БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

В. М. Котов
Е. П. Соболевская
Г. П. Волчкова

Теория алгоритмов. Организация перебора и приближенные алгоритмы

Рекомендовано
Учебно-методическим объединением
по естественно-научному образованию в качестве
учебно-методического пособия для студентов
учреждений высшего образования, обучающихся
по специальности «информатика»

МИНСК
БГУ
2022

УДК 510.51(075.8)
ББК 22.14я73-1
К73

Р е ц е н з е н т ы:
кафедра программного обеспечения
информационных технологий Белорусского государственного
университета информатики и радиоэлектроники
(заведующий кафедрой кандидат технических наук *Н. В. Лапицкая*);
кандидат физико-математических наук, профессор *П. В. Гляков*

Котов, В. М.

К73 Теория алгоритмов. Организация перебора и приближенные алгоритмы : учеб.-метод. пособие / В. М. Котов, Е. П. Соболевская, Г. П. Волчкова. – Минск : БГУ, 2022. – 151 с.
ISBN 978-985-881-326-0.

Учебно-методическое пособие содержит теоретический материал и задачи для самостоятельного решения. В издании изложены методы организации перебора, разработки и анализа приближенных алгоритмов, алгоритмов локального поиска, а также алгоритмов для задач с неполной информацией.

Предназначено для студентов учреждений высшего образования, обучающихся по специальности «информатика».

УДК 510.51(075.8)
ББК 22.14я73-1

ISBN 978-985-881-326-0

ВВЕДЕНИЕ

В учебно-методическом пособии изложены способы организации перебора, построения и анализа приближенных алгоритмов, приведены различные модельные задачи комбинаторной оптимизации. Данный материал поможет студентам получить знания, необходимые для успешного усвоения дисциплин, которые изучаются на факультете прикладной математики и информатики (ФПМИ), а также приобрести практические навыки.

Задачи для самостоятельного решения, а также указания к их решению завершают каждую главу учебно-методического пособия.

Практические занятия предполагают разработку алгоритма с последующим высоким уровнем его реализации на языке программирования. Для проверки работоспособности программ на ФПМИ используется система автоматического тестирования, функциональность которой позволяет организовать дистанционное обучение, самостоятельную и контролируемую работу студентов. Применяя систему автоматического тестирования, преподаватель освобождается от рутинной работы, связанной с проверкой решений, и получает возможность уделять больше внимания вопросам алгоритмизации.

Глава 1

ОРГАНИЗАЦИЯ ПОЛНОГО ПЕРЕБОРА

1.1. Построение дерева решений

Алгоритм полного перебора используют в случае, когда для решения задачи не удается разработать эффективный алгоритм решения.

Основной принцип, на котором базируются методы полного перебора вариантов, состоит в разбиении начальной задачи P_0 на подзадачи $P_1, P_2, ..., P_k$ с последующей попыткой разрешить каждую из этих подзадач.

Данное разбиение можно представить в виде дерева, вершины которого изображают подзадачи (рис. 1.1).

Выражение «решить подзадачу» имеет одно из следующих значений:

1) найти оптимальное решение;

2) показать, что значение решения, лучшего из найденных, не хуже оптимального решения подзадачи;

3) показать, что подзадача является недопустимой.

Смысл разбиения задачи P_0 в том, что подзадачи проще решить, так как они имеют меньший размер или обладают структурой, не присущей первоначальной задаче P_0.

В случае если подзадачу P_i нельзя решить, то она разбивается на новые подзадачи $P_{i_1}, P_{i_2}, ..., P_{i_r}$ (рис. 1.2). Такое разбиение (ветвление) повторяется для каждой подзадачи, которая не может быть решена.

На любом этапе разбиения (ветвления) полное множество подзадач, требующих решения, представляется множеством концевых вершин

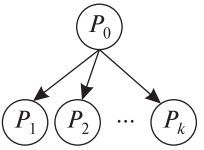


Рис. 1.1

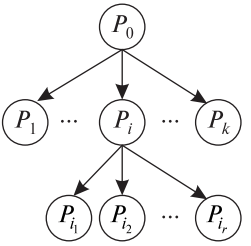


Рис. 1.2

всех путей, исходящих из корня дерева решений, т. е. из начальной задачи P_0. Концевые вершины P_1, ..., P_{i_1}, P_{i_2}, ..., P_{i_r}, ..., P_k называются висячими (см. рис. 1.2).

Если поиск исчерпан, то полученное множество подзадач может представлять все пространство исходной задачи. Таким образом, если задача P_i разбита на r подзадач, то

$$\{P_{i_1}\} \cup \{P_{i_2}\} \cup \ldots \cup \{P_{i_r}\} = \{P_i\},$$

где $\{P_i\}$ обозначает множество всех допустимых решений задачи P_i.

Данное соотношение должно быть применено к каждому разбиению, поэтому $\{P_0\} = \cup\{P_j\}$, где P_j – висячая вершина дерева.

Желательно избегать дублирования построенных решений, т. е. разбивать задачу P_i на подзадачи P_{i_1}, P_{i_2}, ..., P_{i_r} таким образом, чтобы

$$\{P_{i_s}\} \cap \{P_{i_q}\} = \varnothing \qquad (1.1)$$

для любых двух подзадач P_{i_s} и P_{i_q} при $s \neq q$.

Условие (1.1) не является необходимым для полноценного поиска при помощи дерева решений, однако оно имеет большие выгоды с вычислительной точки зрения, так как:

- для задачи оптимизации P_0 наилучшим будет решение одной и только одной подзадачи, являющейся висячей вершиной;
- для задачи полного перебора объединение множеств решений подзадач, представленных висячими вершинами, дает множество всех решений задачи P_0 без дублирования.

В данном случае необходимо иметь механизм отсева повторяющихся подзадач. Однако если количество подзадач велико, то традиционные методы отсева (например, использование памяти) могут не работать, поэтому приходится многократно пересчитывать некоторые подзадачи. Таким образом, нехватка памяти приводит к многократному увеличению времени работы алгоритма.

Рассмотрим задачу P_i с n переменными, где некоторая переменная x_i может принимать только четыре возможных значения: a, b, c и d.

Покомпонентное ветвление. Возможно разбиение P_i на четыре подзадачи – P_{i_1}, P_{i_2}, P_{i_3}, P_{i_4}, причем мы полагаем $x_i = a$ для P_{i_1}; $x_i = b$ – для P_{i_2}; $x_i = c$ – для P_{i_3}; и $x_i = d$ – для P_{i_4} (рис. 1.3).

Каждая из подзадач содержит $n - 1$ переменных и, возможно, допускает более простое решение, чем задача P_i.

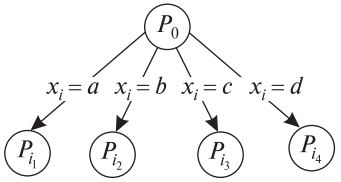


Рис. 1.3

Разбиение по некоторому признаку. Возможно другое разбиение P_i на две подзадачи – P_{i_1}, P_{i_2}, где мы полагаем $x_i = a$ для P_{i_1}, $x_i \neq a$ – для P_{i_2}. Значит, $x_i = b$, c или d (рис. 1.4).

Еще одно возможное разбиение P_i на две подзадачи – P_{i_1}, P_{i_2}, где $x_i = a$ или b для P_{i_1}, $x_i = c$ или d – для P_{i_2} (рис. 1.5).

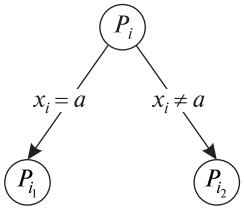


Рис. 1.4

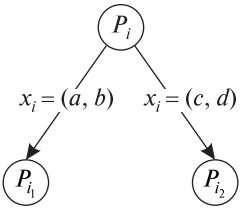


Рис. 1.5

Все ветвления являются допустимыми и удовлетворяют условию (1.1). Какому из них отдать предпочтение, зависит от природы решаемой задачи, причем возможности первых двух типов используются чаще остальных. Однако подзадачи этих подзадач могут и не обладать свойством (1.1). Например, подзадачи P_{i_1} и P_{i_2} оптимальны на подзадаче $x_i = a$.

1.2. Способы обхода дерева решений

Из вышесказанного следует, что любая подзадача, представляемая вершиной и не поддающаяся решению, может быть разбита на подзадачи. Существует две основные стратегии разбиения в зависимости от способа выбора следующей вершины для продолжения процесса ветвления.

Фронтальное ветвление. Начальная задача P_0 разбивается на подзадачи P_1, P_2, ..., P_k, образуя фронт ветвления. Из этих подзадач выбирается наиболее перспективная и снова разбивается на подзадачи, увеличивая количество вершин фронта, которые должны исследоваться до задач следующего уровня.

Вид дерева решений при стратегии фронтального ветвления показан на рис. 1.6.

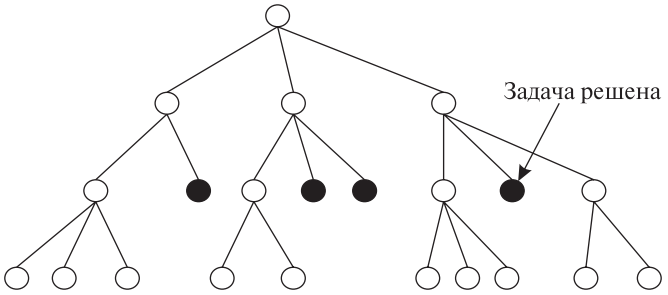


Рис. 1.6

Одностороннее ветвление. При данной стратегии ветвление осуществляется в последней выбранной подзадаче. Такой процесс продолжается до порождения подзадачи, которую можно решить. В этом месте делается возврат к предпоследней порожденной подзадаче, и ветвление продолжается в соответствующей вершине.

При одностороннем ветвлении полученные на каждом этапе задачи хранятся в стеке вместе с исходной задачей.

Вид дерева решений при таком типе ветвления показан на рис. 1.7, где порядок исследования полученных подзадач отражен соответствующей нумерацией.

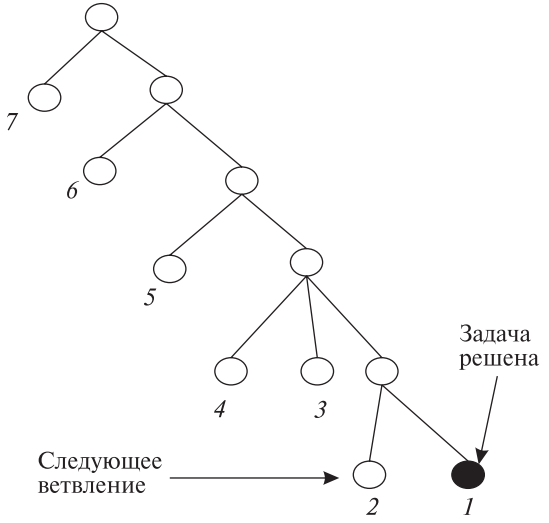


Рис. 1.7

Такая стратегия часто называется *поиском с возвратом*. Основные ее достоинства – простота реализации и минимизация используемой памяти, так как, в отличие от фронтального ветвления, нет необходимости хранить формат вершин кандидатов для ветвления. В *поиске в глубину* строится глубинное дерево поиска, т. е. расширяется некоторый частичный путь до тех пор, пока из его конечной вершины можно еще куда-нибудь двигаться. Если расширить частичный путь не удается, то происходит возврат по дереву на шаг назад и делается попытка движения в другом направлении. При поиске в глубину доступны лишь те вершины, которые ранее никогда не посещались. Для алгоритма полного перебора вариантов при расширении частичного решения такого ограничения нет.

Как при одностороннем, так и при фронтальном ветвлении выбор очередной вершины не полностью определен. При одностороннем ветвлении, когда задача P_i разбивается на подзадачи P_{i_1}, P_{i_2}, ..., P_{i_r}, очередное ветвление, как уже отмечалось, производится в одной из только что порожденных подзадач. Если не указано, в какой именно, то любая из них может рассматриваться как последняя порожденная. При фронтальном ветвлении все подзадачи данного уровня должны исследоваться до задач следующего уровня, однако порядок их исследования на одном уровне четко не установлен.

Функция ветвления позволяет определить вершину, которая будет использоваться при следующем ветвлении. Для вершины, соответствующей подзадаче P_j, эта функция – некоторая мера вероятности того, что оптимальное решение всей задачи P_0 является решением для P_j. Вершина, соответствующая подзадаче с бóльшими шансами на оптимальное решение, должна пользоваться правом преимущественного выбора при очередном ветвлении. Можно указать несколько эвристических мер этой вероятности, одна из которых связана с вычислением для вершин нижних или верхних границ. Для такой меры вершина с более низкой нижней границей (для случая минимизации) считается имеющей бóльшую вероятность. Можно использовать функцию ветвления так, чтобы она полностью определяла выбор следующей вершины для ветвления. Например, если значениями функции ветвления выступают границы вершин (нижние и верхние), то всегда можно производить ветвление в той висячей вершине, нижняя граница которой наименьшая. Эта стратегия является гибридом описанных ранее подходов ветвления, хотя часто называется *поиском по ширине*.

Предположим, что для графа (рис. 1.8) необходимо найти простую цепь между вершинами 1 и 6, используя алгоритм поиска в глубину.

В результате получаем корневое дерево поиска в глубину, в котором нет повторяющихся вершин (рис. 1.9).

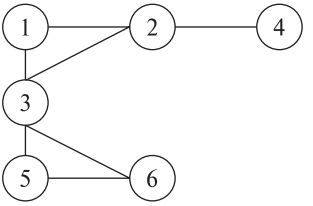


Рис. 1.8

Теперь построим для графа, приведенного на рис. 1.8, все простые цепи, соединяющие вершины 1 и 6, используя алгоритм полного перебора вариантов с односторонним ветвлением, начатом в вершине 1. Если выбраны некоторые вершины графа $v_1, v_2, \ldots, v_{k-1}$ в качестве частичного решения, то при выборе очередной вершины v_k множество возможных выборов представляет собой вершины графа, которые смежны с вершиной v_{k-1} и отличны от вершин $v_1, v_2, \ldots, v_{k-1}$. Ветвление в некоторой вершине v_k заканчивается, когда $v_k = 6$ (частичное решение является решением задачи) или текущее частичное решение из вершины v_k не сможет быть больше расширено.

Применяя данную стратегию для графа на рис. 1.8, получим древовидную структуру (рис. 1.10).

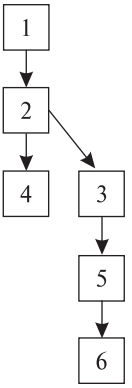


Рис. 1.9

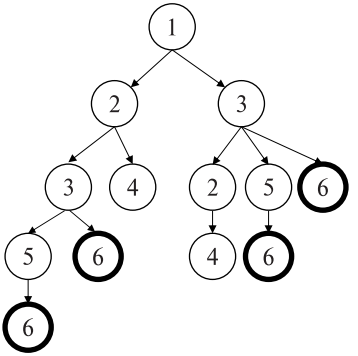


Рис. 1.10

Сгенерированы четыре различные тривиальные цепи, соединяющие вершины 1 и 6; некоторые вершины повторяются:

$$1 - 2 - 3 - 5 - 6;$$

$$1 - 2 - 3 - 6;$$

$$1 - 3 - 5 - 6;$$

$$1 - 3 - 6.$$

1.3. Сокращение числа подзадач, необходимых для решения: отсев возможных вариантов ветвления

При организации перебора вариантов решения основной целью является сокращение количества решаемых подзадач. Рассмотрим три стратегии отсева неперспективных вариантов ветвления:

1) по повторению (исключение повторяющихся подзадач);

2) недопустимости (некоторая вершина является недопустимой);

3) рекорду (если заранее можно определить, что некоторая подзадача не даст решения лучше, чем построенное алгоритмом ранее (рекорд)).

Отсев по повторению. *Кликой* графа $G = (V, E)$ называется максимальное по включению подмножество вершин графа, любые две вершины которого соединены ребром.

Для графа (см. рис. 1.8) имеются три клики:

1, 2, 3;
3, 5, 6;
2, 4.

Одним из алгоритмов построения *множества* всех *клик графа* является поиск с возвратом. Каждая вершина дерева поиска будет соответствовать полному подграфу графа (полный подграф задается множеством его вершин), а каждая дуга дерева поиска – вершине графа. Корень дерева – пустое множество вершин. Предположим, что некоторой вершине дерева поиска поставлен в соответствие полный подграф графа с множеством вершин *C*. Пусть вершина *w* графа смежна с каждой из вершин множества *C,* тогда вершина $C \cup \{w\}$ будет сыном вершины *C*, а дуга, идущая от *C* к $C \cup \{w\}$, будет соответствовать вершине *w*. Если не существует вершин, смежных с каждой из вершин множества *C*, то ветвление в соответствующей вершине дерева завершено и данная висячая вершина порождает клику графа.

Дерево решений для графа на рис. 1.8 будет иметь вид, как на рис. 1.11.

На рисунке видно, что поиск с возвратом привел к порождению 14 клик, но только три из них различны:

$$\{1, 2, 3\}, \{2, 4\}, \{3, 5, 6\}.$$

Число клик в графе может расти экспоненциально относительно числа вершин, поэтому важно при построении дерева решений избегать повторений. Для этого следует использовать следующие теоремы.

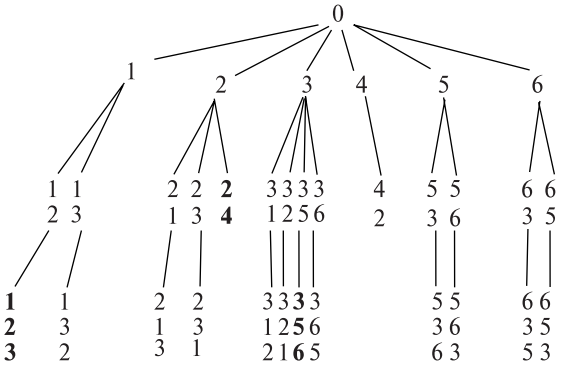


Рис. 1.11

Теорема 1.1. Пусть C – некоторая вершина дерева поиска с возвратом, а $C \cup \{w\}$ – ее первый сын, который должен быть исследован. Предположим, что все поддеревья вершины $C \cup \{w\}$ уже исследованы и порождены все клики, включающие вершины множества $C \cup \{w\}$. Тогда необходимо из оставшихся сыновей вершины C исследовать только те, которые не смежны с вершиной w графа (рис. 1.12).

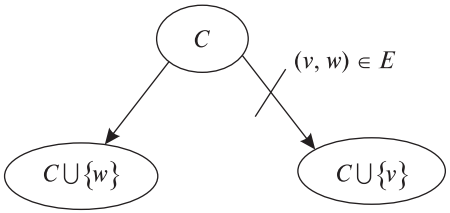


Рис. 1.12

Теорема 1.1 применяется исключительно к первому исследуемому сыну вершины поиска C.

Теорема 1.2. Пусть C – некоторая вершина дерева поиска с возвратом и $\underline{C}$ – ее предок. Если все поддеревья вершины $\underline{C} \cup \{w\}$ уже исследованы и порождены все клики, включающие вершины множества $\underline{C} \cup \{w\}$, то все неисследованные поддеревья с корнями в вершине $C \cup \{w\}$ можно проигнорировать (рис. 1.13).

С учетом приведенных теорем 1.1 и 1.2 (отсев по повторению) дерево решений поиска с возвратом для графа, приведенного на рис. 1.8, будет иметь вид, представленный на рис. 1.14.

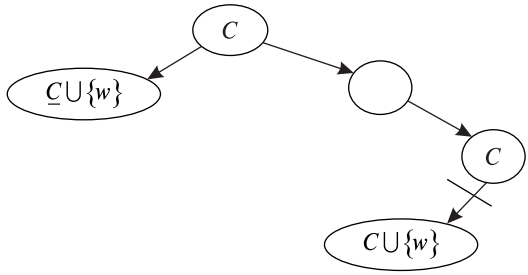


Рис. 1.13

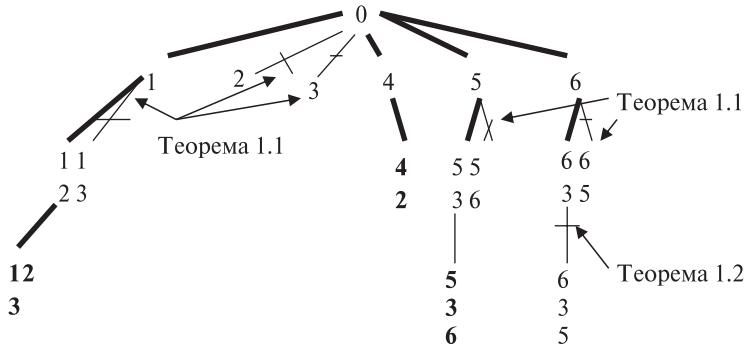


Рис. 1.14

Алгоритм можно улучшить, если более аккуратно выбирать корень первого исследуемого поддерева. В соответствии с теоремой 1.1 такой вершиной должна быть та, которая смежна с наибольшим количеством вершин, что позволит отсечь наибольшее число поддеревьев. Более простым и используемым способом отсева по повторению является построение подзадач в лексикографическом порядке, который гарантирует однозначность решений. Однако такой подход не всегда сочетается с другими способами отсечения.

Отсев по недопустимости. Рассмотрим задачу определения множества всех контуров ориентированного графа (орграфа) $D=(V, E)$. Для того чтобы определить, содержит ли неориентированный граф (далее – граф) цикл или орграф – контур, достаточно выполнить алгоритм поиска в глубину с пометкой ребер (дуг). Граф (орграф) содержит цикл (контур) только тогда, когда в нем существует хотя бы одно ориентированное как обратное ребро (дуга).

Несложной является задача построения *фундаментального множества циклов графа* $G = (V, E)$ – минимального множества циклов, из которого могут быть построены все циклы графа. Для решения этой задачи выполняется поиск в глубину, при этом все ребра графа классифицируются как древесные или обратные. Во время поиска будет построено глубинное дерево, которому принадлежат все ребра графа, ориентированные как древесные. При этом каждое ребро, ориентированное как обратное, при добавлении к глубинному дереву поиска порождает ровно один цикл. Мощность фундаментального множества циклов графа G равна $|E| - |V| + 1$.

Для построения фундаментального множества циклов для графа, изображенного на рис. 1.15, выполним алгоритм поиска в глубину, начиная с вершины 1. На рисунке ребра (2, 1), (3, 2), (4, 3), которые были классифицированы как древесные, выделены жирными линиями. Ребра (1, 3) и (2, 4) – обратные.

На рис. 1.16 показано фундаментальное множество циклов графа, приведенного на рис. 1.15.

Любой цикл в графе может быть представлен в виде линейной комбинации фундаментальных циклов. В нем присутствуют только ребра, участвующие в линейной комбинации нечетное число раз. Для графа (см. рис. 1.15) цикл (1, 2, 4, 3, 1) является линейной комбинацией циклов (1, 2, 3, 1) и (2, 4, 3, 2).

Более сложной выступает задача построения множества всех контуров орграфа. Умение решать эту задачу для орграфа позволит решать ее и для графа, заменив каждое ребро графа двумя дугами с противоположными направлениями. Количество всех контуров полного орграфа с n вершинами может иметь порядок $(n-1)!$, поэтому важной является задача отсечения повторяющихся контуров, а также отсев неперспективных ветвлений, т. е. таких путей, которые заведомо не дадут контура. Для отсева повторяющихся контуров используется *корень контура* –

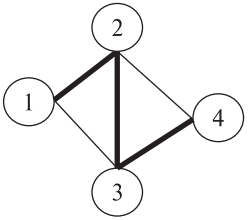


Рис. 1.15

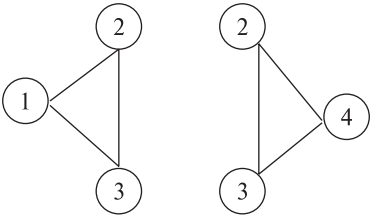


Рис. 1.16

вершина с наименьшим номером среди всех вершин этого контура. Например, для контура $3 \to 2 \to 6 \to 3$ корнем служит вершина 2.

Алгоритм 1.1. Строим последовательно контуры с корнями в вершинах 1, 2, ..., n. Для построения всех контуров с корнем в вершине s поступаем следующим образом.

1. Считаем, что все вершины орграфа не заблокированы, т. е. доступны.

2. Поиском в глубину (идем в вершины, которые не заблокированы) строим путь $(s, v_1, v_2, \ldots, v_k)$, где $v_i > s \ \forall 1 \le i \le k$ (*отсечение по повторению*). Как только вершина присоединяется к пути, она сразу же блокируется (за исключением вершины s). Блокирование вершин v_i, присоединяемых к пути, выполняется для того, чтобы не допустить прохождение контуров с корнями в вершинах v_i.

3. Контур с корнем в вершине s построен, если на некотором шаге алгоритма $v_{k+1} = s$. После построения контура $(s, v_1, v_2, \ldots, v_k, s)$ рассматриваем следующую дугу, выходящую из вершины v_k, и если она ведет в незаблокированную вершину, то продолжаем поиск в глубину из этой вершины. Если же оказывается, что все дуги, выходящие из вершины v_k, исследованы, то возвращаемся в вершину v_{k-1} и изучаем выходящие из нее пути. В случае когда из некоторой вершины v_k был найден хотя бы один контур с корнем в стартовой вершине s, то при возврате вершина v_k должна быть разблокирована (иначе вершина остается заблокированной даже после возвращения из нее). Если на некотором этапе произошло разблокирование некоторой вершины v_i, то при этом вершины, не принадлежащие рассматриваемому пути $s \to v_1 \to \ldots \to v_i$ и являющиеся предшественниками вершины v_i (началами дуг, входящих в вершину v_i), также должны быть разблокированы. Таким образом, разблокирование некоторой вершины пути может привести к цепочке разблокирования других ранее заблокированных вершин (отсев по недопустимости).

4. Построение контура с корнем в вершине s завершится при попытке вернуться во время поиска в глубину за вершину s. После того как все контуры с корнем в вершине s будут построены, она может быть удалена из орграфа вместе с инцидентными ей дугами.

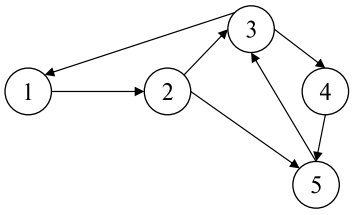


Рис. 1.17

Работу алгоритма проиллюстрируем на следующем примере. Для орграфа, изображенного на рис. 1.17, построим множество всех контуров с корнем в вершине 1.

При поиске контуров с корнем в вершине 1 построим путь $1 \rightarrow$ $\rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, после чего все вершины, за исключением вершины 1, станут заблокированными, а контур с корнем в вершине 1 не построится. Осуществится возврат из вершины 5, но при этом она останется заблокированной. Аналогично произойдет и с вершиной 4 (4 – разблокирована, 5 – разблокирована, 3 – разблокирована).

Для пути $1 \rightarrow 2 \rightarrow 3$ существует еще одна непросмотренная дуга $3 \rightarrow 1$, приводящая к контуру. По этой причине после возврата из вершины 3 она станет разблокированной, что повлечет за собой разблокирование вершины 5 (вершина 5 не принадлежит пути $1 \rightarrow 2 \rightarrow 3$ и является предшественником вершины 3 в орграфе), это приведет к разблокированию вершины 4. При возврате из вершины 3 в вершину 2 будут разблокированы вершины 3, 4 и 5 (рис. 1.18).

Продолжим поиск в глубину из вершины 2 по ранее не исследованным дугам. Из вершины 2 идем по непросмотренной дуге $2 \rightarrow 5$ в незаблокированную вершину 5 (блокируем ее), а затем из вершины 5 – в незаблокированную вершину 3 (блокируем ее), далее – в вершину 1.

Получаем контур $1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1$. Исследуем дуги, выходящие из вершины 3. Двигаемся из вершины 3 по еще не просмотренной дуге в незаблокированную вершину 4 (блокируем), а далее пути нет, так как все вершины заблокированы ($1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4$). Возвращаемся из вершины 4, оставляя ее заблокированной. Осуществляем возврат из вершины 3, так как все ребра, выходящие из этой вершины обследованы. Вершина 3 становится разблокированной (из нее был контур в вершину 1). Осуществляем возврат из вершины 5 (разблокирована), что приводит к разблокировке вершины 4. Затем осуществляем возврат из вершины 2 (разблокирована), так как все дуги, выходящие из этой вершины, исследованы, после чего осуществляется возврат из вершины 1. Алгоритм построения контуров с корнем в вершине 1 завершен (см. рис. 1.18). В результате построены два контура с корнем в вершине 1:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 1,$$
$$1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1.$$

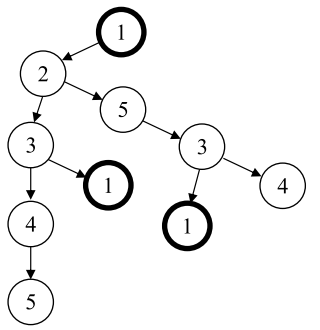


Рис. 1.18

Пример 1.1. Необходимо построить множество контуров для ориентированного графа, изображенного на рис. 1.19.

Решение. Последовательность действий построения множества контуров с корнем в вершине 1 приведена на рис. 1.20.

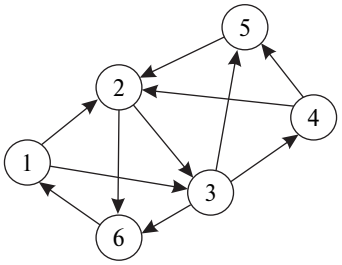


Рис. 1.19

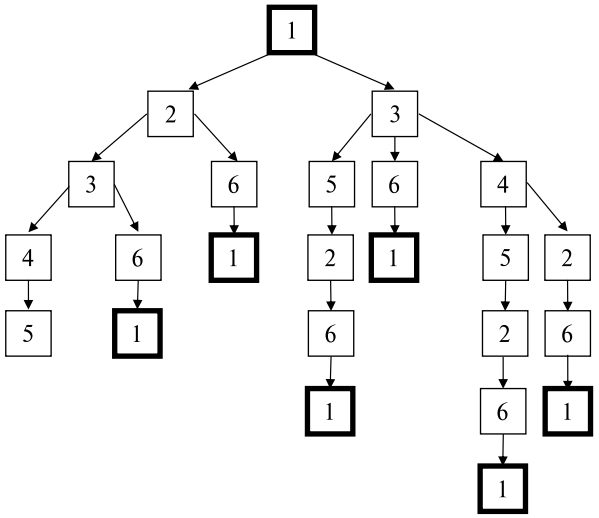


Рис. 1.20

Построены контуры с корнем в вершине 1:

$$1 \to 2 \to 3 \to 6 \to 1,$$
$$1 \to 2 \to 6 \to 1,$$
$$1 \to 3 \to 5 \to 2 \to 6 \to 1,$$
$$1 \to 3 \to 6 \to 1,$$
$$1 \to 3 \to 4 \to 5 \to 2 \to 6 \to 1,$$
$$1 \to 3 \to 4 \to 2 \to 6 \to 1.$$

Удалим из орграфа, приведенного на рис. 1.19, вершину 1 вместе с инцидентными ей дугами. Для модифицированного орграфа постро-

им множество контуров с корнем в вершине 2 (рис. 1.21). Последовательность действий построения множества контуров с корнем в вершине 2 показана на рис. 1.22.

Построены контуры с корнем в вершине 2:

$$2 \to 3 \to 4 \to 5 \to 2,$$
$$2 \to 3 \to 5 \to 2.$$

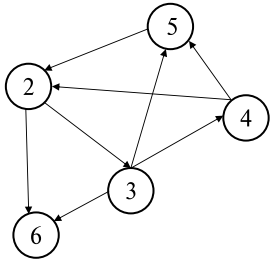


Рис. 1.21

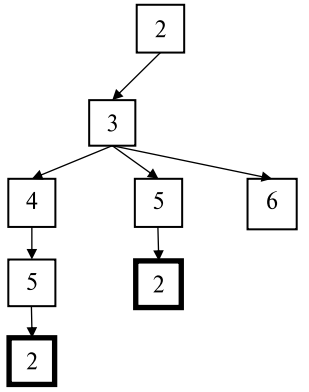


Рис. 1.22

Удалим из орграфа (см. рис. 1.21) вершину 2 вместе с инцидентными ей дугами. Для модифицированного орграфа (рис. 1.23) построим множество контуров с корнем в вершине 3. Полученный орграф – ациклический, поэтому алгоритм построения всех контуров для орграфа, приведенного на рис. 1.19, заканчивает свою работу.

Отсев по рекорду (применение оценок). Предположим, что каждому решению (в том числе частичному) соответствует некоторая целевая функция $F(x_1, x_2, \ldots, x_k)$, причем

$$F(x_1, x_2, \ldots, x_k) \le F(x_1, x_2, \ldots, x_{k+1}).$$

Во многих задачах дискретной оптимизации целевая функция удовлетворяет равенству

$$F(x_1, x_2, \ldots, x_k) = F(x_1, x_2, \ldots, x_{k-1}) + f(x_k) =$$
$$= f(x_1) + f(x_2) + \cdots + f(x_k),$$

где $f(x_k) \ge 0,\ k = 1, \ldots, n.$

Такая функция является сепарабельной.

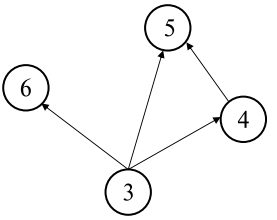


Рис. 1.23

В задаче минимизации необходимо найти все решения с минимальным значением целевой функции, а в задаче максимизации – с максимальным значением.

При организации перебора вариантов решения строится дерево вариантов, причем частичным решениям соответствуют внутренние вершины дерева, а решениям исходной задачи – листья. Цель организации поиска – обход дерева, что позволит построить все оптимальные решения, просмотрев минимальное число вершин дерева. Самое простое решение поставленной задачи – полный обход дерева. Однако для многих задач возможно значительно сократить число рассматриваемых вершин – достаточно определить, какие из подзадач целесообразно рассматривать (ветвить), а какие можно игнорировать.

Определим основные понятия:

1) F^* – значение оптимального решения задачи;

2) F – рекорд, который соответствует значению наилучшего известного решения. Рекорд характеризует приближение к оптимальному решению, поэтому важно удачно выбрать начальное рекордное решение. Обычно его находят с помощью приближенных методов решения исходной задачи. Если не известно ни одного допустимого решения исходной задачи, то не остается ничего другого, как принять значение F равным бесконечности для задачи минимизации, 0 – для задачи максимизации (считая, что $F^* \geq 0$);

3) вершине x дерева ветвления соответствует некоторое частичное решение, в котором часть переменных $x = (x_1, x_2, \ldots, x_k)$ зафиксирована, а остальные переменные $\bar{x} = (x_{k+1}, \ldots, x_n)$ свободны, т. е. пока еще не определены. Значение частичного решения x определим как

$$F(x) = f(x_1) + f(x_2) + \cdots + f(x_k).$$

Подзадача для вершины x – это задача на свободных, пока еще не зафиксированных, переменных (в наших обозначениях на переменных $\bar{x} = (x_{k+1}, \ldots, x_n)$) при некоторой фиксированной части переменных $x = (x_1, x_2, \ldots, x_k)$.

Для подзадачи определим нижнюю оценку (НО) и верхнюю оценку (ВО) :

$$\text{НО}(\bar{x}) \leq \text{опт}(\bar{x}) \leq \text{ВО}(\bar{x}),$$

где $\text{опт}(\bar{x})$ – значение оптимального решения подзадачи x (рис. 1.24).

Рис. 1.24

Для вычисления оценки существует несколько способов. Часто рассматривается более общая задача, которая легко решается. Например, от дискретных переменных переходят к непрерывным либо убирают (ослабляют) некоторые ограничения, тем самым гарантируя, что значение оптимального решения общей задачи не хуже значения оптимального решения исходной задачи. Затем значение оптимального решения более общей задачи берется в качестве оценки.

Если в задаче минимизации для вершины дерева x выполняется неравенство

$$F < F(x) + \text{НО}(\bar{x}),$$

то

$$F < F(x) + \text{опт}(\bar{x}),$$

поэтому частичное решение x не приведет к оптимальному решению. Следовательно, для вершины x можно производить отсечение по рекорду.

В задаче минимизации можно всегда взять $\text{НО}(\bar{x}) = 0$. Тогда отсечение частичного решения по рекорду будет происходить только в случае, если значение целевой функции для частичного решения больше значения рекорда.

Если в задаче максимизации для вершины дерева x выполняется неравенство

$$F > F(x) + \text{ВО}(\bar{x}),$$

то

$$F > F(x) + \text{опт}(\bar{x}),$$

следовательно, частичное решение x не приведет к оптимальному решению и можно выполнить отсечение вершины x по рекорду. В задаче максимизации можно взять $\text{ВО}(\bar{x}) = +\infty$. В этом случае отсечения частичного решения по рекорду не происходят никогда, происходит только пересчет рекорда, следовательно, выполняется полный обход дерева решений.

Для построения одного оптимального решения можно использовать более сильное отсечение с нестрогим неравенством: $F \le F(x) + \text{НО}(\bar{x}) -$ для задачи на минимум и $F \ge F(x) + \text{ВО}(\bar{x})$ – для задачи на максимум. Метод организации перебора вариантов решения с отсечениями по рекорду часто называют методом *ветвей и границ*.

Рассмотрим задачу минимизации.

Теорема 1.3. Предположим, нижняя оценка подзадачи совпадает со значением ее оптимального решения, т. е.

$$\text{НО}(\bar{x}) = \text{опт}(\bar{x}),$$

тогда это позволит генерировать для задачи минимизации только оптимальные решения, отсекая в дереве все неперспективные частичные решения.

