максимум суммы среди всех таких путей). На каждом шаге можно двигаться к соседнему по диагонали числу влево и вниз или вправо и вниз. Число строк в треугольнике k. Все числа в треугольнике $a_{i,j}$ – целые в интервале между 0 и 99 включительно.

Формат входных данных

В первой строке записано число k строк треугольника $(1 \le k \le 1000)$. Следующие k строк входного файла задают сам треугольник. В i-й строке $(1 \le i \le k)$, описывающей треугольник, содержится i чисел, записанных через пробел.

Формат выходных данных

Выведите одно целое число – искомую максимальную сумму.

входной файл	*выходной файл*
5 7 3 8 8 1 0 2 7 4 4 4 5 2 6 5	30

Задача 56. Фишка.

Фишка может двигаться по полю длины n только вперед. Длина хода фишки не более k. Необходимо найти число различных маршрутов, по которым фишка может пройти поле от позиции 1 до позиции n. Разработанный алгоритм должен иметь трудоемкость $O(n)$.

Формат входных данных

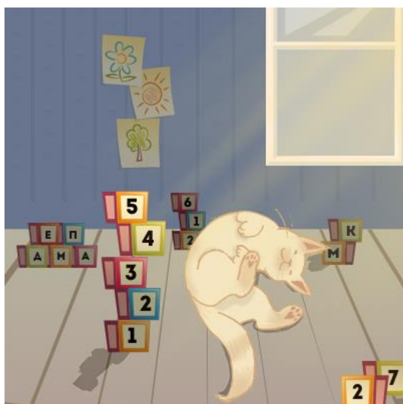
Первая строка содержит число t наборов входных данных $(1 \le t \le 100)$. Следующие t строк содержат по два числа: n и k, где n – длина текущего поля, а k – максимальная длина хода фишки $(1 \le k < n \le 100\,000)$.

Формат выходных данных

Необходимо для каждого набора входных данных вывести остаток от деления числа различных маршрутов на 1 000 000 007.

входной файл	*выходной файл*
2 4 2 8 1	3 1

Задача 57. Андрей играет в кубики.



После сдачи ЦТ по математике Андрей решил поиграть в кубики. У него есть N кубиков, каждый характеризуется величинами A_i и B_i. При этом на каждом кубике написан его номер. Андрей хочет построить башенку из N кубиков, причем все кубики должны идти по порядку, а самым нижним кубиком должен быть кубик с номером 1. Изначально на полу стоит N башенок из одного кубика. Андрей может соединять две башенки, при условии, что после соединения этих башенок кубики в получившейся башенке будут идти по порядку $i, i+1, i+2, \ldots$ На кубике с меньшим номером должен лежать кубик с большим номером. Так как это необычные кубики, для соединения башенок нужно гладить кота. Пусть l – кубик с наименьшим номером среди всех кубиков в двух соединяемых башнях, а r – с максимальным. Чтобы соединить башенки, Андрею нужно погладить кота $A_l \cdot B_r$ раз. Для того, чтобы не мучить бедное животное, Андрею нужно знать: какое минимальное количество раз надо погладить кота, чтобы собрать башенку из N кубиков.

Формат входных данных

На вход в первой строке подается одно целое положительное число $(N \leq 500)$ количество кубиков. Далее в N строках следуют по два целых положительных числа $A_i \leq 10^7, B_i \leq 10^7$.

Формат выходных данных

Выведите одно целое неотрицательное число – ответ на задачу.

входной файл	*выходной файл*
3 34 29 29 4 4 15	646

Задача 58. Свертка строки.

Билл пытается компактно представить последовательность заглавных букв алфавита от *A* до *Z*, сворачивая повторяющиеся подпоследовательности внутри нее. Например, одним из способов представить последовательность *AAAAAAAAAABABABCCD* является 10(*A*)2(*BA*)*B*2(*C*)*D*. Он формально определяет свернутую последовательность и операцию разворачивания следующим образом:

– Последовательность, содержащая единственный символ от *A* до *Z*, считается сжатой последовательностью. Разворачивание этой последовательности дает тот же единственный символ.

– Если *S* и *Q* – свернутые последовательности, то *SQ* также свернутая последовательность. Если *S* разворачивается в *S'*, а *Q* разворачивается в *Q'*, то *SQ* разворачивается в *S' Q'*.

– Если S – свернутая последовательность, то $X(S)$ также свернутая последовательность, где X – десятичное представление целого числа, большего 1. Если S разворачивается в S', то $X(S)$ разворачивается в S', повторяющуюся X раз.

Исходя из этого определения, легко развернуть данную свернутую последовательность. Однако Билл гораздо более заинтересован в обратном преобразовании. Он хочет свернуть данную последовательность таким образом, чтобы результирующая свернутая последовательность содержала наименьшее возможное число символов.

Формат входных данных

На вход подается единственная строка длины от 1 до 700, состоящая из символов от *A* до *Z*.

Формат выходных данных

Выведите кратчайшую возможную свернутую последовательность, которая разворачивается в последовательность, данную во входном файле. Если таких последовательностей несколько, выведите любую из них.

входной файл	*выходной файл*
`NEERCYESYESYESNEERCYESYESYES`	`2(NEERC3(YES))`

Задача 59. Репортаж.

Группа альпинистов покорила много вершин и возвратилась в родной город. Одна из местных газет решила написать статью об их походе. Как выяснилось, в процессе похода альпинисты n раз останавливались на ночлег на той или иной высоте. Поскольку главный редактор газеты настаивает, чтобы название статьи было «Восхождение и спуск», решено было не упоминать о некоторых днях похода, рассказав лишь о восхождении, причем если статья будет рассказывать о x_1-м, x_2-м, ..., $x_{2\cdot k+1}$-м днях $\left(x_1 < x_2 < \ldots < x_{2\cdot k+1}\right)$, то должно выполняться условие: $h_{x_1} < h_{x_2} < \ldots < h_{x_k} > h_{x_{k+1}} > h_{x_{k+2}} > \ldots > h_{x_{2\cdot k+1}}$.

Найдите максимальное k, для которого можно соответствующим образом выбрать $(2\cdot k+1)$ дней.

Формат входных данных

Первая строка содержит целое число n – количество дней в походе $(1 \le n \le 100)$.

Следующая строка содержит n целых чисел $h_1, h_2, \ldots, h_n$ $(0 \le h_i \le 10\,000)$.

Формат выходных данных

В первой строке выведите число k. Затем выведите $2\cdot k+1$ число – номера дней, репортаж о которых следует включить в статью, в возрастающем порядке.

Если возможных ответов несколько, выведите любой.

входной файл	*выходной файл*
`7` `0 3 1 10 7 2 1`	`2` `1 2 5 6 7`

Задача 60. Оптимальное разрезание.

Необходимо разрезать прямоугольник размера $x \times y$ на детали прямоугольной формы размера $x_1 \times y_1$ и $x_2 \times y_2$, чтобы отходы были минимальными. Возможны только вертикальные и горизонтальные разрезы. Т. е. после первого разреза получается два прямоугольника, которые впоследствии также можно разрезать. Прямоугольники можно поворачивать на 90 градусов.

Формат входных данных

Первая строка входного файла содержит шесть целых чисел x, y, x_1, y_1, x_2, y_2 $(1 \le x,\ y \le 300, 1 \le x_1,\ x_2 \le x, 1 \le y_1,\ y_2 \le y)$.

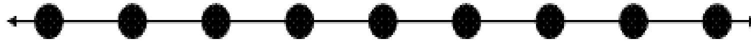
Формат выходных данных

Выведите минимальную сумму площадей остатков.

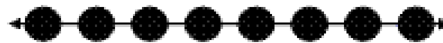
входной файл	*выходной файл*
`10 10 2 9 3 4`	`10`

Задача 61. Вредная лягушка.

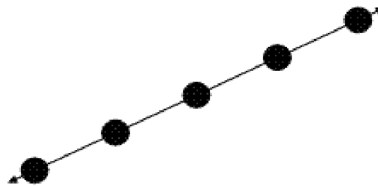
Маленькие вредные лягушки «ченгайгури» стали в Корее легендарными. По ночам они прыгают по рисовым полям, втаптывая в землю кустики риса и нанося урон посевам. Утром, заметив поврежденные кустики, вы захотели определить путь той лягушки, которая нанесла наибольший урон. Лягушка всегда прыгает по полю по прямой линии, причем длина каждого прыжка одинакова.



Разные лягушки могут иметь разную длину прыжка



и разные направления.



Кустики риса на вашем поле посажены строго вдоль перпендикулярных линий на одинаковом расстоянии друг от друга, как это показано на рисунке 2.8.

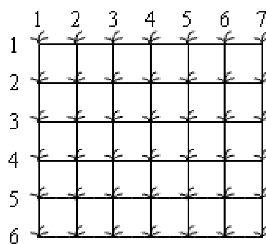


Рисунок 2.8– Посадка риса.

Вредные лягушки проскакали через все ваше поле, причем путь лягушки начинался за пределами поля с одной его стороны, а заканчивался за его пределами с другой стороны, как это показано на рисунке 2.9.

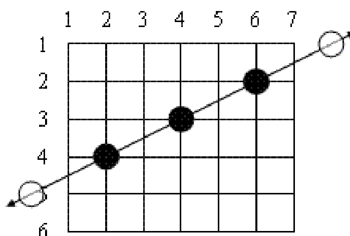


Рисунок 2.9 – Путь лягушки.

По полю может прыгать много лягушек, перепрыгивая от одного рисового кустика к другому. При каждом прыжке лягушка втаптывает в землю очередной кустик риса, как на рисунок 2.10.

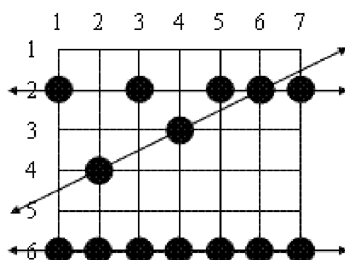


Рисунок 2.10 – Пути нескольких лягушек.

Отметим, что за ночь одно и то же растение может быть затоптано более чем одной лягушкой. Конечно, вы не сможете увидеть прямые линии, вдоль которых прыгали лягушки, и какие-либо следы их перемещения за пределами поля. Для примера, показанного на рисунке 2.10, вы увидите картину поврежденного поля, показанную на рисунке 2.11.