Доказательство. Рассмотрим частичное решение $x = \varnothing$, в котором ни одна из переменных не зафиксирована. Тогда подзадача для x – это задача на переменных $\bar{x} = (x_1, \ldots, x_n)$, т. е. исходная задача.

Более того, так как по условию теоремы

$$\text{НО}(\bar{x}) = \text{опт}(\bar{x}) = F^*,$$

то нижняя оценка исходной задачи совпадает со значением ее оптимального решения. Таким образом, вычислив для подзадачи $\bar{x} = (x_1, \ldots, x_n)$ нижнюю оценку, мы получим значение оптимального решения исходной задачи.

Теперь, выбирая в качестве рекорда F значение оптимального решения исходной задачи F^*, мы рассмотрим только те частичные решения, для которых

$$F^* = F(x) + \text{опт}(\bar{x}),$$

тем самым гарантируя получение оптимальных решений и отсекая все неперспективные частичные решения.

Для задачи максимизации справедлива аналогичная теорема.

Теорема 1.4. Допустим, верхняя оценка подзадачи совпадает со значением ее оптимального решения, т. е.

$$\text{ВО}(\bar{x}) = \text{опт}(\bar{x}),$$

тогда это позволит генерировать для задачи максимизации только оптимальные решения, отсекая в дереве все неперспективные частичные решения.

Доказательство данной теоремы аналогично доказательству теоремы 1.3.

В случае совпадения оценки с оптимальным решением при генерации всех оптимальных решений трудоемкость будет зависеть от количества решений, трудоемкости вычисления оценки и высоты дерева перебора. Проиллюстрируем отсев по рекорду на следующем примере.

Пример 1.2. Имеется клеточное поле размером $n \times m$, на котором расставлены фигуры. Необходимо найти все кратчайшие маршруты коня между двумя заданными позициями: s – стартовой и t – финишной.

Решение. В данной задаче вершинам дерева перебора вариантов соответствуют позиции шахматной доски. Корень дерева – стартовая позиция s. Частичное решение для вершины дерева x – это путь из корня дерева в вершину x, т. е. последовательность ходов коня из позиции s шахматной доски в позицию x. Значение целевой функции на частичном решении $F(x)$ определяется как длина пути из корня дерева s в вершину x, при этом $F(s) = 0$. Сыновья вершины x – всевозможные пути, которые получаются добавлением к частичному решению, соответствующему вершине x, одного хода конем:

$$F(v) = F\big(\text{отец}(v)\big) + 1.$$

Пусть x – некоторая вершина дерева, тогда подзадача для данной вершины – это путь коня из позиции x шахматной доски в точку финиша f.

$\text{НО}(\bar{x})$ для подзадачи определим как длину кратчайшего пути коня на шахматной доске из позиции x в позицию f. Нижняя оценка подзадачи совпадает со значением оптимального решения:

$$\text{НО}(\bar{x}) = \text{опт}(\bar{x}).$$

Это обеспечивает просмотр только таких вершин дерева, которые соответствуют кратчайшему пути коня.

В качестве рекорда возьмем длину кратчайшего пути коня из стартовой позиции шахматной доски в финишную.

Пусть x – некоторая вершина дерева, тогда данная вершина рассматривается в качестве перспективной, если

$$F^* = F(x) + \text{НО}(\bar{x}) = F(x) + \text{опт}(\bar{x}),$$

и неперспективной – в противном случае (отсечение по рекорду).

Очередное оптимальное решение задачи построено, когда $x = f$.

Для эффективного вычисления рекорда и нижних оценок подзадач сопоставим шахматную доску с графом и выполним поиск в ширину

(конем) из точки финиша f в точку старта s. При этом все достижимые позиции шахматной доски из точки финиша f получат метки, которые являются нижними оценками подзадач, соответствующих данным позициям. Метка, которая будет присвоена позиции s, есть рекорд F^*.

Трудоемкость описанного алгоритма – это трудоемкость вычисления рекорда и нижних оценок подзадач, а также построения кратчайших путей. Поскольку трудоемкость поиска в ширину есть $O(n)$, где n – количество пустых клеток на доске, а трудоемкость алгоритма восстановления путей есть $O(kl)$, где k – количество кратчайших путей, l – длина кратчайшего пути, то трудоемкость всего алгоритма есть $O(n+kl)$. Продемонстрируем работу алгоритма на примере.

Пример 1.3. Пусть $n=m=4$, $s=(1,1)$, $f=(4,4)$. Необходимо построить все кратчайшие маршруты коня из s в f.

Решение. После выполнения поиска в ширину из позиции f в позицию s матрица нижних оценок подзадач будет иметь следующий вид:

i/j	*1*	*2*	*3*	*4*
1	2	3	2	5
2	3	4	1	2
3	2	1	4	3
4	5	2	3	0

Рекорд $F^*=2$. Дерево решений для примера представим на рис. 1.25, где видно, что вершины x_1, x_2 являлись перспективными, а для вершин x_3, x_4, x_7, x_8 было выполнено отсечение по рекорду:

$$F^*=2=F(x_1)+\mathrm{HO}(\bar{x}_1)=1+1=2,$$

$$F^*=2=F(x_2)+\mathrm{HO}(\bar{x}_2)=1+1=2,$$

$$F^*=2<F(x_3)+\mathrm{HO}(\bar{x}_3)=2+2=4,$$

$$F^*=2<F(x_4)+\mathrm{HO}(\bar{x}_4)=2+2=4,$$

$$F^*=2<F(x_7)+\mathrm{HO}(\bar{x}_7)=2+2=4,$$

$$F^*=2<F(x_8)+\mathrm{HO}(\bar{x}_8)=2+2=4.$$

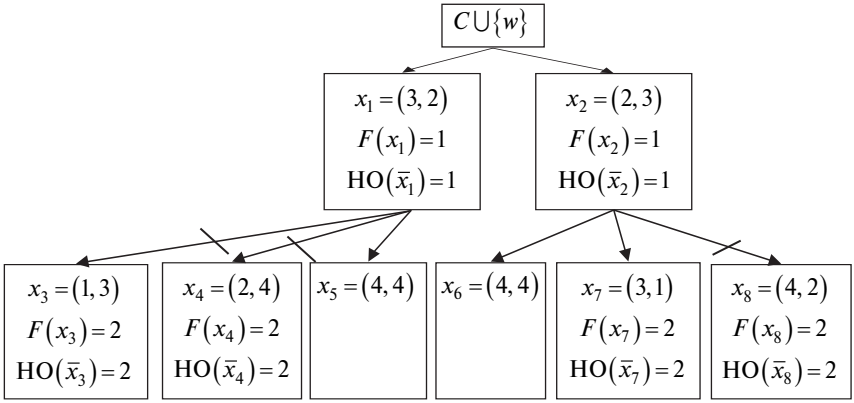


Рис. 1.25

Вершины x_5 и x_6 порождают оптимальные решения задачи:

$$x_5 : s = (1,1) \to (3,2) \to (4,4) = f,$$
$$x_6 : s = (1,1) \to (2,3) \to (4,4) = f.$$

1.4. Способы построения нижних и верхних оценок

Как было отмечено ранее, в задаче минимизации в качестве нижней оценки всегда можно взять число 0, а в задаче максимизации в качестве верхней оценки – бесконечность. Однако для того чтобы отсечь наибольшее число неперспективных вариантов ветвления, надо стремиться к построению максимально близкой оценки значения оптимального решения задачи.

Один из способов построения оценок состоит в использовании значения решения, построенного алгоритмом, для которого известна *гарантированная* оценка.

Предположим, что алгоритм A имеет гарантированную оценку P, если значение построенного решения x^A удовлетворяет неравенству

$$F(x^A) \geq PF(x^{\text{опт}}),\ P \leq 1,$$

для задачи на максимум, и

$$F(x^A) \leq PF(x^{\text{опт}}),\ P \geq 1,$$

для задачи на минимум.

Для задачи на максимум гарантированная оценка $P \le 1$, полученная алгоритмом A, означает, что алгоритмом A будет построено решение не менее $P \times 100$ % от оптимального (рис. 1.26).

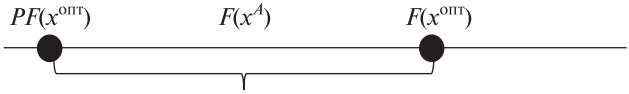


Рис. 1.26

Например, если $P = 1/4$, то построенное алгоритмом A решение составляет не менее 25 % оптимального.

Для задачи на минимум гарантированная оценка $P \ge 1$, полученная алгоритмом A, означает, что алгоритм A может ошибиться не больше чем в P раз.

Например, если $P = 3$, то алгоритм A может ошибиться не больше чем в три раза (рис. 1.27).

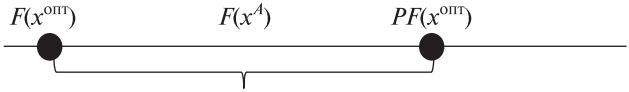


Рис. 1.27

Предположим, решается задача максимизации и для нее известен алгоритм A, который имеет гарантированную оценку P. Тогда справедливо неравенство

$$F\left(x^{\text{опт}}\right) \le F\left(x^A\right)\Big/P,\ P \le 1,$$

а значение $F\left(x^A\right)\Big/P$ можно взять в качестве верхней оценки. На рис. 1.28 область значений, которые может принимать верхняя оценка $\text{ВО}\left(x^{\text{опт}}\right)$, выделена серой заливкой.

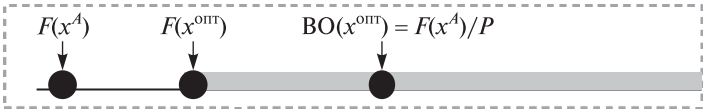


Рис. 1.28

Предположим, что решается задача минимизации и для нее известен алгоритм A, который имеет гарантированную оценку P. Тогда справедливо неравенство

$$F\left(x^{\text{опт}}\right) \ge F\left(x^A\right)\Big/P,\ P \ge 1,$$

и значение $F\left(x^A\right)/P$ можно взять в качестве нижней оценки. На рис. 1.29 область значений, которые может принимать нижняя оценка $\text{НО}\left(x^{\text{опт}}\right)$, выделена серой заливкой.

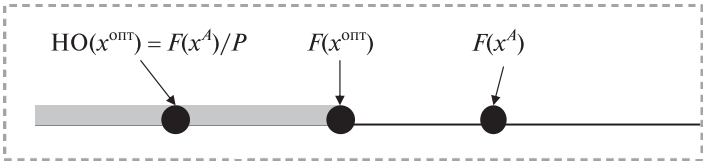


Рис. 1.29

Другой способ постоения оценок состоит в том, что в исходной задаче ослабляются некоторые условия и решается более общая задача. Например, если при решении целочисленных задач ослабить условие целочисленности, это приведет к расширению области допустимых значений. Величина оптимального решения на расширенной области не хуже, чем на исходной. Если существует алгоритм, который может эффективно решать новую задачу, то построенное таким алгоритмом решение будет верхней оценкой в случае максимизационной задачи, и нижней – в случае минимизационной.

Рассмотрим одну из самых известных *NP*-трудных задач комбинаторной оптимизации – *симметричную* (*неориентированную*) задачу коммивояжера (англ. *travelling salesman problem*). Задан полный взвешенный граф. Необходимо найти кратчайший маршрут, который начинается в некоторой вершине графа, проходит через каждую вершину только один раз и возвращается в начальную вершину. Другими словами, задача состоит в поиске гамильтонова цикла минимального веса.

В симметричной задаче коммивояжера даны расстояния между любыми двумя городами и матрица расстояний симметрична: $d\left(x, y\right) = d\left(y, x\right)$, т. е. все пары ребер между одними и теми же вершинами имеют одинаковую длину.

Симметричную задачу коммивояжера называют *метрической*, если для матрицы стоимостей выполняется неравенство треугольника, т. е. длина ребра между вершинами x и y никогда не бывает длиннее пути через промежуточную вершину k:

$$d\left(x, y\right) \le d\left(x, k\right) + d\left(k, y\right).$$

В задаче коммивояжера на плоскости каждой вершине графа соответствует точка на плоскости, а вес ребра между вершинами равен рас-

стоянию между соответствующими точками. Существует несколько вариантов задачи. Это зависит от метрики, в которой вычисляется расстояние между двумя точками на плоскости:

- геометрическая – расстояние на плоскости между точками $x=(x_1, x_2)$ и $y=(y_1, y_2)$ вычисляется в метрике в евклидовом пространстве:

$$d(x, y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2};$$

- прямоугольная – расстояние между точками решетки $x=(x_1, x_2)$ и $y=(y_1, y_2)$ вычисляется в манхэттенской метрике и равно сумме расстояний по оси ординат и абсцисс (задача возникает на практике, когда передвижение в обоих направлениях выполняется последовательно):

$$d(x, y)=|x_1-x_2|+|y_1-y_2|.$$

При построении оценок для задачи коммивояжера можно использовать некоторые подходы.

Допустим, мы находимся в положении, когда выбрано *k* ребер и множество удовлетворяет следующим двум требованиям:

- не образуется циклов длины меньше *n*;
- не образуется вершин степени 3.

Другими словами, выбранное множество ребер – набор простых открытых цепей. Множество ребер, еще не выбранных, назовем *свободными*. Свободное ребро, которое может быть добавлено к текущему состоянию, не нарушая требований, назовем *допустимым*.

Для задачи коммивояжера на плоскости при выборе допустимого ребра можно к двум условиям добавить третье: ребро не должно пересекаться с ранее выбранными ребрами. Это позволит отсечь неперспективные варианты ветвления, так как существует решение, в котором можно заменить пересекающиеся ребра на непересекающиеся, инцидентные тем же вершинам, при этом получая лучшее решение.

Предположим, что выбраны ребра $\{a, b\}$ и $\{c, d\}$, которые пересекаются в точке *x* (рис. 1.30).

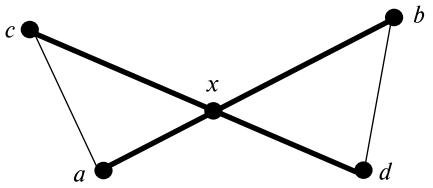


Рис. 1.30

Оценим их суммарную длину, учитывая неравенство треугольника:

$$|ab|+|cd|=|ax|+|xb|+|cx|+|xd|=$$

$$=\left(|ax|+|cx|\right)+\left(|xb|+|xd|\right)\geq|ca|+|bd|.$$

Таким образом, существует лучшее решение, куда входят непересекающиеся ребра $\{c,a\}$ и $\{b,d\}$, а выбор пересекающихся ребер является неперспективным вариантом ветвления.

В задаче коммивояжера для подзадач (задачи на свободных ребрах) можно вычислить нижние оценки, используя следующие правила:

1) $\mathrm{HO}(\bar{x})=0$, т. е. ничем не дополняем текущее состояние;

2) пусть $\min_e$ – длина минимального свободного ребра, тогда, учитывая, что гамильтонов цикл состоит из *n* ребер, а выбрано пока только *k*, полагаем

$$\mathrm{HO}(\bar{x})=\min_e(n-k);$$

3) нижнюю оценку можно усилить, если в качестве $\min_e$ взять длину минимального допустимого ребра (в результате будет отсекаться больше неперспективных вариантов решений).

В задаче коммивояжера можно ослабить условие поиска гамильтонова цикла, убирая часть ограничений, что приведет к расширению множества допустимых решений. Пусть $\hat{F}$ – решение задачи на расширенной области, тогда мы можем использовать $\hat{F}$ в качестве нижней оценки $\mathrm{HO}\left(x^{\text{опт}}\right)=\hat{F}$, поскольку $\hat{F}\leq\hat{F}\left(x^{\text{опт}}\right)$ (рис. 1.31).

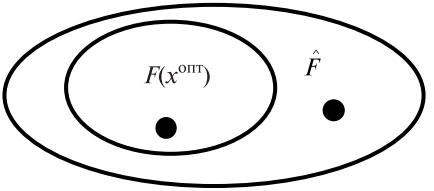


Рис. 1.31

Ослабить условие поиска гамильнонова цикла (связный подграф на всех вершинах, который является одним циклом) можно одним из двух *способов:*

1) убрать ограничение «цикл». Для задачи коммивояжера получим задачу поиска связного подграфа минимального веса, покрывающего все вершины графа. Это широко известная задача построения мини-

мального остовного дерева, которую можно решить, используя алгоритм Прима или Краскала;

2) убрать ограничения «один цикл» и связность. Для задачи коммивояжера получим задачу построения набора простых циклов минимального суммарного веса, покрывающих все вершины. Это известная задача построения в графе 2-фактора, для которой имеется эффективный точный алгоритм (напомним, что k-фактор – регулярный остовный подграф степени k, т. е. степени всех вершин остовного подграфа одинаковы и равны k).

Пример 1.4. Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлено k черных фигур. Необходимо расставить минимальное число белых ладей, чтобы пробивались все свободные позиции. В качестве примера – шахматное поле, изображенное на рис. 1.32 (занятые позиции на рисунке имеют серую заливку).

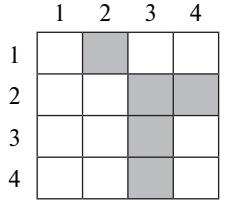


Рис. 1.32

Одна из возможных оптимальных расстановок ладей приведена на рис. 1.33.

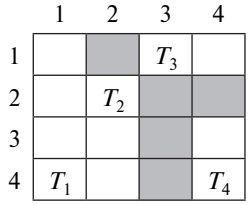


Рис. 1.33

Решение. Вершинам дерева перебора вариантов соответствуют некоторые расстановки ладей. В корне дерева перебора находится заданная расстановка черных фигур. Ветвление осуществляется следующим образом: пусть x – некоторая вершина дерева, тогда множеством ее сыновей будет множество расстановок с добавленной ладьей к уже

имеющейся расстановке, описываемой данной вершиной *x*. Таким образом, на глубине *l* дерева решений на шахматной доске будет находиться *l* ладей.

Обход осуществим в лексикографическом порядке. Такой способ обхода гарантирует отсутствие повторений и возможность получения любой целесообразной расстановки. Рекорд вычислим после первого завершенного обхода дерева (построен первый «лист») и впоследствии он может быть улучшен.

Рассмотрим возможные отсечения.

По доминированию. Если клетка, в которую возможна постановка ладьи на некотором шаге, уже пробивается, то ставить ладью в эту клетку не имеет смысла. Поставив ладью в первую непробивающуюся клетку, которая следует в лексикографическом порядке за данной, мы получим бóльшее множество пробивающихся клеток.

Построение оценок. Разобьем поле на вертикали и горизонтали. Одной горизонтали принадлежат соседние клетки одной строки, между которыми нет фигур, одной вертикали – смежные клетки одного столбца, между которыми нет фигур. Для шахматной доски (см. рис. 1.32) показаны семь горизонталей и пять вертикалей (рис. 1.34):

- первая горизонталь состоит из одной клетки (1, 1), вторая – (1, 3) и (1, 4), третья – (2, 1) и (2, 2), четвертая – (3, 1) и (3, 2), пятая – (3, 4), шестая – (4, 1) и (4, 2), седьмая – (4, 4);
- первая вертикаль состоит из клеток (1, 1), (2, 1), (3, 1) и (4, 1), вторая – (2, 2), (3, 2) и (4, 2), третья – (1, 3), четвертая – (1, 4), пятая – (3, 4), (4, 4).

Рис. 1.34

Рассмотрим способы построения нижних оценок.

1. Допустим, что нижняя оценка подзадачи равна 0. Тогда, если на некотором шаге видно, что для частичного решения, соответствующего вершине *x*, количество ладей больше либо не меньше полученного рекорда, выполним отсечение вершины *x* по рекорду.

2. Поставим в свободную клетку ладью и посчитаем число свободных клеток n_0, которые она пробивает (учитываем при подсчете и ту клетку, в которой стоит ладья). Проделаем то же для каждой свободной клетки. Пусть $n_{\max}$ – максимальное значение среди всех n_0, а s_0 – число всех свободных клеток. Тогда в качестве нижней оценки возьмем величину

$$\text{НО}\left(x^{\text{опт}}\right) = \left\lfloor \frac{s_0}{n_{\max}} \right\rfloor.$$

Для шахматного поля, приведенного на рис. 1.32, получим следующую оценку:

$$n_{\max} = 5,\ s_0 = 11,\ \text{НО}\left(x^{\text{опт}}\right) = 2.$$

3. Построим двудольный граф. Вершины первой доли – горизонтали шахматной доски, которые не заняты фигурами; вершины второй доли – вертикали доски, которые не заняты фигурами. Ребро между двумя вершинами двудольного графа проводится в том случае, если соответствующие вертикаль и горизонталь пересекаются (рис. 1.35).

Таким образом, каждому ребру двудольного графа будет соответствовать позиция шахматной доски (x, y), в которой нет фигур. Если поставить ладью в позицию (x, y) (выбрать соответствующее ребро двудольного графа), то этой ладьей будут пробивать все позиции горизонтали x и вертикали y, поэтому существует оптимальное решение, при котором в каждой горизонтали и вертикали будет стоять не более 1 ладьи.

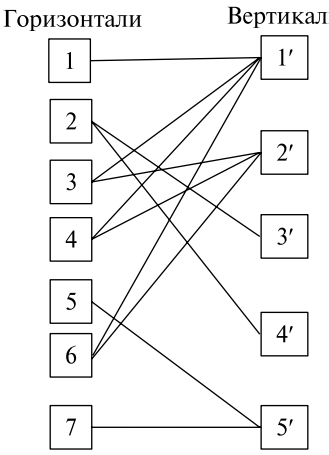


Рис. 1.35

Под *максимальным паросочетанием* понимается такое, которое не является подмножеством никакого другого паросочетания (максимальное по включению).

Для двудольного графа, приведенного на рис. 1.36, имеется два максимальных паросочетания:

$$\{\{1, 3\}\} \text{ и } \{\{1, 4\}, \{2, 3\}\}.$$

Под *наименьшим максимальным* понимается паросочетание минимальной мощности среди всех максимальных паросочетаний. Для двудольного графа (см. рис. 1.36) наименьшее максимальное паросочетание – $\{\{1, 3\}\}$.

Под *наибольшим* (*по мощности*) понимается паросочетание с наибольшим числом ребер среди всех паросочетаний графа. Для двудольного графа (см. рис. 1.36) наибольшее паросочетание – $\{\{1, 4\}, \{2, 3\}\}$. Любое наибольшее паросочетание является максимальным, но обратное верно не всегда.

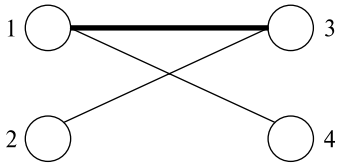


Рис. 1.36

Точной оценкой минимального числа ладей при расстановке, когда все свободные поля пробиваются, является мощность наименьшего максимального паросочетания в сгенерированном двудольном графе. Более того, по такому паросочетанию можно восстановить решение: расстановкой являются белые ладьи, соответствующие ребрам, входящим в наименьшее максимальное паросочетание. Изолированные точки отдельно не рассматриваются, так как они в любом случае обойдутся при обходе в лексикографическом порядке. В этом случае граф будет иметь столько компонент связности, сколько изолированных компонент будет иметь начальная расстановка. Оценка сначала вычисляется для каждой компоненты связности, после чего неравенства складываются почленно и получается оценка для всего графа.

Задача нахождения наименьшего максимального паросочетания является *NP*-полной, поэтому построим нижнюю оценку. Предположим, что k – мощность наименьшего максимального паросочетания M, т. е. $f\left(x^{\text{опт}}\right) = k$.

Множество конечных вершин ребер, которые вошли в максимальное паросочетание M, порождает некоторое вершинное покрытие V' графа G. Мощность этого вершинного покрытия равна $2k$.

Предположим, что для нашего двудольного графа найдена мощность k_0 минимального вершинного покрытия. Тогда справедливы следующие неравенства:

$$2k \geq k_0,\ f\left(x^{\text{опт}}\right) \geq \frac{k_0}{2} \geq \left\lfloor \frac{k_0}{2} \right\rfloor,$$

и величина $\lfloor k_0 / 2 \rfloor$ может быть взята в качестве нижней оценки для оптимального решения задачи, т. е.

$$\text{НО}\left(x^{\text{опт}}\right) = \left\lfloor \frac{k_0}{2} \right\rfloor.$$

Для двудольного графа существуют алгоритмы, которые позволяют вычислить мощность наименьшего вершинного покрытия за полино-

миальное время. С этой целью можно воспользоваться теоремой венгерского математика Денеша Кенига (1931 г.): для двудольного графа число вершин наименьшего вершинного покрытия равно числу ребер наибольшего (по мощности) паросочетания. Для нахождения наибольшего паросочетания в двудольном графе $G = (V, E)$ можно использовать одну из реализаций алгоритма Куна, в основе которой лежит алгоритм построения максимального потока в сети (время работы алгоритма – $O(|V||E|)$).

Для двудольного графа (см. рис. 1.35) мощность наибольшего паросочетания равна 4, значит, по теореме Кенига мощность наименьшего вершинного покрытия также равна 4 и нижняя оценка для оптимального решения задачи может быть получена по формуле

$$НО\left(x^{опт}\right) = \left\lfloor \frac{k_0}{2} \right\rfloor = \left\lfloor \frac{4}{2} \right\rfloor = 2.$$

Пример 1.5. На клеточном поле размером $n \times m$, где в некоторых позициях расставлено k черных фигур, необходимо расставить максимальное число белых ладей так, чтобы они не били друг друга.

Решение. Разобьем шахматную доску на вертикали и горизонтали и построим двудольный граф. Если ладья ставится в некоторую позицию доски (выбрано некоторое ребро двудольного графа), то эта ладья пробивает все позиции соответствующей вертикали и горизонтали, в свободных клетках которых в оптимальном решении не должно стоять других ладей. Следовательно, выбранное множество ребер должно образовывать паросочетание. Для того чтобы число ладей было максимальным, паросочетание должно быть наибольшим по мощности.

1.5. Минимальные связующие деревья

Рассмотрим проблему построения минимальных связующих деревьев, которая часто называется задачей Штейнера, и покажем, что для ее решения можно использовать динамическое программирование, хотя оно не является полиномиальным алгоритмом (рекомендуется учитывать структурные особенности задачи).

Исторически имеется несколько формулировок задачи Штейнера.

Задача Штейнера на евклидовой плоскости. Дано множество терминальных вершин на плоскости. Требуется найти кратчайшую сеть, соединяющую заданное множество вершин.

Важное условие: для минимизации длины связывающей сети можно добавлять еще точки, называемые *точками Штейнера* (ТШ). Кратчайшая сеть, содержащая точки Штейнера, называется *деревом Штейнера* (ДШ). Задача поиска деревьев Штейнера играет важную роль в геодезии, проектировании дорожных сетей и т. д. Задача Штейнера является *NP*-полной.

Известно несколько комбинаторных алгоритмов (Мелзака – Кокейна, Дрейфуса – Вагнера), дающих оптимальное решение задачи Штейнера за экспоненциальное время. Подобные алгоритмы плохо подходят для практического применения при больших объемах входных данных, поэтому существует большое количество алгоритмов построения деревьев Штейнера, находящих приближенные решения.

Р. Курант и Г. Роббинс наряду с формулировкой задачи для произвольного n привели два важных свойства:

1) в минимальном дереве Штейнера (МДШ) число ТШ не превышает $n - 2$;

2) степень ТШ равна 3 и инцидентные ей ребра образуют друг с другом углы 120°.

Однако данных свойств недостаточно для непосредственного построения конечной процедуры поиска МДШ, так как для любой пары точек (D, B) множество точек плоскости P, при которых угол DPB равен 120°, бесконечно.

Впервые конечная процедура построения МДШ была предложена З. Мелзаком в 1961 г. Ключевым здесь является факт: если две точки D и B соединены ребрами с точкой Штейнера P, то третье ребро, инцидентное P, проходит через вершину C. Она образует равносторонний треугольник с вершинами D и B и лежит в разных с точкой P полуплоскостях, образованных прямой, которая проходит через D и B. Таким образом, если МДШ с n вершинами таково, что вершины D и B непосредственно соединены ребрами с ТШ, то заменой пары D и B на C осуществляется редукция к задаче Штейнера с $n - 1$ вершинами.

Есть два способа выбора точки C, а также $n(n-1)/2$ способов выбора пары D, B. Учитывая, что число ТШ не превосходит $n - 2$, получается конечная процедура поиска МДШ.

Дерево Штейнера с $n - 2$ точками Штейнера называется полным (ПДШ).

Если МДШ не является ПДШ, то существует его декомпозиция на ПДШ меньшей размерности.

Дальнейшие исследования по построению точных алгоритмов решения задачи проведены Е. Кокине.

Задача Штейнера на плоскости с прямоугольной метрикой (ЗШПМ). Даная задача получается в результате замены в предыдущей постановке евклидовой метрики на прямоугольную (манхэттеновскую). В этом случае ДШ покрывает терминалы, используя только вертикальные и горизонтальные отрезки на плоскости. Расстояние между двумя точками с координатами (x_i, y_i) и (x_j, y_j) равно

$$|x_i - x_j| + |y_i - y_j|.$$

Вновь допускаются ТШ и требуется найти МДШ. В первой же публикации М. Ханана, посвященной ЗШПМ, доказано, что эта задача является частным случаем задачи Штейнера на графах (ЗШГ), которая будет рассмотрена ниже.

Дано два множества действительных чисел:

$$X = \{x_1, \ldots, x_n\},$$

$$Y = \{y_1, \ldots, y_n\}.$$

Вершинами графа-решетки $G = (V, E)$ являются точки плоскости, первая координата которых принадлежит множеству X, а вторая – Y. Упорядоченные наборы первых и вторых координат терминальных вершин образуют множества X и Y, получается граф-решетка, которая обозначается через $G(A)$.

М. Ханан доказал, что МДШ в ЗШМП является подграфом $G(A)$. Таким образом, все результаты, относящиеся к ЗШГ, применимы и к рассматриваемой постановке. Однако данная задача традиционно рассматривается отдельно.

Задача Штейнера на графах. Пусть дан граф $G = (V, E)$, где множество вершин V состоит из двух непересекающихся множеств: множества терминалов V' и множества возможных точек Штейнера S, т. е.

$$V = V' \cup S.$$

Ребрам графа приписаны неотрицательные веса, а под длиной дерева понимается сумма весов ребер, входящих в это дерево. Задача состоит в нахождении такого подграфа G', который является деревом, покрывающим все терминальные вершины, возможно, некоторые вершины из S, при этом он имеет минимальную длину среди всех подобных подграфов. Этот подграф вновь называется МДШ.

Пусть $ST(X \cup \{v\})$ соответствует длине дерева Штейнера на множестве вершин $X \cup \{v\}$, $v \in V \setminus X$, а $ST_v(X \cup \{v\})$ соответствует длине

дерева Штейнера на множестве вершин $X \cup \{v\}$ при условии, что степень вершины v не меньше 2.

Следовательно,

$$ST\left(X \cup \{v\}\right) \le ST_v\left(X \cup \{v\}\right).$$

Алгоритм Дрейфуса – Вагнера основан на использовании двух рекуррентных соотношений:

1) $ST_v\left(X \cup \{v\}\right) = \min\left(ST\left(X' \cup \{v\}\right) + ST\left(X \setminus X' \cup \{v\}\right)\right)$,

где минимум берется по всевозможным разбиениям множества X на подмножества X' и $X \setminus X'$, $v \in V \setminus X$;

2) $ST\left(X \cup \{v\}\right) = \min\left\{\min\left\{p(v,w) + ST(X)\right\}\right\}$, если $w \in X$,

$\min\left\{p(v,w) + ST_w(X)\right\}$, если $w \in V \setminus X$.

Во второй формуле минимум берется по всем вершинам w, а величина $p(v,w)$ соответствует длине минимального пути между вершинами v и w.

Трудоемкость алгоритма оценивается величиной

$$n3^k,$$

где k соответствует мощности дерева Штейнера.

Задачи для самостоятельного решения

1. Минимальное число коней

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлены черные фигуры (m – число столбцов, n – число строк).

Необходимо расставить минимальное число белых коней, чтобы пробивались все свободные позиции.

Формат входных данных

Первая строка задает размеры поля: целые числа m и n ($1 \le m, n \le 100$), вторая строка содержит число k занятых клеток ($0 \le k \le mn - 2$). В последующих k строках находятся координаты занятых клеток – пары целых чисел x_i и y_i ($1 \le x_i \le m$, $1 \le y_i \le n$).

Формат выходных данных

В первой строке должно находиться число p расстановок, следующая строка должна быть свободной. Далее выведите p досок размером $n \times m$, разделенных пробелом: 0 – свободная клетка, 1 – конь, 2 – препятствие.

Входной файл	Выходной файл	Входной файл	Выходной файл
5 5 5 2 2 2 4 3 3 4 2 4 4	2 00100 02120 10201 02120 00100		 00100 02020 11211 02020 00100

2. Минимальное число ладей

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлено k черных фигур.

Необходимо расставить минимальное число белых ладей, чтобы пробивались все свободные позиции.

Формат входных данных

В первой строке находятся целые числа n, m и k ($1 \le n, m \le 7$, $0 \le k \le nm$). В каждой из последующих k строк находится по одному целому числу x ($0 \le x \le nm$) – координата черной фигуры (нумерация координат – в лексикографическом порядке). Гарантируется, что все k координат фигур различны.

Формат выходных данных

В первой строке выведите число p всех расстановок, в каждой из последующих p строк – по одной расстановке ладей. Каждая расстановка должна быть отсортирована в лексикографическом порядке.

Входной файл	Выходной файл
3 3 2 5 6	5 1 7 1 8 1 9 2 7 3 7

3. Минимальное число ферзей

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлено k черных фигур.

Необходимо расставить минимальное число белых ферзей, чтобы пробивались все свободные позиции.

Формат входных данных

В первой строке находятся целые числа n, m и k ($1 \le n, m \le 7$, $0 \le k \le nm$). В каждой из последующих k строк находится целое число x –

координата черной фигуры (нумерация координат – в лексикографическом порядке, $1 \le x \le nm$).

Формат выходных данных

Выведите по одной в строке расстановки, отсортированные в лексикографическом порядке, в последней строке – число всех расстановок.

Входной файл	Выходной файл
3 3 2 5 6	1 7 1 8 1 9 2 7 2 8 2 9 3 7 3 8 8

4. Минимальное число слонов

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлено k черных фигур.

Необходимо расставить минимальное число белых слонов, чтобы пробивались все свободные позиции.

Формат входных данных

В первой строке находятся целые числа n, m и k ($1 \le n, m \le 7$, $0 \le k \le nm$). В каждой из последующих k строк находится целое число x – координата черной фигуры (нумерация координат производится по строкам, $1 \le x \le nm$).

Формат выходных данных

Выведите по одной в строке расстановки, отсортированные между собой в лексикографическом порядке. Числа внутри каждой расстановки также должны идти в порядке возрастания. В последней строке – число всех расстановок.

Входной файл	Выходной файл
3 3 2 5 6	1 3 4 7 9 1

5. Расстановка коней

Необходимо расставить минимальное число коней, чтобы они атаковали каждую свободную клетку доски не менее k раз. Клетка считается свободной, если она изначально пустая и в нее не ставится конь.

Формат входных данных

Первая строка содержит число k, вторая – ширину, затем высоту доски, третья – число n занятых клеток доски (препятствий). Следующие n строк содержат координаты занятых точек (строка, столбец).

Формат выходных данных

В первой строке выведите число m решений. В каждой из следующих m строк выведите решение: разделенные пробелом координаты клеток, куда следует расставить коней. Координаты должны быть упорядочены по возрастанию (приоритет координаты x (строка) больше, чем приоритет координаты y (столбец)). Строка завершается пробелом.

Входной файл	Выходной файл
1 3 3 1 1 1	8 0 0 0 1 0 2 0 0 0 1 1 0 0 0 1 0 2 0 0 1 0 2 1 2 0 2 1 2 2 2 1 0 2 0 2 1 1 2 2 1 2 2 2 0 2 1 2 2

6. Расстановка ферзей

Необходимо расставить минимальное число ферзей, чтобы они атаковали каждую свободную клетку доски не менее k раз. Клетка считается свободной, если она изначально пустая. Следует учитывать, что на доске возможны препятствия, которые сузят область пробитых клеток у ферзей. При этом другие ферзи такими препятствиями друг для друга не являются. Клетка, занятая ферзем, также должна быть атакована не менее k раз, включая атаку стоящего на ней ферзя.

Формат входных данных

Первая строка содержит число k, вторая – ширину w, затем высоту h доски, третья – число n занятых клеток доски (препятствий). Следующие n строк содержат координаты занятых точек (строка, столбец). Гарантируется, что $1 \le k$, $1 \le w$, $1 \le h$, $0 \le n$, $n + k \le wh$, $wh \le 100$, $w + h \le 25$.

Формат выходных данных

В первой строке выведите m решений. Если решений нет, то $m = 0$. В каждой из следующих m строк выведите решение: разделенные пробелом координаты позиций, куда следует расставить ферзей. Позиции нужно выводить в порядке возрастания x-координат. В случае равенства

x-координат – в порядке возрастания y-координат. Строка не должна заканчиваться пробелом.

Входной файл	Выходной файл
2 4 4 2 1 2 2 1	4 0 0 0 3 3 0 3 3 0 0 0 3 2 2 3 0 0 3 1 1 3 0 3 3 0 0 1 1 2 2 3 3

7. Максимальное число ладей

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлены черные фигуры. Необходимо расставить максимальное число белых ладей, чтобы они не били друг друга.

Формат входных данных

В первой строке указаны через пробел n, m и k, где n и m – размеры поля, k – число черных фигур. В каждой из последующих k строк находится два числа – i и j, разделенных пробелом, где (i, j) – координаты черной фигуры (считается, что левая верхняя клетка имеет координаты (1, 1)).

Формат выходных данных

Для одной расстановки выведите построчно координаты всех белых ладей в формате (j, пробел, i), затем – комбинацию символов `<--->`, в конце – число полученных расстановок.

Входной файл	Выходной файл	Входной файл	Выходной файл
3 3 4 2 2 2 3 3 2 3 3	3 1 1 3 <---> 3 1 1 2 <--->		2 1 1 3 <---> 2 1 1 2 <---> 4

8. Максимальное число ферзей

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлены черные фигуры. Необходимо расставить максимальное число белых ферзей, чтобы они не били друг друга.

Формат входных данных

В первой строке находится целое число n $(1 \le n \le 10)$, во второй – целое число m $(1 \le m \le 10)$, в третьей – число b черных фигур. В каждой

из последующих b строк находится по два числа, разделенных пробелом, – координаты черной фигуры (левая верхняя клетка имеет координаты (0, 0)).

Формат выходных данных

В первой строке выведите максимальное число ферзей, во второй – число p решений. В каждой из последующих p строк выведите последовательность чисел, разделенных пробелами, где каждая пара чисел – координаты ферзя. Последовательности выведите в лексикографическом порядке.

Входной файл	Выходной файл
4 4 2 0 1 1 2	4 2 0 0 0 2 2 3 3 1 0 2 1 0 2 3 3 1

9. Максимальное число слонов

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлены черные фигуры.

Необходимо расставить максимальное число белых слонов, чтобы они не били друг друга.

Формат входных данных

В первой строке находится целое число m, во второй – целое число n $(1 \le m, n \le 10)$. В каждой из последующих m строк находится по n чисел 0 или 1, разделенных пробелом: 0 – свободная клетка, 1 – занятая.

Формат выходных данных

В первой строке выведите число k решений, во второй – число слонов в решении. В каждой из последующих k строк выведите последовательность чисел, разделенных пробелами, где каждая пара чисел – координаты слона в лексикографическом порядке.

Входной файл	Выходной файл
4 6 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0	5 12 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 1 2 2 2 3 2 5 3 5 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 2 2 3 2 5 3 0 3 5 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 2 2 5 3 0 3 2 3 5 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 2 3 0 3 2 3 4 3 5 0 0 0 2 0 3 0 5 1 0 1 5 2 0 2 2 2 3 2 5 3 0 3 5

10. Кратчайшие маршруты коня

Имеется шахматное поле размером $n \times m$, где некоторые позиции могут быть заняты фигурами.

Необходимо найти все кратчайшие маршруты коня между двумя заданными позициями.

Формат входных данных

В первой строке находится число n столбцов $\left(1 \le n \le 10^6\right)$, во второй – число m строк $\left(1 \le m \le 10^6,\ mn \le 2 \cdot 10^6\right)$, в третьей – число k фигур $\left(0 \le k,\ nk \le 200,\ mk \le 200\right)$. В каждой из последующих k строк находится по два числа: номера столбца и строки (начиная с 0), где находится фигура. В последних двух строках указаны начальная и конечная позиции коня (номера столбца и строки начинаются с 0).

Формат выходных данных

Если решение существует, то последовательно выведите все возможные кратчайшие маршруты в следующем формате. В первой строке для каждого маршрута выведите число s ходов коня, а в следующих $s+1$ строках – клетки этого маршрута (номера столбца и строки начинаются с 0), в конце файла – символ перевода строки. Если решений нет, то выведите *no_solutions*.

Поскольку порядок вывода путей неоднозначен, то необходимо его определить. Примите клетку $(0,0)$ за левый верхний угол, а $\left(n-1,\ m-1\right)$ – за правый нижний. Рассмотрите первый ход. Пути выведите в порядке направления движения: вправо – вправо – вниз, вправо – вниз – вниз, влево – вниз – вниз, влево – влево – вниз, влево – влево – вверх, влево – вверх – вверх, вправо – вверх – вверх, вправо – вправо – вверх. Если же есть несколько решений, у которых первый ход совпадает, то аналогичным образом сортируйте их по второму ходу, далее – по третьему и т. д.

Входной файл	Выходной файл
5 5 3 1 2 2 1 2 4 0 4 4 0	4 0 4 2 3 4 4 3 2 4 0 4 0 4 2 3 1 1 3 2 4 0

11. Кратчайшие маршруты ладьи

Имеется шахматное поле размером $n \times m$, где некоторые позиции могут быть заняты фигурами.

Необходимо найти все кратчайшие маршруты ладьи между двумя заданными позициями.

Формат входных данных

В первой строке находится число n столбцов, во второй – число m строк, в третьей – число k фигур. В каждой из последующих k строк находится по два числа: номера столбца и строки (начиная с 0) – координаты фигуры. В последних двух строках указаны начальная и конечная позиции ладьи (номера столбца и строки начинаются с 0).

Формат выходных данных

Если решение существует, то последовательно выведите все возможные кратчайшие маршруты в следующем формате. В первой строке для каждого маршрута выведите число s ходов ладьи, а в следующих $s+1$ строках – клетки этого маршрута (номера столбца и строки начинаются с 0), в конце файла – символ перевода строки. Если решений нет, то выведите *no_solutions* (без символа перевода строки).

Поскольку порядок вывода маршрутов неоднозначен, то необходимо его определить. Примите клетку $(0,0)$ за левый верхний угол, $(n-1,\ m-1)$ – за правый нижний. Рассмотрите первый ход.

Маршруты выведите в порядке направления движения (главный параметр сортировки): вправо – вниз – влево – вверх; по длине хода (вторичный параметр сортировки). Если же есть несколько решений, у которых первый ход совпадает, то аналогичным образом их сортируйте по второму ходу, далее по третьему и т. д.

Входной файл	Выходной файл
8 6 8 1 1 1 4 3 0 3 5 4 0 4 5 6 1 6 4 0 0 7 5	3 0 0 0 2 7 2 7 5 3 0 0 0 3 7 3 7 5

12. Кратчайшие маршруты ферзя

Имеется шахматное поле размером $n \times m$, где некоторые позиции могут быть заняты фигурами.

Необходимо найти все кратчайшие маршруты ферзя между двумя заданными позициями. Под кратчайшим маршрутом понимается тот, который содержит минимально возможное число ходов ферзя.

Формат входных данных

В первой строке находится число n столбцов, во второй – число m строк, в третьей – число k черных фигур.

В каждой из последующих k строк находится по два числа (номера столбца и строки, где находится черная фигура, начинаются с 0). В последних двух строках указаны начальная и конечная позиции ферзя (номера столбца и строки начинаются с 0). Гарантируется, что $1 \le n,\ m \le 1024,\ nm \le 60\,000$.

Формат выходных данных

Если решение существует, то последовательно выведите все возможные пути в следующем виде: в первой строке – число s ходов ферзя, в следующих $s+1$ строках – клетки пути (номера столбца и строки начинаются с 0), в конце файла – символ перевода строки. Если решений нет, то выведите *no_solutions* (без символа перевода строки).

Поскольку порядок вывода путей неоднозначен, то необходимо его определить. Примите клетку $(0, 0)$ за левый верхний угол, $(m-1,\ n-1)$ – за правый нижний. Рассмотрите первый ход.

Пути выводите в порядке направления движения (главный параметр сортировки): вправо, вправо – вниз, вниз, влево – вниз, влево, влево – вверх, вверх, вправо – вверх; длине хода (вторичный параметр сортировки). Если же число решений, у которых первый ход совпадает, больше одного, то аналогичным образом их сортируйте по второму ходу, далее – по третьему.

Входной файл	Выходной файл
3 3 3 0 0 1 1 2 2 0 2 2 0	3 0 2 1 2 2 1 2 0 3 0 2 0 1 1 0 2 0

13. Эйлеровы циклы

Необходимо найти все эйлеровы циклы в неориентированном графе.

Формат входных данных

Первая строка содержит число n вершин и число m ребер графа $\left(1 \le n \le 100,\ 0 \le m \le n(n-1)/2\right)$. Следующие m строк содержат по одному ребру, заданному как два номера концевых вершин, разделенных пробелом.

Формат выходных данных

В первой строке выведите число k эйлеровых циклов в заданном графе. В каждой из следующих k строк выведите по одному эйлерову циклу в виде подпоследовательности, содержащей все вершины, разделенные пробелами. Для каждого цикла выберите только одну соответствующую последовательность вершин – лексикографически минимальную. Циклы выводите в лексикографическом порядке. Если $k > 10\,000$, выведите только первые 10 000 циклов.

Входной файл	Выходной файл
5 7 1 4 1 5 2 4 2 5 3 4 3 5 4 5	6 1 4 2 5 3 4 5 1 1 4 2 5 4 3 5 1 1 4 3 5 2 4 5 1 1 4 3 5 4 2 5 1 1 4 5 2 4 3 5 1 1 4 5 3 4 2 5 1

14. Разрезы доски

Необходимо найти все возможные разрезы доски размером $n \times n$ ($n = 2k$) на две одинаковые по форме связные фигуры. *Доской* назовем множество точек координатной плоскости *OXY*, для которых $\max\{|x|, |y|\} \le k$; *границей доски* – множество точек координатной плоскости *OXY*, для которых $\max\{|x|, |y|\} \le k$; *разрезом доски* – ломаную, удовлетворяющую следующим аксиомам:

- ее конечные точки лежат на границе доски, а все остальные – внутри;
- координаты конечных точек любого звена этой ломаной – целые числа;
- любое ее звено параллельно одной из осей;
- ломаная незамкнута и не имеет самопересечений и самокасаний.

Фигуру назовем *связной*, если для любых двух ее внутренних точек существует кривая, соединяющая эти точки и состоящая только из внутренних точек этой фигуры. Две фигуры одинаковы по форме, если при помощи преобразований параллельного переноса, поворота и симметрии можно получить совпадающие фигуры.

Формат входных данных

В единственной строке находится натуральное число k $(1 \le k \le 4)$.

Формат выходных данных

Выведите все различные разрезы доски по одному в строке. Если ломаная $L_1 L_2 \ldots L_n$ – разрез доски (причем считается, что все звенья ломаной имеют длину 1), то формат выходной строки следующий:

$$x_1 y_1 \cdot d_2 d_3 \ldots d_n$$

(между x_1 и y_1, y_1 и d_2 должно стоять по одному пробелу),

где x_1, y_1 – целочисленные координаты начальной точки L_1 разреза;

$$d_i = \text{R, если } x_i - x_{i-1} = 1,\ y_i - y_{i-1} = 0;$$

$$d_i = \text{U, если } x_i - x_{i-1} = 0,\ y_i - y_{i-1} = 1;$$

$$d_i = \text{L, если } x_i - x_{i-1} = -1,\ y_i - y_{i-1} = 0;$$

$$d_i = \text{D, если } x_i - x_{i-1} = 0,\ y_i - y_{i-1} = -1,$$

где (x_i, y_i) – координаты точки L_i.

Входной файл	Выходной файл
1	-1 0 RR 0 -1 UU

15. Магические квадраты

Необходимо составить из костяшек одного набора домино все магические квадраты размером $n \times n$ (квадратные числовые таблицы, в которых суммы элементов во всех строках, столбцах и на двух главных диагоналях совпадают). Костяшку можно класть только горизонтально, она занимает две клетки по горизонтали и одну по вертикали. Каждая костяшка из набора в каждом отдельно взятом квадрате используется не более одного раза.

Формат входных данных

Единственная строка содержит число n $(1 \le n \le 4)$.

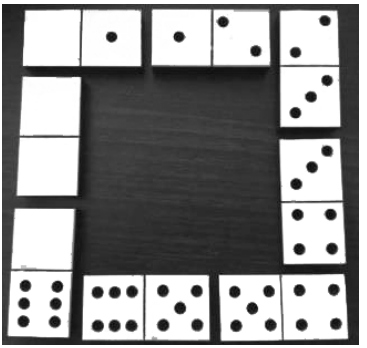
Формат выходных данных

Выведите полученные квадраты следующим образом. Каждый магический квадрат должен занимать $n + 1$ строк: первая строка содержит восемь символов «-» (дефисоминус, ASCII-код 45) и служит для разделения квадратов, а последующие четыре строки – соответствующие строки самого квадрата (в каждой – по четыре числа, обозначающих числа на костяшках домино). Пустой клетке костяшки соответствует число 0. Соседние числа разделяются одним пробелом. После последнего квадрата строка `--------` не выводится, но выводится перед первым квадратом. Последняя строка файла имеет вид Count = x, где x – число найденных магических квадратов. Порядок вывода магических квадратов не имеет значения.

Входной файл	Выходной файл
4	-------- 0 0 6 0 1 4 0 1 1 2 0 3 4 0 0 2 -------- и так далее

16. Цепочки домино

Необходимо составить из стандартного набора костяшек домино все возможные замкнутые цепочки. Костяшки соприкасаются гранями сторон с одинаковыми числами. Под замкнутой цепочкой подразумевается прямоугольная рамка. При этом разные прямоугольные рамки, полученные «сворачиванием» одной и той же линейной последовательности костей домино, считаются одинаковыми цепочками.



Формат входных данных

Единственная строка содержит длину цепочки – число костяшек домино, из которых составляется замкнутая цепочка.

Формат выходных данных

Если входные данные заданы некорректно, выведите строку *wrong count*; если корректно – выведите все различные цепочки в любом порядке (две цепочки различны, если не принадлежат одной группе симметрии, т. е. одна не может быть получена из другой циклическим сдвигом и/или разворотом), а в последней строке выведите количество цепочек.

Входной файл	Выходной файл	Входной файл	Выходной файл
2	wrong count		6 6.0 3 3.0 4 4.0 и так далее 210
4	2 2.0 3 3.0 4 4.0 5 5.0		

17. Таблица из костей домино

Даны две таблицы с числами.

Необходимо покрыть их с помощью набора костяшек домино. Костяшки в наборе повторяться не должны.

Формат входных данных

В первой строке находится два целых неотрицательных числа первой таблицы: m_1 и n_1 – число строк и столбцов соответственно. В каждой из последующих m_1 строк находится по n_1 чисел, разделенных пробелом, – это элементы первой таблицы.

Далее следуют два целых неотрицательных числа второй таблицы: m_2 и n_2 – число строк и столбцов соответственно. В каждой из последующих m_2 строк находится по n_2 чисел, разделенных пробелом, – элементы второй таблицы.

Формат выходных данных

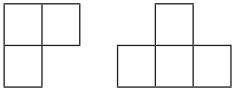
В первой строке должно быть YES, если решение есть для данной задачи, и NO – решения нет. Если решение существует, то выведите число k всевозможных расстановок костяшек домино по элементам таблиц. Затем выведите k различных расстановок костей домино для данных таблиц: по $m_1 + m_2$ строк – строки первой и второй таблиц (таблицы разделены одной пустой строкой), где элементами являются порядковые номера костей домино, покрывающих данную ячейку исходной таблицы, причем поле заполняется слева направо и сверху вниз, начиная выкладывать кости с меньшим порядковым номером. Варианты расстановки разделяются двумя пустыми строчками.

Входной файл	Выходной файл	Входной файл	Выходной файл
2 2 2 3 2 5 2 2 6 5 0 1	YES 4 1 2 1 2 3 4 3 4 1 2 1 2 3 3 4 4		 1 1 2 2 3 4 3 4 1 1 2 2 3 3 4 4

18. Замощение фигурами

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлены черные фигуры.

Необходимо замостить его такими фигурами:

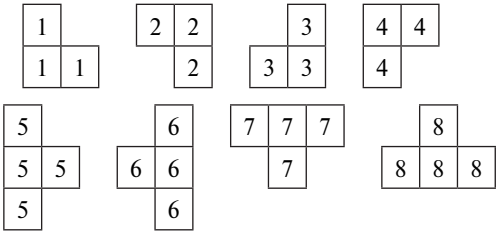


Формат входных данных

В первой строке находится число m столбцов $(2 \le m \le 20)$, во второй – число n строк $(2 \le n \le 20)$. Сначала указывается число столбцов, а затем – строк. Последующие n строк длины m описывают исходное клеточное поле: 0 – свободная клетка, 1 – занятая.

Формат выходных данных

В первой строке выведите число k решений данной задачи (0, если решений нет). Далее выведите k вариантов замощения исходного поля фигурами, которые отображаются следующим образом:



Занятая клетка обозначается символом 0. Варианты замощения должны отделяться друг от друга пустой строкой.

Входной файл	Выходной файл
5 5 00000 00000 00001 00000 00000	0 12222 11232 22330 12122 11112 12222 11232 22330 12443 11433 и так далее

19. Раскраска графа

Задан неориентированный граф без петель и кратных ребер.

Необходимо раскрасить вершины этого графа в минимально возможное число цветов, чтобы смежные вершины имели разные цвета.

Формат входных данных

В первой строке указано число n вершин графа. Если граф состоит только из вершин, то больше в файле ничего нет. Пронумеруем вершины графа произвольным образом. Следующие строки входного файла содержат по два числа i и j через пробел, если вершины i и j соединены ребром, при этом $i \neq j$.

Формат выходных данных

Каждому цвету поставьте в соответствие произвольный номер, начиная с 1 и далее по порядку. Число цветов не больше n. В первой строке выведите минимальное количество цветов, в которые данный граф раскрашивается при выполнении требований, указанных в условии. Во второй строке – номера цветов через пробел в порядке следования вершин, т. е. сначала номер цвета 1-й вершины, затем 2-й вершины и т. д.

Входной файл	Выходной файл
5 1 2 1 5 2 3 3 4 4 5	3 1 2 1 2 3

20. Расстановка фишек

Имеется клеточное поле размером $n \times m$, где в некоторых позициях расставлены черные фигуры.

Необходимо расставить на клеточном поле всевозможными способами фишки таким образом, чтобы в каждой линии (горизонтальной, вертикальной, диагональной) располагалось четное число фишек.

Формат входных данных

В первой строке находятся целые числа n и m ($n \geq 1$, $m \geq 1$). Последующие n строк содержат таблицу размером $n \times m$ с элементами: 0 – пустое место, (–1) – занятое черной фигурой.

Формат выходных данных

Каждый вариант расстановки фишек записывается в виде таблицы размером $n \times m$ с элементами: 0 – свободная клетка; (–1) – занятая черной фигурой; 1 – клетка, на которой стоит фишка. Таблицы разделяйте пустыми строками.

Входной файл	Выходной файл
4 4 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 -1 0 1 1 0 0 1 0 1 1 0

21. Детали – станки

Имеется n деталей и m станков. Каждая деталь характеризуется одинаковым временем обработки на всех станках. На каждом станке последовательно в каждый момент времени обрабатывается только одна деталь.

Необходимо определить такое назначение деталей на станки, чтобы время окончания обработки последней детали было бы минимальным.

Формат входных данных

В первой строке находятся два целых числа: n и m ($1 \leq n, m \leq 20$), где n – число деталей, m – число станков; во второй строке – n чисел (от 1 до 10^7) – время обработки каждой детали.

Формат выходных данных

В первой строке выведите минимальное время окончания обработки всех деталей. Далее последовательно опишите одно из возможных назначений деталей на станки: m строк, каждая из которых описывает

назначение деталей на соответствующий станок (1-я строка – 1-й станок, 2-я – 2-й и т. д.). Значения, которые содержит данная строка, – время обработки, соответствующее деталям, назначенным на данный станок. В случае отсутствия деталей на каком-либо станке строка, соответствующая назначению деталей на текущий станок, должна содержать единственное число (–1).

Входной файл	Выходной файл
4 2 2 3 4 2	6 3 2 2 4

22. Склад

Компания имеет большой прямоугольный склад, где есть рабочий и менеджер. Стороны склада в порядке обхода называются соответственно расположению на схеме: левая, верхняя, правая и нижняя. Площадь склада разбита на квадраты с одинаковой площадью, составляющие строки и столбцы, которые пронумерованы целыми числами $1, 2$ и т. д. сверху вниз и слева направо соответственно. На складе находятся контейнеры, имеющие попарно различные идентификационные номера. Каждый контейнер занимает ровно один квадрат. Склад настолько большой, что число прибывающих контейнеров меньше, чем число строк и число столбцов. Контейнеры не удаляются со склада, но иногда прибывают новые. Вход на склад – в левом верхнем углу. Рабочий разместил контейнеры вокруг входа таким образом, чтобы можно было найти их по их идентификационным номерам. Допустим, k – идентификационный номер следующего контейнера, который нужно вставить. Рабочий идет вдоль первой строки и ищет первый контейнер с идентификационным номером больше чем k. Если он не находит такой контейнер, то ставит его непосредственно за самым правым контейнером в строке. Если такой контейнер t найден, то он заменяется контейнером k, а контейнер t вставляется в следующую строку с помощью того же метода. Если рабочий достигает строки, в которой нет ни одного контейнера, он ставит контейнер в позицию самого левого квадрата этой строки. Предположим, что контейнеры $3, 4, 9, 2, 5, 1$ прибыли на склад в таком порядке. Тогда размещение контейнеров на складе будет следующим:

1	4	5
2	9	
3		

К рабочему подходит менеджер, и происходит следующий диалог:

– Контейнер 5 прибыл перед контейнером 4?

– Нет, это невозможно.

– Вы можете рассказать мне порядок прибытия контейнеров по их размещению?

– Нет. Контейнеры, находящиеся на складе, могли прибыть в порядке 3, 2, 1, 4, 9, 5, или в порядке 3, 2, 1, 9, 4, 5, или в одном из 14 других порядков.

Необходимо помочь менеджеру написать программу, которая по данному размещению контейнеров вычислит все возможные порядки их прибытия.

Формат входных данных

Первая строка содержит *r* строк, в которых размещены контейнеры. Следующие *r* строк содержат информацию о строках 1, 2, ..., *r*, начиная с верхней, в следующем порядке: в начале каждой из них находится число *m* контейнеров; за ним находятся *m* целых чисел – идентификационные номера контейнеров в строке, начиная слева (все идентификационные номера *i* контейнеров удовлетворяют условию $1 \le i \le 50$. Число *n* контейнеров на складе удовлетворяет условию $1 \le i \le 15$.

Формат выходных данных

Выведите все возможные порядки прибытия контейнеров (по одному в строке). Каждая из строк должна содержать *n* целых чисел, т. е. идентификационных номеров контейнеров в соответствующем порядке прибывания. Все строки должны быть уникальны.

Входной файл	Выходной файл
2 2 1 2 1 3	3 1 2 1 3 2

23. Упаковка семи деталей

Петя и Вася очень любят решать головоломки. Они уже разгадали не одну сотню японских кроссвордов, судоку и других подобных задач. Однажды мальчики собрались вместе и долго думали, какую же головоломку им решать. Петя достал с полки семь прямоугольников из бумаги. Они уложили их в прямоугольник минимальной площади, долго думали и решили упростить задачу. Для этого установили в ней следующее ограничение: располагать прямоугольники только параллельно осям координат, включая прямоугольник минимальной площади, в который они хотели вставить все эти прямоугольники.

Дело было вечером, поэтому Вася и Петя разошлись по домам, но напоследок поспорили: кто быстрее разложит семь прямоугольни-

ков в прямоугольник минимальной площади, договорившись о размерах этих семи прямоугольников. Помогите мальчикам решить эту задачу.

Необходимо написать программу, которая находила бы расстановку для любых размеров. Для заданных размеров прямоугольников надо определить минимальную площадь прямоугольника, в который можно уложить эти прямоугольники.

Формат входных данных

В каждой из семи строк имеется по два целых числа – w и h, задающих размеры прямоугольников $(0 \le w,\ h \le 100)$.

Формат выходных данных

В первой строке выведите минимальную площадь прямоугольника, в который можно уложить данные прямоугольники.

Входной файл	Выходной файл
5 3 8 2 3 4 1 6 5 3 8 4 3 5	114

24. Минимальные остовные деревья

Необходимо построить все минимальные остовные деревья в графе.

Формат входных данных

В первой строке находятся два целых числа – n и m, разделенные пробелом, где n – число вершин, m – число ребер графа. В последующих m строках находится по три целых числа – i, j и d, где d – вес ребра, соединяющего вершины i и j.

Формат выходных данных

В первой строке выведите число k минимальных остовных деревьев данного графа, в последующих k строках – по $n - 1$ целому числу, каждое из которых соответствует номеру ребра. Числа разделены пробелом.

Номера ребер определяются порядком их ввода.

Входной файл	Выходной файл
4 4 1 2 4 2 3 4 3 4 4 1 4 4	4 1 2 3 4 3 2 1 4 3 1 2 4

В примере ребра {1, 2}, {2, 3}, {3, 4}, {1, 4} имеют соответственно порядковые номера 1, 2, 3 и 4.

25. Распределенная задача с сервером

Есть сервер и два процессора. Существует n_1 задач, которые могут выполняться только на первом процессоре, и n_2 задач, которые могут выполняться только на втором процессоре.

Перед выполнением каждая задача сначала должна загрузиться с сервера на процессор, что занимает время $s(k, i)$ для задачи с номером i, выполняемой только на процессоре k. При этом сервер и процессор k могут заниматься только загрузкой этой задачи. Сразу после загрузки процессор начинает выполнение поступившей задачи, что занимает у него время $p(k, i)$.

Затем на процессор может загружаться следующая предназначенная для него задача, т. е. в каждый момент времени сервер может заниматься загрузкой не более одной задачи, а каждый процессор – выполнять не более одной задачи. Одновременно загружать одну задачу и выполнять другую невозможно.

Необходимо найти такое расписание, при котором выполнялись бы все задачи, а время окончания выполнения последней было бы наименьшим.

Формат входных данных

В первой строке записаны числа n_1 и n_2. Далее идет n_1 пар чисел: $p(1,i)$ и $s(1, i)$, затем – n_2 пар чисел: $p(2, i)$ и $s(2, i)$.

Формат выходных данных

В первую строку выведите время окончания выполнения последней задачи. В последующих $n_1 + n_2$ строках – порядок, в котором задачи загружаются с сервера и задаются номером процессора и номером задачи. Нумеруются задачи с 1 в порядке, заданном входным файлом.

Входной файл	Выходной файл
1 2 2 5 4 1 5 5	15 2 2 1 1 2 1

26. Нераспределенная задача с сервером

Есть один сервер и два процессора, а также n задач, каждая из которых сначала должна быть загружена с сервера на любой процессор и сразу исполняться. При этом сервер может заниматься не более одной загрузкой, а каждый процессор – исполнением не более одной за-

дачи в каждый момент времени. Процессор не может одновременно исполнять одну задачу и загружать другую. Для задачи i известно время s_i загрузки с сервера на процессор $(s_i > 0)$ и время p_i исполнения на процессоре $(p_i > 0),\ i = 1, 2, ..., n$.

Необходимо найти такое расписание, при котором выполнялись бы все задачи, а время окончания выполнения последней было бы наименьшим.

Формат входных данных

В первой строке записано число n. Далее идет n строк, в каждой – по паре чисел: s_i и p_i.

Формат выходных данных

В первой строке выведите время окончания выполнения последней задачи. В последующих $n_1 + n_2$ строках – порядок, в котором задачи загружаются с сервера. Задача задается номером процессора и номером задачи. Нумеруются задачи с 1 в порядке, заданном входным файлом.

Входной файл	Выходной файл
4 12 3 8 7 3 12 1 14	33 3 1 4 2

27. Open-shop+

Имеется n работ и m приборов. Требуется составить такой маршрут обслуживания всех работ на каждом приборе, чтобы время выполнения последней работы было минимальным. В любой момент времени каждая работа должна обслуживаться не более чем одним прибором, а каждый прибор – выполнять не более одной работы. Никакой прибор не может простаивать при наличии работы, готовой к выполнению на нем.

Формат входных данных

В первой строке записаны два числа – n и m. Далее следует n строк по m чисел в каждой – время выполнения i-й работы на j-м приборе.

Формат выходных данных

Выведите минимальное время выполнения всех n работ на m приборах.

Входной файл	Выходной файл
3 4 4 12 4 6 8 6 2 10 2 4 18 3	27

28. Мультиразрез

Задан взвешенный неориентированный граф и зафиксировано некоторое подмножество T его вершин.

Необходимо удалить из графа множество ребер с минимальным суммарным весом, чтобы все вершины множества T оказались в разных компонентах связности.

Формат входных данных

Первая строка содержит число n вершин и число m ребер графа. Следующие m строк файла содержат тройки чисел (u, v, w), свидетельствующие о наличии ребра между вершинами u и v с весом w $(1 \le u, v \le n, 0 \le w \le 1\,000\,000)$. В строке $m + 2$ находится число k вершин, входящих во множество T $(2 \le k \le n)$. В следующей строке через пробел записано k различных чисел от 1 до n, которые являются номерами вершин, входящих во множество T.

Формат выходных данных

В первой строке выведите минимальный возможный вес удаляемого множества, во второй – число r ребер, входящих в это множество. В следующих r строках выведите номера ребер, входящих в это множество (по одному в строке). Ребра нумеруются с единицы в таком порядке, как были заданы во входном файле. Если существует несколько решений с минимальным весом удаляемого множества, выведите любое из них.

Входной файл	Выходной файл	Входной файл	Выходной файл
6 7 1 2 3 2 3 1 2 6 3 5 6 1	9 4 2 4 5	3 5 5 3 4 2 4 5 3 3 1 3 5	6

29. Job-shop

Есть n станков и 2 работы. Каждая работа состоит из s_i этапов, которые характеризуются парой (k, t) чисел, где k – номер станка, t – продолжительность этапа. Для работы порядок этапов строго задан. Каждый этап можно приостановить в любой момент и позже продолжить. В любой момент времени работа может выполняться только на одном станке, который выполняет только одну работу.

Необходимо составить такое расписание, чтобы все работы были выполнены за минимальное время.

Формат входных данных

В первой строке содержится единственное число n $(2 \le n \le 5)$, в следующих двух – на первом месте записано число s_i $(0 \le s_i \le 3000)$, а за ним – s_i пар (k, t) через пробел.

Формат выходных данных

Выведите минимальное время выполнения всех n работ на m приборах.

Входной файл	Выходной файл
4 3 1 1 2 1 3 3 4 1 1 2 3 4 1 3 1	7

30. Пятнашки

Имеется клеточное поле размером $n \times m$, где в клетках произвольным образом расставлены $nm - 1$ костяшек, пронумерованных целыми числами 1, 2, ..., $nm - 1$. Одна клетка свободная. На нее можно передвигать по горизонтали или вертикали соседние костяшки так, чтобы они располагались по возрастанию (слева направо по строкам, сверху вниз по столбцам), а правая нижняя угловая клетка стала пустой.

Необходимо определить кратчайшее решение. Гарантируется, что решение существует.

Формат входных данных

В первой строке находится число n строк, во второй – число m столбцов. В каждой из последующих nm строк находятся числа от 0 до $nm - 1$, которые расположены в соответствующих клетках. Порядок обхода клеток: от верхней угловой, слева направо по строкам, сверху вниз по столбцам. Число 0 означает, что клетка пустая. Гарантируется, что $nm \le 64$.

Формат выходных данных

В первой строке выведите минимальное число k ходов. В следующих k строках запишите номера костяшек, которые передвигались.

Входной файл	Выходной файл	Входной файл	Выходной файл
3 3 1 0 3 4	3 2 5 8	2 6 7 5 8	

31. *n* работ, 1 исполнитель

Есть *n* работ и 1 работник. Для каждой работы известно время поступления и время выполнения. Штраф за работу – разность между временем ее начала и временем поступления.

Необходимо минимизировать суммарный штраф.

Формат входных данных

В первой строке выводится минимальный возможный штраф. В следующих строках – все последовательности выполнения работ, на которых достигается минимальный штраф. Последовательность каждой работы должна быть отсортирована в лексикографическом порядке. Номер работы отсчитывается от 1.

Формат выходных данных

Выведите минимальное время выполнения всех *n* работ на *m* приборах.

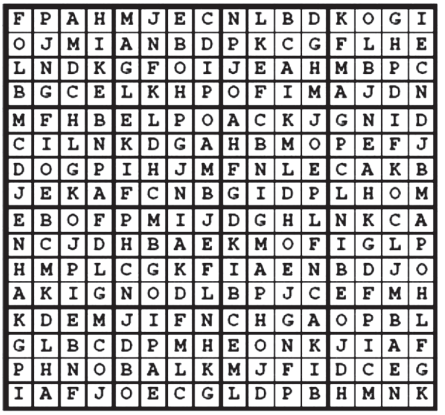
Входной файл	Выходной файл
5 0 10 0 1 1 1 2 1 3 1	4 2 3 4 5 1

32. Судоку

Судоку – это головоломка из ячеек, сгруппированных в 16 квадратов 4 × 4, где некоторые квадраты заполнены буквами от *A* до *P* (первые 16 букв латинского алфавита), как показано на рисунке.

		A					C						O		I
	J			A		B		P		C	G	F		H	
		D			F		I		E					P	
	G		E	L		H					M		J		
				E					C			G			
	I			K		G	A		B				E		J
D		G	P			J		F					A		
	E				C		B			D	P			O	
E			F		M			D			L		K		A
	C									O		I		L	
H		P		C			F		A			B			
			G		O	D				J					H
K				J					H		A		P		L
		B			P			E			K			A	
	H			B			K			F	I		C		
		F				C			D			H		N	

Игра заключается в том, чтобы заполнить все пустые ячейки буквами от *A* до *P* так, чтобы каждая из них встретилась только один раз в строке, столбце и квадрате. Первоначальная расстановка букв такова, что удовлетворяет ограничениям, указанным выше, и гарантирует существование решения (см. рисунок).



F	P	A	H	M	J	E	C	N	L	B	D	K	O	G	I
O	J	M	I	A	N	B	D	P	K	C	G	F	L	H	E
L	N	D	K	G	F	O	I	J	E	A	H	M	B	P	C
B	G	C	E	L	K	H	P	O	F	I	M	A	J	D	N
M	F	H	B	E	L	P	O	A	C	K	J	G	N	I	D
C	I	L	N	K	D	G	A	H	B	M	O	P	E	F	J
D	O	G	P	I	H	J	M	F	N	L	E	C	A	K	B
J	E	K	A	F	C	N	B	G	I	D	P	L	H	O	M
E	B	O	F	P	M	I	J	D	G	H	L	N	K	C	A
N	C	J	D	H	B	A	E	K	M	O	F	I	G	L	P
H	M	P	L	C	G	K	F	I	A	E	N	B	D	J	O
A	K	I	G	N	O	D	L	B	P	J	C	E	F	M	H
K	D	E	M	J	I	F	N	C	H	G	A	O	P	B	L
G	L	B	C	D	P	M	H	E	O	N	K	J	I	A	F
P	H	N	O	B	A	L	K	M	J	F	I	D	C	E	G
I	A	F	J	O	E	C	G	L	D	P	B	H	M	N	K

Необходимо написать программу, которая решит судоку.

Формат входных данных

На входе 16 строк по 16 символов в строке, где *i*-я строка ввода соответствует *i*-й строке решетки судоку и имеет длину 16 символов, начиная с первой позиции в строке. Символы в каждой строке имеют значения `A`, `B`, ..., `P` и «`-`», где символ «`-`» обозначает свободную позицию решетки.

Формат выходных данных

Выведите все решения головоломки, каждое из которых состоит из 16 строк по 16 символов в каждой. Решения можно выводить в любом порядке, различные – разделять свободной строкой.

В конце файла (после последнего решения и перевода строки после него) выведите (`-1`). Гарантируется, что существует хотя бы одно решение головоломки. Обязательно соблюдайте формат вывода, описанный выше.

Входной файл	Выходной файл

`--A----C-----O-I`
`-J--A-B-P-CGF-H-`
`--D--F-I-E----P-`
`-G-EL-H----M-J--`
`----E----C--G---`
	`FPAHMJECNLBDKOGI`
	`OJMIANBDPKCGFLHE`
	`LNDKGFOIJEAHMBPC`
	`BGCELKHPOFIMAJDN`
	`MFHBELPOACKJGNID`

Входной файл	Выходной файл
-I--K-GA-B---E-J D-GP--J-F----A-- -E---C-B--DP--O- E--F-M--D--L-K-A -C--------O-I-L- H-P-C--F-A--B--- ---G-OD---J----H K---J----H-A-P-L --B--P--E--K--A- -H--B--K--FI-C-- --F---C--D--H-N-	CILNKDGAHBMOPEFJ DOGPIHJMFNLECAKB JEKAFCNBGIDPLHOM EBOFPMIJDGHLNKCA NCJDHBAEKMOFIGLP HMPLCGKFIAENBDJO AKIGNODLBPJCEFMH KDEMJIFNCHGAOPBL GLBCDPMHEONKJIAF PHNOBALKMJFIDCEG IAFJOECGLDPBHMNK -1

33. Поиск гамильтонова цикла

Имеется полный граф с n вершинами. Каждая вершина задается своими целочисленными координатами на плоскости. Расстояние $d(i, j)$ между вершинами i и j задается следующим образом: $d(i, j) = |x_i - x_j| + |y_i - y_j|$.

Нужно найти гамильтонов цикл минимального веса в этом графе.

Формат входных данных

В первой строке находится число n вершин в графе. Далее следуют n строк с координатами вершин, i-я из которых содержит координаты i-й вершины – целые числа, по модулю не превосходящие 10^9.

Формат выходных данных

В первой строке выведите вес найденного гамильтонова цикла, во второй – номера вершин в порядке обхода в цикле (n + 1 число).