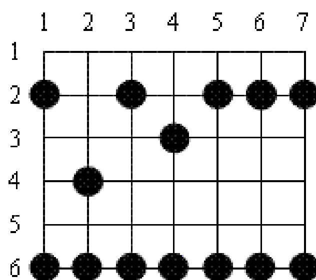


Рисунок 2.11 – Повреждение поля лягушками с рисунка 2.10.

Исходя из рисунка 2.11, вы сможете восстановить все возможные пути, по которым лягушки могли бы перемещаться через ваше поле. Нас интересуют только те лягушки, которые повредили не менее трех рисовых кустиков при своем перемещении через поле. Таким образом, мы определили понятие «путь лягушки». Можно заметить, что три пути, показанные на рисунке 2.10, являются путями лягушек (имеются также другие возможные пути лягушек). Вертикальный путь, проходящий вниз вдоль столбца 1 мог бы быть путем лягушки, имеющей длину прыжка 4, однако при этом могло быть повреждено только 2 кустика риса, и такой путь нас не интересует. Рассмотрим диагональный путь, проходящий через кустики, находящиеся на пересечении строки 2 и

столбца 3, строки 3 и столбца 4, строки 6 и столбца 7. На этом пути находятся три поврежденных кустика, но путь неравномерный и содержит интервалы неравной длины; таким образом, этот путь также не является путем лягушки. Отметим также, что вдоль линии движения лягушки могут встречаться поврежденные кустики, которые не были повреждены именно этой лягушкой (например, кустик в точке (2, 6) на горизонтальном пути вдоль строки 2 на рисунке 2.11), и на поле могут находиться поврежденные кустики, происхождение повреждения которых нельзя объяснить никаким путем лягушки. Рассмотрим все возможные пути лягушек. Ваша программа должна определить число поврежденных кустиков риса на пути той лягушки, которая повредила больше всех кустиков (то есть нанесла самый большой урон рисовой плантации). На рисунке 2.11 таким мог быть путь вдоль строки 6, а ответом является число 7.

Формат входных данных

Первая строка входного файла содержит два целых числа r и c – число строк и столбцов на рисовом поле $\left(1 \le r, c \le 10^9\right)$. Вторая строка содержит число n поврежденных рисовых кустиков $\left(3 \le n \le 7\,000\right)$. Каждая из следующих n строк содержит два целых числа – номер строки (от 1 до r включительно) и номер столбца (от 1 до c включительно), определяющие позицию поврежденного кустика. Эти числа разделяются одним пробелом. Каждый поврежденный кустик описан ровно один раз.

Формат выходных данных

Выведите число кустиков, лежащих на пути лягушки, которая нанесла наибольший урон, если такой путь существует. В противном случае, необходимо вывести `0`.

Приведенному примеру входных и выходных данных соответветствуют рисунки 2.11 и 2.10.

входной файл	*выходной файл*
6 7 14 2 1 6 6 4 2 2 5 2 6 2 7 3 4 6 1 6 2 2 3 6 3 6 4 6 5 6 7	7

Рассмотрим еще один пример, демонстрирующий условие задачи. Примеру соответветствуют рисунки 2.12 и 2.13.

входной файл	*выходной файл*
6 7 18 1 1 6 2 3 5 1 5 4 7 1 2 1 4 1 6 1 7 2 1 2 3 2 6 4 2 4 4 4 5 5 4 5 5 6 6	4

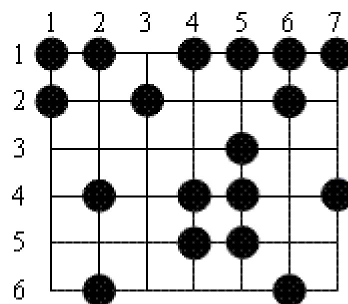


Рисунок 2.12 – Повреждение поля.

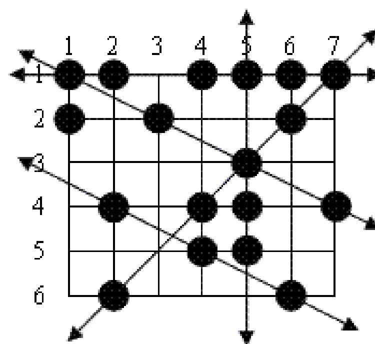


Рисунок 2.13 – Максимальное количество поврежденных кустиков 4.

Задача 62. Микрочип.

Микрочип, производимый на некотором заводе, имеет форму плоского квадрата со стороной ***a*** микрометров. На нижнюю грань выведены контакты, причем координаты этих контактов в системе координат, в которой оси параллельны сторонам чипа, а единичный отрезок имеет длину 1 мкм, являются целыми числами. Для успешной распайки необходимо от каждого контакта протянуть проводящую дорожку к одной из сторон чипа для последующего закрепления на ноге интегральной схемы. Однако используемый технологический процесс позволяет создавать только прямые дорожки, идущие от контакта к краю чипа без изгибов и параллельные сторонам кристалла, причем невозможно проложить одну дорожку под или над другой. Поэтому Вам необходимо определить, в какую сторону выводить каждый из контактов, чтобы полученные дорожки не пересекались, а суммарная их длина была минимальной.

Формат входных данных

Первая строка входного файла содержит натуральное число a – длина стороны микрочипа в микрометрах $(1 \le a \le 30)$. Во второй строке находится число n контактов на нижней стороне чипа $(1 \le n \le 38)$. В последующих n строках следуют пары целых чисел в диапазоне от 1 до $a-1$ – соответственно абсциссы и ординаты контактов во введенной системе координат.

Формат выходных данных

Выведите в первой строке о минимальную суммарную длину необходимых дорожек. В последующий строках поясните, в какую сторону выводить дорожку для каждого из контактов: в $(i+1)$-й строке выведите одно из слов `UP` (англ. «вверх»), `DOWN` (англ. «вниз»), `LEFT` (англ. «налево»), `RIGHT` (англ. «направо») – направление выведения дорожки i-го контакта.

В случае неоднозначного ответа выводите любой, обеспечивающий минимальную суммарную длину дорожек.

входной файл	*выходной файл*
15 15 11 13 6 4 13 3 4 12 5 5 4 9 1 14 3 11 9 7 2 12 1 10 12 2 7 6 8 8 12 1	0 UP DOWN RIGHT UP LEFT LEFT UP LEFT RIGHT LEFT LEFT RIGHT DOWN UP DOWN

Задача 63. Али-баба и монеты.

На дороге в некоторых местах разбросаны золотые монеты. Для каждой монеты известно ее местоположение, которое задается одним целым числом – расстоянием в метрах от начальной отметки.

Все монеты расположены правее начальной отметки на различном расстоянии от начала.

Али-баба бегает по дороге и собирает монеты, начиная делать это в момент времени 0. За одну секунду он пробегает ровно один метр. У каждой монеты есть крайний срок, до которого ее нужно подобрать, иначе монета исчезнет.

Али-баба должен собрать все монеты и сделать это за минимально возможное время.

Он может стартовать в любой точке на прямой, собирать монеты в произвольном порядке, но обязательно должен успеть собрать все монеты и минимизировать затраченное на это время.

Формат входных данных

Первая строка входного файла содержит целое число монет n $(1 \le n \le 10\,000)$.

В каждой из последующих n строк содержатся по два целых числа d и t, первое из которых задает положение монеты $(1 \le d \le 10\,000)$, а второе – крайний срок в секундах $(1 \le t \le 10\,000)$, до которого нужно успеть собрать эту монету (в момент времени ровно t подобрать монету также возможно).

Монеты перечислены в порядке возрастания расстояния от начала (слева направо).

Формат выходных данных

Выведите единственное число – минимальное время, за которое Али-баба может собрать все монеты.

Если Али-баба не может собрать все монеты – выведите строку `No solution`.

входной файл	*выходной файл*
5 1 3 3 1 5 8 8 19 10 15	11
5 1 5 2 1 3 4 4 2 5 3	No solution

Задача 64. Олимпиада покемонов.

Каждый год проходит Олимпиада покемонов. Соревнование проходит в два тура, каждый день покемоны соревнуются в различных видах активности: беге, соревнованиях и даже в решении задач. После подведения итогов олимпиады для чествования победителей проводится церемония награждения, во время которой участников поочередно вызывают на сцену для вручения дипломов и памятных сувениров.

Считается, что церемония закрытия была красивой, если все победители, выйдя на сцену и заняв свое место, оказываются выстроенными по росту. Но сценарий церемонии награждения требует выхода на сцену призеров в строгом порядке. Будем считать, что на сцену должны выйти *M* покемонов, причем рост выходящего i-го по порядку покемона равен H_i нанобетров.

Чтобы церемония закрытия прошла красиво, организаторы решили использовать пьедестал особой формы. Пьедестал – это скрепленные блоки, каждый из которых является параллелепипедом различной высоты. Всего при конструировании пьедестала было использовано *N* блоков $(M \le N)$. Все блоки пронумерованы (в направлении от входа на сцену) целыми числами от 1 до *N*, высота *j*-го из них равна B_j нанобетров. На каждый блок может стать только один покемон.

РЕЗУЛЬТАТ СОРЕВНОВАНИЙ

No	УЧАСТНИК	РОСТ	No ПЬЕДЕСТАЛА
1	ПИКАЧУ	10	1
2	ЧЕРЕПАШКА	17	3
3	ПАНДА	10	4
4	ПТИЧКА	10	5

ВХОДНЫЕ ДАННЫЕ:
4 5
10 17 10 10
10 20 5 15 20

Во время церемонии награждения, очередной покемон должен пройти вдоль пьедестала, поздравить всех находящихся на сцене победителей, и занять свое место.

Чтобы этот процесс был организованным, было принято решение заранее сообщить участникам, где им необходимо стать на сцене.

Более формально, требуется определить набор различных позиций

$$1 \le p_1 \le p_2 \le \ldots \le p_M \le N ,$$

куда должны стать победители олимпиады.

Напомним, что для того, чтобы церемония прошла красиво должно выполняться условие:

$$H_1 + B_{p_1} < H_2 + B_{p_2} < \ldots < H_M + B_{p_M} .$$

Напишите программу, которая определит место на сцене для каждого победителя олимпиады так, чтобы церемония награждения прошла красиво, или сообщит, что такого набора мест не существует.

Формат входных данных

Первая строка входного файла содержит два натуральных числа N и M $\left(2 \le N \le 2\,594,\, 2 \le M \le 1\,022\right)$ – количество победителей олимпиады и количество блоков, из которых составлен пьедестал.

Вторая строка содержит M целых чисел H_i $\left(1 \le H_i \le 10^9\right)$.

Третья строка содержит N целых чисел B_j $\left(1 \le B_j \le 10^9\right)$.

Числа в строках входного файла разделены одиночными пробелами.

Формат выходных данных

В единственной строке выведите M разделенных одиночными пробелами различных целых чисел p_i – номера блоков, куда должны стать победители олимпиады, чтобы церемония награждения прошла красиво (числа должны следовать по возрастанию).

Если существует несколько вариантов размещения победителей на пьедестале, то выведите любой из них.

Если провести церемонию закрытия красиво невозможно, то выведите –1 (минус один).

входной файл	*выходной файл*
3 5 21 12 19 10 20 10 20 10	1 2 4
3 5 20 20 20 10 10 10 30 30	-1

Задача 65. Джокеры

Никто не умеет менять свой цвет так, как хамелеон, поэтому не удивительно, что в Колоретто – самой цветной настольной игре – на картах изображены забавные хамелеончики. В Колоретто цвета меняются, как в калейдоскопе, и цвет, буквально только что приносивший вам победные очки, может вдруг оказаться лишним.

Игроки собирают цветные карты хамелеончиков (всего в игре *k* цветов), но лишь *t* цветов карт принесут в конце игры призовые очки, а вот остальные цвета, если они у игрока оказались, обернутся штрафными очками. Но в Колоретто то, какие карты достанутся игроку, зависит от его соперников, которые в той же мере зависят от его выбора. Так что игрокам постоянно приходится предугадывать действия других игроков и иногда рисковать, а вовремя выбрать правильный цвет будет для них не менее важно, чем для хамелеона, прячущегося на грядке с клубникой.

В игре используется по *c* карт каждого цвета и *j* джокеров (остальные карты из оригинальной игры мы опустим). За время игры некоторое подмножество этих карт оказывается у игрока. Вам же нужно помочь ему распределить джокеров.

Подсчет очков для каждого игрока происходит независимо. За каждый цвет игрок получает число очков (положительное или отрицательное) в соответствии с числом карт этого цвета у него. За *x* карт одного цвета причитается a_x очков.

Каждый игрок сам выбирает, какие *t* цветов приносят ему положительное число очков. Все прочие цвета приносят ему отрицательное число очков.

Игрок сам определяет, какому цвету соответствует каждый из его джокеров (джокеры могут быть добавлены и к одному цвету, и к разным цветам – по выбору игрока), а затем подсчитываются очки.

Формат входных данных

В первой строке входного файла записаны четыре целых числа *k*, *c*, *j*, *t* $\left(2 \le k, c, j, t \le 100, t \le 100, t \le k\right)$.

Во второй строке записано $c + j + 1$ чисел $a_i \left(0 \le i \le c + j, 0 \le a_i \le 1000\right)$.

В третьей строке записано число *n* карт у игрока $\left(1 \le n \le k \cdot c + j\right)$.

В последней строке записаны *n* чисел.

Цветные карты задаются числами от 1 до *k*, джокеры – нулями.

Формат выходных данных

Выведите максимальное число очков, которое может набрать игрок.

входной файл	*выходной файл*
7 9 3 3 0 1 3 6 10 15 21 21 21 21 21 21 21 16 1 4 3 3 4 1 4 7 1 0 3 1 1 3 1 7	39

Задача 66. Фирма Macrohard

Пете Булочкину крупно повезло: он, наконец, устроился на работу в фирму «Macrohard». Он хочет показать себя с самой лучшей стороны, поэтому к первому своему заданию отнесся весьма ответственно. Задание состоит в том, чтобы написать поисковую систему. Пете заранее известен набор чисел $A_1, A_2, \ldots, A_k$ (все A_i – различные целые числа), назовем их ключами. Система должна обрабатывать запросы типа: «Содержится ли среди ключей число s ?».

Известно, что число s может быть любым целым числом от 1 до n. Руководство фирмы сказало Пете, что ему нужно использовать как можно меньше памяти.

Поразмыслив, Петя решил, что оптимальным решением поставленной задачи будет использование двоичных деревьев поиска, описание которых приведено ниже.

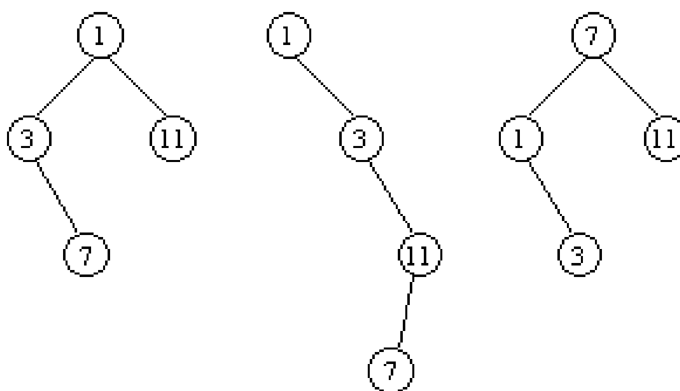


Рисунок 2.14 – Двоичные деревья поиска.

Двоичное дерево может быть пустым, или состоять из вершины, к которой присоединены два двоичных дерева, то есть левое и правое поддерево (в этом случае вершину, к которой присоединяются деревья, называют корнем).

Если в каждую вершину поместить по ключу, причем так, чтобы в разных вершинах были различные ключи, то получим двоичное дерево для заданного набора ключей.

Будем говорить, что дерево является двоичным деревом поиска, если левое и правое поддеревья являются двоичными деревьями поиска, а также любой ключ из левого поддерева, выходящего из корня, меньше ключа, записанного в корне, а любой ключ из правого поддерева – больше.

Для того, чтобы проверить, содержится ли в заданном двоичном дереве поиска ключ s, используется следующий алгоритм:

1. Положить текущую вершину равной корню дерева.

2. Проверить, совпадает ли s с ключом, записанным в текущей вершине. Если да, то ключ s найден. Иначе перейти к шагу 3.

3. Если s меньше ключа, записанного в текущей вершине, то положить текушую вершину равной корню левого поддерева, иначе – равной корню правого поддерева (если соответствующие поддеревья отсутствуют, то алгоритм заканчивает работу, выдавая, что ключ sss в дереве отсутствует). Перейти к шагу 2.

Стоимостью поиска ключа s назовем количество выполненных шагов 2 вышеописанного алгоритма.

Например, для правого дерева на рисунке 2.14 стоимость поиска ключа 7 равна 1, стоимость поиска ключей 1, 8, 9, 10, 11 равна 2, стоимость поиска ключей 2, 3, 4, 5, 6 равна 3.

Стоимостью заданного двоичного дерева для диапазона поиска от 1 до n назовем сумму стоимостей поиска каждого из ключей от 1 до n в этом дереве.

Например, стоимость правого дерева на рисунке для диапазона поиска от 1 до 11 равна 26.

Петя хочет построить для своей поисковой системы двоичное дерево поиска минимальной стоимости.

По введенным числам n, k, $A_1, A_2, \ldots, A_k$ определить минимальную стоимость C двоичного дерева поиска для набора ключей $A_1, A_2, \ldots, A_k$ и диапазона поиска от 1 до n.

Формат входных данных

В первая строка входного файла содержит одно целое число n $\left(1 \le n \le 10^7\right)$ – ограничения сверху на числа.

Во второй строке записано одно целое число $k\left(1 \le k \le 300\right)$ – количество чисел.

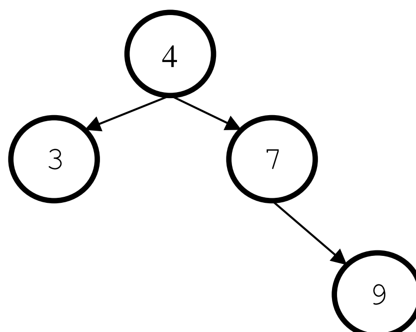
Затем следуют k строк, i-ая из них содержит число A_i $\left(1 \le A_i \le n\right)$.

Формат выходных данных

Выведите единственное число C – минимальную стоимость двоичного дерева поиска.

входной файл	*выходной файл*
10 4 9 3 7 4	22

Оптимальным деревом поиска для приведенного примера будет следующее:



Задача 67. Ним.

Родители мальчика Вовы часто ему повторяют, что со спичками играть нельзя. Однако любознательный мальчик постоянно придумывает новые игры со спичками, не рискуя зажигать их. Не так давно папа Вовы рассказал ему занимательную игру «Ним», в которой камушки разложены по нескольким кучкам, и два игрока, делая ходы поочередно, могут брать произвольное число камушков из любой непустой кучки. Тогда же папа научил его определять, может ли он выиграть (при оптимальной стратегии противника), если перед его ходом определенное число камушков в кучках. Но игры с камушками совсем не привлекают Вову, а спичек у него дома очень-очень много. Вот он и решил посчитать, сколько существует начальных позиций в игре, в которых выигрывает первый игрок. Для начала он легко решил эту задачу для одной кучки спичек. А сейчас его интересует, сколькими способами можно разложить спички в n кучек, если в каждой должно быть менее 2^m спичек и при этом в разных кучках должно быть различное число спичек.

Замечание.

В условии задачи подразумевается классический вариант игры «ним», в котором два игрока делают ходы по очереди, за один ход каждый игрок должен из любой одной кучи взять любое положительное целое число спичек, а если сделать ход не может (не осталось непустых кучек), то проигрывает. Несложно показать, что если игра начинается с кучками размера $a_1, a_2, \ldots, a_n$ спичек, то первый игрок может гарантировать себе победу тогда и только тогда, когда $a_1 \oplus a_2 \oplus \cdots \oplus a_n \neq 0$, где операция $\oplus$ –это поразрядное сложение по модулю два в двоичной системе счисления. Например, позиция (5, 6, 7) выигрышная, потому что $5_1 \oplus 6 \oplus 7 = 101_2 \oplus 110_2 \oplus 111_2 \neq 0$, а позиция (5, 6, 3) – проигрышная, потому что $5_1 \oplus 6 \oplus 3 = 101_2 \oplus 110_2 \oplus 011_2 = 0$.

Формат входных данных

Первая строка входного файла два целых числа n и m $(1 \leq n, m \leq 10\,000\,000)$.

Формат выходных данных

Выведите искомое число способов по модулю 1 000 000 007.

входной файл	*выходной файл*
4 2	0
3 3	168

Задача 68. Ожерелье.