Входной файл	Выходной файл
5 3 7 7 2 6 3 8 8 4 7	22 1 5 4 2 3 1

34. Job-shop: *m* работ

Есть n станков и m работ. Каждая работа состоит из s_i этапов. Этап характеризуется парой (k, t) чисел, где k – номер станка, t – продолжительность этапа. Для каждой работы порядок этапов строго задан. Лю-

бой этап можно приостановить и позже продолжить с того же момента. В каждый момент времени любая работа может выполняться только на одном станке, который может выполнять только одну работу.

Необходимо составить такое расписание, чтобы все работы были выполнены за минимальное время, а также найти гамильтонов цикл минимального веса в этом графе.

Формат входных данных

В первой строке числа n и m выведите через пробел. В следующих m строках сначала записано число s_i, а за ним – s_i пар (k, t) целых чисел ($1 \le k \le m$, $1 \le t \le 10^9$).

Формат выходных данных

Выведите минимальное время выполнения.

Входной файл	Выходной файл
4 2 3 1 1 2 1 3 3 4 1 1 2 3 4 1 3 1	7

35. Задача коммивояжера на плоскости

Положение городов задается координатами (возможно, нецелыми) на плоскости. Расстояние считается в Евклидовой метрике.

Формат входных данных

В первой строке находится число n. В каждой из последующих n строк находятся два действительных числа – координаты i-го города, x_i и y_i, не более чем с двумя знаками после запятой $(|x_i|, |y_i| \le 10^6)$.

Формат выходных данных

Выведите одно действительное число – длину цикла (с точностью до 10^{-4}).

Входной файл	Выходной файл
4 0 0 1 0 1 1 0 1	4.000000

36. Изоморфизм графов

Даны два неориентированных графа без петель.

Требуется проверить, являются ли графы изоморфными, а также найти пример изоморфизма и их число.

Формат входных данных

В первой строке находятся числа n_1 и m_1 – количество вершин и ребер первого графа соответственно $(1 \le n_1 \le 16)$. В каждой из следующих m_1 строк расположена пара (u, v) чисел от 0 до $n_1 - 1$, означающая, что вершины u и v соединены ребром. Далее аналогичным образом задается и второй граф.

Формат выходных данных

Выведите в единственной строке NO, если графы не являются изоморфными. В противном случае выведите в первой строке YES, во второй – номера вершин второго графа в порядке их соответствия вершинам первого графа, т. е. попарно различные значения $\varphi(0), \varphi(1), \ldots, \varphi(n_1 - 1)$, для которых верно, что вершины u и v первого графа смежны только тогда, когда смежны вершины $\varphi(u)$ и $\varphi(v)$ второго графа. Если таких функций φ несколько, выведите любую. В третьей строке выведите число изоморфизмов первого графа на второй, т. е. число таких функций φ.

Входной файл	Выходной файл
5 3 1 2 2 3 3 4 5 3 1 3 3 2 2 4	YES 0 1 3 2 4 2

37. Сбалансированные скульптуры

Скульптура порядка n – полимино (фигура из одинаковых квадратных плиток, соединенных по общим сторонам, которая может содержать дыры) со следующими свойствами:

- состоит из $n + 1$ плиток: *блоков* (n плиток) и *основания* (оставшаяся плитка);
- центр основания находится в позиции (0, 0);
- все блоки имеют положительную y-координату, т. е. основание – нижняя плитка.

Скульптура называется *сбалансированной*, если центр масс всех ее блоков вместе имеет нулевую x-координату. Скульптуры, являющиеся зеркальным отражением относительно y-оси, не считаются различными.

Требуется найти все сбалансированные скульптуры заданного порядка, которые можно получить путем добавления блоков. Повороты и сдвиги не допускаются.

Формат входных данных

Первая строка содержит число n – требуемый порядок сбалансированной скульптуры $(1 \le n \le 18)$, вторая – число k – порядок начальной скульптуры $(1 \le k \le n)$. Следующие $k + 1$ строки содержат пары $(x,\ y)$ целочисленных координат блоков $(0 \le |x|,\ y \le k)$.

Формат выходных данных

В первой строке выведите общее число s сбалансированных скульптур порядка n, далее – в произвольном порядке $\min\{s, 10\,000\}$ скульптур. Каждую скульптуру выведите как псевдографическое изображение с использованием символов «_» (свободная клетка) и «#» (плитка). Изображение должно содержать скульптуру полностью и не содержать строк или столбцов без плиток. Разные скульптуры необходимо разделять пустой строкой. Для несимметричной скульптуры выведите любое из двух ее отражений.

Входной файл	Выходной файл
3 1 0 0 0 1	2 # # # # ### _#_

38. Испытание шаха

На доске размером $m \times m$ нужно расставить некоторое количество шахматных фигур – ферзей, ладей, слонов, коней и королей – таким образом, чтобы ни одна из них не била другие.

Необходимо написать программу, которая будет решать эту головоломку.

Формат входных данных

Первое число во входном файле задает размер доски m $(2 \le m \le 12)$. Следующие пять целых неотрицательных чисел K, Q, R, B, N задают соответственно количество королей, ферзей, ладей, слонов и коней.

Их требуется расставить. Общее количество фигур не превосходит M^2. Фигуры подобраны так, что искомая расстановка существует.

Формат выходных данных

Выведите доску с расставленными фигурами в виде m строк по m символов в каждой. Пустые поля обозначаются символом «.» (точка), поля с королями – K, ферзями – Q, ладьями – R, слонами – B, конями – N.

Входной файл	Выходной файл
4 1 0 0 1 2	.... B.NN .... K...

39. Задача категории «Б»

Квадратная таблица $n \times n$ $(n > 4,\ n \neq 6)$ называется красивой, если в каждой клетке матрицы записано 0 или 1, при этом сумма в любой строке и любом столбце диагонали не превосходит 1, а сумма чисел во всей таблице равна n.

Требуется найти любую красивую квадратную таблицу размером $n \times n$.

Формат входных данных

На вход подается одно целое положительное число n $(n \leq 2000)$.

Формат выходных данных

Выведите таблицу $n \times n$ без пробелов. Если существует несколько ответов, вы можете вывести любой.

Входной файл	Выходной файл
5	00010 10000 00100 00001 01000

Указания к решению задач

1. Минимальное вершинное покрытие графа, вершинами которого являются все свободные клетки. Существует ребро (x, y), если при постановке коня в клетку x будет пробиваться клетка y.

2. Двудольный граф с вершинами первой доли – вертикали шахматной доски, второй доли – горизонтали; ребро между двумя вершина-

ми двудольного графа проводится в том случае, если соответствующие вертикаль и горизонталь пересекаются.

3. Минимальное вершинное покрытие графа, вершинами которого будут все свободные клетки.

4. Минимальное вершинное покрытие графа, вершинами которого будут все свободные клетки.

5. Двудольный граф: вершинам первой доли которого соответствуют диагонали слева направо, вершинам второй доли – диагонали справа налево; в двудольном графе будем проводить ребро (a, b) в том случае, если a и b пересекаются.

6. Двудольный граф: вершинам первой доли соответствуют диагонали слева направо, вершинам второй доли – диагонали справа налево; в двудольном графе будем проводить ребро (a, b) в том случае, если a и b пересекаются.

7. Двудольный граф, соответствующий нашей доске: вершины – горизонтальные (вертикальные) блоки (последовательность свободных клеток доски, расположенных одна за другой по горизонтали (вертикали)), между вершинами a и b существует ребро, если на доске существует свободная клетка, принадлежащая блокам, соответствующим вершинам a и b, т. е. клетки доски отображаются в ребра графа.

8. Поиск максимального паросочетания в графе, ребрами которого являются вертикали, горизонтали и диагонали шахматной доски.

9. Проведем на доске все диагонали справа налево и сверху вниз (множество A). Каждой диагонали сопоставим вершину графа. Аналогично проведем все диагонали слева направо и сверху вниз (множество B) и каждой диагонали сопоставим вершину графа. В полученном графе найдем максимальное парасочетание, что и будет наибольшим количеством расставленных слонов, которые не бьют друг друга.

10. Двудольный граф, соответствующий нашей доске: вершины – клетки, не занятые черными фигурами; ребро между вершиной A и B устанавливается в том случае, если конь за один ход может попасть из вершины A в вершину B.

11. Двудольный граф, соответствующий доске: вершины – клетки, не занятые черными фигурами; ребро между вершиной A и B устанавливается в том случае, если ладья за один ход может попасть из вершины A в вершину B.

12. Двудольный граф, соответствующий доске: вершины – клетки, не занятые черными фигурами; ребро между вершиной A и B устанавливается в том случае, если ферзь за один ход может попасть из вершины A в вершину B.

13. Простой перебор.

14. Ломаную линию следует начинать с центра доски и учитывать симметрию.

15. В дереве решений у каждой вершины может быть до семи потомков, каждое ребро имеет метку из множества {0, 1, 2, 3, 4, 5, 6}. Вершина дерева глубины k задает частично построенный квадрат.

16. Поиск простых циклов, начинающихся в вершинах (a, a) в ориентированном графе, построенном по следующему правилу: каждой доминошке, кроме доминошек вида (a, a), соответствуют две вершины графа (a, b) и (b, a). В доминошку (a, b) входят дуги из всех доминошек со второй цифрой a, а выходят – с первой цифрой b; в доминошку (b, a) – наоборот.

17. Дерево решений – бинарное.

18. В качестве текущей точки привязки фигур выступает клетка, допускающая минимальное количество фигур для размещения.

19. В дереве перебора: вершина – номер вершины в графе; ребро – номер цвета, в который выкрашиваем вершину. Каждая вершина имеет $k + 1$ сыновей, где k – количество цветов, уже использованных для раскраски графа.

20. Строим дерево перебора. Его вершиной будет комбинация значений свободных переменных из множества {0, 1}. Корнем будет комбинация всех свободных переменных, приравненных нулю. На каждом шаге приравниваем текущую переменную нулю (левая ветка) или единице (правая ветка). Всего вершин в дереве будет 2^k. Высота дерева составит k.

21. Вершиной дерева перебора будет n-вектор. Тогда значение i-й координаты этого вектора будет совпадать с номером станка, на который мы назначили i-ю деталь. В корень дерева мы положим 0-вектор.

22. Простой перебор.

23. Нужно учесть свойство симметричности и тот факт, что на границах искомого прямоугольника (в углах) должны быть три заданных прямоугольника, кроме случая, когда все прямоугольники расположены в линию.

24. Вершина дерева решений – подмножество ребер графа, образующих дерево. Сыновья вершины в дереве решений – все деревья, получаемые из родительского дерева добавлением одного ребра.

25. Необходимо построить нижнюю оценку, рассматривая, например, задачу, когда работы можно прервать.

26. Необходимо построить нижнюю оценку, рассматривая, например, задачу, когда работы можно прервать.

27. Использовать нижнюю оценку, рассматривая задачу с возможными прерываниями.

28. Простой перебор.

29. Использовать нижнюю оценку, рассматривая задачу с возможными прерываниями.

30. Поиск в ширину, начиная с финишной вершины.

31. Использовать нижнюю оценку, рассматривая задачу с прерываниями.

32. Простой перебор.

33. Использовать в качестве стартовой точки выпуклую оболочку и процедуру «растущий цикл».

34. Аналогично задаче 29.

35. Аналогично задаче 33.

36. Простой перебор.

37. Простой перебор.

38. Использовать идеи задач 7–9.

39. Использовать задачу 7.

Глава 2

ЭВРИСТИЧЕСКИЕ И ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ

2.1. Основные понятия

Поскольку точные полиномиальные алгоритмы решения задач, принадлежащих классу *NP*, неизвестны, следует рассмотреть альтернативные возможности, дающие лишь достаточно разумные ответы. Из *NP*-трудности задачи следует необходимость построения для ее решения эвристических или приближенных алгоритмов и применения схем направленного перебора вариантов (метод ветвей и границ, динамическое программирование).

Предположим, что дана задача максимизации:

$$F(x) \to \max,\ x \in D.$$

Пусть $x^{\text{опт}}$ дает максимум фунционалу $F(x)$, т. е.

$$x^{\text{опт}} = \arg\left(\max F(x)\right).$$

Предположим, что есть алгоритм A, который на множестве D строит другое решение:

$$x(A) \overset{\text{def}}{=} x^A.$$

Тогда, возможно, $F\left(x^{\text{опт}}\right) \cong F\left(x^A\right)$.

Если алгоритм A – точный, то получим строгое равенство

$$F\left(x^{\text{опт}}\right) = F\left(x^A\right).$$

Определение 2.1. Абсолютная погрешность алгоритма A на входном наборе данных определяется как величина

$$\left|F\left(x^{\text{опт}}\right) - F\left(x^A\right)\right|.$$

Определение 2.2. Относительная погрешность алгоритма A на входном наборе данных определяется как величина

$$\frac{\left|F\left(x^{\text{опт}}\right) - F\left(x^A\right)\right|}{\left|F\left(x^{\text{опт}}\right)\right|}.$$

Иногда для оценки качества алгоритма A рассматривают функцию

$$A = \frac{\inf\left|F\left(x^{A}\right)\right|}{\left|F\left(x^{\text{опт}}\right)\right|},$$

где inf берется по всем возможным наборам входных данных.

Определение 2.3. Гарантированной оценкой точности решения, получаемого при помощи алгоритма A, называется:

• для задачи максимизации – нижняя оценка (inf) величины Δ_A:

$$P \le \Delta_A = \frac{\left|F\left(x^{A}\right)\right|}{\left|F\left(x^{\text{опт}}\right)\right|} \le 1;$$

• задачи на минимум – верхняя оценка (sup) величины Δ_A:

$$1 \le \Delta_A = \frac{\left|F\left(x^{A}\right)\right|}{\left|F\left(x^{\text{опт}}\right)\right|} \le P.$$

Определение 2.4. Алгоритм называется ***приближенным***, если для него известна абсолютная (относительная) погрешность, не зависящая от входных данных, либо гарантированная оценка точности.

Существует следующая классификация приближенных алгоритмов:

• *субоптимальный приближенный* – величина Δ_A отлична от 0 (для задач максимизации) или от ∞ (для задач минимизации);

• ε-*приближенный* (PTAS) – для любого $\varepsilon > 0$ строит решение с относительной погрешностью ε, причем трудоемкость этого алгоритма есть функция, которая зависит от длины входа и величины $1/\varepsilon$);

• *быстрый* ε-*приближенный* (FPTAS) – для любого $\varepsilon > 0$ строит решение с относительной погрешностью ε, причем трудоемкость этого алгоритма есть полином от длины входа и величины $1/\varepsilon$.

Пример 2.1. Пусть $p_1, p_2, \ldots, p_n$ – длительности программ. Имеются два носителя одинаковой длины L. Необходимо вместить как можно больше программ на носители.

Поскольку значение $F\left(x^{\text{опт}}\right)$ неизвестно, то, как правило, оценивается величина

$$\frac{F\left(x^{A}\right)}{\text{ВО}(I)}$$

для задачи максимизации и

$$\frac{F\left(x^{A}\right)}{\text{НО}(I)}$$

для задачи минимизации, где $\text{ВО}(I)$ и $\text{НО}(I)$ – верхняя и нижняя оценки оптимального решения для индивидуальной задачи (на определенном наборе входных данных).

В этом случае для задачи максимизации

$$\frac{F\left(x^{A}\right)}{F\left(x^{\text{опт}}\right)} \geq \frac{F\left(x^{A}\right)}{\text{ВО}(I)}$$

в силу $\text{ВО}(I) \geq F\left(x^{\text{опт}}\right)$ для задачи на максимум,

$$\frac{F\left(x^{A}\right)}{F\left(x^{\text{опт}}\right)} \leq \frac{F\left(x^{A}\right)}{\text{НО}(I)}$$

в силу $\text{НО}(I) \leq F\left(x^{\text{опт}}\right)$ для задачи на минимум.

Таким образом, для вычисления гарантированной оценки (в большинстве случаев) требуется определить верхние (нижние) оценки оптимального решения и только потом попытаться найти соотношение между этими оценками и решением, построенным предлагаемым алгоритмом. При таком подходе очень важно найти оценки, наиболее близкие к оптимальному решению.

Решение. Предположим, что существует такое k, что

$$\sum_{i=1}^{k} p_k \leq 2L, \ \sum_{i=1}^{k+1} p_k > 2L.$$

Тогда

$$F\left(x^{\text{опт}}\right) \leq k.$$

Предположим, что существует алгоритм A, для которого

$$F\left(x^{A}\right) \geq k-1.$$

Тогда A – приближенный алгоритм с абсолютной погрешностью

$$\left|F\left(x^{\text{опт}}\right)-F\left(x^{A}\right)\right|\le\left|k-\left(k-1\right)\right|\le 1.$$

Рассмотрим несколько типов приближенных алгоритмов. Все они полиномиальны по времени.

2.2. Жадные (градиентные) алгоритмы

Жадные (градиентные) алгоритмы (англ. *greedy algorithm*) являются наиболее популярной эвристикой для большого количества прикладных задач и имеют широкое практическое распространение. Именно поэтому важнейшие вопросы – качество (точность) и условия, при которых алгоритмы данного типа дают точные решения. Рассмотрим наиболее популярные алгоритмы градиентного типа для различных модельных задач дискретной оптимизации, дадим оценки их точности, а также проанализируем вопрос о том, когда алгоритмы градиентного типа строят точные решения.

2.2.1. Жадный алгоритм для задачи о коммивояжере

Задано n городов, попарно соединенных дорогами. Для каждой из них (i, j) задается стоимость проезда – c_{ij}. Коммивояжер должен посетить все n городов по одному разу и вернуться в начальный город, при этом суммарная стоимость проезда должна быть минимальной.

Алгоритм 2.1. Предположим, что задан полный граф. Будем перебирать все ребра графа в порядке возрастания их весов и для каждого ребра проверять два условия:

1) в результате добавления данного ребра к ранее выбранным не образуется цикл, если это не завершающее ребро пути (для завершающего ребра будет получен цикл, который проходит через все вершины графа);

2) ребро после добавления к ранее выбранным ребрам не будет являться третьим, инцидентным некоторой вершине.

Если для ребра выполняются эти условия, то оно добавляется к ранее выбранному множеству ребер.

Пример 2.2. Необходимо решить задачу коммивояжера для графа, приведенного на рис. 2.1.

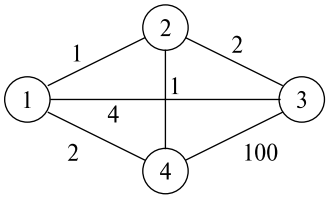


Рис. 2.1

Решение. Жадный приближенный алгоритм сначала выберет ребро {1, 2}, затем ребро {2, 4} и присоединит их к выбранному множеству ребер. Следующим ребром минимального веса будет {1, 4}, но для него не выполняется пункт 1 из условий присоединения ребер, и оно не является завершающим ребром пути (цикл $1 \to 2 \to 3 \to 1$ не проходит через все вершины графа), поэтому будет отвергнуто. Ребро {2, 3} также будет отвергнуто, поскольку для него не выполняется пункт 2 из условий присоединения, так как вершина 2 в этом случае была бы инцидентна трем ребрам. Ребро {1, 3} успешно присоединяется. Ребро {4, 3} присоединяется и является завершающим ребром пути. Полученный цикл проходит через все вершины графа. Жадный алгоритм сгенерировал путь $1 \to 2 \to 4 \to 3 \to 1$, вес которого 106 (на рис. 2.2 выбранные ребра выделены).

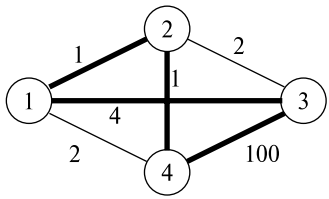


Рис. 2.2

Данное решение не является оптимальным: есть как минимум один более короткий путь: $1 \to 2 \to 3 \to 4 \to 1$, вес которого 105.

2.2.2. Жадный алгоритм для задачи о рюкзаке

Имеется набор объектов. Для каждого объекта i заданы (a_i, c_i), где a_i – объем и c_i – стоимость. Мы хотим заполнить рюкзак объемом W так, чтобы суммарная стоимость вошедших в него предметов была максимальна.

Алгоритм 2.2. Отсортируем список объектов в зависимости от их стоимости и объема (удельная стоимость):

$$\frac{c_i}{a_i} \geq \frac{c_{i+1}}{a_{i+1}}.$$

Затем будем последовательно и упорядоченно заполнять рюкзак объектами.

Если очередной объект не помещается, то откладываем его и переходим к рассмотрению следующего. Описанный процесс продолжаем до тех пор, пока рюкзак не заполнится или будет исчерпан весь список объектов.

Пример 2.3. Заполнить рюкзак объемом $W = 80$ объектами, сведения о которых приведены в табл. 2.1.

Таблица 2.1

(a_i, c_i)	(20, 40)	(20, 80)	(20, 50)	(15, 45)	(30, 105)	(35, 35)	(20, 10)	(10, 45)
c_i / a_i	2	4	2,5	3	3,5	1	0,5	4,5
$\frac{c_i}{a_i} \geq \frac{c_{i+1}}{a_{i+1}}$	(10, 45) 1	(20, 80) 2	(30, 105) 3	(15, 45) 4	(20, 50) 5	(20, 40) 6	(35, 35) 7	(20, 10) 8

Решение. Учитывая, что емкость рюкзака 80, мы сможем упаковать в соответствии с нашим алгоритмом первые четыре объекта отсортированного списка. Общий объем упакованных объектов – 75, стоимость – 275. Это решение не является оптимальным, так как существует лучшее, которое дают первые три и пятый объекты отсортированного списка: объем – 80, стоимость – 280.

2.2.3. Жадный алгоритм для задачи о суммах элементов подмножеств

Имеется набор предметов различных размеров и некоторая верхняя положительная граница их суммарного объема L. Необходимо найти такой набор предметов, сумма размеров которых была бы наиболее близка к L.

Алгоритм 2.3. Для построения приближенного алгоритма будем использовать жадный алгоритм. Отсортируем список предметов в порядке убывания их размеров и будем последовательно дополнять предметы к пустому множеству.

Если после добавления очередного предмета ко множеству суммарный объем предметов, входящих в сгенерированное множество, превышает L, то отбрасываем его и переходим к рассмотрению следующего. Описанный процесс продолжаем до тех пор, пока суммарный размер всех предметов не станет равным L или будет исчерпан упорядоченный список объектов.

Пример 2.4. Объемы предметов a_i приведены в табл. 2.2. Предельная сумма объемов предметов множества $L = 55$.

Таблица 2.2

a_i	1	14	22	7	27	11
$a_i \geq a_{i+1}$	$\frac{27}{1}$	$\frac{22}{2}$	$\frac{14}{3}$	$\frac{11}{4}$	$\frac{7}{5}$	$\frac{1}{6}$

Необходимо найти такой набор предметов, сумма объемов которых была бы наиболее близка к предельной.

Учитывая, что объем множества $L = 55$, мы сможем добавить в него в соответствии с нашим алгоритмом три предмета объемом 27, 22 и 1. Общий объем сгенерированного множества – 50. Это решение не является оптимальным, так как существует лучшее, которое дают пять предметов объемом 22, 14, 11, 7 и 1. В этом случае объем – 55, значит данное решение оптимально.

Таблица 2.3

Номер прохода	Размер начального подмножества	Добавляемый элемент	Сумма элементов сгенерированного множества
0	0	27, 22, 1	50
1	{27} {22} {14} {11} {7} {1}	22,1 27, 1 27, 11, 1 27, 14, 1 27, 14, 1 27, 22	50 50 53 53 49 50
2	{27, 22} {27, 14} {27, 11} {27, 7} {27, 1} {22, 14} {22, 11} {22, 7} {22, 1} {14, 11} {14, 7} {14, 1} {11, 7} {11, 1} {7, 1}	1 11, 1 14, 1 14, 1 22 11, 7, 1 14, 7, 1 14, 11, 1 27 27 1 27, 1 27, 11 27, 1 27, 14 27, 14	50 53 53 49 50 55! оптимум 55 55 50 53 49 53 46 53 49

Алгоритм 2.4. Рассмотрим еще один алгоритм, который также является жадным при каждом из своих проходов.

На первом проходе начинаем с пустого множества и добавляем в него элементы по убыванию объемов упорядоченного списка предметов. Если очередной предмет при добавлении приводит к тому, что суммарный объем предметов, входящих в сгенерированное множество, превышает L, то откладываем его и переходим к рассмотрению следующего. Описанный процесс продолжаем до тех пор, пока суммарный размер всех предметов не станет равным L или будет исчерпан упорядоченный список объектов.

На втором проходе начинаем свою работу со всевозможных двухэлементных множеств и добавляет к ним элементы по объемам из упорядоченного списка предметов.

На третьем проходе аналогичным образом исследуем все трехэлементные множества и т. д.

Количество проходов ограничено временем, отведенным на работу алгоритма. Если время позволяет, то на $n + 1$ проходе, где n – количество предметов, алгоритм построит оптимальное решение, если оно не построено ранее, возможно, на некотором проходе сумма элементов сгенерированного множества совпадет с L.

Пример 2.5. Для приведенного примера (см. табл. 2.2) упорядоченность предметов по убыванию объемов следующая:

$$27, 22, 14, 11, 7, 1.$$

Предельная сумма объемов предметов множества $L = 55$.

Выполним последовательность шагов жадного алгоритма 2.4. Уже на втором проходе будет построено оптимальное решение задачи (табл. 2.3).

2.2.4. Жадный алгоритм для задачи о раскраске графа

Требуется определить минимальное количество цветов c, в которые можно раскрасить вершины графа так, чтобы концы каждого его ребра были окрашены в разные цвета. Для данной задачи доказано, что число красок, даваемое лучшим приближенным полиномиальным алгоритмом, более чем вдвое превышает оптимальное. Рассмотрим простой алгоритм последовательной раскраски произвольного графа с n вершинами.

Алгоритм 2.5. Последовательно рассмотрим все вершины исходного графа, чтобы определить, каким цветом их раскрасить.

Предположим, что мы хотим решить, в какой цвет должна быть покрашена вершина *i*.

1. Первоначально номер цвета для этой вершины полагаем $c = 1$.

2. Пока в графе существуют вершины, смежные с вершиной *i* и покрашенные цветом *c*, увеличим номер цвета окраски вершины *i* на единицу: $c = c + 1$.

3. Окрасим вершину с номером *i* в цвет *c*.

Максимально возможное число красок, используемое для раскраски графа данным алгоритмом, равно степени графа, увеличенной на 1 (степень графа – максимум из степеней его вершин).

Пример 2.6. Необходимо решить задачу раскраски графа, приведенного на рис. 2.3.

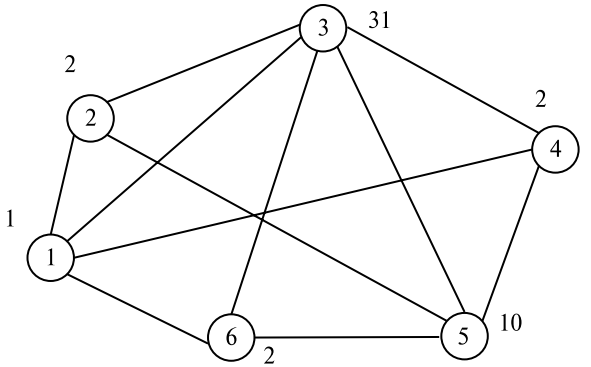


Рис. 2.3

Решение. Раскраска вершин проводится в соответствии с приведенным алгоритмом. Вершины графа просматриваются в порядке возрастания их номеров. Как видно из рисунка, минимальное число потребовавшихся красок равно 3.

2.3. Локальный поиск

2.3.1. Описание эвристики

Для любой задачи алгоритм локального поиска начинает работу от некоторого исходного решения, найденного каким-то другим алгоритмом или выбранного случайно, и представляет собой итеративный процесс. На каждом шаге локального спуска происходит переход

от текущего решения к соседнему с лучшим значением целевой функции до тех пор, пока не будет достигнут локальный оптимум. Соседнее решение не обязательно должно быть наилучшим в окрестности (в некотором множестве соседних решений), а вот критерий оценки решения не должен меняться по ходу этого итеративного процесса.

Структурой, или *семейством*, всех окрестностей для индивидуальной задачи x оптимизационной проблемы V называется отображение N, которое каждому допустимому решению s задачи ставит в соответствие множество решений $N(s)$. Структура окрестностей может быть достаточно сложной, а отношение соседства – не всегда симметричным, когда s может быть соседом s', а s' может не быть соседом s.

Для решения s поиск более хорошего решения в окрестности $N(s)$ возможны два способа:

1) выбор наилучшего решения во всей окрестности;

2) выбор первого лучшего решения, когда окрестность каким-либо образом перебирается и прекращается, как только найдено лучшее решение.

Локальным поиском для оптимизационной проблемы V называется алгоритм, который для любой индивидуальной задачи строит локально оптимальное решение относительно структуры окрестностей N.

Пусть P – дискретная оптимизационная проблема, тогда структуру окрестностей можно задать с помощью ориентированного графа.

Графом соседства (*окрестностей*) (*neighborhood graph*) для индивидуальной задачи x называется взвешенный ориентированный граф $G_N(x) = (V(x), U)$, где множество дуг $U = \{(s_1, s_2) | s_2 \in N(s_1)\}$. Веса в этом графе приписаны вершинам и равны соответствующим значениям целевой функции.

Граф окрестностей $G_N(x)$ иногда называют еще *ландшафтом* целевой функции, или просто ландшафтом (*landscape, fitness landscape*). При определении структуры окрестностей N важно следить, чтобы получающийся граф $G_N(x)$ был связным, т. е. для каждой пары вершин s и s' существовал путь из s в s'. Это свойство является важным при анализе асимптотического поведения алгоритмов, например вероятностных метаэвристик.

Если же это свойство не выполняется, то стараются получить хотя бы свойство связности: из любой вершины существует путь в вершину с минимальным значением целевой функции. Если же и этого свойства нет, то достижение глобального оптимума локальными методами ста-

вится под сомнение и следует либо ограничиться локальными оптимумами, либо переопределить функцию окрестности.

Переходу от одного текущего решения к другому в графе соседства соответствует дуга, поэтому локальному поиску для задачи x соответствует путь в графе соседства, который заканчивается в локально-оптимальной вершине $s_г$ свойством: для любой вершины s, где $(s_г, s) \in U$, значение целевой функции не лучше, чем в вершине $s_г$. В худшем случае локальный поиск может исследовать экспоненциальное число решений прежде, чем достигнет локального оптимума. По этой причине в эвристике локального поиска часто используется правило остановки.

Алгоритмы локального поиска основаны на следующей общей схеме.

```
        Input: Индивидуальная задача x;
              Output: Решение s;
begin
s : = исходное допустимое решение s0;
repeat
    выбрать еще не рассмотренное решение s' ∈ N(s);
    если F(s') лучше F(s), то s := s';
until все решения из TV не будут рассмотрены;
return s
end.
```

Для данной индивидуальной задачи поведение этого алгоритма зависит от следующих факторов:

1) структура окрестностей N. Размер окрестности любого решения должен выбираться на основе компромисса между целью получения хорошего улучшения при каждом переходе к новому текущему решению и целью ограничения времени просмотра одной окрестности. Обычно для любого решения s окрестность $N(s)$ порождается с помощью некоторой операции локального изменения s;

2) начальное решение s_0. Его можно находить с помощью конструктивной эвристики, которая выдает хорошее допустимое решение, или с помощью процедуры случайной генерации;

3) стратегия выбора новых решений. Например, все решения из $N(s)$ просматриваются, выбирается лучшее из них и сравнивается с s. В случае если s не локально оптимальное, то осуществляется переход к наилучшему соседу или к первому лучшему решению, найденному в окрестности.

Алгоритмы локального поиска широко применяются для решения *NP*-трудных задач дискретной оптимизации. Вместе с тем многие полиномиально разрешимые задачи могут легко решаться локальным спуском. При подходящем выборе полиномиальной окрестности соответствующая теорема может быть сформулирована так: допустимое решение не является глобальным оптимумом, если оно может быть улучшено некоторым локальным образом.

Ниже приводится несколько примеров, которые указывают на важность локального поиска при построении оптимизационных алгоритмов и на достаточно общий характер этого подхода.

1. *Линейное программирование*. Геометрически алгоритм симплексметода можно интерпретировать как движение по вершинам многогранника допустимой области. Вершина не является оптимальной, если существует смежная с ней вершина с меньшим значением целевой функции. Алгебраически, предполагая невырожденность задачи, базисное допустимое решение не является оптимальным, если оно может быть улучшено локальным изменением базиса, т. е. заменой одной базисной переменной на небазисную. Получающаяся таким образом окрестность является точной и имеет полиномиальную мощность.

2. *Минимальное остовное дерево*. Остовное дерево не является оптимальным, если можно получить новое остовное дерево с меньшим суммарным весом посредством локальной перестройки, добавляя одно ребро и удаляя из образовавшегося цикла другое. Операция локальной перестройки задает отношение соседства на множестве остовных деревьев. Окрестность любого дерева имеет полиномиальную мощность, а функция окрестности является точной.

3. *Наибольшее паросочетание*. Паросочетание не является наибольшим, если существует увеличивающий путь. Два паросочетания называются соседними, если их симметрическая разность образует путь. Определенная таким образом окрестность является точной и имеет полиномиальную мощность. Аналогичные утверждения справедливы для взвешенных паросочетаний, совершенных паросочетаний минимального веса, задач о максимальном потоке и потоке минимальной стоимости.

На каждом шаге локального спуска структура окрестностей N задает множество возможных направлений движения. Очень часто это множество состоит из нескольких элементов, благодаря чему и имеется определенная свобода в выборе следующего решения. Правило выбора может оказать существенное влияние на трудоемкость алгоритма и результат его работы. Например, в задаче о максимальном потоке ал-

горитм Форда – Фалкерсона, который тоже можно рассматривать как вариант локального спуска, имеет полиномиальную временную сложность при выборе наименьшего пути по количеству дуг (для увеличения потока) и экспоненциальную временную сложность без гарантии получить глобальный оптимум при произвольном выборе пути.

Таким образом, при разработке алгоритмов локального поиска важно не только правильно определить окрестность, но и верно задать правило выбора направления спуска. На первый взгляд кажется, что в окрестности надо брать элемент с наименьшим значением целевой функции. Однако разумным оказывается не только такой выбор, но и движение в «абсурдном» направлении, когда несколько шагов с ухудшением могут привести (и часто приводят) к лучшему локальному оптимуму.

Множество $N(s)$ как можно меньшей мощности позволяет сократить трудоемкость одного шага, однако более широкая окрестность приводит к лучшему локальному оптимуму. Именно поэтому при создании алгоритмов приходится искать оптимальный баланс между этими противоречивыми факторами.

2.3.2. Сложность локального поиска

Анализ вычислительной сложности локального поиска в последние годы интенсивно ведется в двух направлениях: эмпирическом и теоретическом, которые дают существенно разные оценки возможностям локального поиска.

Для многих *NP*-трудных задач локальный поиск позволяет находить решения, близкие по целевой функции к глобальному оптимуму. Трудоемкость алгоритмов часто оказывается полиномиальной, причем степень полинома достаточно мала. Так, для задачи о разбиении множества вершин графа на две равные части разработаны алгоритмы локального поиска со средней трудоемкостью $O(n\log n)$, n – число вершин, которые дают всего несколько процентов погрешности.

Для задачи коммивояжера алгоритмы локального поиска являются наилучшими с практической точки зрения. Один из таких алгоритмов с окрестностью Лина – Кернигхана в среднем имеет погрешность около 2 %, а максимальная размерность решаемых задач достигает 1 000 000 городов. На случайно сгенерированных задачах такой колоссальной размерности итерационная процедура Джонсона позволяет находить решения с отклонением около 0,5 % за несколько минут. Для реальных задач с числом городов до 100 000 существуют алгоритмы с аналогичными характеристиками.

Для задач теории расписаний, размещения, покрытия, раскраски и многих других *NP*-трудных задач алгоритмы локального поиска показывают превосходные результаты. Более того, их гибкость при изменении математической модели, простота реализации и наглядность превращают локальный поиск в мощное средство для решения практических задач.

Рассмотрим 2-оптимальную (2-опт) эвристику для задачи о коммивояжере. (Для удобства будем рассматривать только неориентированные графы с симметричной целочисленной неотрицательной матрицей расстояний $w(i, j), 1 < i,\ j < n$). Метод основан на следующей 2-обменной окрестности: для любого цикла C 2-обменная окрестность $N(C)$ – это множество всех циклов C', которые могут быть получены из C после удаления двух ребер $\{x, y\},\ \{v, z\}$ и добавления двух новых ребер $\{x, v\},\ \{z, y\}$. Процесс продолжается до тех пор, пока не происходит улучшение целевой функции.

Алгоритм 2-опт можно модифицировать в алгоритм локального поиска с большими окрестностями. Например, 3-опт эвристика основывается на 3-обменных окрестностях. Для тура C его 3-обменная окрестность $N(C)$ – это множество всех циклов t', которые могут быть получены из C после замены не более чем трех ребер. Алгоритм 3-опт эвристики имеет лучшее приближение, но хуже по трудоемкости.

Существуют такие начальные циклы, при использовании которых эвристика дает решение хуже оптимального в произвольное число раз.