Мальчик Вася любит играть со старинным ожерельем, составленным из n камней различных цветов (все камни различаются по размеру и форме). Камни нанизаны на цепочку так, что никакие два камня одинакового цвета не находятся рядом; камни, расположенные по разные стороны от замка, застегивающего ожерелье, также имеют разный цвет. Увы, все вещи когда-нибудь ломаются… Так произошло и с ожерельем: одно неосторожное движение – и цепочка разорвалась, а камни рассыпались по полу. Васе удалось найти и собрать все рассыпавшиеся камни, но он не помнит, в каком порядке они шли в ожерелье…

Сможете ли Вы подсчитать, сколько способов сборки ожерелья, удовлетворяющих описанным правилам, существует? Каждый способ сборки однозначно описывается порядком следования камней, начиная от замка, скрепляющего цепочку, по часовой стрелке (камни одного цвета считаются различными, так как все они имеют различную форму).

Формат входных данных

Первая строка входного файла два целых числа n и k – число камней и число цветов соответственно $(1 \le k \le n \le 450)$.

Формат выходных данных

Выведите остаток от деления искомого числа способов сборки ожерелья на $10^9 + 7$.

входной файл	выходной файл
4 3 1 2 3 1	8

Задача 69. Кувшинки.

В одном очень длинном и узком пруду по кувшинкам прыгает лягушка. Кувшинки в пруду расположены в один ряд. Лягушка начинает прыгать с первой кувшинки ряда и хочет закончить на последней. Но в силу вредности характера лягушка согласна прыгать только вперед через одну или через две кувшинки. Например, с кувшинки номер 1 она может прыгнуть лишь на кувшинки номер 3 и номер 4. На некоторых кувшинках сидят комарики. А именно, на i-й кувшинке сидят a_i комаров. Когда лягушка приземляется на кувшинку, она съедает всех комариков, сидящих на ней. Лягушка хочет спланировать свой маршрут так, чтобы съесть как можно больше комаров. Помогите ей: подскажите, какое максимальное число комаров она может съесть за свое путешествие.

Формат входных данных

Первая строка входного содержит число n кувшинок в пруду $(1 \le n \le 1\,000)$. Вторая строка содержит n чисел, разделенных одиночными пробелами: i-е число сообщает, сколько комаров сидит на i-й кувшинке.

Все числа целые, неотрицательные и не превосходят 1 000.

Формат выходных данных

Выведите максимальное число комаров, которое может съесть лягушка. Если лягушка не может добраться до последней кувшинки, то выведите -1.

входной файл	выходной файл
6 1 100 3 4 1000 0	5
2 8 9	-1

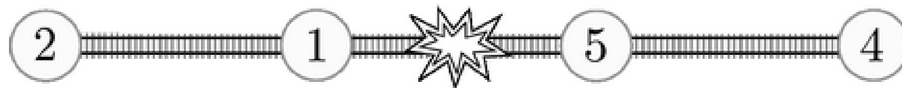
Задача 70. Захват Байтланда.

Инопланетные захватчики, атакующие Байтланд, планируют разрушить главную скоростную железнодорожную ветку страны. Ветка представляет собой цепочку из n станций, связанных между собой $n-1$ железнодорожными перегонами. Для каждой станции известен ее «уровень важности» с точки зрения всеобщего дела обороны Байтланда – число от 1 до 5. Станция ценна тем, что обеспечивает транспортную связь с другими станциями. Поэтому стратегическая значимость отдельно взятой станции считается равной нулю, а значимость цепочки станций определяется как сумма произведений «уровней важности» станций для всех пар в цепочке, связанных прямо или через другие станции.

Например, для железнодорожной сети (для станций указаны их «уровни важности») стратегическая значимость вычисляется по формуле $2\cdot 1+2\cdot 5+2\cdot 4+1\cdot 5+1\cdot 4+5\cdot 4=49$.

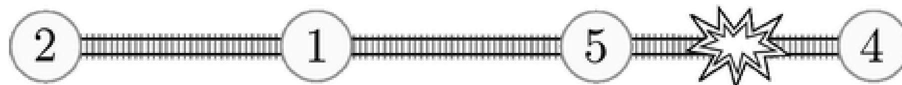
Ресурсы у захватчиков ограниченны. Инопланетяне могут атаковать только пути между станциями, но не сами станции, которые хорошо охраняются.

Посмотрим, что случится, если пришельцы разрушат железную дорогу, соединяющую вторую и третью станции.



Железнодорожная система распадется на две не связанные между собой части, и ее стратегическая значимость упадет до величины $2\cdot 1+5\cdot 4=22$.

Однако если атаковать дорогу между третьей и четвертой станциями, то можно достичь стратегической значимости $2\cdot 1+2\cdot 5+1\cdot 5=17$.



С точки зрения захватчиков так поступать выгоднее – при равных затратах достигается больший разрушительный эффект.

У пришельцев есть возможность разрушить не более m железнодорожных перегонов между станциями. Определите, какого минимального значения стратегической значимости железнодорожной системы можно с их помощью добиться.

Формат входных данных

Первая строка входного содержит целые числа n и m разделенные пробелом, – число станций $(1\le n\le 300)$ и число перегонов между станциями $(0\le m<n)$, которые могут быть разрушены.

Во второй строке записаны через пробел n целых чисел в диапазоне от 1 до 5 включительно – «уровни важности» станций в порядке их следования вдоль скоростной железной дороги.

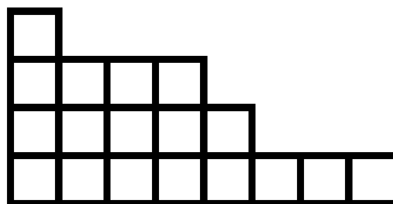
Формат выходных данных

Выведите одно число – ответ задачи.

входной файл	*выходной файл*
4 1 2 1 5 4	17
4 2 2 1 5 4	2
5 0 1 2 3 4 5	85

Задача 71. Лесенки.

Лесенкой будем называть конструкцию из кубиков, в которой каждый следующий уровень состоит из строго большего числа кубиков, чем предыдущий уровень, если считать уровни сверху вниз.



Необходимо подсчитать число лесенок, которые можно построить ровно из n кубиков.

Формат входных данных

Первая строка входного содержит число n $(1 \le n \le 500)$.

Формат выходных данных

Выведите искомое число лесенок.

входной файл	*выходной файл*
3	2

Задача 72. Ожидаемая цена.

Дерево – это неориентированный граф, в котором любые две вершины связаны ровно одной цепью.

Случайным образом выбирается одно помеченное дерево на n вершинах, равновероятно среди всех таких деревьев.

Определим стоимость дерева как

$$\min_{u=1}^{n} \sum_{v=1}^{n} d(u,v),$$

где $d(u,v)$ – число ребер в (u,v)-цепи.

Найдите математическое ожидание стоимости выбранного дерева.

Формат входных данных

Единственная строка входного файла содержит два целых числа n и m $\left(3 \le n \le 5\,000, 900\,000\,011 \le m \le 1\,000\,000\,007, \text{число } m - \text{ простое}\right)$.

Формат выходных данных

Можно показать, что ответ можно представить в виде несократимой дроби p/q, где p и q – положительные взаимнопростые числа, а $q \not\equiv 0 \pmod{m}$.

Выведите одно целое число *x*, для которого верно $x \equiv p \cdot q^{-1} \pmod{m}$ и $0 \le x < m$, где q^{-1} – число, обратное к q по модулю m.

входной файл	*выходной файл*
4 900000011	675000012
7 1000000007	363182020
4999 950000017	506366868

Рекомендации.

Точные ответы для первого и втор○го тестовых примеров: $\frac{15}{4}$, $\frac{23\,916}{2\,401}$.

Пусть ξ – дискретная случайная величина, принимающая значения $x_1, x_2, \ldots, x_k$ с вероятностями $p_1, p_2, \ldots, p_k$ соответственно. Тогда *математическое ожидание* величины ξ может быть вычислено по формуле:

$$\varepsilon\xi = \sum_{i=1}^{n} p_i \cdot x_i.$$

Число a называется *обратным к* числу b по модулю m, если $(a \cdot b) \equiv 1 \pmod{m}$.

Для простого m верно, что $a = b^{(m-2)}$. (Следствие из малой теоремы Ферма).

***Некоторые определения*.**

Пусть задан граф G, являющийся деревом.

Диаметром $d(G)$ дерева G называется длина (в ребрах) самого большого кратчайшего маршрута в дереве между двумя вершинами.

Эксцентриситетом $e(V)$ вершины v называется число $\max_{u \in V(G)} d(u, v)$, где $V(G)$ – множество вершин дерева, то есть расстояние между v и наиболее удаленной от нее вершиной.

Радиусом дерева $r(G)$ называется минимальный из эксцентриситетов его вершин.

Центральной вершиной называют вершину v, для которой $e(V) = r(G)$.

Центром дерева называют множество всех центральных вершин.

Центроидом дерева называется вершина, при выборе которой в качестве корня порядок поддерева каждого ее сына на превосходит $\left\lfloor \frac{n}{2} \right\rfloor$, где n – число вершин в дереве.

Задача 73. Разноцветный поезд.

В городе готовится к открытию новая линия метро. Платформы станций на ней будут значительно длиннее стандартных. По задумке градоначальника первым по линии должен проехать торжественный разноцветный поезд. Поэтому глава города передал работникам вагонного депо схему окраски вагонов в порядке от головы состава.

На двух запасных путях в депо уже стоит необходимое число вагонов. Каждый из них окрашен в один из цветов. Цвет обозначается натуральным числом, не превосходящим 1 000 000.

Рассмотрим рисунок 2.15.

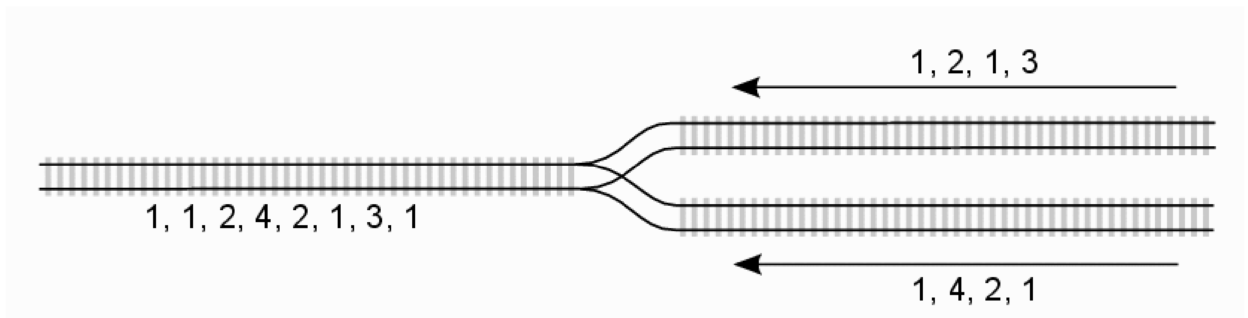


Рисунок 2.15 – Схема окраски вагонов.

Справа находятся два запасных пути, на каждом из которых слева направо расположены вагоны. Работники депо могут за одну операцию прицепить в хвост поезда либо самый левый из находящихся на первом пути вагонов (если такие есть), либо самый левый из находящихся на втором пути. Они повторяют эти операции до тех пор, пока ни на одном из запасных путей не останется вагонов.

И, естественно, у работников депо возник вопрос, можно ли построить описанным выше образом поезд, соответствующий цветовой схеме, которую им передал градоначальник.

В примере, приведенном рисунке 2.15, мы можем получить поезд 1, 1, 2, 4, 2, 1, 3, 1 или поезд вида 1, 2, 1, 1, 4, 2, 1, 3, но не можем получить поезд вида 1, 2, 1, 2, 1, 2, 3, 4 или поезд вида 1, 1, 1, 2, 1, 2, 3, 4.

Формат входных данных

Первая строка входного содержит два натуральных числа n_1 и n_2 – число вагонов на первом и втором запасном пути соответственно $\left(1 \le n_1, n_2 \le 3000\right)$. Вторая строка содержит n_1 натуральных чисел, не превосходящих 10^6, которые описывают цвета вагонов на первом пути в порядке слева направо. Третья строка содержит n_2 натуральных чисел, не превосходящих 10^6, – цвета вагонов на

втором пути в порядке слева направо. Четвертая строка содержит $n_1 + n_2$ натуральных чисел, описывающих цвета вагонов нового поезда по плану градоначальника (в аналогичном формате).

Формат выходных данных

Выведите строку `possible`, если возможно построить поезд в соответствии с цветовой схемой, переданной градоначальником, или `not possible` в противном случае.

входной файл	*выходной файл*
4 4 1 2 1 3 1 4 2 1 1 1 2 4 2 1 3 1	possible
4 4 1 2 1 3 1 4 2 1 1 1 1 2 1 2 3 4	not possible

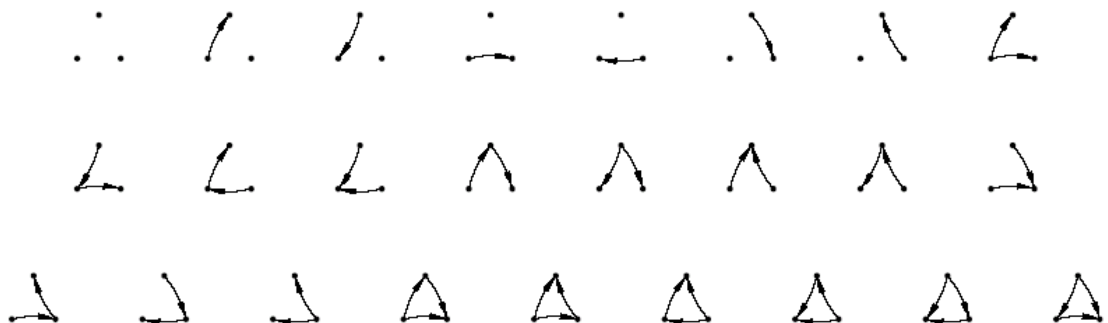
Задача 74. Промежуточная позиция.

Вася Пупкин и Петя Васечкин играют в следующую игру: на листе бумаги есть n точек. Они нумеруются от 1 до n. Участники игры поочередно рисуют по одной стрелке между любыми двумя точками. При этом стрелка от x к y может быть проведена только при выполнении двух условий:

– еще нет стрелок от x к y;

– двигаясь по стрелкам, из y нельзя дойти к x.

Проигрывает тот, кто не имеет возможности сделать ход. Вася Пупкин для выработки выигрышной стратегии захотел узнать, сколько всего различных позиций может появиться на листе бумаги в процессе игры. Однако сам подсчитать это число он не смог. Вам придется ему помочь.



Формат входных данных

Первая строка входного содержит число n точек $(1 \le n \le 100)$.

Формат выходных данных

Выведите искомое число позиций.

входной файл	*выходной файл*
3	25

Задача 75. Графы с мостами.

Найдите число помеченных связных графов на n вершинах, не более чем с b мостами.

Формат входных данных

В единственной строке входных данных записаны два целых числа n и b $\left(2 \le n \le 50, 0 \le b \le n \cdot (n-1)/2\right)$.

Формат выходных данных

Выведите искомое число графов, взятое по модулю 1 000 000 007.

входной файл	*выходной файл*
`2 1`	`1`
`4 0`	`10`
`4 1`	`22`

Задача 76. Карточная игра.

Вам, конечно же, известно, что в сложных карточных играх (таких, например, как преферанс) вероятность выигрыша зависит не только от умений и навыков игрока, но и от выпавшего расклада карт.

Карточная игра, в которой участвует n игроков, состоит из нескольких туров, в каждом туре карты сдаются по-новому. *Сила руки* i-го игрока $(1 \le i \le n)$ в отдельном туре равна значению непрерывной случайной величины, равномерно распределенной на интервале $[a_i, b_i]$. Тур выигрывает игрок, у которого сила руки, определенная описанным случайным образом, будет наибольшей. Если наибольшая сила окажется у нескольких игроков, в туре фиксируется ничья.

Определите вероятность победы в туре для каждого игрока.

Формат входных данных

Первая строка содержит целое число n $(2 \le n \le 300)$. Каждая из последующих n строк содержит два целых числа a_i и b_i – границы интервала для силы руки каждого игрока$\left(0 \le a_i < b_i \le 10^9\right)$.

Формат выходных данных

Выведите n строк, i-я из которых содержит одно действительное число – вероятность победы в туре i-го игрока. Абсолютная погрешность не должна превосходить10^{-9}.

входной файл	*выходной файл*

`5`
`0 5`
`1 5`
`2 5`
`3 5`
`4 5`
	`0.069722222222`
	`0.088472222222`
	`0.121111111111`
	`0.192638888889`
	`0.528055555556`

77. Шаблон и слово.

Будем рассматривать слова из больших латинских букв и шаблоны, состоящие из больших латинских букв и символов «?» и «*».

Говорят, что слово подходит под шаблон, если в шаблоне можно заменить каждый символ «?» на большую латинскую букву, а каждый символ «*» – на последовательность (возможно, пустую) больших латинских букв, так, чтобы получилось требуемое слово.

Напишите программу, которая определит, подходит ли слово под шаблон.

Формат входных данных

В первых двух строках записаны шаблон и слово:

– в одной строке записан шаблон – последовательность больших латинских букв «?» и «*»,

– в другой – слово, состоящее только из больших латинских букв (строки короче 256 символов).

Формат выходных данных

Выведите YES, если слово подходит, или NO, если нет.

входной файл	*выходной файл*
ABBCDA A*CDA	YES

78. BitTorrent.

BitTorrent – пиринговый (peer-to-peer) сетевой протокол для передачи больших файлов, в котором узлы действуют и как клиенты и как серверы, в отличие от централизованной клиент-серверной архитектуры, где клиентские узлы запрашивают ресурсы у центрального сервера.

Действительно, протокол BitTorrent позволяет группе хостов одновременно раздавать и получать друг у друга файлы, что снижает нагрузку и зависимость от каждого клиента-источника и обеспечивает избыточность данных.

А именно, раздача (так называемый торрент) включает в себя набор файлов, разбитых на фрагменты, как показано на рисунке 2.16.

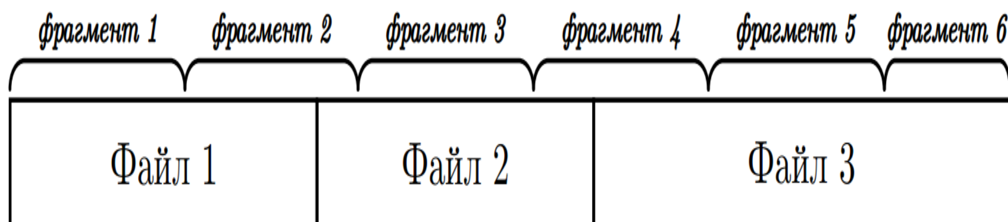


Рисунок 2.16 – набор файлов (торент).

Например, пакет файлов общим размером в 10 МиБ может быть разделен на десять фрагментов размера 1 МиБ или на сорок фрагментов по 256 КиБ.

Как только хост (так называемый пир) получает новый фрагмент торрента, он становится источником этого фрагмента для других хостов, которым также необходим этот фрагмент.

Как правило, фрагменты загружаются непоследовательно и переставляются в правильном порядке самими пирами.

Каждый хост самостоятельно управляет тем, какие фрагменты должны быть загружены.

Фрагменты имеют одинаковый размер в рамках одного торрента, за исключением последнего, который может иметь меньший размер.

Фрагменты скачиваются и закачиваются только целиком.

Вы хотите скачать один торрент, но приближается ваш месячный лимит входящего интернет-трафика, и на скачивание всех файлов остатка может не хватить.

Тем не менее, вы не желаете ждать до следующего месяца и хотите заполучить хотя бы некоторые файлы из торрента.

Какое наибольшее число отдельных файлов можно скачать, уложившись в лимит?

Формат входных данных

В первой строке через пробел записано три целых числа N, P и L, где N – количество файлов в торренте $\left(1 \le N \le 3\,000\right)$, P – размер фрагмента в КиБ $\left(1 \le P \le 1\,000\right)$, L – месячный остаток интернет-трафика в КиБ $\left(1 \le L \le 1\,000\,000\right)$.

Во второй строке через пробел записано N чисел $S_1, S_2, \ldots, S_N$, где S_i – размер i- го файла в КиБ $\left(1 \le S_i \le 100\,000\right)$.

Файлы перечислены в порядке следования в торренте.

Формат выходных данных

Выведите одно число — максимальное количество файлов, которые можно скачать.

входной файл	*выходной файл*
3 3 13 5 5 7	2
7 2 16 6 11 3 3 8 1 8	4

Замечание.