Теорема 2.1. Для любых

$$k > 2,\ n > 2k + 8 \ \text{ и } \ \alpha > \frac{1}{n}$$

существует такой пример x задачи коммивояжера с n городами $\{1, 2, \ldots, n\}$, что эвристика k-опт для входа x с начальным циклом $C_0 = (c_1, c_2, \ldots, c_n)$ находит цикл, стоимость которого $F(X)$ удовлетворяет неравенству

$$F(x) > \alpha^* F(C_0).$$

2.4. Приближенные алгоритмы с гарантированной оценкой точности

Рассмотрим задачу минимизации целевого функционала $F(x)$ на множестве X. Обозначим

$$F^* = \min\{F(x) \mid x \in X\}.$$

Предположим, что $F^* > 0$. Пусть приближенный алгоритм H находит элемент множества X со значением F^H целевого функционала. Введем в рассмотрение величину $\Delta_H = F^H / F^*$, которая зависит от алгоритма H и от набора входных данных задачи. Под гарантированной оценкой точности решения, получаемого при помощи алгоритма H, будем понимать верхнюю оценку величины Δ_H при любом наборе входных данных задачи.

В дальнейшем обозначения F^*, F^H, Δ_H будут использоваться при рассмотрении конкретных задач и конкретных приближенных алгоритмов. Смысл этих обозначений не изменяется.

Рассмотрим *NP*-трудную в сильном смысле задачу $P // \sum w_i C_i$. Она состоит в отыскании расписания, минимизирующего взвешенную сумму моментов завершения обслуживания n требований в системе из m параллельных идентичных приборов. Для каждого требования j задана длительность обслуживания p_j и вес (относительная важность) w_j. Расписание однозначно определяется разбиением множества требований на подмножества $N_1, N_2, \ldots, N_m$, где требования множества N_1 обслуживаются прибором 1 с указанием порядка обслуживания требований в каждом подмножестве. При помощи перестановочного метода нетрудно показать, что достаточно ограничиться рассмотрением расписаний, при которых требования каждого подмножества упорядочены по неубыванию значений p_j / w_j. Таким образом, расписание однозначно определяется разбиением множества требований на m подмножеств.

Опишем приближенный алгоритм A решения задачи $P // \sum w_i C_i$, который состоит в следующем.

Перенумеруем требования в порядке SWPT так, что

$$\frac{p_1}{w_1} \le \ldots \le \frac{p_n}{w_n}.$$

Организуем m подмножеств: $N_1, N_2, \ldots, N_m$, полагая вначале

$$N_1 = N_2 = \ldots = N_m = \varnothing.$$

Будем последовательно, начиная с первого, распределять требования 1, 2, ..., n по подмножествам. Пусть требования 1, 2, ..., k, $k < n$ распределены, где 0 – фиктивное требование, соответствующее начальной ситуации $p_0 = 0$. Требование $k+1$ назначается следующим образом. Обозначим $P_l = \sum_{j \in N_{l_l}} p_j$, $l = 1, 2, ..., m$. Находим такое подмножество N_l,

где $P_l = \min\limits_{1 \le h \le m} P_h$. Полагаем $N_l = N_l \cup \{k+1\}$ и, если $k+1 \ne n$, переходим к назначению требования $k+2$.

Временная сложность алгоритма A равна $O(n \log n)$, так как значения $P_1, P_2, \ldots, P_m$ можно хранить в виде древовидной структуры данных *куча* или сбалансированного поискового 2–3-дерева.

Обозначим через F_1^* и F_n^* минимальное значение функционала $\sum w_i C_i$ при $m = 1$ и $m = n$ соответственно.

Тогда

$$F_1^* = \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i, \quad F_n^* = \sum_{j=1}^{n} w_j p_j.$$

Найдем верхнюю и нижнюю оценки значения F^*. Очевидно, что $F^* \le F^A$. Поскольку на итерации j алгоритма A выполняется соотношение

$$P_l = \min_{1 \le h \le n} \{P_h\} \le \frac{1}{m} \sum_{i=1}^{j-1} p_i,$$

то имеем

$$F^A \le \sum_{j=1}^{n} w_j \left(\frac{1}{m} \sum_{i=1}^{j-1} p_i + p_j \right) = \frac{1}{m} \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i + \left(1 - \frac{1}{m}\right) \sum_{j=1}^{n} w_j p_j =$$

$$= \frac{1}{m} F_1^* + \frac{m-1}{m} F_n^*.$$

Для получения нижней оценки значения F^* понадобится ряд вспомогательных утверждений.

Лемма 2.1. Пусть $b_1, b_2, \ldots, b_n$ – действительные числа, тогда

$$\left(\sum\nolimits_{j=1}^{n} b_j \right)^2 \le n \sum\nolimits_{j=1}^{n} b_j^2.$$

Лемма 2.2. Допустим, что $p_j / w_j = C$, $j = 1, 2, \ldots, n$, тогда

$$F_1^* = \frac{1}{2C} \left(\sum_{j=1}^{n} p_j \right)^2 + \frac{1}{2} F_n^*.$$

Лемма 2.3. Предположим, что $p_j / w_j = C$, $j = 1, 2, \ldots, n$, тогда

$$F^* - \frac{1}{2} F_n^* \ge \frac{1}{m} \left(F_1^* - \frac{1}{2} F_n^* \right).$$

Теорема 2.2. Для любого расписания справедливо неравенство

$$\sum w_j C_j \geq \frac{1}{m} F_1^* + \frac{m-1}{2m} F_n^*.$$

Доказательство. Воспользуемся индукцией по числу r различных значений p_j / w_j. При $r = 1$ справедливость утверждения следует из леммы 2.3. Пусть теорема верна при всех $r \leq k-1$, $k \geq 2$, и имеется k различных значений p_j / w_j и $p_1 / w_1 = \ldots = p_q / w_q > p_{q+1} / w_{q+1}$.

Положим, $\gamma = \dfrac{w_{q+1} p_1}{\left(p_{q+1} w_1\right)}$ и рассмотрим вспомогательную задачу, в которой веса обозначены $\overline{w}_j$, и для первых q требований $\overline{w}_j = \gamma w_j$, $j = 1, 2, \ldots, q$. Очевидно, что во вспомогательной задаче количество различных значений $p_j / \overline{w}_j$ равно $k-1$. Для вспомогательной задачи введем обозначения $\overline{F}_1^*$ и $\overline{F}_n^*$, аналогичные обозначениям – для исходной задачи. В силу индуктивного предположения для любого расписания

$$\sum \overline{w}_j C_j - \frac{1}{2} \overline{F}_n^* \geq \frac{1}{m} \left(\overline{F}_1^* - \frac{1}{2} \overline{F}_n^* \right).$$

Введем обозначения:

$$X = F^* - \frac{1}{2} F_n^*, \quad \overline{X} = \overline{F}^* - \frac{1}{2} \overline{F}_n^*, \quad Y = \sum w_j C_j - \frac{1}{2} F_n^*,$$

$$\overline{Y} = \sum \overline{w}_j C_j - \frac{1}{2} \overline{F}_n^*, \quad \overline{X}^q = \sum_{j=1}^{q} \overline{w}_j \sum_{i=1}^{j} p_i - \frac{1}{2} \sum_{j=1}^{q} \overline{w}_j p_j,$$

$$\delta\left(\overline{X}\right) = \overline{X} - \overline{X}^q, \quad \overline{Y}^q = \sum_{j=1}^{q} \overline{w}_j C_j - \frac{1}{2} \sum_{j=1}^{q} \overline{w}_j p_j, \quad \delta\left(\overline{Y}\right) = \overline{Y} - \overline{Y}^q.$$

Из леммы 2.3 следует, что $m\overline{Y} \geq \overline{X}$ и $m\overline{Y}^q \geq \overline{X}^q$, а из равенств $\overline{w}_j = \gamma w_j$, $j = \overline{1,q}$ следует, что $X = \dfrac{1}{\gamma} \overline{X}^q + \delta\left(\overline{X}\right)$ и $Y = \dfrac{1}{\gamma} \overline{Y}^q + \delta\left(\overline{Y}\right)$.

Возможны два случая:

1) $\delta\left(\overline{X}\right) \leq m\delta\left(\overline{Y}\right)$;

2) $\delta\left(\overline{X}\right) > m\delta\left(\overline{Y}\right)$.

В первом случае получаем $X \leq mY$, т. е.

$$\frac{1}{m} \left(F_1^* - \frac{1}{2} F_n^* \right) \leq \sum w_j C_j - \frac{1}{2} F_n^*.$$

Во втором случае из неравенства $m\bar{Y}^q \geq \bar{X}^q$ следует, что $m\delta(\bar{X})\bar{Y}^q > > m\bar{X}^q\delta(Y)$.

Поскольку $\gamma < 1$, получаем $(1-\gamma)\delta(\bar{X})\bar{Y}^q > (1-\gamma)\bar{X}^q\delta(Y)$ или $\delta(\bar{X})\bar{Y}^q + + \gamma\bar{X}^q\bar{Y}^q\delta(Y) > \bar{X}^q\delta(\bar{Y}) + \gamma\delta(\bar{X})\bar{Y}^q$. Прибавляя к обеим частям этого неравенства $\bar{X}^q\bar{Y}^q + \gamma\delta(\bar{X})\delta(\bar{Y})$, получаем $(\bar{X}^q + \delta(\bar{X}))(\bar{Y} + \gamma\delta(\bar{Y})) > > (\bar{X}^q + \gamma\delta(\bar{X}))(\bar{Y}^q + \delta(\bar{Y}))$.

Однако $\bar{X}^q + \delta(\bar{X}) = \bar{X}$, $\bar{Y}^q + \delta(\bar{Y}) = \bar{Y}$, $\bar{Y}^q + \gamma\delta(\bar{Y}) = \gamma Y$, $\bar{X}^q + + \gamma\delta(\bar{X}) = \gamma X$. Поэтому последнее неравенство принимает вид $\gamma\bar{X}\bar{Y} > \gamma X\bar{Y}$.

Из $m\bar{Y} \geq \bar{X}$ и последнего неравенства следует, что $\gamma m\bar{Y}Y > \gamma X\bar{Y}$. Таким образом, $mY > X$.

Из теоремы следует, что

$$F^* \geq \frac{1}{m}F_1^* + \frac{m-1}{2m}F_n^* \geq F^A - \frac{m-1}{2m}F_n^*.$$

Полагая, что $m = n$, получаем $F_n^* \geq \frac{2}{n+1}F_1^*$, откуда следует $F^* \geq \geq \frac{m+n}{m(n+1)}F_1^*$.

Поскольку $F_n^* \leq F^*$,

$$\Delta_A = \frac{F^A}{F^*} \leq \frac{F^* + \frac{m-1}{2m}F_n^*}{F^*} \leq \frac{3m-1}{2m}.$$

2.4.1. Задача об упаковке в контейнеры

Рассмотрим *NP*-трудную в сильном смысле задачу об упаковке в контейнеры, которая в терминах теории расписаний может быть сформулирована следующим образом. В условиях задачи

$$\frac{P}{d_j} = \frac{d}{C_j} \leq d_j$$

предполагается, что количество приборов m равно числу требований n, а общий директивный срок d не меньше максимальной длительности обслуживания требований $\max\limits_j p_j$.

Необходимо отыскать минимальное число приборов и соответствующее расписание, при котором все требования будут обслужены к директивному сроку d.

Приведем описание четырех приближенных алгоритмов решения задачи об упаковке в контейнеры. В алгоритмах рассматривается незначительная перестановка π требований, которые последовательно отбираются из этой перестановки и назначаются на приборы в соответствии с определенными правилами. Резервом времени R_j прибора l называется разница между d и суммарной длительностью обслуживания требований, назначенных на прибор l.

В алгоритмах B_1 и B_2 перестановка π является произвольной. На шаге k алгоритма B_1 или B_2 выбирается требование, расположенное на месте k в перестановке π. В алгоритме B_1 это требование назначается на прибор с наименьшим номером среди приборов с достаточным резервом времени для завершения этого требования в срок. В алгоритме B_2 это требование назначается на прибор с наименьшим достаточным резервом времени.

Алгоритмы B_3 и B_4 отличаются от алгоритмов B_1 и B_2 соответственно лишь тем, что требования в перестановке π расположены в порядке невозрастания значений p_j.

Алгоритм B_i может быть реализован за время $O(n \log m_i)$, где m_i – количество приборов, найденное алгоритмом. Каждый из приборов обслуживает хотя бы одно требование. Очевидно, что $m_i \le n$, $i = 1, 2, 3, 4$.

Оценим точность получаемых решений при помощи алгоритмов $B_1 - B_4$. Не ограничивая общности, будем считать, что $d = 1$ и $p_j \le 1$, $j = 1, 2, \ldots, n$. Докажем ряд вспомогательных утверждений для алгоритмов B_1 и B_2.

Введем в рассмотрение функцию $\psi(x)$, $x \in [0, 1]$:

$$\psi(x) = \begin{cases} \frac{6}{5}x, \ x \in \left[0, \frac{1}{6}\right], \\ \frac{9}{5}x - \frac{1}{10}, \ x \in \left(\frac{1}{6}, \frac{1}{3}\right], \\ \frac{6}{5}x + \frac{1}{10}, \ x \in \left(\frac{1}{3}, \frac{1}{2}\right], \\ 1, \ x \in \left(\frac{1}{2}, 1\right]. \end{cases}$$

Далее будем рассматривать расписания, построенные алгоритмом B_1 либо B_2.

Лемма 2.4. Допустим, что на некоторый прибор назначены требования 1, 2, ..., j. Тогда $\sum_{i=1}^{j}\psi(p_i)\le\frac{17}{10}$. Не ограничивая общности, предположим, что $p_1\ge\ldots\ge p_j$.

Если $p_1\le\frac{1}{2}$, то из определения $\psi(x)$ следует, что $\psi(p_i)\le\frac{3}{2}p_i$, $i=1, 2, ..., j$. Тогда $\sum_{i=1}^{j}\psi(p_i)\le\frac{3}{2}\sum_{i=1}^{j}p_i\le\frac{3}{2}$.

Если $p_1\le\frac{1}{2}$, то $\sum_{i=2}^{j}p_i<\frac{1}{2}$.

В данном случае достаточно показать, что $\sum_{i=2}^{j}\psi(p_i)\le\frac{7}{10}$.

Возможны следующие варианты:

1) $p_2\in\left(\frac{1}{3},\frac{1}{2}\right]$, $p_3\in\left(0,\frac{1}{6}\right]$;

2) $p_2\in\left(\frac{1}{6},\frac{1}{3}\right]$, $p_3\in\left(0,\frac{1}{6}\right]$;

3) $p_2,\ p_3\in\left(\frac{1}{6},\frac{1}{3}\right]$, $p_4\in\left(0,\frac{1}{6}\right]$;

4) $p_2\in\left(0,\frac{1}{6}\right]$.

При первом и втором вариантах имеем соответственно

$$\sum_{i=2}^{j}\psi(p_j)<\frac{6}{5}p_2+\frac{1}{10}+\frac{6}{5}\left(\frac{1}{2}-p_2\right)=\frac{7}{10},$$

$$\sum_{i=2}^{j}\psi(p_i)<\frac{9}{5}p_2-\frac{1}{10}+\frac{6}{5}\left(\frac{1}{2}-p_2\right)=\frac{1}{2}+\frac{3}{5}p_2\le\frac{7}{10}.$$

При третьем варианте имеем

$$\sum_{i=2}^{j}\psi(p_i)<\frac{9}{5}(p_2+p_3)-\frac{2}{10}+\frac{6}{5}\left(\frac{1}{2}-p_2-p_3\right)=\frac{2}{5}+\frac{3}{5}(p_2+p_3)<\frac{7}{10},$$

поскольку $p_2+p_3<\frac{1}{2}$.

При четвертом варианте имеем

$$\sum_{i=2}^{j}\psi(p_i)<\frac{6}{5}\sum_{i=2}^{j}p_i<\frac{6}{10}.$$

Для прибора l обозначим через $\psi(h)$ максимальный резерв времени для приборов $h = 1, 2, \ldots, l-1$, т. е. $Q_l = \max\limits_{1 \le h \le l-1} R_h$. Положим, $Q_0 = 0$.

Лемма 2.5. Для любого требования, назначенного на прибор l с $R_l \ge 1/2$, справедливо $p_i > Q_l$.

Доказательство. Для алгоритма B_1 $p_i > Q_l$ для любого требования, назначенного на прибор l. Рассмотрим алгоритм B_2.

Пусть $Q_l < 1/2$. Тогда $R_h < 1/2$, $h = 1, 2, \ldots, l-1$ и длительность обслуживания первого назначенного требования на прибор l больше, чем Q_l. Обозначим эту длительность через τ. Если $\tau > 1/2$, то лемма доказана.

Если $\tau \le 1/2$, то $R_l \ge 1/2$ и длительность второго назначенного требования на прибор l больше, чем Q_l. Обозначим эту длительность через τ'. Если $\tau + \tau' \le 1/2$, то лемма 2.5 доказана. Если $\tau + \tau' > 1/2$, то длительность третьего назначенного на прибор l требования больше, чем Q_l. Повторяя приведенные рассуждения конечное число раз, получаем истинность утверждения леммы.

При $Q_l \ge 1/2$ имеем $\tau > Q_l \ge 1/2$, что и требуется.

Лемма 2.6. Допустим, на прибор l назначены требования 1, 2, ..., j и $Q_l < 1/2$. Если $\sum\limits_{i=1}^{j} p_i \ge 1 - Q_l$, то $\sum\limits_{i=1}^{j} \psi(p_i) \ge 1$.

Доказательство. Не ограничивая общности, предположим, что $p_1 \ge \ldots \ge p_j$. Если $p_1 > 1/2$, то $\psi(p_1) = 1$ и справедливость утверждения леммы доказана.

Пусть $p_1 \le 1/2$. Поскольку $\sum\limits_{i=1}^{j} p_i \ge 1 - Q_l > 1/2$, имеем $l \ge 2$. Из леммы 2.5 и $p_1 \le 1/2$ следует, что $p_2 > Q_l$.

Возможны варианты:

1) $Q_l \in \left(0, \frac{1}{6}\right]$;

2) $Q_l \in \left(\frac{1}{6}, \frac{1}{3}\right]$;

3) $Q_l \in \left(\frac{1}{3}, \frac{1}{2}\right]$.

В условиях первого варианта

$$\sum_{i=1}^{j} \psi(p_i) \ge \frac{6}{5} \sum_{i=1}^{j} p_i \ge \frac{6}{5}(1 - Q_l) \ge \frac{6}{5} \cdot \frac{5}{6} = 1.$$

Пусть в условиях второго варианта $j = 2$. Поскольку $p_1 + p_2 > 1/2$, имеем либо $p_2 \geq 1/3$, либо $p_1 \geq 1/3$ и $Q_l < p_2 < 1/3$. В первом случае

$$\psi(p_1) + \psi(p_2) = \frac{6}{5}p_1 + \frac{9}{5}p_2 = \frac{6}{5}(p_1 + p_2) + \frac{3}{5}p_2 >$$

$$> \frac{6}{5}(1 - Q_l) + \frac{3}{5}Q_l = \frac{6}{5} - \frac{3}{5}Q_l \geq 1,$$

поскольку $Q_l \leq 1/3$.

Пусть в условиях второго варианта $j \geq 3$. Если $p_2 \geq 1/3$ или $p_1 \geq 1/3$ и $Q_l < p_2 < 1/3$, то, как показано выше, $\psi(p_1) + \psi(p_2) \geq 1$.

Предположим, что $Q_l < p_2 \leq p_1 < 1/3$.

Тогда

$$\sum_{i=1}^{j} \psi(p_i) \geq \frac{9}{5}p_1 + \frac{9}{5}p_2 - \frac{1}{5} + \frac{6}{5}\sum_{i=3}^{j} p_i \geq$$

$$\geq \frac{9}{5}(p_1 + p_2) - \frac{1}{5} + \frac{6}{5}(1 - Q_l - p_1 - p_2) =$$

$$= \frac{3}{5}(p_1 + p_2) - \frac{1}{5} + \frac{6}{5}(1 - Q_l) > \frac{6}{5} - \frac{1}{5} = 1.$$

В условиях третьего варианта из $j \geq 2$ следует $p_1 > 1/3$, $p_2 > 1/3$ и далее $\sum_{i=1}^{j} \psi(p_i) \geq \frac{6}{5}(p_1 + p_2) + \frac{1}{5} > 1$. Лемма 2.6 доказана.

Лемма 2.7. Допустим, на прибор l назначены требования 1, 2, ..., j и $Q_l < 1/2$. Если $\sum_{i=1}^{j} \psi(p_i) = 1 - \alpha$ и $\alpha > 0$, то $p_i \leq 1/2$, $i = 1, 2, ..., j$, и $\sum_{i=1}^{j} p_i \leq 1 - Q_l - \frac{5}{9}\alpha$ при $j \geq 2$.

Доказательство. Не ограничивая общности, предположим, что $p_1 \geq p_2 > Q_l$. Если $p_i > 1/2$, то $\psi(p_i) = 1$, что противоречит условию леммы.

Пусть $l \geq 2$. Поскольку $p_1 \leq 1/2$, то, учитывая лемму 2.6, имеем

$$p_1 \geq p_2 > Q_l, \ \sum_{i=1}^{j} p_i = 1 - Q_l - \gamma,$$

где $\gamma > 0$.

Поскольку $\sum_{i=1}^{j} p_i + \gamma < 1$, можно выбрать числа $\delta_1 \leq 1/2$ и $\delta_2 \leq 1/2$, при этом $\delta_1 \geq p_1$, $\delta_2 \geq p_2$ и $\delta_1 + \delta_2 = p_1 + p_2 + \gamma$.

Заменим требования с длительностями p_1 и p_2 на требования с длительностями δ_1 и δ_2. Очевидно, что алгоритмы B_1 и B_2 распределят новое множество требований по приборам так же, как и исходное с заменой на приборе l требований с длительностями p_1 и p_2 на требования с длительностями δ_1 и δ_2. В силу леммы 2.6 $\sum_{i=3}^{l}\psi(p_i)+\psi(\delta_1)+\psi(\delta_2)\ge 1$.

Для любых x и y $(0\le x<y\le 1/2)$ выполняется соотношение

$$\psi(y)\le\psi(x)+\frac{9}{5}(y-x).$$

Значит,

$$\psi(\delta_1)+\psi(\delta_2)\le\psi(p_1)+\psi(p_2)+\frac{9}{5}\gamma.$$

Отсюда получаем

$$\sum_{i=1}^{j}\psi(p_i)+\frac{9}{5}\gamma\ge 1.$$

Следовательно,

$$\frac{9}{5}\gamma\ge\alpha \text{ и } \gamma\ge\frac{5}{9}\alpha.$$

Лемма 2.7 доказана.

Теорема 2.3. Справедливы неравенства

$$m_1<\frac{17}{10}m^*+2,\ \ m_2<\frac{17}{10}m^*+2.$$

Доказательство. Обозначим $\Psi=\sum_{i=1}^{n}\psi(p_i)$.

Из леммы 2.5 следует, что $\frac{17}{10}m^*\ge\Psi$.

Пусть требования 1, 2, ..., n распределены по приборам. Обозначим через N_l множество требований, назначенных на прибор l.

Пусть $\psi(N_l)=\sum_{i\in N_l}\psi(p_i)$.

Из последовательности приборов, на каждый из которых назначено хотя бы одно требование, выделим подпоследовательность $l'_1, l'_2, \ldots, l'_k$ всех приборов, у которых $\psi(N'_l)=1-\alpha_h,\ \alpha_h>0,\ h=1, 2, \ldots, k$.

Для любого $i\in N_{l'_h}$, $h=1, 2, \ldots, k$, выполняется $p_i\le 1/2$, поскольку в противном случае $\psi(N_{l'_h})\ge 1$.

Отсюда следует, что $Q_{l'_h} < 1/2,\ h = 1, 2, ..., k.$

Покажем, что $Q_{l'_{h+1}} \ge Q_{l'_h} + \frac{5}{9}\alpha_h,\ h = 1, 2, ..., k.$

Очевидно, что $Q_{l'_1} = 0,$ и в силу леммы 2.7 $\sum\limits_{i \in N_{l'_1}} p_i \le 1 - \frac{5}{9}\alpha_1.$

Тогда $Q_{l'_2} \ge \frac{5}{9}\alpha_1.$ В силу леммы 2.7 имеем $\sum\limits_{i \in N_{l'_2}} p_i \le 1 - \frac{5}{9}(\alpha_1 + \alpha_2).$

Поэтому $Q_{l'_3} \ge \frac{5}{9}(\alpha_1 + \alpha_2) = Q_{l'_2} + \frac{5}{9}\alpha_2$ и т. д.

Таким образом,

$$\sum_{h=1}^{k-1} \alpha_h \le \frac{9}{5} \sum_{h=1}^{k-1} \left(Q_{l'_{h+1}} - Q_{l'_h} \right) = \frac{9}{5} \left(Q_{l'_k} - Q_{l'_1} \right) < \frac{9}{10},$$

поскольку $Q_{l'_h} < 1/2,\ h = 1, 2, ..., k.$ Отсюда, а также из $\alpha_k < 1$ получаем $\sum\limits_{h=1}^{k} \alpha_h < 2.$ Как нетрудно убедиться, $\psi(N_l) \ge 1,$ если $\sum\limits_{j \in N_l} p_j = 1.$ Пусть Θ – множество всех таких приборов l, что $\sum\limits_{j \in N_l} p_j = 1.$ Тогда

$$m_1 \le \sum_{l \in \Theta} \psi(N_l) + k \le \sum_{l \in \Theta} \psi(N_l) + \sum_{h=1}^{k} \psi\left(N_{l'_h}\right) + \sum_{h=1}^{k} \alpha_h < \Psi + 2 \le \frac{17}{10} m^* + 2.$$

Аналогично $m_2 < \frac{17}{10} m^* + 2.$ Теорема 2.3 доказана.

Из теоремы 2.3 следует, что $\Delta_H \le \frac{17}{10} + \frac{2}{m^*}$ для алгоритма $H \in \{B_1, B_2\}.$ Поскольку $m^* \ge P,$ где $P \sum\limits_{j=1}^{n} p_j,$ то $\Delta_H \le \frac{17}{10} + \frac{2}{P}.$

Напомним, что здесь предполагается $d = 1$ и $p_j \le 1,\ j = 1, 2, ..., n.$ Если d и $p_j \le 1$ ($j = 1, 2, ..., n$) – произвольные числа, то

$$\Delta_H \le \frac{17}{10} + \frac{2}{\lceil P/d \rceil}.$$

Перейдем к рассмотрению алгоритмов B_3 и B_4.

Теорема 2.4. Справедливы неравенства

$$m_3 \le \frac{11}{9} m^* + 4 \ \text{ и } \ m_4 \le \frac{11}{9} m^* + 4.$$

Доказательство. Схема доказательства теоремы 2.4 аналогична схеме доказательства теоремы 2.3. Оно сводится к построению такой весовой функции, аналогичной $\psi(x)$, что суммарный вес требований, назначенных на один и тот же прибор, не превосходит некоторой константы $C \geq 1$, и число m_3 или m_4 может превосходить суммарный вес всех требований не более чем на некоторую константу C'.

В случае алгоритмов B_1 и B_2 получаем $C = 17/10,\ C' = 2,$ а в случае алгоритмов B_3 и B_4 получаем $C = 9/11,\ C' = 4.$

В случае когда d и $p_j \leq 1,\ j = 1, 2, ..., n$ – произвольные числа, из теоремы 2.3 следует, что $\Delta_H \leq \frac{11}{9} + \frac{4}{\lceil P/d \rceil},\ H \in \{B_3,\ B_4\}.$

2.4.2. Задача распределения работ на конечное число одинаковых процессоров

Рассмотрим задачу распределения работ n на одинаковых процессорах m таким образом, чтобы минимизировать загрузку максимально загруженного процессора. Пусть p_i соответствует времени выполнения i-й работы на процессоре. Попробуем оценить алгоритм, на каждом шаге которого работа загружается на минимально загруженный процессор.

Предположим, что k работ уже распределены и происходит назначение работы $k + 1$. Поскольку суммарная длительность уже распределенных работ равна $p_1 + \ldots + p_k$, то существует процессор, загрузка которого не превышает величину

$$ZK = \frac{p_1 + \ldots + p_n}{m}.$$

В данном случае все процессоры равно загружены. Тогда после назначения работы $k + 1$ на этот процессор его загрузка не превысит величины $Z_k + p_{k+1} = Z_k + p_{k+1}/m + (m-1)p_{k+1}/m = Z_{k+1} + (m-1)p_{k+1}/m.$ Это соотношение справедливо на каждом шаге.

Теперь необходимо найти нижнюю границу оптимального решения. Первую границу можно взять как среднюю загрузку:

$$LB \geq \frac{p_1 + \ldots + p_n}{m},$$

причем $LB \geq Z_k$ для любого k. Таким образом, учитывая предыдущие соотношения, на каждом шаге новая загрузка процессора не превышает величины

$$LB + \frac{(m-1)p_{k+1}}{m}.$$

Теперь осталось воспользоваться тем, что нижняя граница оптимального решения не меньше p_k для любого k, т. е.

$$LB \geq \max\{p_k\}.$$

Следовательно, если предположить, что $LB = \max\{p_k\}$, то алгоритм «в минимально загруженный» для любой последовательности $p_1, p_2, \ldots, p_n$ строит решение, в котором загрузка максимального процессора не превосходит величины $LB + (m-1)LB/m = (2-1/m)LB$.

2.4.3. Задача коммивояжера

В общем случае для задачи коммивояжера построение полиномиального алгоритма с любой гарантированной оценкой точности будет означать построение точного полиномиального алгоритма, поэтому рассматривают версии задач, когда возможно построение приближенного алгоритма с гарантированной оценкой.

1. *Задача коммивояжера на максимум*. Эта версия будет подробно рассмотрена в главе 3.

2. *Метрическая задача коммивояжера*. Главным условием в данном случае предполагается выполнение так называемого неравенства треугольника, когда на матрицу весов накладывается условие, согласно которому для любой тройки вершин i, j, k выполняется

$$C_{i,j} + C_{j,k} > C_{i,k}.$$

Данное соотношение выполняется в любом метрическом пространстве, например, когда вершинам соответствуют точки на плоскости, а расстояния между вершинами равны расстоянию между соответствующими точками в какой-то метрике, в частности евклидовой. Эта версия будет рассмотрена в п. 2.5.

2.5. Методики построения приближенных алгоритмов с гарантированной оценкой точности

Приведем отдельные техники, которые позволяют строить и оценивать приближенные алгоритмы с гарантированной оценкой.

Одним из основных методов разработки приближенных алгоритмов является релаксация. Идея состоит в том, чтобы построить решение данной проблемы P, используя оптимальное или неоптимальное решение другой задачи P'. При таком подходе существуют две возможности.

1. В качестве задачи P' подразумевается ограничение пространства решений задачи P путем отказа от подмножества возможных решений. Наиболее распространенным подходом является решение одной подзадачи, но существуют алгоритмы, которые сначала решают несколько подзадач, затем выдают лучшее из этих решений. Оптимальное или неоптимальное решение исходной задачи P строится одной из стандартных методологий.

2. В данном случае подразумевается расширение пространства возможных решений за счет включения ранее недопустимых. Для этого необходимо решить более общую задачу P''. Алгоритм аппроксимации для P выполняет задачу P'' (оптимально или неоптимально), а затем преобразует в такое решение, которое допустимо для P.

Тесно связан с методом релаксации подход, обеспечивающий ее преобразование к допустимому решению исходной задачи. Идея преобразования состоит в том, чтобы трансформировать недопустимое решение релаксационной задачи к допустимому выполнению исходной задачи. Существуют различные методы преобразования. Алгоритмы аппроксимации, основанные на методологии линейного программирования, подпадают под эту категорию. Проиллюстрируем данный метод двумя классическими задачами: построение минимального дерева Штейнера и задача коммивояжера. Дерево Штейнера и задача коммивояжера являются классическими задачами комбинаторной оптимизации. Алгоритмы выступают наиболее известными методами аппроксимации для любой задачи.

Определим задачу о дереве Штейнера над полным метрическим графом с ребрами $G = (V, E, w)$, где V – множество n вершин; E – множество $m = \left(n^2 - n\right)/2$ ребер; w – весовая функция для ребер $E \to R^+$. Поскольку граф метрический, то набор весов удовлетворяет неравенству треугольника, т. е. для каждой пары вершин $i, j, w(i, j)$ меньше или равно сумме веса ребер на любом пути от вершины i до вершины j.

Задача о дереве Штейнера состоит из метрического графа $G = (V,\ E)$ и подмножества вершин $T \subseteq V$. Проблема состоит в том, чтобы найти дерево, включающее все вершины T, а также некоторые другие вершины в графе, чтобы сумма веса ребер в дереве была наименее возможной. Проблема дерева Штейнера – в *NP*-сложной задаче. При таком подходе можно решать аппроксимационную задачу построения минимального остовного дерева на множестве вершин T в модифицированном графе, в котором веса ребер между соответствующими вершинами

равны длинам кратчайших путей между вершинами в исходном графе. В этом случае используется сведéние исходной задачи Штейнера к более частной задаче построения минимального остовного дерева. Данный подход применяется при построении аппроксимационных схем (PTAS, FPTAS). Продемонстрируем это на примере задачи о рюкзаке, которая формулируется следующим образом:

$$\sum c_i x_i \to \max,$$

$$\sum a_i x_i \le B,$$

$$x_i = 0 \bigcup 1.$$

Вместо решения исходной задачи, для которой можно использовать динамическое программирование с оценкой трудоемкости $\mathrm{O}\left(n \min\left(B, cx^{\text{опт}}\right)\right)$, решается огрубленная задача, в которой новые коэффициенты целевой функции c_i заменяются на

$$c_i = \left[\frac{c_i}{k}\right],$$

где k – некоторое число. Здесь $cx^{\text{опт}}$ – это величина оптимального решения исходной задачи.

Затем решается огрубленная задача с помощью динамического программирования, однако трудоемкость выражается как

$$\mathrm{O}\left(n \min\left(B, \frac{cx^{\text{опт}}}{k}\right)\right).$$

При правильном выборе параметра k относительная погрешность построенного решения не превышает ε, а трудоемкость будет ограничена полиномом от величин n и $1/\varepsilon$.

Лучшие алгоритмы для метрической задачи коммивояжера в качестве алгоритма аппроксимации также используют минимальное остовное дерево, но в этом случае решается более общая задача, так как ослабляются ограничения. Широко известен алгоритм аппроксимации связующего дерева с двойным минимальным весом. Формальный подход заключается в построении эйлерова цикла в мультиграфе (графе с несколькими ребрами между вершинами), состоящем из двух копий ребер остовного дерева. Эйлеров цикл – это путь, который начинается и заканчивается в одной и той же вершине и посещает каждое ребро мультиграфа один раз. Алгоритм строит оставное дерево минимально-

го веса, копирует все ребра в дереве, а затем генерирует эйлеров цикл весом, равным удвоенному весу остовного дерева минимального веса.

Оптимальное решение задачи коммивояжера имеет вес, превышающий минимальный вес остовного дерева, из чего следует, что вес тура, который генерируется алгоритмом, не более чем в два раза превышает значение оптимального решения. Допустимое решение (гамильтонов цикл) получится из эйлерова цикла путем вычеркивания повторяющихся вершин, кроме первой и последней. При условии неравенства треугольника такая трансформация не приводит к увеличению длины получаемого допустимого решения относительно длины эйлерова цикла. Сердюков и Христофидес модифицировали вышеуказанный подход таким образом, чтобы созданные туры имели вес не больше 1,5 веса от оптимального тура.

Заметим, существует множество различных способов преобразования связующего дерева в мультиграф Эйлера. Идея состоит в том, чтобы сделать это с помощью добавления наименьшего количества ребер с наименьшим общим весом. Таким набором может служить совершенное паросочетание минимального веса, которое покрывает вершины нечетной степени в минимальном остовном дереве. В этом случае суммарный вес ребер в мультиграфе, построенном по алгоритму Сердюкова – Христофидеса, не более чем в 1,5 раза больше веса оптимального тура.

Время работы алгоритма Христофидеса – $O\left(n^3\right)$, бóльшая его часть необходима для построения совершенного паросочетания с минимальным весом. Этот подход аналогичен подходу, применяемому Дж. Эдмондсом и К. Джонсоном для задачи китайского почтальона, которая состоит в том, чтобы построить цикл с минимальным весом (возможно, с повторяющимися ребрами), содержащим каждое ребро в графе хотя бы один раз. Лучший алгоритм для решения этой проблемы занимает $O\left(n^3\right)$ времени и использует кратчайшие пути и алгоритмизацию построения совершенного паросочетания минимального веса.

2.6. Оптимальность градиентного алгоритма

Задача о минимальном остовном дереве обладает фундаментальным структурным свойством, которое позволяет получить быстрое алгоритмическое решение. Таким же свойством обладает целый класс задач оптимизации, известных как задачи на *матроидах*. Сначала введем некоторые определения.

Определение 2.5. Системой подмножеств $S=(E,\ J)$ называется конечное множество E вместе с семейством J подмножеств, которое замкнуто относительно включения (если $I\in J$ и $I'\subseteq I$, то $I'\in J$).

Элементы семейства J называются *независимыми*.

Пример 2.7. Задано конечное множество $E=\{1,2,3,4\}$ и семейства подмножеств этого множества

$$J_1=\{(2,3),2,3\},\ J_2=\{(2,3),(1,2),3\}.$$

Семейство J_1 является замкнутым семейством подмножеств и $S=(E,J_1)$ – система подмножеств. Семейство J_2 не является замкнутым.

Определение 2.6. Для каждого элемента $e\in E$ задан вес $w(e)\geq 0$. Комбинаторная задача оптимизации для системы подмножеств (E,J) состоит в нахождении независимого подмножества, которое имеет наибольший общий вес.