Вместо десятичных приставок кило-, мега- и т. д., обозначающих $10^3, 10^6$ и т. д. единиц, международная электротехническая комиссия в марте 1999 года предложила использовать двоичные приставки кеби-, меби- и т. д., обозначающие 2^{10}, 2^{20} и т. д. единиц.

79. Система оценивания.

Сегодня 18 февраля 2099 года. Вы только что пришли на первое алгоритмическое занятие на своем факультете и с ужасом обнаружили, что за следующие несколько месяцев вам придется не только решать задачи, но и делать это быстро! Не улучшает ситуацию и то, что система оценивания очень запутанная и непонятная: за решение одной и той же задачи в разные моменты времени дается кардинально различное количество баллов (но если задача сдана чуть позже, то баллов за нее больше не дадут – и на том спасибо, не нужно еще больше ломать голову!).

Вы уже получили список из n задач на семестр. Задачи достаточно старые, так что у вас уже есть информация от предыдущих курсов, сколько времени требуется на решение каждой из задач: i-я задача требует t_i минут. Зная систему оценивания и точную длительность семестра, найдите, какое максимальное количество баллов вы можете получить.

Задачи можно сдавать в любом порядке. Вы можете распределять всю длительность семестра между решением любой из задач и отдыхом как угодно (одновременно можно решать не более одной задачи), при условии, что до момента сдачи задачи на нее затрачено не менее t_i минут. Сдача задачи происходит мгновенно, каждую задачу можно сдавать не более одного раза. Вы не обязаны сдавать все задачи.

Формат входных данных

Первая строка входных данных содержит два числа n и T $\left(1 \le n \le 20, 1 \le T, T \cdot n \le 10^6\right)$ – количество доступных задач и длительность семестра в минутах.

Следующая строка содержит n чисел $t_1, t_2, \ldots, t_n$ $\left(1 \le t_i \le 10^6\right)$ – время в минутах, которое необходимо потратить на каждую из задач для ее решения.

Остальные строки содержат описание системы оценивания.

i-я из этих строк содержит n целых чисел $p_{i,1}, p_{i,2}, \ldots, p_{i,n}$ $\left(1 \le p_{i,j} \le 10^7\right)$, что означает, что сдача j-й задачи после i минут семестра дает $p_{i,j}$ баллов.

Гарантируется, что $p_{i,j} > p_{i+1,j}$ для всех допустимых значений i и j.

Формат выходных данных

Выведите одно число – максимальное количество баллов, которое возможно получить.

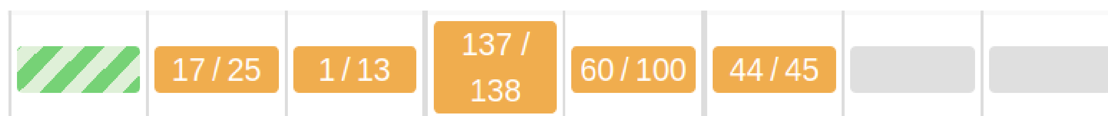
входной файл	*выходной файл*
2 2 1 1 11 12 2 1	14

80. WA 138

Чтобы получить зачет, надо решать задачи. Много. Задачи я дам. Чтобы получить ОК по задаче, нужно починить решение на одном тесте. Сто тридцать восьмом. Тест я не дам. Скажите спасибо, что еще один тест не добавил.

WA 138 Solution 846513

Result Wrong Answer (138)
Score 137 of 138



Будучи в отчаянии перед надвигающимся дедлайном, вы посылаете много решений в систему с одной целью – с помощью `assert(x >= 998244353)`, различных констант и бинпоиска хоть как-то узнать единственное целое число из входных данных.

Но вот у вас осталась одна попытка до дедлайна, а пока удалось только узнать, что число лежит на отрезке $[L, R]$…

Вы подозреваете, что для того, чтобы ваше решение упало, искомое число x из входных данных должно обладать следующим свойством: если взять такое число y, запись которого в a-ичной системе счисления выглядит так же, как десятичная запись x, то число $y - x$ должно делиться на b. Кроме этого, сумма цифр в десятичной записи числа x должна быть равна c.

Теперь вас интересует, сколько таких чисел x существует на отрезке $[L, R]$. Если таких не очень много, у вас останется надежда найти баг до дедлайна...

Формат входных данных

Первая строка входных данных содержит два целых числа L, R $\left(0 \le L \le R \le 10^{18}\right)$.

Вторая строка содержит три целых числа a, b, c $(11 \le a \le 99, 2 \le b \le 400, 0 \le c \le 200)$.

Формат выходных данных

Выведите одно целое число – количество целых чисел x, удовлетворяющих условию задачи.

входной файл	*выходной файл*
`8 108 11 2 9`	`5`
`998 244353` `19 16 24`	`895`

81. Ящерицы-филологи.

Как-то раз группа ящериц-филологов решила исследовать необычное поселение, найденное ими почти случайно – молодой ученый Ящеркович провалился по землю из-за обрушения грунта (к счастью, не очень глубоко) во время раскопок, проводившихся ящерицами-археологами, и наткнулся на большой подземный город.

Ящеркович предположил, что это не что иное, как поселение древних пикачу. На табличке даже было написано название города, но вот незадача – в зашифрованном виде.

Из многочисленных древних источников известно, что пикачу представляли все названия городов в виде пары чисел (N, K). Чтобы восстановить название города, достаточно найти K-е слово из N слогов в лексикографическом порядке.

Хорошо, что старший историк Ящерный неплохо знаком с основами языка древних пикачу. Так как все покемоны умеют произносить только свое имя, то любое их слово состоит из трех слогов: «pi», «ka» и «chu».

Однако не все так просто. Язык древних покемонов отличается от языка современных, поскольку, например, ни в одну слове древних пикачу не могли встретиться подстроки «pipipi», «kakaka» и «chukapi».

Помогите Ящерковичу узнать название древнего города.

Формат входных данных

На вход подаются два целых положительных числа $N \le 40$ и $K \le 2 \cdot 10^{18}$ – количество слогов в слове и его лексикографический порядок соответственно.

Гарантируется, что такое слово существует.

Формат выходных данных

Выведите одну строку – ответ на задачу.

входной файл	*выходной файл*
3 20	pikachu
3 22	pikapi
5 155	pikapikachu

Замечание.

В первом примере город в переводе называется «Пикачу», что несколько примитивно, но в то же дело практично, так как сразу понятно, кто в нем обитает.

Во втором примере название города трудно перевести, однако известно, например, что в аниме «Покемон» этим словом Пикачу называл своего тренера.

В третьем примере город называется «Меня зовут пикачу», что является несколько странным выбором для названия.

82. Орган Хаммонда.

Александр хочет научиться играть на органе Хаммонда.

Он уже нашел мелодию, которую будет играть.

Для простоты он выписал последовательность нот A длины N из целых чисел, которые означают номер клавиш: чем больше номер, тем правее клавиша на клавиатуре.

Александр очень умный и понимает, что самое важное – правильная аппликатура, то есть правильный выбор, каким пальцем какую ноту играть.

Если выбрать неудобные пальцы, то потом можно потратить уйму времени на попытки научиться играть мелодию и все равно в итоге не преуспеть.

Обозначим пальцы на руке числами от 1 до 5.

Назовем аппликатурой любую последовательность B номеров пальцев.

Назовем аппликатуру удобной, если для любого $1 \le i \le N-1$ выполнено следующее:

- Если $A_i < A_{i+1}$, то $B_i < B_{i+1}$.
- Если $A_i = A_{i+1}$, то $B_i \ne B_{i+1}$.
- Если $A_i > A_{i+1}$, то $B_i > B_{i+1}$.

Найдите любую удобную аппликатуру.

Гарантируется, что она существует.

Формат входных данных

На вход в первой строке подается одно целое положительное число

$$N \le 10^5.$$

Далее во второй строке следуют N целых положительных чисел

$$A_i \le 2 \cdot 10^5.$$

Формат выходных данных

Выведите любую удобную аппликатуру.

входной файл	*выходной файл*
5 2 2 5 3 3	5 4 5 4 1
7 1 6 7 8 10 4 1	1 2 3 4 5 4 1
15 5 7 5 4 1 3 1 3 1 3 1 3 1 3 1	4 5 4 3 2 5 4 5 4 5 4 5 4 5 1

83. Сигнатура перестановки.

Студент Вася всерьез увлекся комбинаторикой и планирует в будущем защитить кандидатскую диссертацию по теме, связанной с перестановками. Напомним, что, упрощенно говоря и не вдаваясь в теорию групп, перестановкой длины n называется упорядоченный набор $(a_1, a_2, \ldots, a_n)$, составленный из чисел $1, 2, \ldots, \ldots n$ без повторений. Как обычно бывает в математической науке, если придумать какое-то новое (пусть никому и не нужное) понятие, то затем можно опубликовать в рецензируемых журналах несколько статей, посвященных свойствам этого понятия и его связи с другими ранее придуманными (и столь же полезными) понятиями. Вася придумал такую штуку, как сигнатура перестановки. Вот как она строится.

Для перестановки длины n сигнатура является строкой, состоящей из $n-1$ символа. Каждый символ в последовательности характеризует возрастание или убывание значения при движении вдоль перестановки.

Так, если на i-м месте стоит `U` (от up – вверх), то $a_i < a_{i+1}$. Если же на i-м месте записан символ `D` (от down – вниз), то $a_i > a_{i+1}$. Возможен также символ `?` – он говорит о том, что соотношение между a_i и a_{i+1} неизвестно.

Например, перестановка $(3, 1, 2, 7, 4, 6, 3, 5)$ подходит под сигнатуры `DUUDUD`, `?U?D?D` или `??????`.

Научный интерес Васи заключается в том, чтобы по заданной сигнатуре определить, сколько перестановок ей удовлетворяют. Поможете молодому ученому справиться с этой задачей?

Формат входных данных

На входе задано t строк $(1 \le t \le 100)$, в каждой строке записана одна сигнатура перестановки. Длина каждой сигнатуры – от 1 до 1 000 символов.

Формат выходных данных

Выведите ровно t строк: для каждой сигнатуры укажите число перестановок, которые подходят под эту сигнатуру, по модулю $10^9 + 7$.

входной файл	*выходной файл*
UU UD DU DD ?D ??	1 2 2 1 3 6

Замечание.

Перестановка $(1, 2, 3)$ имеет сигнатуру UU. Перестановки $(1, 3, 2)$, $(2, 3, 1)$ имеют сигнатуру UD. Перестановки $(3, 1, 2)$, $(2, 1, 3)$ имеют сигнатуру DU. Перестановка $(3, 2, 1)$дает сигнатуру DD. Сигнатура `?D` может значить или `UD`, или `DD`. Строка `??` допускает все перестановки длины 3.

83. Забор.

Представьте бесконечный забор, состоящий из пронумерованных натуральными числами дощечек белого цвета, и робота iBrush, который каждый день может некоторые из этих дощечек перекрашивать в черный цвет.

Изначально робот находится возле дощечки с номером 1 и уже покрасил ее в черный цвет. iBrush может за один день сделать не более *d* перемещений (одно перемещение – сдвинуться к соседней дощечке слева, если такая есть, или справа), при этом робот может перекрасить дощечку в черный цвет, а может и не перекрашивать. Одно «но»: в конце дня робот обязательно красит дощечку, даже если она уже была черная.

Определите общее число возможных финальных раскрасок забора после *n* дней.

Формат входных данных

В единственной строке входного файла записаны два целых числа *n* и *d* $(1 \le n, d \le 200)$.

Формат выходных данных

Выведите общее число финальных раскрасок по модулю 998 244 353.

входной файл	*выходной файл*
3 1	4
1 3	8
2 2	11
3 3	224

85. Уравнение.

Дано *n* положительных целых чисел $a_1, a_2, \ldots, a_n$. Найдите число решений уравнения $a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_n \cdot x_n = m$ в положительных числах $x_1, x_2, \ldots, x_n$. Так как искомое число решений может быть слишком большим, найдите его остаток от деления на 998 244 353.

Формат входных данных

В первой строке входного файла через пробел записаны два целых числа *n* и *m* $\left(1 \le n \le 100, 0 \le m \le 2^{63} - 1\right)$.

Во втрой строке записаны *n* целых положительных чисел $a_1, a_2, \ldots, a_n$ $\left(\sum_{i=1}^{n} a_i \le 100\right)$.

Формат выходных данных

Выведите остаток от деления искомого числа решений уравнения на 998 244 353.

входной файл	*выходной файл*
4 21 1 2 3 4	27
4 2019 21 9 20 19	18142

86. Компьютерная игра.

Вася очень любит играть в компьютерные игры, а родители этому не очень рады...

Как известно, папа у Васи силен в математике. А за последние несколько лет он еще и в программировании набил руку. Папа написал для Васи развивающую компьютерную игру, чтобы сын играл с пользой для ума.

В этой игре необходимо пройти все уровни, но можно это делать не только в последовательном порядке. Игра начинается на первом уровне, а заканчивается тогда, когда игрок оказывается на последнем уровне и все остальные уровни он уже прошел.

Игрок может оказаться на каждом уровне только один раз, а на последнем – когда прошел все остальные.

Переход между уровнями сопровождается дополнительными загадками. Если игрок переходит с уровня i на уровень j, то правильное решение загадки принесет ему $|i-j|$ бонусных очков.

Вася решает все загадки правильно, но играть интереснее, когда каждый раз уровни проходишь в разном порядке.

Сегодня он решил пройти уровни в таком порядке, чтобы количество набранных бонусных очков было в промежутке от A до B включительно.

Помогите ему определить, сколько существует различных способов выбрать порядок уровней, пройти их в этом порядке и набрать подходящее количество очков.

Формат входных данных

В единственной строке входного файла записаны три целых числа n, A и B $(2 \le n \le 30, 1 \le A \le B \le 1000)$.

Формат выходных данных

В единственной строке выведите количество подходящих способов пройти уровни игры.

Ответ может быть достаточно большим, поэтому выведите его по модулю 1 000 002 017.

входной файл	*выходной файл*
`3 2 4`	`1`
`5 4 6`	`3`
`2 2`	`11`
`10 9 9`	`1`

87. Бинарное дерево поиска.

У вас есть четыре массива a, b, c, d из n элементов.

Пусть T – некоторое бинарное дерево поиска на n вершинах с ключами $1, 2, \ldots, n$.

Пусть $f_T(i, j)$ – величина, равная 1, если $i \neq j$ и вершина с ключом i – предок вершины с ключом j в дереве T, и 0 иначе.

Определим *значение* дерева T следующим образом:

$$value(T) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(a_i \cdot b_i \cdot f_T(j, i) + c_i \cdot d_i \cdot f_T(i, j)\right).$$

Найдите максимальное значение, которое может иметь бинарное дерево поиска на n вершинах с ключами $1, 2, \ldots, n$.

Формат входных данных

Первая строка входного файла содержит одно число n $(1 \le n \le 300)$.

Следующие четыре строки содержат описание массивов a, b, c, d соответственно. Каждое описание содержит n целых чисел $x_1, x_2, \ldots, x_n$ $(-40 \le x_i \le 40)$ элементы соответствующего массива.

Формат выходных данных

Выведите одно целое число – максимальное значение $value(T)$ по всем бинарным деревьям поиска T на n вершинах с ключами $1, 2, \ldots, n$.

входной файл	*выходной файл*
1 0 1 0 1	0
4 1 2 3 4 2 3 4 1 3 4 2 1 4 1 2 3	41
7 1 4 1 -1 1 4 1 1 8 1 40 1 8 1 1 8 1 40 1 8 1 1 4 1 -1 1 4 1	512

Замечание.

Прямые левые обходы для некоторых оптимальных деревьев имеют вид:

для второго примера: 1, 2, 3,4;

для третьего примера: 4, 2, 1, 3, 6, 5, 7.

3. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

3.1. Набор тестовых заданий для проверки теоретических знаний

По всем разделам учебной дисциплины Алгоритмы и структуры данных» для проверки теоретических знаний в iRunner разработаны тестовые задания.

Тест может включать задания разных видов: вопросы, где нужно выбрать один правильный ответ из предложенных; вопросы, где правильных ответов несколько и требуется отметить множество вариантов; вопросы, где нужно ввести текстовый ответ.

Отвечать на вопросы можно в любом порядке, все ответы автоматически сохраняются. Система ведет учет времени.

Отличительной особенностью iRunner является использование автоматизированных генераторов. Это позволяет создать большую базу вопросов для тестов. Каждый студент получает уникальный набор заданий, тем самым исключается возможность списывания.

Студенты в Образовательной платформе iRunner на каждом практическом занятии выполняют тестовые задания, преподаватель проводит разбор допущенных ошибок.

В рамках самоконтроля студенты имеют возможность выполнять тестовые задания в удобное для них время из дома/общежития, видеть правильно или нет они ответили, а также ознакомиться с правильными вариантами ответов.

Учебная дисциплина «Алгоритмы и структуры данных» завершается прохождением итогового теста, который, как правило, содержит 20 вопросов по основным разделам дисциплины и рассчитан на 30 минут. При прохождении итогового теста студентам не разрешено пользоваться вспомогательными материалами.

Примеры тестовых заданий.

1. Выберите верные утверждения:

- ☒ $n \cdot \log n = \Omega(n)$
- ☐ $n! = \Theta(2^n)$
- ☒ $n = \mathrm{O}(n)$
- ☐ $n^2 = \Omega(2^{42})$
- ☐ $n^2 = \mathrm{O}(n \cdot \log n)$
- ☐ нет верных утверждений

2. Решить рекуррентное уравнение методом итераций:

$$\begin{cases} T(n) = 2 \cdot T\left(\dfrac{n}{2}\right) + 100 \cdot n^2,\ n = 2^k,\ k \geq 1; \\ T(1) = 7. \end{cases}$$

Решение.
Решим рекуррентное уравнение методом итераций.

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 100 \cdot n^2 = \left[T\left(\frac{n}{2}\right) = 2 \cdot T\left(\frac{n}{2^2}\right) + 100 \cdot \left(\frac{n}{2}\right)^2 \right] =$$

$$= 2^2 \cdot T\left(\frac{n}{2^2}\right) + 100 \cdot \frac{n^2}{2^1} + 100 \cdot \frac{n^2}{2^0} = \ldots =$$

$$= 2^m \cdot T\left(\frac{n}{2^m}\right) + 100 \cdot n^2 \cdot \left(\frac{1}{2^0} + \frac{1}{2^1} + \ldots + \frac{1}{2^{m-1}}\right) =$$

$$= 2^m \cdot T\left(\frac{n}{2^m}\right) + 100 \cdot n^2 \cdot 2 \cdot \left(1 - \frac{1}{2^m}\right) = \left[\frac{n}{2^m} = 1,\ n = 2^m\right]$$

$$= n \cdot T(1) + 100 \cdot n^2 \cdot 2 \cdot \left(1 - \frac{1}{n}\right) =$$

$$= 7 \cdot n + 200 \cdot n^2 - 200 \cdot n = 200 \cdot n^2 - 193 \cdot n$$

3. Заданы *n* элементов. Справедливы ли утверждения:

– время работы алгоритма сортировки выбором в худшем случае есть $O(n \cdot \log n)$?

– время работы алгоритма дихотомического поиска элемента в упорядоченном массиве в худшем случае есть $O(\log n)$)?

Варианты ответов:

☐ справедливы оба утверждения

☐ оба утверждения неверны

☐ справедливо только первое утверждение

☒ справедливо только второе утверждение

3.2. Набор тестов для проверки работоспособности программ

В рамках учебной дисциплины «Алгоритмы и структуры данных» студенты решают алгоритмические задачи. Решением задачи является программа, написанная на некотором высокоуровневом языке программирования.

Задачи подразбиты на четыре темы, которые покрывают все разделы учебной дисциплины:

1) рекуррентные соотношения;
2) деревья поиска;
3) структуры данных и графы;
4) алгоритмы на графах;
5) строковые алгоритмы.

Набор разнообразных задач, которые предлагаются для решения в iRunner, формируется преподавателями и постоянно расширяется.

В настоящий момент в рамках учебной дисциплины «Алгоритмы и структуры данных» в системе iRunner разработано более 350 задач разного уровня сложности.

Задачи делятся на две категории: общие и индивидуальные.

Общие задачи выполняют все учащиеся (в рамках учебной дисциплины каждый учащийся должен выполнить не менее 30 общих задач). Алгоритмы решения общих задач подробно разбираются на лекциях, обосновывается корректность алгоритмов, приводятся псевдокоды.

Цель выполнения общих задач – отработать базовые алгоритмические навыки. Считается, что любой выпускник ФПМИ должен уметь реализовывать основные алгоритмы.