Задавать систему подмножеств можно выписывав все независимые подмножества из E, используя подходящее представление. Однако такой способ представления может оказаться очень неэффективным, так как в семействе подмножеств J содержится до $2^{|E|}$ подмножеств. Следовательно, все рассматриваемые системы подмножеств будем представлять с помощью алгоритма, который по данному подмножеству I множества E решает, верно ли, что $I\in J$, т. е. является независимым. Цель заключается в поиске алгоритмов решения комбинаторных задач оптимизации для некоторых систем подмножеств. Рассмотрим общую схему жадного (градиентного) алгоритма.

Жадный алгоритм

1. $I=\varnothing$.
2. Пока $E\neq\{\varnothing\}$, выполнять следующую последовательность шагов:
   а) пусть $e\in E$ и имеет наибольший вес $w(e)$;
   б) удалить e из E, т. е. $E=E\setminus e$;
   в) если $I\cup e\in J$, то $I=I\cup e$.

Некоторые комбинаторные задачи для систем подмножеств могут быть решены корректно (точно) жадным алгоритмом, а другие – нет (получается приближенное решение).

Определение 2.7. Пусть $M=(E,J)$ – система подмножеств. Будем M называть матроидом, если жадный алгоритм корректно решает любую индивидуальную комбинаторную задачу для системы подмножеств M.

Пример 2.8. Пусть E – множество ребер графа $G=(V,E)$, каждому ребру графа приписан вес $w(e)$ и семейство паросочетаний графа X. Семейство X замкнуто относительно включения, так как если из некоторого паросочетания удалить ребро, то оно также останется паросочетанием. Таким образом, $S=(E,X)$ – система подмножеств. Комбинаторная задача оптимизации для системы S заключается в нахождении паросочетания графа максимального веса.

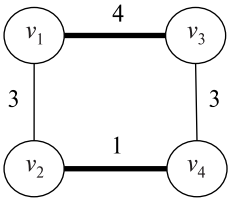


Рис. 2.4

Рассмотрим граф, для которого нужно найти паросочетание максимального веса (рис. 2.4).

Применим жадный алгоритм. Он построит следующее решение: $I=\{(v_1,v_3),(v_2,v_4)\}$, которое имеет вес $w(I)=5$ (на рисунке выбранные ребра выделены). Однако построенное жадным алгоритмом решение не является оптимальным, так как существует множество $I'=\{(v_1,v_2),(v_3,v_4)\}$, которое имеет вес $w(I')=6$. Поскольку $w(I)<w(I')$, то жадный алгоритм не решает корректно задачу о максимальном взвешенном паросочетании, а следовательно, рассмотренная система S не является матроидом.

Пример 2.9. Задан орграф $G=(V,E)$, для каждой дуги $e\in E$ определен вес $w(e)\geq 0$. Требуется найти во множестве E такое подмножество B наибольшего веса, где любые две дуги из B не имеют общего конца.

Рассмотрим систему подмножеств $S=(E,X)$. Подмножество B множества E входит в семейство X только тогда, когда никакие две дуги из B не имеют общего конца. Очевидно, что X – замкнуто, так как если из множества дуг, любые две из которых не имеют общего конца, выбросить некоторую дугу, то оставшееся множество дуг будет удовлетворять требуемому свойству, т. е. никакие две дуги в полученном множестве не будут иметь общего конца.

Рассмотрим ориентированный граф (рис. 2.5).

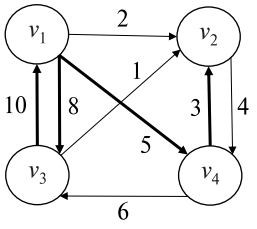


Рис. 2.5

Применим жадный алгоритм, который строит следующее решение (на рисунке выбранные дуги выделены):

$$I=\{(v_3,v_1),(v_1,v_3),(v_1,v_4),(v_4,v_2)\}.$$

Анализируя орграф, приведенный выше, можно заметить, что при выборе дуги, входящей в вершину v_1, не имеет смысла выбирать никакую другую дугу, кроме той, которая имеет наи-

больший вес. Это связано с тем, что данный выбор локален, так как он не влияет на допустимость выборов в других вершинах. То же самое справедливо для всех остальных вершин, поэтому в данной задаче жадный алгоритм будет давать оптимальное решение.

Поскольку данную комбинаторную задачу жадный алгоритм решает корректно, то рассмотренная в примере система подмножеств $S=(E, X)$ является матроидом, который называется *матроидом разбиения*.

Пример 2.10. Задан граф $G=(V, E)$, для каждого ребра $e \in E$ определен вес $w(e) \geq 0$.

Рассмотрим в графе множества ребер, которые являются объединениями вершинно непересекающихся путей. Семейство этих множеств ребер обозначим через *J* (множество ребер *X* независимо, т. е. принадлежит *J*, если оно объединяет вершинно непересекающиеся пути). Комбинаторная задача оптимизации для некоторой системы подмножеств *E* и *J* заключается в нахождении подмножества ребер вершинно непересекающихся путей, имеющего наибольший вес.

На рис. 2.6, *а* изображен исходный граф. Применим жадный алгоритм и получим следующее решение (рис. 2.6, *б*):

$$I=\{\{v_1, v_4\}, \{v_4, v_2\}, \{v_2, v_3\}\},\ w(I)=18.$$

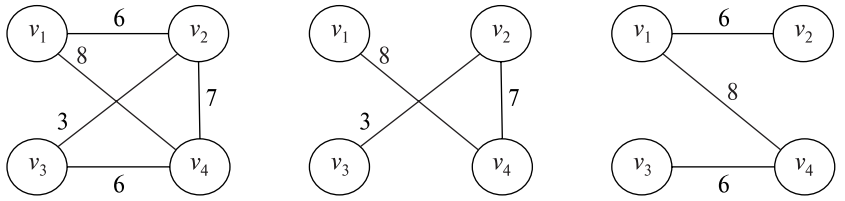


Рис. 2.6

Данное решение не является наилучшим, так как оптимально следующее независимое множество (рис. 2.6, *в*):

$$I'=\{\{v_2, v_1\}, \{v_1, v_4\}, \{v_4, v_3\}\},\ w(I)=20.$$

Приведенный контрпример говорит о том, что рассмотренная система подмножеств не является матроидом.

Пример 2.11. Задана матрица A $n \times m$. Пусть E – множество столбцов этой матрицы $\{e_1, e_2, \ldots, e_m\}$. Тогда $S=(E, J)$ – система подмножеств,

где J – семейство линейно независимых множеств столбцов матрицы A. Жадный алгоритм корректно решает комбинаторную задачу оптимизации, связанную с этой системой подмножеств. Рассмотренная система подмножеств называется *матричным матроидом*.

Задача о минимальном остовном дереве также решалась жадным алгоритмом. Однако эта задача не может быть сформулирована в терминах системы подмножеств, так как если обозначить через J семейство остовных деревьев некоторого графа, то оно не будет являться замкнутым относительно включения (любое подмножество ребер остовного дерева уже не будет являться остовным деревом исходного графа).

Тем не менее существует другая задача о минимальном остовном дереве, но в то же время она может быть сформулирована как комбинаторная задача оптимизации для системы подмножеств.

Пример 2.12. Задан граф $G=(V, E)$. Каждому его ребру поставлен в соответствие некоторый вес $w(e)\geq 0$. Требуется найти лес – ациклический подграф графа G, имеющий максимальный общий вес.

Предположим, что граф является связным. Поскольку веса положительны, то любой оптимальный вес можно сделать максимальным по включению, т. е. остовным деревом графа G. Кроме того, все остовные деревья графа G имеют $|V|-1$ ребер, поэтому можно построить функцию расстояний для E, полагая

$$d_e = W - w(e),\ \forall e \in E,$$

где

$$W = \max_{e\in E} w(e) \text{ для всех } e \in E.$$

При этом сумма расстояний $d(T)$ любого остовного дерева T будет связана с общим весом $w(T)$ дерева T следующим соотношением:

$$w(T) = (|V|-1)W - d(T).$$

Отсюда следует, что минимальное остовное дерево в графе G с расстояниями d совпадает с лесом максимального веса с весами w. Если граф не является связным, то лес максимального веса в графе $G=(V, E)$ с весами w является объединением минимальных остовных деревьев для всех связных компонент графа G с расстояниями d, где d определяется как описано выше.

Таким образом, задачу о минимальном остовном дереве и задачу о лесе максимального веса можно рассматривать как одну и ту же. Следующая теорема является аналогом основного свойства минимального остовного дерева.

Теорема 2.5. Допустим, $\{(U_1, T_1), (U_2, T_2), \ldots, (U_k, T_k)\}$ – остовный лес на множестве V, а $\{v, u\}$ – ребро, исходящее из U_1 и имеющее наибольший вес. Тогда среди всех лесов, содержащих $T = U_{i=1}^{k} T_i$, существует оптимальный, содержащий $\{v, u\}$.

Таким образом, задачу о лесе максимального веса можно решить, используя любой из двух алгоритмов для минимального остовного дерева.

Теорема 2.6. Жадный алгоритм корректно решает задачу о лесе максимального веса.

Пример 2.13. Рассмотрим задачу построения минимального остовного леса для графа (рис. 2.7). Каждому ребру на рисунке поставлен в соответствие вес w.

Данная задача эквивалентна задаче о минимальном остовном дереве в графе (рис. 2.8). Каждому ребру графа поставлено в соответствие расстояние d по формуле

$$d_e = W - w(e), \ \forall e \in E,$$

где

$$W = \max_{e \in E} w(e) = 15.$$

Минимальное остовное дерево графа с матрицей весов d построено, а ребра, вошедшие в дерево, выделены (см. рис. 2.8). Решая эти задачи жадным алгоритмом, получим лес максимального веса (рис. 2.9).

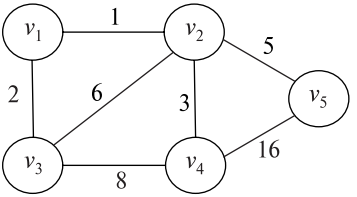


Рис. 2.7

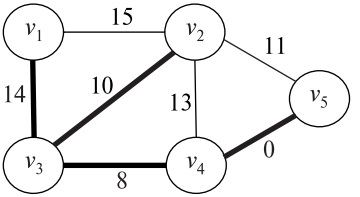


Рис. 2.8

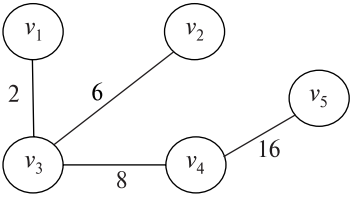


Рис. 2.9

Рассмотренная в примере 2.13 система подмножеств является матроидом, который называется *графическим*. Докажем основную теорему об эквивалентных определениях матроида.

Теорема 2.7. Предположим, $M=(E,J)$ – система подмножеств. Тогда следующие утверждения эквивалентны:

1) M – матроид;

2) если $I_p, I_{p+1} \in J$ (независимые подмножества), где $|I_p| = p$, $|I_{p+1}| = p+1$, то существует элемент $e \in I_{p+1} - I_p$, при котором $I_p \cup e \in J$;

3) если $A \subseteq E$ и I, I' – максимальные по включению подмножества множества A, то $|I| = |I'|$, т. е. максимальные по включению независимые подмножества множества A являются также максимальными и по количеству элементов.

Доказательство. $1 \Rightarrow 2$.

Пусть M – матроид, но второе утверждение не выполняется, т. е. существуют два независимых подмножества I_p, I_{p+1}, где $|I_p| = p$, $|I_{p+1}| = p+1$, но ни для какого элемента $e \in I_{p+1} - I_p$ множество $I_p \cup e \notin J$, т. е. не является независимым.

Рассмотрим функцию весов w на E, которую определим по следующему правилу:

$$w(e) = \begin{cases} p+2, & e \in I_p, \\ p+1, & e \in I_{p+1} - I_p, \\ 0, & e \notin I_p \cup I_{p+1}. \end{cases}$$

Оценим вес множеств I_p и I_{p+1}:

$$w(I_{p+1}) \geq (p+1)(p+1) = p^2 + 2p + 1 > p(p+2) = w(I_p).$$

Таким образом, получим

$$w(I_{p+1}) > w(I_p). \tag{2.1}$$

Из неравенства (2.1) следует, что множество I_p неоптимально.

Применим жадный алгоритм, так как M – матроид. Алгоритм должен построить оптимальное множество для данной функции весов, но он начнет с элементов множества I_p (у них самые большие веса) и все их выберет (так как множество I_p независимо). После этого алгоритм попытается добавлять элементы из множества $I_{p+1} - I_p$, но, по предположению, таких элементов e не существует $\forall e \in I_{p+1} - I_p$, $I_p \cup e \notin J$. Затем останутся лишь элементы нулевого веса, которые не изменят веса оптимального множества, построенного жадным алгоритмом. Таким об-

разом, жадный алгоритм построит множество I' в качестве оптимального, которое имеет такой же вес, что и множество I_p.

Однако в силу (2.1) выполняется условие $w(I') = w(I_p) < w(I_{p+1})$, что противоречит оптимальности множества I', построенного жадным алгоритмом. Следовательно, предположение о том, что ни для какого элемента $e \in I_{p+1} - I_p$ множество $I_p \cup e \notin J$, неверно. Таким образом, утверждение 2) теоремы верно.

Доказательство $2 \Rightarrow 3$.

Предположим, что второе утверждение выполняется. Пусть I, I' – два независимых максимальных по включению подмножества множества $A \subseteq E$. Пусть третье утверждение теоремы не выполняется, т. е. $|I| < |I'|$. Тогда удалим из большего множества I' количество элементов, равное $(|I'| - |I| - 1)$. Получим множество $I'' \subseteq I'$ с количеством элементов

$$|I''| = |I'| - (|I'| - |I| - 1) = |I| + 1.$$

Заметим, что множество I'' является независимым подмножеством множества A (любое подмножество независимого множества также является независимым). Таким образом, имеем два независимых подмножества I, I'', их мощности отличаются на 1; так как выполняется второе утверждение, то существует элемент $e \in I'' - I (e \in A)$, при котором $I \cup e \in J$.

Заметим, что $I \cup e \in A$ и является независимым подмножеством множества A, однако множество I – максимальное по включению независимое подмножество множества A. Приходим к противоречию. Следовательно, сделанное предположение о том, что мощность множества I меньше мощности множества I', неверно (аналогично приходим к противоречию при предположении, что мощность множества I' меньше мощности множества I). Таким образом, $|I| = |I'|$, и третье утверждение теоремы верно.

Доказательство $3 \Rightarrow 1$.

Предположим, что третье утверждение верно, и покажем, что жадный алгоритм корректно решает любую комбинаторную задачу на M, тогда система M будет матроидом. Проведем доказательство от противного. Пусть существует такая функция весов $w(e)$, $\forall e \in E$, при которой жадный алгоритм строит независимое множество $I \in J$, максимальное по включению $I = \{e_1, e_2, \ldots, e_i\}$, в то время как существует другое независимое максимальное по включению множество $\bar{J} = \{e'_1, e'_2, \ldots, e'_j\}$, при котором

$$w(\bar{J}) > w(I). \qquad (2.2)$$

Таким образом, множество, построенное жадным алгоритмом, не является оптимальным. Предположим, что элементы множеств I и $\bar{J}$ упорядочены так, что

$$w(e_1) \geq w(e_2) \geq \ldots \geq w(e_i),$$

$$w(e'_1) \geq w(e'_2) \geq \ldots \geq w(e'_j).$$

Учитывая, что $I, \bar{J} \in J$ являются максимальными по включению, то в силу выполнения третьего утверждения теоремы (при $E = A$) выполняется условие $|I| = |\bar{J}|$, поэтому $i = j$, и мы можем считать, что

$$I = \{e_1, e_2, \ldots, e_i\},$$

$$\bar{J} = \{e'_1, e'_2, \ldots, e'_j\}.$$

Покажем, что для всех $m = 1, 2, ..., i$ верно неравенство

$$w(e_m) \geq w(e'_m). \tag{2.3}$$

Тогда $w(I) \geq w(\bar{J})$, и мы придем к противоречию с неравенством (2.2).

Доказательство проведем по индукции.

Пусть $m = 1$. Неравенство (2.3) верно, так как жадный алгоритм всегда выбирает элемент с самым большим весом $w(e_1) \geq w(e'_1)$. Предположим, что неравенство выполняется для $m \leq i - 1$, т. е. $w(e_s) \geq w(e'_s)$, $\forall s = 1, 2, \ldots, i - 1$.

Нам надо показать, что неравенство верно для $m = i$, т. е. что $w(e_i) \geq w(e'_i)$. Допустим, это не так, т. е.

$$w(e_i) < w(e'_i). \tag{2.4}$$

Рассмотрим множество

$$A = \{e \in E \,|\, w(e) \geq w(e'_i)\}.$$

Множество $\{e_1, e_2, \ldots, e_{i-1}\}$ – максимальное по включению независимое подмножество множества A. Поскольку при условии существования множества $\{e_1, e_2, \ldots, e_{i-1}, e\} \in J$ и в силу неравенства (2.4) должно было бы выполняться неравенство $w(e) \geq w(e'_i) > w(e_i)$, то жадный алгоритм взял бы на своем очередном шаге при построении оптимального множества не элемент e_i, а элемент e. Однако существует независимое множество $\{e'_1, e'_2, \ldots, e'_m\} \in A$ мощностью на 1 больше, чем множество $\{e_1, e_2, \ldots, e_{i-1}\}$. Приходим к противоречию с третьим утверждением теоремы. Следовательно, неравенство (2.4) не выполняется, и индукция по m доказана.

Таким образом, $w(I) \geq w(\bar{J})$, что противоречит предположению (2.2). Следовательно, предположение о существовании множества $\bar{J}$ неверно,

поэтому жадный алгоритм корректно решает любую задачу на M, а следовательно, M – матроид.

В примере 2.7 задача о максимальном взвешенном паросочетании не является матроидом, так как для нее не выполняется третье утверждение теоремы: максимальные по включению независимые подмножества множества A являются также максимальными и по количеству элементов. Максимальное по включению паросочетание не всегда будет максимальным по количеству элементов.

Определение 2.8. Пусть $M=(E, J)$ – матроид и $A \subseteq E$. Ранг $r(A)$ множества A в M – это мощность максимальных по включению независимых подмножеств множества A. Максимальные по включению независимые подмножества множества E называются *базисами*.

В примере 2.13 рассмотренная система являлась матричным матроидом, так как для нее выполняется третье утверждение теоремы. Это следует из хорошо известного факта линейной алгебры: все максимальные линейно независимые подмножества множества векторов E имеют одинаковую мощность. В алгебре она называется рангом подматрицы A, определяемой множеством столбцов E.

Матричные матроиды образуют более широкий класс, чем графические матроиды и матроиды разбиения. Любой графический матроид или матроид разбиения можно представить матричным матроидом, если подобрать соответствующую матрицу A над некоторым полем K. Однако это верно не для всех матроидов, поэтому теория матроидов является существенным обобщением линейной алгебры.

Определение 2.9. Подмножество D множества E, не входящее в J, называется *зависимым*, минимальное по включению зависимое подмножество C в E называется *циклом*.

Теорема 2.8. Допустим, $I \in J$ и $e \in E$. Тогда либо $I+e \in J$, либо $I+e$ содержит единственный цикл. Если мы рассматриваем графический матроид, то циклы матроида – это теоретико-графовые циклы графа G. Ранг подмножества E' из E равен $|V|-c(E')$, где $c(E')$ – число связных компонент графа $G'=(V, E')$.

Задачи для самостоятельного решения

1. Станки – детали с тремя параметрами

Имеется n деталей и m станков. Каждая деталь характеризуется тремя параметрами: временем доставки, обработки и поступления на склад. Станок обрабатывает любую деталь сразу, все станки одинаковы.

Необходимо определить порядок обработки деталей на станках за минимальное время, когда все детали будут на складе.

Формат входных данных

В первой строке через пробел идут числа n деталей и m станков $(1 \le n,\ m \le 300\,000)$. В каждой из последующих n строк записаны время доставки, обработки и поступления на склад очередной детали (неотрицательные целые числа, не превосходящие 10^{13}).

Формат выходных данных

В первой строке выведите минимальное время, полученное алгоритмом. В каждой последующей строке в порядке назначения на станки запишите номер детали и номер станка, на который она назначена. Полученное время не должно превосходить оптимальное больше чем в два раза.

Входной файл	Выходной файл
7 3 2 4 1 0 4 2 1 6 0 1 2 3 0 0 4 3 4 1 2 5 0	11 5 1 2 1 3 2 1 3 4 1 6 1 7 3

2. Станки – детали с двумя параметрами

Имеется n деталей и m станков. Каждая деталь характеризуется двумя параметрами: временем доставки и обработки. Станок обрабатывает любую деталь сразу, все станки одинаковы.

Необходимо определить порядок обработки деталей на станках за минимальное время.

Формат входных данных

В первой строке записаны натуральные числа n и m $(1 \le n,\ m \le 300\,000)$. В каждой из последующих n строк указано время доставки и обработки очередной детали (неотрицательные целые числа, не превосходящие 10^{13}).

Формат выходных данных

В первой строке выведите минимальное время, полученное алгоритмом. В каждой последующей строке в порядке назначения на станки запишите два номера: детали и станка, на который она назначена. Полученное время не должно превосходить удвоенное оптимальное.

Входной файл	Выходной файл
7 3 10 6 1 15 10 10 5 15 14 8 6 7 4 12	37 5 1 3 2 4 3 1 2 7 3 2 1 6 2

3. Станки двух типов

Имеется n деталей и $2m$ станков двух типов. Деталь обрабатывается в две стадии: на станках первого и второго типа. Любая деталь характеризуется двумя параметрами: временем обработки на станках первого и второго типа. Станок обрабатывает каждую деталь сразу, станков разных типов одинаковое число.

Необходимо определить порядок обработки деталей на станках, когда все детали будут обработаны за минимальное время.

Формат входных данных

В первой строке записано число n деталей $(1 \le n \le 300\,000)$, во второй – число m станков одного типа $(1 \le m \le 300\,000)$. Последующие n строк содержат по два неотрицательных вещественных числа, записанных через пробел, – время обработки каждой детали на станке первого и второго типа.

Формат выходных данных

В первой строке выведите минимальное время обработки детали с точностью до пяти разрядов после запятой. В каждой последующей строке в порядке назначения на станки запишите номера детали и станка, на который она назначена. Станки первого типа имеют номера от 1 до m, второго – от $m + 1$ до $2m$. Полученное время не должно быть больше удвоенного оптимального.

Входной файл	Выходной файл	Входной файл	Выходной файл
5 3 2 3 1 4 4 2 5 1 2 2	8.00000 1 1 2 2 5 3 2 5 3 2 1 4 5 6		4 3 3 4 4 4

4. Наименее загруженный работник

Имеется n работ и 2570m работников. Каждая работа характеризуется временем выполнения.

Требуется распределить работы между работниками, чтобы минимально загруженный работник выполнял максимальный объем работы.

Формат входных данных

В первой строке указаны целые числа n и m $(1 \le n,\ m \le 300\,000)$. В каждой из последующих n строк записано время выполнения работы (неотрицательное целое число, не превосходящее 10^{13}).

Формат выходных данных

В первой строке выведите общее время всех работ, которые выполнит минимально загруженный работник, в последующих n строках для каждой работы – номер работника, который ее выполнит. Время работы минимально загруженного работника не должно быть меньше оптимального более чем в полтора раза.

Входной файл	Выходной файл
2 2 1 4	1 2 1

5. Максимальное число работников

Имеется n работ. Каждая работа характеризуется временем ее исполнения (все работники одинаковы). Необходимо так распределить n работ, чтобы каждый работник был загружен не менее x единиц времени, при этом число занятых работников было как можно больше.

Формат входных данных

В первой строке записано положительное целое число x, а затем в каждой строке – время выполнения очередной работы (неотрицательное целое число, не превосходящее 10^{13}). Число работ не превышает 300 000. Сумма времени выполнения всех работ не меньше x.

Формат выходных данных

В первой строке выведите число k занятых работников, далее для каждой работы в отдельной строке – номер работника (от 1 до k), который будет выполнять соответствующую работу. Число занятых работников должно быть не меньше 2/3 от оптимального либо меньше оптимального на 1.

Входной файл	Выходной файл
6 1 3 2 3 5 2 2	3 1 2 3 2 1 3 3

6. Наиболее загруженный работник

Имеется n работ и m работников. Каждая работа характеризуется длительностью ее исполнения любым работником.

Необходимо так распределить n работ, чтобы максимальная загруженность работника была как можно меньше.

Формат входных данных

В первой строке задается число m работников $\left(1 \le m \le 300\,000\right)$, в следующей – последовательно продолжительность работы каждого (неотрицательные целые числа, не превосходящие 10^{13}). Число работ не превосходит 300 000.

Формат выходных данных

В первой строке выведите полученное время работы наиболее загруженного работника, превосходящее минимальное возможное не более чем в 4/3 раза, во второй строке – для каждой работы номер ее исполнителя.

Входной файл	Выходной файл
3 1 1 1 2 2 2	3 1 2 3 3 2 1

7. Сервер и процессоры

Имеется n программ, m одинаковых процессоров и один сервер.

Каждая программа характеризуется временем скачивания данных с сервера и временем выполнения ее на процессоре.

Необходимо так организовать выполнение программ на процессорах, чтобы время завершения программы было минимальным.

С сервера не могут одновременно скачиваться данные более чем для одной программы, и работа не может быть приостановлена. Процессор в любой момент времени может осуществлять только одну опе-

рацию (скачивание или выполнение) ровно с одной программой, которую выполнит только после получения данных для нее. Выполнение программы не может быть приостановлено.

Формат входных данных

В первой строке указано число m процессоров $(1 \le m \le 300\,000)$, во второй – число n программ $(1 \le n \le 300\,000)$. В каждой из последующих n строк записано время загрузки программы и время ее выполнения (неотрицательные целые числа, не превосходящие 10^{13}).

Формат выходных данных

В первой строке выведите время выполнения всех программ, превосходящее минимально возможное не более чем в два раза. В каждой из последующих n строк запишите номер очередной выполняемой программы и номер процессора, на котором будет выполняться эта программа.

Входной файл	Выходной файл
3 6 5 3 4 2 4 3 2 1 8 5 5 4	29 5 1 6 2 1 3 2 1 3 2 4 3

8. Грузовики и камни

Имеется n камней и m машин в очереди. Камни характеризуются массой, машины – грузоподъемностью. Известно, что любой камень помещается в любую машину, и если искомое размещение существует, то общее число машин превосходит минимально возможное не больше чем в два раза.

Необходимо определить порядок загрузки, при котором используется минимальное число машин.

Формат входных данных

В первой строке задано число m грузовиков $(1 \le m \le 300\,000)$, в следующих m строках – грузоподъемности грузовиков (целые числа от 1 до 10^{13}). В следующей строке записано число n камней $(1 \le n,\ n+m \le 500\,000)$, далее в n строках – массы камней (целые числа от 1 до 10^{13}).

Формат выходных данных

Если решение существует, то выведите $2m+1$ строк. В первой строке запишите число m. Для каждой машины в первой строке должна быть указана грузоподъемность, а во второй – последовательно массы камней, положенных в грузовик (пустая строка, если в грузовик ничего не положено). Число используемых машин не должно превышать минимально возможное более чем в два раза. Если решений нет, то выведите no solution.

Входной файл	Выходной файл
6 15 14 17 18 16 20 10 5 6 8 3 5 3 9 4 5 3	6 20 9 8 3 18 6 5 5 17 5 4 3 3 16 15 14

9. Упаковка предметов в контейнеры

Имеется n предметов и много контейнеров. Каждый предмет характеризуется массой и объемом. Грузоподъемность и объем контейнера известны.

Необходимо упаковать предметы в минимальное число контейнеров.

Формат входных данных

В первой строке содержится число n предметов $(1 \le n \le 300\,000)$, далее в n строках записано по два целых числа: масса и объем каждого предмета. Грузоподъемность и объем каждого контейнера одинаковы и равны 100, а каждый предмет помещается в контейнер.

Формат выходных данных

Выведите число k контейнеров, которое не превосходит минимально возможное, в каждой из последующих k строк – характеристики предметов, попавших в один контейнер.

Входной файл	Выходной файл
8 25 28 1 2 56 10 89 90 12 55 51 41 51 10 49 1	6 89, 90; 56, 10; 51, 41; 51, 10; 12, 55; 25, 28; 1, 2; 49, 1;

10. Задача о грузоподъемности машины

Имеется *n* городов, каждый из которых является потребителем либо поставщиком продукции. Для города *k* число x_k характеризует спрос (отрицательное) или предложение (положительное).

Необходимо определить минимальную грузоподъемность машины и маршрут, чтобы объехать все города по разу и удовлетворить потребности (сумма спроса равна сумме предложения).

Формат входных данных

В первой строке записано целое число n $\left(1 \le n \le 300\,000\right)$, во второй – последовательность $\left(x_k\right)$ целых чисел, разделенных пробелами $\left(\left|x_k\right| \le 10^{13}\right)$.

Формат выходных данных

В первой строке выведите необходимую грузоподъемность машины, которая не превосходит удвоенную минимально возможную. Во второй строке запишите последовательность городов, разделенных пробелами, в том порядке, в котором машина будет их объезжать.

Входной файл	Выходной файл
15 -4 16 -3 -5 -8 -9 -3 -8 1 7 4 -5 8 4 5	16 16 -9 8 -8 7 -8 -5 5 -5 4 -4 4 -3 1 -3

11. Задачи с предписанным распределением по процессорам

Имеется *n* программ, *m* одинаковых процессоров и один сервер. Каждая программа характеризуется временем скачивания данных с сервера и временем выполнения ее на процессоре.

Необходимо организовать выполнение программ на процессорах так, чтобы время завершения последней программы было минималь-

ным. Распределение программ по процессорам известно заранее. С сервера не могут одновременно скачиваться данные более чем для одной программы, и работа не может быть приостановлена. Процессор в любой момент времени может осуществлять только одну операцию (скачивание или выполнение) ровно с одной программой и выполнять программу только после получения данных для нее. Работа не может быть приостановлена.

Формат входных данных

В первой строке находится число n программ $(1 \le n \le 300\,000)$, во второй – число m процессоров $(1 \le m \le 300\,000)$. В каждой из последующих n строк записана тройка целых чисел: время скачивания данных с сервера (x_i), время работы на процессоре (y_i) и номер процессора для i-й программы (z_i). При этом $0 \le x_i,\ y_i \le 10^{13},\ 1 \le z_i \le m$.

Формат выходных данных

В первой строке выведите полученное время завершения выполнения программ, которое не превосходит удвоенное оптимальное, во второй – номера программ в порядке скачивания данных с сервера.

Входной файл	Выходной файл
4 3 3 10 1 4 5 3 7 6 2 5 5 1	24 1 2 3 4

12. Задача коммивояжера на плоскости

Решите задачу с евклидовой метрикой, если положения городов задаются их координатами.

Формат входных данных

В первой строке указано целое число n городов $(3 \le n \le 3000)$. В каждой из последующих n строк записана пара (x, y) действительных чисел – координаты городов (целые числа, по модулю не превосходящие 10^9).

Формат выходных данных

В первой строке выведите длину маршрута с точностью до трех знаков после запятой. Эта длина не должна превосходить оптимальную более чем в два раза. Во второй строке запишите вершины полученного маршрута в порядке посещения.

Входной файл	Выходной файл
4 0 0 1 1 0 1 1 0	4.000 3 1 4 2

13. Сумма векторов

Необходимо указать такой порядок суммирования векторов в R^m, при котором все частичные суммы попадают в шар минимального радиуса в евклидовой метрике. Известно, что сумма всех векторов равна $\vec{0}$.

Формат входных данных

В первой строке записано число n векторов $(1 \le n \le 3000)$, во второй – размерность m пространства $(1 \le m \le 10)$. Далее выведено n строк, содержащих по m целых чисел, – координаты каждого вектора (число не превосходит 10^9 по абсолютной величине).

Формат выходных данных

Выведите n строк, содержащих номера векторов в порядке суммирования. Радиус шара, определяемый предложенным порядком суммирования, не должен превосходить минимально возможный более чем в $2m$ раз.

Входной файл	Выходной файл
4 2 -4 2 2 1 3 -1 -1 -2	1 3 4 2

14. Детали с суммарным временем обработки

Имеется m станков и детали, о которых известно только суммарное время s их обработки. Каждая деталь характеризуется временем обработки, все станки одинаковы и обрабатывают любую деталь сразу. При этом нет никакой информации о следующей по очереди детали и их количестве. Поступившая деталь должна быть сразу назначена на один из станков.

Необходимо определить порядок обработки деталей на станках, при котором на все детали затрачено минимальное время.

Формат входных данных

В первой строке записано целое число m станков $(1 \le m \le 300\,000)$, во второй – целое число s – суммарное время обработки $\left(0 \le s \le 10^{18}\right)$, в третьей – положительные целые числа, разделенные пробелом,– информация о времени обработки очередной детали. Число деталей не превосходит 300 000.

Формат выходных данных

В первой строке выведите полученное время обработки деталей, которое не превосходит оптимальное более чем в $2m/(m+1)$ раз, во второй – для каждой детали номер станка, на котором она обрабатывается.

Входной файл	Выходной файл
4 154 12 10 5 7 16 11 15 17 18 9 14 3 10 7	39 1 2 2 3 3 1 3 4 4 1 2 4 2 1

15. Работники со списком работ

Дано m работников и n работ. Каждый из них может выполнить любую работу и характеризуется скоростью исполнения, работа – временем, необходимым работнику с единичной скоростью для ее выполнения.

Надо реализовать алгоритм, позволяющий распределить работу таким образом, чтобы минимизировать время ее выполнения самым загруженным (по времени) работником. При этом все работники выполняют почти одинаковое число работ (для некоторого k каждый работник выполняет k либо $k-1$ работ).

Формат входных данных

В первой строке указаны разделенные пробелом число n работ и число m работников $(1 \le n,\ m \le 3000)$, далее – n строк действительных чисел – время выполнения задания работником единичной скорости. Затем следует m строк действительных чисел, характеризующих скорость выполнения работы.

Формат выходных данных

В первой строке выведите время выполнения работы самым загруженным работником, далее для каждой работы в отдельной строке – номер ее исполнителя (целое число от 1 до m). Полученное время работы не должно превосходить удвоенное оптимальное.

Входной файл	Выходной файл
4 2 1.0 3.0 3.0 5.0 1.0 1.0	6.0 1 2 2 1

16. Назначение работ рабочим с различной производительностью

Имеется n работ и m работников. Каждая работа характеризуется длительностью ее исполнения работником единичной производительности, для каждого из них она известна.

Необходимо так распределить работу среди работников, чтобы минимизировать время выполнения последней работы.

Формат входных данных

В первой строке задано число n работ $(1 \le n \le 3000)$, во второй – n натуральных чисел, которые характеризуют время ее выполнения работником единичной производительности. В третьей строке записано число m работников $(1 \le m \le 3000)$, в четвертой – m натуральных чисел, характеризующих производительность каждого работника.

Формат выходных данных

В первой строке выведите время выполнения всех работ с пятью знаками после запятой, во второй – для каждой работы номер ее исполнителя (число от 1 до m). Полученное время выполнения всех работ не должно превосходить удвоенное оптимальное.

Входной файл	Выходной файл
9 2 2 2 2 2 2 2 2 4 16 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2.00000 2 3 4 5 6 7 8 9 1

17. *n* работ, 1 исполнитель

Есть n работ и один работник. Для каждой из них существует время выполнения и время подготовки. Штраф за работу – разность между реальным и возможным временем начала работы.

Необходимо минимизировать суммарный штраф.

Формат входных данных

Первая строка содержит число n работ $(1 \le n \le 300\,000)$, следующие n строк – по два целых числа от 0 до 10^8: время задержки и время выполнения.

Формат выходных данных

В первой строке выведите число n работ и полученный штраф, во второй – номера работ в порядке выполнения. Штраф не должен превосходить минимально возможный более чем в $n - 1$ раз.

Входной файл	Выходной файл
5 0 10 0 1 1 1 2 1 3 1	5 4 2 3 4 5 1

18. Обработка в две стадии

Имеется n деталей, m станков первого и один станок второго типа. Каждая деталь обрабатывается в две стадии: сначала на станке первого типа, затем на станке второго, и характеризуется временем обработки на каждом станке. Станок обрабатывает любую деталь сразу.

Необходимо определить порядок обработки всех деталей на станках за минимальное время.

Формат входных данных

В первой строке находится число m станков первого типа $(1 \le m \le 300\,000)$, во второй – число n деталей $(1 \le n \le 300\,000)$. В каждой из последующих n строк записано по два целых числа a и b, где a – время обработки i-й детали на станке первого типа, b – на станке второго типа $\left(0 \le a;\ b \le 10^{13}\right)$.

Формат выходных данных

В первой строке выведите полученное время обработки всех деталей. В каждой из следующих m строк запишите число обработанных деталей и их номера в порядке обработки на первом станке, в последней – номера деталей в порядке обработки на втором. Найденное время работы не должно превышать удвоенное оптимальное.

Входной файл	Выходной файл
3 6 20 3 16 8 19 1 16 5 15 20 15 6	58 2 1 2 2 3 4 2 5 6 5 1 2 4 6 3

19. Задача коммивояжера с прямоугольной метрикой

Решите задачу коммивояжера на плоскости, если положение городов задается координатами, а расстояние считается в прямоугольной метрике.

Формат входных данных

В первой строке находится число n городов $(1 \le n \le 10\,000)$. В каждой из последующих n строк записано по два целых числа x_i и y_i – координаты i-го города $(|x_i|, |y_i| \le 10^9)$.