Индивидуальные задачи требуют творческого подхода, алгоритмических и программистских навыков, а также развивают креативное мышление.

Каждый студент в группе по каждой теме получает 1–2 индивидуальные задачи, уровень сложности которой от 4 до 10 баллов (студент может попросить преподавателя назначить ему дополнительные задачи, чтобы повысить свой уровень алгоритмической подготовки). Студенты могут по каждой теме заявить уровень сложности, на который они хотят решать задачи.

Все задачи хранятся в системе iRunner. Каждая задача состоит из двух важнейших частей: из текста условия и набора тестов.

Условия задач в системе по традиции имеют единую структуру и стиль оформления.

В условие обычно входят следующие элементы:

– название задачи;

– уровень сложности по 10-бальной шкале;

– ограничения по времени и памяти, в которые должна уложиться программа;

– легенда (сказка) или формальная постановка;

– описание формата входных данных и ограничения на входные данные;

– описание формата выходных данных;

– примеры (примеры из условия задачи являются первыми тестами при тестировании и доступны студентам для просмотра в процессе тестирования).

Студенты могут читать условия всех задач, находящихся в системе.

Тесты к задаче. Под тестом для задачи понимается набор входных и выходных данных. Тесты могут выглядеть следующим образом:

Тест №1: вход 1, выход 1

Тест №2: вход 2, выход 2

Тест №3: вход 3, выход 5

…

Тест №16: вход 10000000, выход 640540120

Тесты составляются преподавателями. Студентам содержимое тестов недоступно (кроме примеров из условия). Чем больше тестов создано для задачи, тем больше неправильных решений этими тестами отсекается.

Наличие заготовленных тестов позволяет автоматически проверить правильность решения. Для этого нужно запускать программу, передавая ей входные данные, и сравнивать тот ответ, который выдает программа, с правильным ответом, который хранится в базе. Эту работу и делает Образовательная платформа iRunner.

Всего на сегодняшний день по всем задачам в системе хранится более 6,5 тыс. тестов.

3.3. Средства диагностики

Студент пишет программу, которая решает поставленную перед ним задачу. Затем он отправляет свой код в iRunner через специальную форму. На сервере решение проверяется на наборе тестов. Через некоторое время (обычно несколько секунд) студенту выдается вердикт. Например, «Неправильный ответ (5)» означает, что на первых четырех тестах программа отработала верно, а на пятом тесте выдала неверный ответ. Вердикт «Принято» значит, что решение прошло все тесты успешно. Студент может ознакомиться с протоколом тестирования.

Образовательная платформа при этом предоставляет дружественный интерфейс преподавателю курса для назначения студентам задач (индивидуальных, общих, дополнительных, штрафных), установки крайних сроков выполнения задания (для планирования равномерной работы студентов в семестре), отслеживания прогресса, расчета итоговых оценок по курсу (Журнал, Ведомость).

В системе есть и такие функционалы, как «Сообщения» (система обмена сообщениями между преподавателем и студентами), «Электронная очередь»

(позволяет организовывать ход практического занятия: студенты по мере готовности записываются в очередь для беседы путем нажатия кнопки, а преподаватель на паре беседует со студентами в порядке очереди; автоматически собирается статистика о том, когда студент работал с преподавателем).

Преподаватель также имеет возможность задать компиляторы, на которых студент может выполнять задания C# 8 (.NET Core 3.1), GNU C++17 7.3.0 (MinGW), GNU C++20 11.2.0 (MinGW) x64, Java 8, Kotlin 1.5, Microsoft Visual C++ 15.9 (2017), Microsoft Visual C++ 15.9 (2017) x64, PyPy3.6 v7.2.0, Python 3.8 + NumPy и др.).

С целью предотвращения списывания в системе iRunner функционирует *модуль проверки решений на плагиат*, результаты работы которого доступны только преподавателю. Для каждого нового решения, которое прошло тесты, система ищет похожие решения в своей базе. Похожесть оценивается в процентах. Преподаватель затем окончательно решает, плагиат это или нет, и определяет способ наказания для студента (например, назначить штрафную задачу по данной теме).

Компьютерная система не заменяет преподавателя, а помогает ему: автоматизируется рутинная работа по проверке решений, тем самым у преподавателя появляется возможность уделять больше внимания вопросам алгоритмизации и проводить занятия в виде эвристических диалогов. Защита студентом алгоритма решения индивидуальной задачи проводится в вопросно-ответной форме: студенту даются наводящие, уточняющие вопросы, которые вытекают один из другого и, в совокупности, приводят учащегося к разработке эффективного алгоритма.

Система iRunner также используется для организации соревнований по программированию и для проведения тренировок. Знания, полученные в рамках учебных дисциплин по теории алгоритмов, играют ключевую роль при решении олимпиадных задач. Соревнуясь, студенты учатся придумывать сложные алгоритмы и писать программы без ошибок в условиях стресса и ограниченного времени.

Образовательная платформа iRunner функционирует в рамках учебной дисциплины с 2003 года и доказала свою эффективность на практике. Возможность круглосуточной самостоятельной работы студентов на базе образовательной платформы iRunner способствует получению высоких результатов на соревнованиях в области алгоритмизации и спортивного программирования и повышению позиции БГУ в мировых рейтингах.

Результатом внедрения платформы в учебный процесс стала повышенная заинтересованность в практических занятиях по учебной дисциплине «Алгоритмы и структуры данных» как со стороны учащихся, так и со стороны преподавателей [8, 11].

4. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

4.1. Рекомендуемая литература

Основная

1. Сборник задач по теории алгоритмов. Структуры данных : учеб.-метод. пособие / С. А. Соболь [и др.] – Минск : БГУ, 2020. – 159 с.

2. Котов, В. М. Теория алгоритмов. Организация перебора и приближенные алгоритмы : учеб.-метод. пособие / В. М. Котов, Е. П. Соболевская, Г. П. Волчкова. – Минск: БГУ, 2022. – 151 с.

3. Тюкачев, Н. А. C#. Алгоритмы и структуры данных / Н. А. Тюкачев, В. Г. Хлебостроев. – 6-е изд., стер. – Санкт-Петербург : Лань, 2023. – 232 с. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/346067.

Дополнительная

4. Абрамов, С.А. Лекции о сложности алгоритмов / С. А. Абрамов – Москва : МЦНМО, 2009. – 256 с.

5. Алгоритмы: построение и анализ / Т. Кормен [и др.] – Москва : Вильямс, 2005. – 1296 с.

6. Котов, В. М. Алгоритмы и структуры данных : учеб. пособие / В. М. Котов, Е. П. Соболевская, А. А. Толстиков. – Минск : БГУ, 2011. – 267 с.

7. Макконел, Дж. Анализ алгоритмов. Вводный курс / Дж. Макконел – Москва : Техносфера, 2002. – 304 с.

8. Опыт использования образовательной платформы Insight Runner на факультете прикладной математики и информатики Белорусского государственного университета. Роль университетского образования и науки в современном обществе : материалы междунар. науч. конф., Минск, 26–27 февр. 2019 г. / редкол.: А. Д. Король (пред.) [и др.]. – Минск : БГУ, 2019. – С. 263 – 267.

9. Сборник задач по теории алгоритмов : учеб.-метод. пособие / В. М. Котов [и др.] – Минск : БГУ, 2017. – 183 с.

10. Сборник задач по теории алгоритмов. Организация перебора и приближенные алгоритмы : электронный учебно-методический комплекс для специальности: 1-31 03 04 «Информатика» / В. М. Котов, Е. П. Соболевская, Г. П. Волчкова ; БГУ, Фак. прикладной математики и информатики, Каф. дискретной математики и алгоритмики. – Минск : БГУ, 2021. – 144 с. : ил. – Библиогр.: с. 143–144.

11. Соболь, С. А. Методика преподавания дисциплин по теории алгоритмов с использованием образовательной платформы iRUNNER / С. А. Соболь,

В. М. Котов, Е. П. Соболевская // Электронный науч.-методич. Журнал «Педагогика информатики». – 2020 – № 2.

12. Теория алгоритмов : учеб. пособие / П. А. Иржавский [и др.] – Минск : БГУ, 2013. – 159 с.

4.2. Электронные ресурсы

1. Образовательный портал БГУ [Электронный ресурс]. – Режим доступа: https://edufpmi.bsu.by/course/view.php?id=96. – Дата доступа: 02.09.2023.

2. Образовательная платформа Insight Runner [Электронный ресурс]. – Режим доступа: https://acm.bsu.by. – Дата доступа: 02.09.2023.

3. Котов, В. М. Алгоритмы и структуры данных : учеб. пособие / В. М. Котов, Е. П. Соболевская, А. А. Толстиков. – Минск : БГУ, 2011. – 267 с. – (Классическое университетское издание). [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/8522. – Дата доступа: 02.09.2023.

4. Теория алгоритмов : учеб. пособие / П. А. Иржавский [и др.] – Минск : БГУ, 2013. – 159 с. [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/91612. – Дата доступа: 02.09.2023.

5. Сборник задач по теории алгоритмов : учеб.-метод. пособие / В. М. Котов [и др.] – Минск : БГУ, 2017. – 183 с. [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/181529. – Дата доступа: 02.09.2023.

6. Опыт использования образовательной платформы Insight Runner на факультете прикладной математики и информатики Белорусского государственного университета. Роль университетского образования и науки в современном обществе : материалы междунар. науч. конф., Минск, 26–27 февр. 2019 г. / редкол.: А. Д. Король (пред.) [и др.]. – Минск : БГУ, 2019. – С. 263 – 267. Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/231914. – Дата доступа: 02.09.2023.

7. Сборник задач по теории алгоритмов. Структуры данных : учеб.-метод. пособие / С. А. Соболь [и др.] – Минск : БГУ, 2020. – 159 с. Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/255033. – Дата доступа: 02.09.2023.

8. Сборник задач по теории алгоритмов. Организация перебора и приближенные алгоритмы : электронный учебно-методический комплекс для специальности: 1-31 03 04 «Информатика» / В. М. Котов, Е П. Соболевская, Г. П. Волчкова ; БГУ, Фак. прикладной математики и информатики, Каф. дискретной математики и алгоритмики. – Минск : БГУ, 2021. – 144 с. : ил. – Библиогр.: с. 143–144. – Режим доступа: https://elib.bsu.by/handle/123456789/272717. – Дата доступа: 02.09.2023.