Формат выходных данных

В первой строке выведите $n+1$ целых чисел – города в порядке их обхода. Номер первого города в списке должен совпадать с номером последнего. Во второй строке выведите длину маршрута, которая не должна превышать длину оптимального более чем вдвое.

Входной файл	Выходной файл
13 4 4 -4 -4 -4 4 4 -4 3 3 -3 -3 -3 3 3 -3 2 2 -2 -2 -2 2 2 -2 1 1	1 5 9 13 3 7 11 2 6 10 12 8 4 1 46

20. Разделение на зоны

В городе имеется несколько транспортных маршрутов. Известна стоимость проезда между станциями.

Необходимо поделить город на k зон, установить в каждой из них стоимость проезда, а также стоимость проезда между зонами, чтобы максимальное изменение цены проезда между станциями было минимально, т. е. абсолютная величина разности новой и старой стоимости.

Формат входных данных

В первой строке содержатся два целых числа k и n – числа зон и станций соответственно $(1 \le k \le n \le 15,\ k \le 5)$. Далее следует n строк по n чисел. На месте (i, j) стоит стоимость проезда от станции номер i до станции номер j (если 0, то маршрута между этими станциями нет). Стоимости – целые числа от 0 до 10^9.

Формат выходных данных

В первых k строках выведите номера станций по зонам, разделенные пробелом: в i-й строке отсортированные по возрастанию номера станций, принадлежащих i-й зоне, а также стоимость проезда в зоне. Строки должны быть отсортированы по возрастанию, станции нумеруются с нуля. Затем выведите k строк, i-я из которых должна содержать стоимость проезда между i-й зоной и всеми остальными, или 0, если между зонами нет проезда. Максимальное полученное изменение цены проезда не должно превышать удвоенное оптимальное.

Входной файл	Выходной файл
3 5 0 4 0 0 1 4 0 2 5 0 0 2 0 3 0 0 5 3 0 1 1 0 0 1 0	0 1 3 4.5 2 0.0 4 0.0 0.0 2.5 1.0 2.5 0.0 0.0 1.0 0.0 0.0

21. Расписание для последовательных станков

Имеется n деталей и m станков. Для каждой детали известно время и последовательность выполнения на каждом станке.

Необходимо определить порядок обработки деталей (ни один станок не используется более одного раза), при котором последняя обработается как можно раньше. Запрещено останавливать станок до окончания обработки очередной детали.

Формат входных данных

В первой строке записано два целых числа n и m $(1 \le n \le 3000, 1 \le m \le 300)$. Далее следуют n блоков. При этом i-й блок начинается со строки, содержащей число k_i этапов в процессе обработки i-й детали.

Затем следуют k_i строк, задающих последовательность прохождения станков i-й деталью. Каждая строка включает два положительных целых числа – номер станка и время обработки, которое не превосходит 10^{12}.

Формат выходных данных

В первой строке отразите время окончания обработки последней детали. Далее выведите m строк по n чисел в каждой. В i-й строке j-е число соответствует времени, когда i-й станок начнет обработку j-й детали, или равно 0, если j-ю деталь не надо обрабатывать на i-м станке. Затраченное время не должно превосходить оптимальное более чем в $(n+1)/2$ раз.

Входной файл	Выходной файл	Входной файл	Выходной файл
5 5 4 1 10 2 10 3 10 4 10 4 2 10 3 10 4 10 5 10 4 3 10	40 0 0 30 20 10 10 0 0 30 20 20 10 0 0 30 30 20 10 0 0 0 30 20 10 0	4 10 5 10 1 10 4 4 10 5 10 1 10 2 10 4 5 10 1 10 2 10 3 10	

22. Частичный порядок обработки деталей

Имеется n деталей (пронумерованы от 1 до n) и m последовательных станков. Для каждой детали известно время и частичный порядок обработки на каждом станке.

Необходимо определить порядок, при котором последняя деталь обработается как можно раньше.

Формат входных данных

В первой строке указаны целые числа n и m $(2 \le n,\ m \le 100)$. Далее в n строках по m элементов в каждой идут целые числа x $(1 \le x \le 10^{13})$.

Если число x находится в i-й строке на j-м месте, то время обработки i-й детали на j-м станке равно x. В следующей строке идет число k деталей, от которых зависит хотя бы одна другая деталь $(0 \le k \le 100)$. Далее в k строках записаны числа: первое число в строке – номер a детали, далее – номера деталей, зависящих от нее по времени (должны выполнится позже a). Гарантируется существование решения задачи при заданных входных условиях.

Формат выходных данных

Выведите в первой строке минимальное время обработки последней детали. Далее в m строках отразите перестановки чисел от 1 до n – в i-й строке порядка обработки деталей на i-м станке (если вариантов несколько, то выведите любой). Полученное время обработки деталей не должно превосходить оптимальное более чем в $(m+1)/2$ раз.

Входной файл	Выходной файл
4 2 2 2 2 2 2 2 1 2 1 3 2	9 4 3 2 1 4 3 2 1

23. Минимальная стоимость проезда

Имеется n городов и m машин. Изначально машины могут располагаться в любых городах, которые задаются координатами точек на плоскости.

Необходимо объехать каждый город только один раз, при этом минимизировать стоимость проезда. Каждой машине нужно вернуться в тот же город, из которого она выехала.

Формат входных данных

В первой строке записано два числа n и m – числа городов и машин соответственно $(1 \le m \le n \le 10\,000)$. Далее идут n строк, содержащих по паре целых чисел, не превосходящих 10^9 по абсолютной величине, – координаты городов.

Формат выходных данных

В первой строке выведите полученную сумму длин маршрутов. Далее в m строках запишите число городов, посещенных очередной машиной, и их номера в порядке посещения. Сумма длин маршрутов не должна превосходить удвоенную минимально возможную.

Входной файл	Выходной файл
6 1 171 40 172 58 209 95 228 71 226 47 198 41	180.701 6 1 2 3 4 5 6 1

24. Станки – детали

Имеется n деталей и m станков. Каждая деталь характеризуется одинаковым временем обработки на любом станке. Станок последовательно обрабатывает по одной детали в каждый момент времени.

Необходимо определить такой порядок обработки деталей на станках, при котором время окончания обработки последней было бы минимальным.

Формат входных данных

В первой строке содержатся два целых числа n и m, где n – число деталей, а m – станков $(1 \le n \le 100\,000,\ 1 \le m \le 50\,000)$, во второй – n целых чисел от 1 до 100 000 – время обработки каждой детали.

Формат выходных данных

В первой строке выведите время окончания обработки всех деталей, во второй – для каждой детали номер станка, на котором она будет обработана. Полученное время работы не должно превосходить 4/3 от оптимального.

Входной файл	Выходной файл
4 2 2 3 4 2	6 1 1 2 2

В приведенном примере детали с номерами 1 и 2 будут обрабатываться на первом станке, а детали 3 и 4 – на втором, или наоборот. Тогда первый станок завершит выполнение через $t_1 = 5$ единиц времени с начала обработки, а второй – через $t_2 = 6$. Таким образом, минимальное время завершения обработки всех деталей $T = \max\{t_1, t_2\} = 6$.

25. Распределенная задача с сервером

Есть сервер и два процессора. Существует n_1 задач, которые могут выполняться только на первом процессоре, и n_2 задач – только на втором процессоре. Перед выполнением каждую задачу надо загрузить

с сервера на необходимый процессор, что занимает время $s_{k,i}$ $\left(s_{k,i} > 0\right)$. После этого процессор начинает исполнение только что загруженной задачи, что занимает у него время $p_{k,i}$ $\left(p_{k,i} > 0\right)$. В каждый момент времени сервер может заниматься загрузкой и выполнением не более одной задачи. Процессор не может одновременно загружать одну задачу и исполнять другую.

Необходимо найти такое расписание, при котором выполнялись бы все задачи, а время выполнения последней было бы наименьшим.

Формат входных данных

В первой строке записаны числа n_1 и n_2 $\left(1 \le n_1,\ n_2 \le 100\,000\right)$. Далее идет n_1 пар $\left(p_{1,i},\ s_{1,i}\right)$ чисел и n_2 пар $\left(p_{2,i},\ s_{2,i}\right)$ чисел. Все числа положительные целые, не превосходящие 10^{13}.

Формат выходных данных

В первой строке выведите время окончания выполнения последней задачи, в последующих $n_1 + n_2$ строках – порядок, в котором задачи загружаются с сервера.

Задача задается двумя числами – номер процессора и задачи. Задачи нумеруются с 1 в порядке появления во входном файле. Полученное расписание не должно превышать по общему времени оптимальное более чем в полтора раза.

Входной файл	Выходной файл
1 2 2 5 4 1 5 5	15 2 2 1 1 2 1

В примере одна задача для первого процессора: время выполнения – 2, время загрузки – 5, а также две задачи для второго процессора. Задачи можно выполнить за 15 единиц времени. Первой загружается задача 2 для второго процессора, потом задача 1 – для первого, потом задача 1 – для второго.

26. Нераспределенная задача с сервером

Имеется один сервер и два процессора. Есть *n* задач, каждая из которых сначала должна быть загружена с сервера на любой процессор, а затем исполняться на нем. Сервер может заниматься не более чем одной загрузкой и исполнением не более одной задачи в каждый момент времени, при этом одновременное исполнение одной задачи и загруз-

ка другой невозможна. Для задачи i известно время s_i загрузки с сервера на процессор и время p_i исполнения на процессоре.

Необходимо найти такое расписание, при котором выполнились бы все задачи, а время завершения последней было бы минимально возможным.

Формат входных данных

В первой строке указано число n, далее следуют n строк по два положительных целых числа s_i и p_i в каждой $\left(1 \le n \le 300\,000,\ 1 \le s_i,\ p_i \le 10^{13}\right)$.

Формат выходных данных

Выведите в первой строке время завершения обработки всех задач, а во второй — порядок загрузки задач с сервера (перестановку чисел от 1 до n). Полученное время не должно превышать оптимальное более чем в 3/2 раза.

Входной файл	Выходной файл
4 12 3 8 7 3 12 1 14	33 3 1 4 2

27. Мультиразрез

Дан взвешенный неориентированный граф, зафиксировано некоторое подмножество T его вершин.

Необходимо удалить из графа множество ребер с минимальным суммарным весом так, чтобы все вершины множества T оказались в разных компонентах связности.

Формат входных данных

Первая строка содержит число n вершин и число m ребер графа $\left(1 \le n \le 150\right)$. Следующие m строк содержат тройки $\left(u, v, w\right)$ чисел, свидетельствующие о наличии ребра между вершинами u и v весом w $\left(1 \le u,\ v \le n,\ 0 \le w \le 1\,000\,000\right)$. В $(m+2)$-й строке находится число k вершин, входящих во множество T $\left(2 \le k \le n\right)$. Далее записаны k различных чисел от 1 до n — номера вершин, входящих во множество T.

Формат выходных данных

В первой строке выведите вес удаляемого множества, во второй — число r ребер, входящих в это множество, в следующих r строках — номера ребер, входящих в это множество (по одному в строке). Ребра нумеруются с 1 в том же порядке, в котором они были заданы на входе.

Вес удаляемого множества должен отличаться от оптимального не более чем в $2-2/|T|$ раз.

Входной файл	Выходной файл
6 7 1 2 3 2 3 1 2 6 3 5 6 1 3 5 5 3 4 2 4 5 3 3 1 3 5	9 4 2 4 5 6

28. Flow-shop permutation

Имеется *n* работ и *m* приборов. Каждая работа обслуживается приборами в порядке 1, 2, ..., *m*. Обслуживание требования следующим по порядку прибором может быть начато не ранее завершения его обслуживания предыдущим прибором.

Требуется составить такое расписание работ на всех приборах, чтобы время обслуживания последней было минимальным. В любой момент времени каждая работа должна обслуживаться не более чем одним прибором и каждый прибор должен обслуживать не более одной работы. При этом никакой прибор не может простаивать, если некоторая работа готова к выполнению на нем.

Формат входных данных

В первой строке записаны два числа *n* и *m* $(1 \le n,m \le 1000)$. Далее следует *n* строк по *m* чисел в каждой: время выполнения *i*-й работы на *j*-м приборе – неотрицательное целое число, не превосходящее 10^9.

Формат выходных данных

В первой строке выведите время выполнения всех *n* работ на *m* приборах. В каждой из следующих *m* строк запишите по *n* чисел – номера работ в порядке выполнения на соответствующем станке. Полученное время выполнения не должно превышать оптимальное, умноженное на $\lceil m/2 \rceil$.

Входной файл	Выходной файл
2 3 1 4 3 5 3 7	16 1 2 1 2 1 2

29. Open-shop

Имеется n работ и m приборов. В любой момент времени каждая работа должна обслуживаться не более чем одним прибором, а каждый прибор выполнять не более одной работы. При этом никакой прибор не может простаивать, если некоторая работа готова к выполнению.

Требуется составить такой маршрут обслуживания каждой работы на каждом приборе, чтобы время обслуживания последней было минимально.

Формат входных данных

В первой строке записаны числа n и m $(1 \le n,\ m \le 1000)$, далее следуют n строк по m чисел в каждой. В i-й строке j-е число задает время выполнения i-й работы на j-м приборе (каждое число неотрицательное целое, не превосходящее 10^9).

Формат выходных данных

В первой строке выведите время выполнения всех n работ на m приборах.

В каждой из nm строк выведите по два числа: номера работы и станка. Каждая такая пара чисел соответствует началу выполнения определенной работы на соответствующем станке. Все события должны следовать в хронологическом порядке. Полученное время выполнения не должно превышать удвоенное оптимальное.

Входной файл	Выходной файл	Входной файл	Выходной файл
3 4 4 12 4 6 8 6 2 10 2 4 18 3	27 1 4 2 2 3 3 1 2 2 1 2 4		1 3 3 1 3 2 1 1 2 3 3 4

30. Поиск гамильтонова цикла

Имеется полный граф на n вершинах. Каждая вершина задается своими целочисленными координатами на плоскости. Вес ребра $\{u, v\}$ задается следующей формулой:

$$d(u, v) = |x_u - x_v| + |y_u - y_v|.$$

Нужно найти гамильтонов цикл минимального веса в этом графе.

Формат входных данных

Первая строка содержит целое число n $(3 \le n \le 10\,000)$, далее следуют n строк с координатами вершин: i-я строка задает координаты

i-й вершины, разделенные пробелом. Координаты – целые числа, не превосходящие по модулю 100 000.

Формат выходных данных

В первой строке выведите вес найденного гамильтонова цикла, во второй – $(n+1)$ чисел – номера вершин в порядке обхода в цикле. Вес найденного цикла не должен превосходить удвоенный вес оптимального.

Входной файл	Выходной файл
5 3 7 7 2 6 3 8 8 4 7	22 1 5 4 2 3 1

31. Пожарные станции

В стране есть n городов, связанных двусторонними дорогами. Из каждого города можно добраться в любой другой.

Необходимо разместить k пожарных станций в городах таким образом, чтобы максимальное расстояние от города до пожарной станции было наименьшим.

Формат входных данных

В первой строке заданы три числа: число n городов, число m дорог и число k пожарных станций $(1 \le k \le n \le 1000,\ k \le 200)$. В последующих m строках расположено по три натуральных числа i, j и d, где d – длина дороги из города i в город j $\left(1 \le d \le 10^9\right)$. Каждая пара городов соединена не более чем одной дорогой.

Формат выходных данных

В первой строке выведите полученное максимальное расстояние от города до пожарной станции, во второй – k номеров городов, в которых размещаются станции. Полученное расстояние не должно превосходить удвоенное оптимальное.

Входной файл	Выходной файл
6 7 3 1 2 10 1 3 5 2 4 3 3 4 1 4 5 19 4 6 15 5 6 2	3 1 4 5

32. Наименьшее максимальное паросочетание

Паросочетание – независимое множество ребер графа, в котором любые два ребра не смежны (не имеют общих вершин).

Максимальное паросочетание – паросочетание, которое не содержится ни в каком другом паросочетании.

Наименьшее максимальное паросочетание – максимальное паросочетание наименьшей мощности.

Необходимо найти в заданном графе наименьшее максимальное паросочетание.

Формат входных данных

Первая строка содержит число n вершин и число m ребер $(1 \le n \le \le 500\,000,\ 0 \le m \le 500\,000)$.

Каждая из последующих m строк содержит пару номеров концов очередного ребра (числа от 1 до n). Гарантируется, что граф не содержит петель и кратных ребер.

Формат выходных данных

В первой строке выведите мощность максимального паросочетания, не более чем в два раза превосходящую минимальную возможную, во второй – номера (начиная с 1) входящих в него ребер.

Входной файл	Выходной файл
4 3 1 2 2 3 3 4	1 2

33. Всемирная выставка

В некоторых из 11 комнат установлены голографические проекторы, каждый из которых создает объемное изображение не в самой комнате, а в запутанных переходах.

Один проектор, установленный в комнате, может работать по всей длине переходов, ведущих в эту комнату (какими бы запутанными они ни были!), и только в них.

Нужно обеспечить каждый коридор хотя бы одним проектором. Следует минимизировать затраты на проекторы или потратить не более чем вдвое больше по сравнению с минимально возможной суммой.

Формат входных данных

В первой строке содержатся число n комнат и число m переходов $(1 \le n \le 200\,000,\ 0 \le m \le 500\,000)$. В каждой из следующих m строк содер-

жится пара номеров комнат (числа от 1 до n), соединенных очередным переходом. Гарантируется, что каждый переход соединяет две разные комнаты, никакие из них не соединены более чем одним переходом.

Формат выходных данных

В первой строке выведите число k используемых проекторов, во второй – k номеров комнат с проекторами.

Входной файл	Выходной файл
4 6 1 2 1 3 1 4 2 3 2 4 3 4	3 1 2 3

34. Два станка

Есть два одинаковых станка и n деталей. Для каждой детали известно время ее обработки на одном станке, при этом выполнение работы не может быть приостановлено.

Необходимо распределить детали между станками, чтобы время завершения обработки последней из них было минимальным.

Формат входных данных

В первой строке идет число n деталей $(1 \le n \le 300\,000)$, во второй запишите n чисел, разделенных пробелом, – время обработки каждой детали (неотрицательные целые числа, не превышающие 10^{13}).

Формат выходных данных

В первой строке выведите время, полученное алгоритмом, во второй – определите для каждой детали номер станка, на который она назначена. Полученное время обработки не должно превышать минимально возможное более чем на 4 %.

Входной файл	Выходной файл
5 2 2 2 3 3	6 2 2 2 1 1

35. Искусство

Существует интернет-ресурс, на котором определяются рейтинги разных произведений искусства: картин, книг, музыкальных композиций, фотографий или фильмов. Рейтинг каждого произведения опре-

деляется в результате сравнения с другим произведением. Каждое сравнение будем называть *голосом*.

Необходимо ранжировать произведения искусства с учетом результатов голосования. При этом иногда принципиально невозможно расставить все произведения по порядку, чтобы каждый пользователь говорил, что произведение с меньшим рангом хуже, чем с бóльшим. Нужно найти порядок, учитывающий максимальное число голосов.

Формат входных данных

В первой строке записано число n произведений искусства и число m голосов $(1 \le n,\ m \le 300\,000)$. Все произведения искусства пронумерованы числами от 1 до n. В каждой из следующих m строк записана тройка чисел x, y и z $(1 \le x,\ y \le n,\ x \ne y,\ 1 \le z \le 2)$, соответствующих одному голосу, где x и y – номера сравниваемых произведений. Если произведение x понравилось пользователю больше, чем y, то $z = 1$, в противном случае – 2.

Формат выходных данных

В первой строке выведите число голосов, которые будут учтены в предложенном порядке, во второй запишите перестановку чисел от 1 до n – порядок, при котором не меньше половины от максимально возможного числа голосов предпочтение отдается произведению, идущему в этом порядке раньше.

Входной файл	Выходной файл
3 3 1 2 1 2 3 1 1 3 2	1 3 2 1

В примере максимальное число голосов, которые можно учесть, равно двум. Предложенный ответ учитывает только один – третий голос.

Глава 3 АЛГОРИТМЫ ДЛЯ ЗАДАЧИ КОММИВОЯЖЕРА

Пусть $G=(V,E)$ – полный n-вершинный граф (орграф) и $f: E \to R^+$ – весовая функция, заданная на ребрах (дугах).

Гамильтонов цикл (*орцикл*) в графе (орграфе) G – это цикл (орцикл), содержащий каждую вершину из G один раз. Подмножество $T \subset E$ ребер (дуг) в G называется *частичным туром,* если существует гамильтонов цикл (орцикл), содержащий все ребра из T. Ребро (дугу) $e \in E$ назовем *допустимым* (*допустимой*) для частичного тура T, если $T \cup \{e\}$ – частичный тур. Через $f(I)$, где $I \subseteq E$, обозначим суммарный вес ребер (дуг) из I.

Симметричной (несимметричной) задачей коммивояжера на максимум (ЗКМ) назовем задачу нахождения гамильтонова цикла (орцикла) X^* максимального веса $f(X^*)$ в G. Для несимметричной ЗКМ, в отличие от симметричной, не обязательно, чтобы $f(i,j)=f(j,i)$.

Одним из наиболее простых приближенных алгоритмов решения ЗКМ является алгоритм «иди в дальний»: начиная с произвольной вершины, на каждом шаге i к уже построенному частичному туру T_{i-1} присоединяем ребро e_i максимального веса среди допустимых ребер, смежных с последним ребром, включенным в частичный тур T_{i-1}. Трудоемкость алгоритма «иди в дальний» составляет $O(n^2)$ операций. Достижимая оценка погрешности такого алгоритма для симметричной ЗКМ равна 1/2.

Градиентный алгоритм 3.1 решения ЗКМ, в отличие от алгоритма «иди в дальний», не требует связности частичного тура, получаемого на каждом шаге алгоритма.

Алгоритм 3.1. Шаг i, $i = 1, 2, ..., n$. Выбираем ребро (дугу) e_i, имеющее (имеющую) максимальный вес среди допустимых ребер (дуг) для частичного тура $T_{i-1}(T_0=\varnothing)$, и полагаем $T_i = T_{i-1} \cup e_i$. Гамильтонов цикл (орцикл) T_n называем *градиентным* и обозначаем X^g.

Теорема 3.1. Для алгоритма 3.1 справедливы следующие достижимые оценки погрешности: $(1-1/\alpha)/2$, где $\alpha = n$ при нечетном n, $\alpha = n-1$ при четном n (симметричная ЗКМ), для несимметричной ЗКМ – 2/3.

Примеры, которые показывают, что оценки достижимы, приведены на рис. 3.1, *а*, *б*. Волнистой линией изображены ребра из оптимального цикла, сплошной – из цикла, построенного алгоритмом. Веса ребер, не изображенных на рисунке, равны 0.

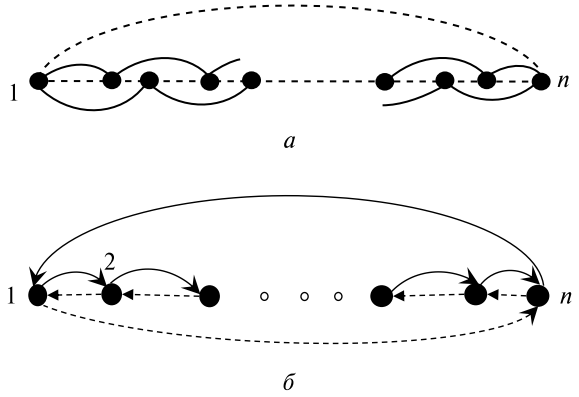


Рис. 3.1

Одноранговая ЗКМ. Для специальных случаев ЗКМ оценки можно уточнить. Рассмотрите для примера задачу на циклических подстановках, которая есть случай ЗКМ, когда веса ребер $f(i, j)$ удовлетворяют свойству $f(i, j) = a_i b_j$. Случай имеет место для одноранговой ЗКМ, когда ранг матрицы весов $\left[f(i, j) \right]$ равен 1. Даже в этом случае задача остается *NP*-трудной.

В несимметричной ЗКМ присоединение дуги e_i к частичному туру T_{i-1} может привести к появлению не более трех новых недопустимых дуг для T_{i-1} из оптимального гамильтонова орцикла. Такие дуги являются запрещаемыми на шаге *i*.

Лемма 3.1. Допустим, u_1, u_2 – запрещаемые на шаге *i* дуги, тогда имеет место неравенство $2f(e_i) \geq f(u_1) + f(u_2)$.

Доказательство вытекает из неравенств $f(e_i) \geq f(u_1)$, $f(e_i) \geq f(u_2)$, справедливых в силу правила выбора дуги e_i алгоритмом 3.1.

Лемма 3.2. Предположим u_1, u_2, u_3 – запрещаемые на шаге *i* дуги, тогда существует дуга e_i, допустимая для частичного тура T_i, причем выполняется неравенство

$$2f(e_i) + f(e_i'') \geq f(u_1) + f(u_2) + f(u_3). \qquad (3.1)$$

Три запрещаемые дуги на i-м шаге могут появиться только в четырех случаях (рис. 3.2, *а–г*). На рисунке показаны: запрещаемые дуги u_1, u_2, u_3 (жирная линия), e_i – дуги, выбираемые для решения и имеющие наибольший вес (сплошная линия).

Отметим, что дуги (v_1, v_3), (v_2, v_4) являются допустимыми для частичного тура T_i, иначе дуга (v_2, v_3) была бы запрещаемой на шаге i. В силу допустимости дуг u_1, u_2, u_3, для частичного тура T_{i-1} во всех случаях справедливы неравенства

$$f(e_i) \geq f(u_k),\ k = 1, 2, 3. \tag{3.2}$$

Доказательство неравенства (3.1) проведем только для случая, изображенного на рис. 3.2, *а*, в остальных случаях неравенство (3.1) доказывается аналогично. Необходимо только во всех формулах v_2' заменить на v_2 или v_3' на v_3.

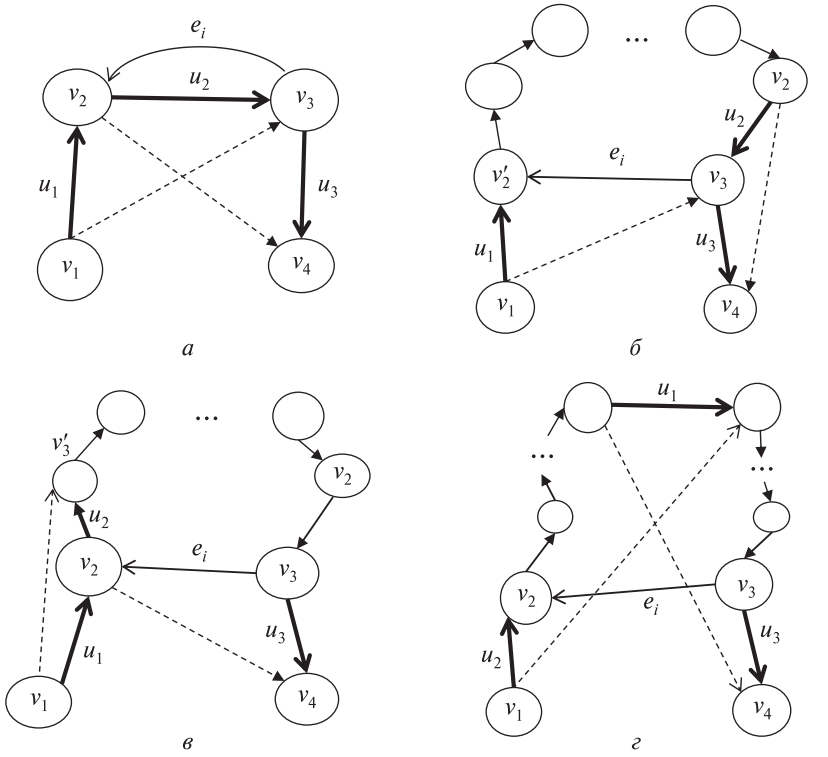


Рис. 3.2

В одноранговой ЗКМ вес каждой дуги (v_i, v_i) есть произведение весов a_i и b_j и вершин v_i и v_j. Веса вершины v_i' обозначим символами a_i' и b_i'.

Пусть $a_1 > a_2$. Тогда, предположив, что $e' = (v_1, v_3)$, имеем

$$f(e_i') = a_1 b_3 \geq a_2 b_3 = f(u_2).$$

Складывая полученное неравенство с неравенствами (3.2) при $k = 1$ и $k = 3$, получаем неравенство (3.1).

Пусть $a_2 \geq a_3'$. Тогда, положив $e' = (v_2, v_4)$, имеем

$$f(e_i') = a_2 b_4 \geq a_3' b_4 = f(u_3)$$

и, складывая с (3.2) при $k = 1$ и $k = 2$, получаем (3.1).

Пусть $b_4 \geq b_3$. Тогда, положив $e_i' = (v_2, v_4)$, имеем

$$f(e_i') = a_2 b_4 \geq a_2 b_3 = f(u_2)$$

и, складывая с (3.2) при $k = 1$ и $k = 3$, получаем (3.1).

Пусть $b_3 \geq b_2'$. Тогда, положив $e_i' = (v_1, v_3)$, имеем

$$f(e_i') = a_1 b_3 \geq a_1 b_2' = f(u_1)$$

и, складывая с (3.2) при $k = 1$ и $k = 3$, получаем (3.1).

Осталось рассмотреть случай, когда $a_1 < a_2 < a_3'$ и $b_2' > b_3 > b_4$. В качестве e_i' можно выбрать любую из дуг: (v_1, v_3) или (v_2, v_3) (например, (v_1, v_3)).

Тогда из неравенства

$$2f(e_i) + f(u_2) \geq a_2 b_2' + f(u_2) + f(u_3)$$

или

$$2a_3' b_2' + a_2 b_3 \geq a_2' b_2' - a_2 b_3 + a_3' b_4',$$

справедливого в силу $a_3' b_2' > a_2' b_2'$ $\left(a_3' > a_2', f(e_i') \geq f(u_3)\right)$, вычитаем неравенство $(a_1 < a_2) b_3 \leq (a_2 - a_1) b_2'$ и получаем неравенство

$$2a_3' b_2 + a_1 b_3 \geq a_1' b_2' + a_2 b_3 + a_3' b_4$$

или

$$2f(e_i) + f(e_i') \geq f(u_1) + f(u_2) + f(u_3).$$

Определим рекуррентно последовательность множеств дуг R_1, R_2, ..., R_n следующим образом. Если на шаге i запрещается не более двух дуг из R_i, то полагаем $R_{i+1} = R_i / \{u_1, u_2\}$, $R_1 = X^*$, где u_1, u_2 – запрещаемые дуги. Если запрещаются три дуги из R_i, например u_1, u_2, u_3, то полагаем $R_1 = X^*$, $R_{i+1} = R_i \cup e_i' \setminus \{u_1, u_2, u_3\}$, где e_i – такая дуга, для которой вы-

полняется (3.1). Нетрудно также заметить, что R_i – частичный тур для любого i.

Лемма 3.3. Допустим, на каждом шаге i алгоритма 3.1 справедливо неравенство

$$2f(T_i)+f(R_{i+1})\geq f^*. \tag{3.3}$$

Доказательство. Нетрудно заметить, что для $i=0$ неравенство (3.3) выполняется, так как $f(T_0)=0,\ f(R_1)=f(X^*)$. Пусть неравенство (3.3) верно для шага i. Докажем его справедливость для шага i.

Если на шаге i запрещаемыми оказались не более двух дуг из R_i, например u_1, u_2, то в силу индукционного предположения, способа определения R_i и леммы 3.1 имеем

$$2f(T_i)+f(R_{i+1})=2f(T_{i-1})+2f(e_i)+f(R_i)-f(u_1)-f(u_2)\geq$$

$$\geq 2f(T_{i-1})+f(R_i)\geq f^*.$$

Если же на шаге i запрещаемыми являются дуги u_1, u_2, u_3 из R_i, то в силу индукционного предположения, способа определения R_i и леммы 3.2 имеем

$$2f(T_i)+f(R_{i+1})=2f(T_{i-1})+2f(e_i)+f(R_i)+f(e_i')-f(u_1)-$$

$$-f(u_1)-f(u_2)-f(u_3)\geq 2f(T_{i-1})+f(R_i)\geq f^*.$$

Теорема 3.2. Достижимая оценка погрешности алгоритма 3.1 для одноранговой несимметричной ЗКМ равна 1/2.

Доказательство вытекает из леммы 3.3 при $i=n$ и с учетом того, что $X^g=T_n,\ f(R_{n+1})=0$. В достижимости оценки легко убедиться на примере, изображенном на рис. 3.3.

Градиентные алгоритмы с форой решения задачи коммивояжера. Данный подход предполагает, что на первом этапе с помощью эвристических правил строится некоторый начальный частичный тур (фора); а на втором этапе этот тур достраивается алгоритмом 3.2 до гамильтонова цикла. В качестве начальных частичных туров могут использоваться решения задачи о паросочетании максимального веса и частичные туры, получаемые при разрыве циклов, которые соответствуют решениям задач о назначении и паросочетании максимального веса.

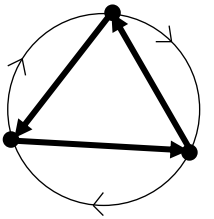


Рис. 3.3

Паросочетанием в графе G называется множество ребер, попарно несмежных между собой. Для нахож-

дения паросочетания максимального веса в графе G существуют эффективные алгоритмы трудоемкости $O(n^3)$. Паросочетание представляет собой частичный тур и может быть использовано в качестве начального частичного тура для алгоритма 3.1.

Алгоритм 3.2.

1. Находим в графе G паросочетание M максимального веса.

2. С помощью алгоритма 3.1, положив $T = M$, достроим паросочетание M до гамильтонова цикла, который обозначим $X^{3.2}$. Алгоритм 3.2 имеет трудоемкость $O(n^3)$ операций.

Теорема 3.3. Неулучшаемая оценка погрешности алгоритма 3.2 для симметричной ЗКМ равна 3/8.

Доказательство. Пусть T_i – частичный тур, получаемый после шага i второго этапа алгоритма 3.2; e_i – ребро, выбираемое на шаге i. Обозначим через e_i множество ребер на X^*, которые не являются допустимыми для частичного тура T_{i-1}, считая, что $x_1 = X^* \cap M$. Покажем, что число запрещаемых ребер не превосходит пяти для всех $i = 1, 2, ..., s$, где s – число шагов второго этапа алгоритма 3.2. Недопустимыми ребрами из X^* для произвольного частичного тура могут быть только те, которые инцидентны внутренним вершинам частичного тура или замыкают в цикл некоторую цепь в частичном туре. При добавлении к частичному туру T_{i-1} ребра e_i число внутренних вершин в новом частичном туре T_i увеличивается не более чем на два, откуда следует, что число запрещенных ребер не больше пяти. Отметим, что любое ребро e_i смежно двум ребрам из M, и если $|x_{i+1}| - |x_i| = 5$, то ребро e_i смежно двум ребрам из $M \setminus X^*$. В справедливости последнего факта легко убедиться, заметив, что внутренним вершинам в T_i, полученным на шаге i второго этапа алгоритма 3.2, должно быть инцидентно четыре ребра из X^*.

Пусть y_1^i – суммарный вес тех ребер e_m при $1 \le m \le i$, добавление которых к частичному туру T_{m-1} приводит к появлению ровно пяти запрещаемых ребер; y_2^i – суммарный вес тех ребер e_m при $1 \le m \le i$, добавление которых к частичному туру T_{m-1} приводит к появлению не более четырех запрещаемых ребер. Очевидно, что

$$y_2^i = \sum_{m=1}^{i} f(e_m) - y_1^i.$$

Считаем, что $y_1^0 = y_2^0 = 0$. Рекуррентно вводим величины y_3^i, y_4^i, $i = 1, 2, ..., s$, полагая $y_3^0 = f(M)$, $y_4^0 = 0$, $y_3^i = y_3^{i-1}$, $y_4^i = y_4^{i-1}$, если $|x_{i+1}| - |x_i| \le 4$

и $y_3^i = y_3^{i-1} - f(e_i') - f(e_i'')$, $y_4^i = y_4^{i-1} + f(e_i') - f(e_i'')$, если $|x_{i+1}| - |x_i| = 5$, где e_i', e_i'' — ребра из паросочетания M, смежные ребру e_i.

По индукции, используя только определение величин y_k^i, $k = 1, 2, 3, 4$ и тот факт, что на каждом шаге i второго этапа алгоритма 3.2 выбирается ребро e_i, имеющее максимальный вес среди допустимых ребер, легко выводятся неравенства

$$5y_1^i + 4y_2^i \geq f(x_{i+1}) - f(M \cap X^*), \tag{3.4}$$

$$y_3^i \geq f(M \cap X^*). \tag{3.5}$$

В силу того что паросочетание M^* имеет максимальный вес, $f(e_i) \leq f(e_i') + f(e_i'')$ для таких i, что $|x_{i+1}| - |x_i| = 5$, откуда следует, что

$$y_4^i \geq y_1^i. \tag{3.6}$$

Кроме того, в силу определения величин y_3^i, y_4^i имеем $y_3^i + y_4^i = f(M)$. Используя очевидное неравенство $f(M) \geq f^*/2$, получаем

$$y_3^i + y_4^i \geq f^*/2. \tag{3.7}$$

Неравенства (3.4)–(3.7) справедливы на любом шаге i второго этапа алгоритма 3.2, в том числе и на последнем шаге s. Но $x_{s+1} = X^*$, поэтому, сложив неравенства (3.4) и (3.5) с утроенным неравенством (3.7), получим $5y_1^s + 4y_2^s + 4y_3^s + 3y_4^s \geq 5f^*/2$. Учитывая неравенство (3.6), имеем $4y_1^s + 4y_2^s + 4y_3^s + 4y_4^s \geq 5f^*/2$, но

$$f(X^{12.2}) = \sum_{i=1}^{s} f(e_i) + f(M) = y_1^s + y_2^s + y_3^s + y_4^s.$$

Из двух последних соотношений вытекает, что $\dfrac{f(X^{12.2})}{f^*} \geq \dfrac{5}{8}$ или $\dfrac{f^* - f(X^{12.2})}{f^*} \leq \dfrac{3}{8}$.

Покажем, что оценка 3/8 неулучшаема. Для этого рассмотрим матрицу весов, в которой $n = 6k$, $f(3i+1, 3i+2) = 2$, $f(3i+2, 3i+3) = f(3i+3, 3i+4) = 1$, $i = 0, \ldots, 2k-1$, $f(4, 6k-3) = \ldots = f(3k-2, 3k+3) = 1$, значения остальных элементов равны 0.

Тогда $X^* = \{(1,2), (2,3), \ldots, (6k-1, 6k), (6k, 1)\}$, $f^* = 8k$, а $M = \{(1,2), (4,5), (7,8), \ldots, (6k-1), (3, 6k), (6k-3,), \ldots, (3k, 3k+3)\}$, ребрами с нену-

левыми весами в $X^{3.2}$ являются ребра $\{(1,2), (4,5), \ldots, (6k-2), (6k-1), (1,6k), (4,6k-3), \ldots, (3k-2,3k+3), (3k,3k+1)\}$.

Таким образом, $f\left(X^{3.2}\right)=4k+k+1$. Следовательно, для ЗКМ с такой матрицей весов гарантированная оценка погрешности равна $3/8-3/4n$.

Подмножество S ребер графа G называется 2-сочетанием, при котором каждой вершине инцидентно точно два ребра из S. Трудоемкость алгоритма нахождения максимального по весу ребер 2-сочетания $O\left(n^4\right)$ требует операций.

Пусть S – 2-сочетание максимального веса в полном графе. Если S не есть гамильтонов цикл, то содержит циклы, состоящие не менее чем из трех ребер. Удалим из каждого такого цикла ребро минимального веса и достроим полученный частичный тур градиентным алгоритмом 3.1 до гамильтонова цикла. Поскольку вес 2-сочетания S не меньше, чем вес гамильтонова цикла максимального веса, и из S удаляется не более 1/3 ребер, то гарантированная оценка погрешности такого алгоритма равна 1/3. В несимметричной ЗКМ, решив задачу о назначении на максимум, можем получить орциклы из двух дуг, и поэтому при разрыве таких орциклов удаляется не более половины дуг. Следовательно, гарантированная оценка погрешности алгоритма 3.1, начинающего работу с решения задачи о назначении (после разрыва циклов) в качестве начального частичного тура, равна 1/2.

Циклы 2-сочетания S, состоящие из трех ребер, назовем *треугольниками*, остальные циклы в S – *колесами*. Ребро, смежное колесу и треугольнику, одновременно будем называть *шипом*.

Алгоритм 3.3. Строим в графе G 2-сочетание S максимального веса. Если в S нет треугольников, то, полагая $T_0=\varnothing$. Находим паросочетание M' максимального веса в графе G с модифицированными по следующему правилу весами ребер:

$$f'(i,j)=\begin{cases} f(i,j), \text{ если } (i,j) \text{ смежно двум треугольникам;} \\ f(i,j)/2, \text{ если } (i,j) - \text{ шип;} \\ f(i,j)+\max\{f(k,j), f(k,i)\}, \text{ если } i,j,k - \\ \text{вершины некоторого треугольника;} \\ 0 - \text{ в остальных случаях.} \end{cases}$$

Образуем частичный тур T_0, включая ребра с ненулевыми весами $f(i,j)$ из паросочетания M', а также по одному ребру из каждого треугольника, имеющего непустое пересечение с M', причем из двух воз-

можных в каждом треугольнике ребер выбираем ребро с максимальным весом $f(i, j)$.

Далее разрываем каждое из колес по следующему правилу: пусть колесо C имеет m ребер $e_1, e_2, ..., e_m$ и $Y = \{y_1, y_2, \ldots, y_m\}$ – его шипы, причем вес шипов, не принадлежащих T_0, полагаем условно равным 0 и считаем, что шип y_i смежен ребрам e_{i-1}, e_i. Если $2f(Y) \le f(C)$, то разрываем колесо C, выбрасывая из него такое ребро e_i и добавляя такие шипы y_i, y_{i+1}, что $f(y_i) + f(y_{i+1}) - f(e_i)$ имеет наибольшую величину. Если же $2f(Y) > f(C)$, то колесо C заменяем при нечетном m на наилучший из частичных туров

$$\{y_{i+1}, \ldots, y_{m+i-1}, e_i, e_{i+2}, \ldots, e_{m+i-1}\},\ i = 1, 2, \ldots, m, \tag{3.8}$$

а при четном m – на лучший из двух частичных туров

$$\{y_1, \ldots, y_m, e_1, e_3, \ldots, e_{m-1}\},\ \{y_1, \ldots, y_m, e_2, e_4, \ldots, e_m\}, \tag{3.9}$$

где $y_{m+i} = y_i$; $e_{m+i} = e_i$; $e_0 = e_m$.

Из ребер T_0 и разорванных колес образуем новый частичный тур T.

Алгоритмом 3.1 достраиваем частичный тур T до гамильтонова цикла, который обозначаем $X^{3.3}$.

Лемма 3.4. Допустим, колесо C состоит из m ребер и Y – некоторое множество его попарно несмежных шипов. Тогда существует частичный тур T', $T' \subseteq C \cup Y$, при котором справедливо неравенство $f(T') \ge f(Y)/2 + 3f(C)/4$.

Доказательство. Рассмотрим два случая: $f(Y) = f(C)/2$ и $f(Y) > f(C)/2$. В первом случае средний вес частичных туров $T_i = C/e_i \cup \cup \{y_1, y_{i+1}\}$, $i = 1, 2, ..., m$ равен

$$\frac{1}{m}\sum_{i=1}^{m} f(T_i) = \frac{2}{m} f(Y) + \frac{m-1}{m} f(z).$$

Следовательно, для лучшего T' из туров T_i верны неравенства

$$f(T') \ge \frac{2}{m} f(Y) + \frac{m-1}{m} f(C) = \frac{2}{m} f(Y) + \frac{3}{4} f(C) + \left(\frac{m-1}{m} - \frac{3}{4}\right) f(C) \ge$$

$$\ge \frac{2}{m} f(Y) + \frac{3}{4} f(C) + \frac{m-4}{2m} f(Y) = \frac{1}{2} f(Y) + \frac{3}{4} f(C).$$

Во втором случае при четном m средний вес частичных туров (3.9) равен $f(Y) + f(Y)/2$. поэтому существует частичный тур T', для которого справедливо

$$f(T') \ge \frac{1}{2} f(C) + f(Y) = \frac{1}{2} f(C) + \frac{1}{2} f(Y) + \frac{1}{2} f(Y) \ge \frac{3}{4} f(C) + \frac{1}{2} f(Y).$$

При нечетном m средний вес туров (3.8) равен

$$\frac{m-1}{m} f(Y) + \frac{m+1}{2m} f(C).$$

Следовательно, и в этом случае существует частичный тур T с весом

$$f(T) \geq \frac{m-1}{m} f(Y) + \frac{m+1}{2m} f(C) =$$

$$= \frac{1}{2} f(Y) + \left(\frac{m-1}{m} - \frac{1}{2} \right) f(Y) + \frac{m+1}{2m} f(C) \geq$$

$$\geq \frac{1}{2} f(Y) + \frac{m-2}{4m} f(C) + \frac{m+1}{2m} f(C) = \frac{1}{2} f(Y) + \frac{2}{4} f(C).$$

Из способа построения частичных туров T_0 и T, алгоритма 3.3 вытекает справедливость следующих утверждений.

Лемма 3.5. $f(T) \geq (3/4) f(K) + f'(M')$, где K – множество ребер всех колес; $f'(M')$ – вес паросочетания M' в графе G с модифицированными весами $f'(i, j)$.

Доказательство. Пусть H – множество шипов. Из способа построения частичного тура и леммы 3.4 имеем

$$f\left(T \cap (H \cup K) \geq \frac{1}{2} f(T \cap H) + \frac{3}{4} f(K) = \frac{1}{2} f(T_0 \cap H) + \frac{3}{4} f(K) \right).$$

Следовательно, с учетом того, что $f'(M') = f(T_0 \setminus H) + f(T_0 \cap H)/2$, имеем требуемое неравенство.

Пусть $\bar{M}$ – паросочетание максимального веса в графе G с весами $f^{-1}(i, j)$, вычисленными так, как и в п. 2 алгоритма 3.1, исходя из матрицы $\{f(i, j)\}$, обладающей свойствами

$$\bar{f}(i, j) \leq f(i, j) \ \forall (i, j) \in \Delta, \ \bar{f}(i, j) = f(i, j) \ \forall (i, j) \in \Delta, \qquad (3.10)$$

где Δ – множество ребер треугольников.

Очевидно

$$f(K) = \bar{f}(K), \ f'(M') \geq \bar{f}'(\bar{M}). \qquad (3.11)$$

Каждому треугольнику может быть смежно 2, 4 или 6 ребер из $X^* \setminus S$. В зависимости от числа смежных ребер множество ребер всех треугольников разобьем на три класса: Δ_2, Δ_4, Δ_6. В класс Δ_i отнесем ребра из тех треугольников, которым смежно i ребер из X^*/S. Через X_{Δ_i}, $X_{\Delta_i} \subset X^* \setminus S$ обозначим множество ребер, каждое из которых смежно хотя бы одному ребру из Δ_i, $i = 2, 4, 6$ и $X_{\Delta_i \cup \Delta_j} = X_{\Delta_{i,j}}$.

Лемма 3.6. Существует матрица весов $\left[f_{i,j} \right]$, обладающая свойствами (3.10) и

$$\bar{f}\left(X_{\Delta_2}\right) \geq \bar{f}\left(X_{\Delta_2} \setminus X^*\right); \tag{3.12}$$

$$\bar{f}\left(X_{\Delta_{2,4}}\right) \geq \bar{f}\left(\Delta_2 \cup \Delta_4 \setminus X^*\right); \tag{3.13}$$

$$\bar{f}\left(X_{\Delta}\right) \geq \bar{f}\left(\Delta \setminus X^*\right); \tag{3.14}$$

$$\bar{f}(S) \geq f\left(X^*\right) = f^*. \tag{3.15}$$

Доказательство. Пусть для матрицы весов не выполняется неравенство (3.12), а верно $\bar{f}\left(X_{\Delta_2}\right) < \bar{f}\left(X_{\Delta_2} \setminus X^*\right)$. Из того, что S является 2-сочетанием максимального веса, следует, что $\bar{f}\left(X^* \setminus \left(X_{\Delta_2} \cup \Delta_2\right)\right) \leq \bar{f}\left(S \setminus \Delta_2\right)$. Складывая эти два неравенства и прибавляя к обеим частям $\bar{f}\left(\Delta_2 \cap X^*\right)$, получаем $\bar{f}\left(X^*\right) < \bar{f}(S)$, поэтому уменьшаем веса ребер из $\Delta_2 \setminus X^*$ так, чтобы для новых весов выполнялось (3.12) при сохранении (3.15).

Пусть теперь не выполняется неравенство (3.13), т. е. $\bar{f}\left(X_{\Delta_{2,4}}\right) \geq$ $\geq \bar{f}\left(\Delta_2 \cup \Delta_4 \setminus X^*\right)$. Складываем это неравенство с неравенством $\bar{f}\left(X^* \setminus \right.$ $\left. \setminus \left(\Delta_2 \cup \Delta_4 \cup X_{\Delta_{2,4}}\right)\right) < \bar{f}\left(S \setminus \left(\Delta_2 \cup \Delta_4\right)\right)$, справедливым в силу оптимальности S, и прибавляем к обеим частям $\bar{f}\left(\left(\Delta_2 \cup \Delta_4\right) \cap X^*\right)$, получаем $\bar{f}\left(X^*\right) < f(S)$. По этой причине можно уменьшить веса ребер из $\left(\Delta_2 \cup \Delta_4\right) \setminus X^*$ так, чтобы для новых весов выполнялись неравенства (3.13) и (3.15). Аналогично, уменьшая веса ребер из $\Delta \setminus X^*$, добиваемся выполнения неравенства (3.14) без нарушения неравенства (3.15).

Множество ребер $H \subset X_{\Delta}$ называется *накрывающим,* если выполняются условия:

1) в H нет двух ребер, инцидентных одной вершине треугольника;

2) в H нет трех ребер, инцидентных вершинам одного треугольника;

3) любой цикл, составленный из ребер $H \cup \Delta$, содержит четное число ребер из H.

Пусть s – общее число треугольников и e_i^1, e_i^2, e_i^3 – ребра треугольника i, $i = 1, 2, ..., s$, со свойством $f\left(e_i^1\right) \geq f\left(e_i^2\right) \geq f\left(e_i^3\right)$.

Лемма 3.7. Существует накрывающее множество H, обладающее свойством

$$\bar{f}'(H) \geq \frac{1}{3} \sum_{i=1}^{s} \bar{f}\left(e_i^3\right).$$

Лемма 3.8. $\bar{f}'(M') \geq 13\bar{f}(\Delta)/18$.

Доказательство. Пусть H – некоторое накрывающее множество. Разобьем его на два множества H_1 и H_2 по следующему правилу: в $H_1(H_2)$ входят ребра из H, попарно несмежные и несмежные одному треугольнику. Это возможно в силу определения накрывающих множеств.

Рассмотрим два паросочетания: в $M_1(M_2)$ включены ребра $H_1(H_2)$ и по одному ребру из каждого треугольника, причем это ребро несмежно с ребром из $H_1(H_2)$; ребра имеют максимальный вес в G с модифицированными весами $f(i,j)$. Тогда существует паросочетание $\bar{M}$, вес которого не меньше, чем средний вес паросочетаний $M_1(M_2)$, поэтому

$$\bar{f}'(\bar{M}) > \frac{1}{2}\left[\bar{f}(M_1) + \bar{f}(M_2)\right] \geq \frac{1}{2}f'(N) + \frac{1}{2}\sum_{i=1}^{s}\left[\bar{f}'(e_i^1) + \bar{f}'(e_i^2)\right].$$

Согласно определению весов $\bar{f}'(i,j)$ имеем

$$\bar{f}'(e_i^1) + \bar{f}'(e_i^2) \geq \bar{f}(e_i^1) + \bar{f}(e_i^2) + \bar{f}(e_i^1) + \bar{f}(e_i^3) = 2\bar{f}(e_i^1) + \bar{f}(e_i^2) + \bar{f}(e_i^3).$$

Таким образом, с учетом леммы 3.7

$$\bar{f}'(M') \geq \frac{1}{6}\sum_{i=1}^{s}\bar{f}(e_i^3) + \frac{1}{2}\sum_{i=1}^{s}\left[2\bar{f}(e_i^1) + \bar{f}(e_i^2) + \bar{f}(e_i^3)\right] =$$

$$= \sum_{i=1}^{s}\frac{13}{18}\bar{f}(e_i^1) + \sum_{i=1}^{s}\left[\frac{1}{2}\bar{f}(e_i^2) + \frac{4}{18}\bar{f}(e_i^1)\right] + \sum_{i=1}^{s}\left[\frac{2}{3}\bar{f}(e_i^3) + \frac{1}{18}\bar{f}(e_i^1)\right] \geq$$

$$\geq \frac{13}{18}\sum_{i=1}^{s}\left[\bar{f}(e_i^1) + \bar{f}(e_i^2) + \bar{f}(e_i^3)\right] = \frac{13}{18}\bar{f}(\Delta).$$

Теорема 3.4. Гарантированная оценка погрешности алгоритма 3.3 для симметричной ЗКМ равна 13/18.

Доказательство. Из лемм 3.5, 3.6, 3.8 и соотношения (3.11) имеем следующую цепочку неравенств:

$$f(X^{12.3}) \geq \frac{3}{4}f(K) + f'(M') \geq \frac{3}{4}\bar{f}(K) + \bar{f}'(M') \geq$$

$$\geq \frac{3}{4}\bar{f}(K) + \frac{13}{18}\bar{f}(\Delta) \geq \frac{13}{18}\bar{f}(K \cup \Delta) = \frac{13}{18}\bar{f}(S) \geq \frac{13}{18}f^*.$$

Глава 4 ВЕРСИИ ЗАДАЧ С НЕПОЛНОЙ ИНФОРМАЦИЕЙ (ОНЛАЙН- И СЕМИОНЛАЙН-ЗАДАЧИ)

В последнее время большой интерес вызывают версии задач с неполной информацией, так называемые онлайн- и семионлайн-задачи. Рассмотрим различные версии для классической задачи теории расписаний.

Имеется m идентичных процессоров, когда требуется распределить работы таким образом, чтобы время завершения последней выполненной было минимально.

В онлайн-версии нет никакой информации о количестве работ, их продолжительности выполнения на процессоре. Работы идут одна за другой, и прежде чем поступит информация о длительности следующей работы или об отсутствии работ, текущая работа должна быть назначена на конкретный процессор, причем это назначение окончательное, т. е. не может быть изменено в дальнейшем.

В целях оценки качества работы алгоритмов для версий задач с неполной информацией обычно используется сравнительный анализ. Для этого сопоставляются значения решений, получаемых алгоритмом для онлайн-версии, с оптимальными значениями соответствующей офлайн-версии (когда все данные заранее известны) и устанавливается гарантированная оценка алгоритма.

При анализе алгоритмов для задач с неполной информацией особую роль играют так называемые нижние границы (НГ) для гарантированных оценок онлайн- и семионлайн-алгоритмов. Смысл НГ для версий задач с неполной информацией состоит в том, что невозможно построить алгоритм, гарантированная оценка которого лучше, чем НГ.

В области онлайн-алгоритмов для задач теории расписания одним из первых результатов является алгоритм «в минимально загруженный», для которого было доказано, что его гарантированная оценка равна $2-1/m$, где m – количество процессоров. Известно также, что в случае $m=2$ этот алгоритм является асимптотически оптимальным, так как НГ точности алгоритмов для этой задачи равна 3/2.

Утверждение. Для онлайн-версии задачи не существует алгоритма, гарантированная оценка которого меньше 3/2.

Доказательство. Предположим, что такой алгоритм существует. Подадим на вход последовательно две работы длительностью 1. Если ал-

горитм назначит их на один процессор, то на следующем шаге дадим информацию, что работы закончились. Значение оптимального решения равно 1, а алгоритм построил решение со значением 2. Гарантированная оценка равна 2, поэтому такое назначение невозможно. Если же алгоритм назначит две поступившие работы на разные процессоры, то на вход подадим последнюю работу с длительностью 2. Значение оптимального решения равно 2, а построенного – 3, поэтому гарантированная оценка не лучше 3/2. Получим противоречие.

Семионлайн-версия задач предполагает наличие дополнительной информации общего характера. Для рассматриваемой задачи известно несколько семионлайн-версий:

- поступление работ в отсортированном порядке;
- оптимальное значение решения;
- общая сумма длительностей выполнения;
- наличие буфера.

Для построения и анализа алгоритма используется несколько ***подходов***.

Динамический пересчет нижней оценки оптимального решения. Известно, что алгоритм «в минимально загруженный» имеет гарантированную оценку, равную $4/3 - 1/(3m)$.

Алгоритм 4.1. Назначим первые m работ на разные процессоры, вычислим $LB = \max\{a_1, a_m + a_{m+1}\}$. Далее определим очередную работу на максимально загруженный процессор, но с условием, что сумма не превосходила $5LB/4$.

Алгоритм 4.1 имеет гарантированную оценку, равную 5/4.

Построение инвариантов. Основная идея подхода состоит в следующем:

- на множестве объектов вводится некоторая классификация;
- определяются инвариантные группы, задаваемые параметрами, связанными с гарантированной оценкой алгоритма;
- определяются множества правил распределения объектов по группам, обеспечивающим поддержание инварианта.

Фиксированное количество групп, согласованность классификации и структура инвариантов позволяют, с одной стороны, обеспечить эффективность алгоритмов (как по времени, так и по памяти), а с другой – упрощают доказательство гарантированной точности алгоритма, так как она уже изначально заложена в инвариантах.

В классической задаче минимизации времени завершения проекта на многопроцессорной системе рассматривается система из m машин

с соответствующими им скоростями $s_1, s_2, \supset, s_m$ и n работ со временем обработки $p_1, p_2, \supset, p_N$. Требуется распределить все n работ таким образом, чтобы время их обработки было минимальным. В онлайн-версии данной задачи все работы поступают последовательно, и каждую из них необходимо назначить на одну из машин сразу после поступления, не имея никакой информации о последующих работах. Мы исследуем случай онлайн-версии задачи, когда $m-k$ процессоров имеют единичную скорость, а у k процессоров, где k – некоторая константа, скорости удовлетворяют соотношениям $1 \le sm-k+1 = \ldots = sm = s \le 2$.

Предлагается параметрический алгоритм, который разбивает работы на классы, процессоры – на группы и вводит такой приоритет назначения очередной работы на процессор, когда гарантируется постоянное наличие процессора, на который можно назначит очередную работу при заданной точности решения задачи. Отличительной особенностью данного алгоритма является тот факт, что предлагаемая схема позволяет построить решение с гарантированной оценкой $2+\alpha$ для любого $\alpha > 0$ при достаточно больших m.

Обозначим через J_j длительность J-го задания в списке σ и будем считать, что задание J_j прибывает на шаге j согласно σ. Обозначим через p_j время обработки задания J_j. Если работа со временем p_j назначается на машину со скоростью s_i, то p_j / s_i – время, необходимое для выполнения данной работы.

Качество онлайн-алгоритма Alg определяется как такое наименьшее число c, при котором для каждого списка заданий σ мы имеем $F(\mathrm{Alg},\sigma) \le c \ \ \mathrm{Opt}(\sigma)$, где $F(\mathrm{Alg}, \sigma)$ – время обработки расписания, которое получается как результат применения алгоритма Alg к списку σ; $\mathrm{Opt}(\sigma)$ – время обработки некоторого оптимального расписания работ σ, вычисляемое при рассмотрении σ как набора заданий. В этом случае говорят, что гарантированная оценка алгоритма Alg равна c.

Рассмотрим параметрическую схему для онлайн-версии задачи теории расписаний с униформ-процессорами. Прежде чем представить основные результаты, введем некоторые обозначения: m – общее количество машин; k – число машин со скоростью $1 < s \le 2$, где k – константа.

Далее опишем стратегию (алгоритм), которая позволит нам назначать (для любого индекса $j = 1, 2, \ldots, \mathrm{Length}(s)$) работу J_j со временем обработки p_j, прибывающей на шаге j согласно списку заказа s, на некоторые машины M_i, $i = 1, 2, \ldots, m$. Мы должны построить расписание таким образом, чтобы J_j были назначены сразу после завершения последнего задания, которое было получено. По сути, нет приоритета

по отношению заданий, поэтому если они назначены на одной машине, то будут выполняться последовательно. Каждый раз, имея дело с текущим заданием J_j входного списка σ, мы обозначаем:

$L_{i,j}$ – текущая загрузка машины i до назначения работы J_j;

$L^*_{i,j}$ – текущая загрузка машины i после назначения работы J_j;

V_j – теоретическое оптимальное решение для офлайн-версии задачи для множества работ $J(j)=\{J_1, J_2, \ldots, J_j\}$.

Легко проверить: если мы обозначим через $q_1, q_2, \ldots, q_j$ время обработки заданий из J_j, отсортированных по невозрастанию, т. е. $q_1 \geq \geq q_2 \geq \cdots \geq q_j$, то будет выполнено следующее утверждение.

Лемма 4.1. Справедливы следующие неравенства:

$$V_j \geq (q_1 + q_2 + \cdots + q_j)(m - k + sk),$$

$$V_j \geq \underset{s}{q_1}, V_j \geq \min\left\{\frac{(q_k + q_{k+1})}{s}, q_{k+1}\right\}.$$

Доказательство очевидно. Важно заметить, что последнее неравенство $V_j \geq \min\left\{\frac{(q_k + q_{k+1})}{s}, q_{k+1}\right\}$ вытекает из предположения $1 \leq s \leq 2$. Следовательно, для каждого шага j имеем

$$LB_j = \mathrm{Sup}\left\{\frac{(q_1 + q_2 + \ldots + q_j)}{(m - k + sk)},\ q_{1/s},\ \min\left\{\frac{(q_k + q_{k+1})}{s},\ q_{k+1}\right\}\right\}. \quad (4.1)$$

Ясно, что величина LB_j является нижней оценкой для оптимального значения офлайн-версии задачи, соответствующей шагу j, причем $LB_{j-1} \leq LB_j$, т. е. LB_j монотонна.

Опишем процесс назначения. Предположим, что некоторое положительное число α дается вместе с тремя целыми числами R, m_1 и m_2, для которых справедливы соотношения

$$(1+\alpha)sk + \left(1 + \frac{\alpha}{2}\right)m_1 \geq sk + m_1 + m_2, \quad (4.2)$$

$$k + m_1 + m_2 = m, \quad (4.3)$$

$$m_2 = Rk, \quad (4.4)$$

$$R \geq \log 1 + \frac{\alpha}{2}\left(\left(1 + \frac{\alpha}{2}\right)\Big/\left(2 + \alpha - s\right)\right). \quad (4.5)$$

В случае если мы зафиксируем k, s и α, а также придадим R и m_1 минимальные возможные значения, то R, m_1 и m_2 полностью определяются значениями k, s и α. Это наблюдение показывает, как m машин может быть разбито на три класса:

- машины со скоростью s называются быстрыми (*Fast*);
- выбираем m_1 машин из $m-k$ машин со скоростью 1 и называем их нормальными (*Normal*);
- остальные m_2 машин со скоростью 1 называем зарезервированными (*Reserved*), их количество равно $m_2 = Rk$.

Зарезервированные машины делятся на R групп G_0, G_1, …, G_{R-1}, где каждая из них содержит ровно k машин.

Аналогичным образом работа J_j, поступающая на шаге j, является:

- *Small* – время выполнения p_j не превосходит величины $\left(1+\frac{\alpha}{2}\right)LB_j$;
- *Large* – в противном случае.

Наконец, мы будем говорить, что работа J_j подходит для машины M_i, $i = 1, 2, ..., m$, если $L_{i,j} + \frac{p_j}{s_i} \le (2+\alpha)LB_j$.

Лемма 4.2. Если работа J_j из класса *Large* не подходит для машины i из класса *Fast*, то справедливо, что $L_{i,j} > (1+\alpha)LB_j$.

Доказательство. Требуемое неравенство следует непосредственно из того факта, что $\frac{p_j}{s_i} = \frac{p_j}{s} \le LB_j$.

Приведенные выше факты позволяют описать онлайн-алгоритм, который будет работать корректно на любом входном наборе для задачи теории расписаний с униформ-процессорами, базируясь на значениях m, k, s в соответствии с соотношениями (4.2)–(4.5).

Основная идея заключается в том, что на каждом шаге j мы будем иметь возможность назначать работы J_j на некоторые машины i таким образом, что будет выполняться неравенство $L^*_{i,j} \le (2+\alpha)LB_j$, причем оно очевидно в случае, когда J_j – это работы из класса *Small*. Таким образом, если J_j – это большая работа (из класса *Large*), то всегда существует машина, для которой эта работа подходит. С этой целью специально подберем параметры, при которых гарантируется, что если работа J_j не подходит для любых машины из классов *Fast* и *Normal*, то она может подойти для некоторых машин из класса *Reserved*.

Алгоритм 4.2 (Assign). Инициализация: полагаем $n = 0$ (n соответствует индексу активной машины из класса *Reserved* в группе G_0; машины в каждой группе G_r пронумерованы от 0 до $k - 1$); полагаем $j = 0$; $LB_j = 0$.

Прочитаем список σ.

Пока список σ не пуст, повторяем следующие шаги: $j = j + 1$.

Берем очередную работу J_j и выполняем шаг j следующим образом:

1) пересчитываем значение LB_j в соответствии с формулой (1.1);

2) если работа J_j подходит для некоторой машины i из классов *Fast* или *Normal*, то назначаем работу j на эту машину;

3) иначе:

• если $n < k$, то назначаем работу J_j на машину с номером n в группе G_0. Пусть i_0 будет номер машины из класса *Normal* с минимальной текущей суммарной загрузкой. Проведем обмен машин n и i_0 между классами *Normal* таким образом, что машина i_0 попадает в группу G_0 с номером n, а машина из группы G_0 – в класс *Normal*. Допустим, $n = n + 1$;

• если $n = k$, то перенумеруем группы $G_0, \ldots, G_{R-1}$ таким образом, что группа с номером r $(1 \le r \le R - 1)$ получает новый номер $r - 1$, а группа с номером 0 – номер $R - 1$. Допустим, $n = 1$.

Предложенный алгоритм работает на входных данных $(M_1, M_2, \ldots, M_m; s_1, s_2, \ldots, s_m)$, для которых $s_i = s \in [1, 2]$, где $i = 1, \ldots, k$; $s_i = 1$ для $i = k + 1, \ldots, m$; m можно представить в виде $m = k + m_1 + m_2 = k + m_1 + {} + Rk$, где числа m_1, m_2 и R удовлетворяют соотношениям (1.2)–(2.5).

Докажем: если k, α фиксированы, а величина m достаточно велика, то гарантированная оценка предлагаемого алгоритма не превосходит величины $2 + \alpha$. Более точно докажем следующее: если последовательность работ σ является входными данными для алгоритма *Assign*, то значение решения $F(Assign, \sigma)$ в построенном расписании не превосходит $(2 + \alpha) LB(\sigma)$, где $LB(\sigma)$ соответствует нижней границе оптимального значения $\mathrm{Opt}(\sigma)$.

Лемма 4.3. На любом шаге j во время выполнения алгоритма *Assign* существует или такая машина i из класса *Fast*, что $L_{i,j} \le (1 + \alpha) LB_j$, или такая машина i из класса *Normal*, что $L_{i,j} \le \left(1 + \frac{\alpha}{2}\right) LB_j$.

Доказательство. Предположим противное. Пусть существует шаг j такой, что для любой машины i из класса *Fast* $L_{i,j} > (1 + \alpha) LB_j$ и для любой машины i из класса *Normal* $L_{i,j} > \left(1 + \frac{\alpha}{2}\right) LB_j$.

Следовательно,

$$\left(p_1 + p_2 + \ldots + p_j\right) = s\left(\sum i \in FastL_{i,j}\right) + \left(\sum i \in Normal \cup ReservedL_{i,j}\right) \gg$$

$$\gg ks\left(1+\alpha\right)LB_j + m_1\left(1+\frac{\alpha}{2}\right)LB_j.$$

Из леммы 4.1 следует, что $\left(p_1 + p_2 + \ldots + p_j\right) \le \left(sk + m_1 + m_2\right)LB_j$, так как условие (1.2) гарантирует, что

$$ks\left(1+\alpha\right)LB_j + m_1\left(1+\frac{\alpha}{2}\right)LB_j = sk + m_1 + m_2.$$

Получили противоречие.

Лемма 4.4. Если текущая работа J_j из класса *Small*, то существует машина в одном из классов *Fast* или *Normal*, для которой эта работа подходит.

Доказательство. Применим лемму 4.2 и рассмотрим машину i как в утверждении Леммы 4.2. Если i из класса *Fast*, то $L_{i,j} \le \left(1+\alpha\right)LB_j$.

Учитывая тот факт, что $\frac{p_j}{s_i} = \frac{p_j}{s} \le LB_j$, получаем $L_{i,j} + \frac{p_j}{s_i} \le \left(2+\alpha\right)LB_j$ как результат.

Если i из класса *Normal*, то $L_{i,j} \le \left(1+\frac{\alpha}{2}\right)LB_j$ и $L_{i,j} + \frac{p_j}{s_i} = L_{i,j} + p_j \le$ $\le \left(2+\alpha\right)LB_j$.

Лемма 4.5. Для $q = 1, \ldots, Q$, выполняется $L_{i(q),j(q)} \le \left(2+\alpha-s\right)LB_{j(q)}$.

Теорема 4.1. Пусть задана величина α и для m, k, s выполняются соотношения (1.2)–(2.5). Тогда для любой входной последовательности σ алгоритм *Assign* строит расписание, для которого выполнено $F\left(Assign, \sigma\right) \le \left(2+\alpha\right)\mathrm{Opt}\left(\sigma\right)$.

Теорема 4.2. Пусть даны скорости s $\left(1 < s \le 2\right)$ машин и количество k машин со скоростью s. Тогда для любого $\alpha > 0$ существует m_0, при котором для каждого $m \ge m_0$ алгоритм *Assign* можно применить таким образом, что $F\left(Assign, \sigma\right) \le \left(2+\alpha\right)\mathrm{Opt}\left(\sigma\right)$.

СПИСОК ЛИТЕРАТУРЫ

Алгоритмы: построение и анализ / Т. Кормен [и др.]. – М. : Вильямс, 2005. – 1296 с.

Волчкова, Г. П. Сборник задач по теории алгоритмов. Организация перебора вариантов и приближенные алгоритмы / Г. П. Волчкова, В. М. Котов, Е. П. Соболевская. – Минск : БГУ, 2008, – 59 с.

Ковалев, М. М. Матроиды в дискретной оптимизации / М. М. Ковалев. – 2-е изд., стер. – М. : Эдиториал УРСС, 2003. – 224 с.

Котов, В. М. Алгоритмы для задач разбиения и упаковки / В. М. Котов. – Минск : БГУ, 2001. – 97 с.

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).

Пападимитриу, Х. Комбинаторная оптимизация. Алгоритмы и сложность / Х. Пападимитриу, К. Стайглиц. – М. : Мир, 1985. – 512 с.

Рейнтгольд, Э. Комбинаторные алгоритмы. Теория и практика / Э. Рейнтгольд, Ю. Нивергельт, Н. Део. – М. : Мир, 1980. – 466 с.

Котов, В. М. Опыт использования образовательной платформы Insight Runner на факультете прикладной математики и информатики Белорусского государственного университета / В. М. Котов, Е. П. Соболевская, С. А. Соболь // Роль университетского образования и науки в современном обществе : материалы междунар. науч. конф., Минск, 26–27 февр. 2019 г. / Белорус. гос. ун-т ; редкол.: А. Д. Король (пред.) [и др.]. – Минск : БГУ, 2019. – С. 263–267.

Сборник задач по теории алгоритмов : учеб.-метод. пособие / В. М. Котов [и др.]. – Минск : БГУ, 2017. – 183 с.

Сборник задач по теории алгоритмов. Структуры данных / С. А. Соболь [и др.]. – Минск : БГУ, 2020. – 159 с.

Танаев, В. С. Теория расписаний. Многостадийные системы / В. С. Танаев, Ю. Н. Сотсков, В. А. Струсевич. – М. : Наука, 1989. – 912 с.

Танаев, В. С. Теория расписаний. Групповые технологии / В. С. Танаев, М. Я. Ковалев, Я. М. Шафранский. – Минск : Ин-т техн. кибернетики НАН Беларуси, 1998. – 289 с.

Теория алгоритмов : учеб. пособие / П. А. Иржавский [и др.]. – Минск : БГУ, 2013. – 159 с.

Approximation scheduling algorithms: a survey / M. Y. Kovalyov [et al.] // Optimization. – 1989. – No. 6. – P. 859–878.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

Глава 1. ОРГАНИЗАЦИЯ ПОЛНОГО ПЕРЕБОРА 4
1.1. Построение дерева решений 4
1.2. Способы обхода дерева решений 6
1.3. Сокращение числа подзадач, необходимых для решения: отсев возможных вариантов ветвления 10
1.4. Способы построения нижних и верхних оценок 23
1.5. Минимальные связующие деревья 32
Задачи для самостоятельного решения 35
Указания к решению задач 65

Глава 2. ЭВРИСТИЧЕСКИЕ И ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ 68
2.1. Основные понятия 68
2.2. Жадные (градиентные) алгоритмы 71
2.2.1. Жадный алгоритм для задачи о коммивояжере 71
2.2.2. Жадный алгоритм для задачи о рюкзаке 72
2.2.3. Жадный алгоритм для задачи о суммах элементов подмножеств 73
2.2.4. Жадный алгоритм для задачи о раскраске графа 75
2.3. Локальный поиск 76
2.3.1. Описание эвристики 76
2.3.2. Сложность локального поиска 80
2.4. Приближенные алгоритмы с гарантированной оценкой точности 81
2.4.1. Задача об упаковке в контейнеры 85
2.4.2. Задача распределения работ на конечное число одинаковых процессоров 92
2.4.3. Задача коммивояжера 93
2.5. Методики построения приближенных алгоритмов с гарантированной оценкой точности 93
2.6. Оптимальность градиентного алгоритма 96
Задачи для самостоятельного решения 105

Глава 3. АЛГОРИТМЫ ДЛЯ ЗАДАЧИ КОММИВОЯЖЕРА 131

Глава 4. ВЕРСИИ ЗАДАЧ С НЕПОЛНОЙ ИНФОРМАЦИЕЙ (ОНЛАЙН- И СЕМИОНЛАЙН-ЗАДАЧИ) 143

СПИСОК ЛИТЕРАТУРЫ 150

Учебное издание

Котов Владимир Михайлович
Соболевская Елена Павловна
Волчкова Галина Петровна

**ТЕОРИЯ АЛГОРИТМОВ.
ОРГАНИЗАЦИЯ ПЕРЕБОРА
И ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ**

Учебно-методическое пособие

Ответственный за выпуск *Т. С. Петроченко*
Художник обложки *М. А. Сарасек*
Технический редактор *В. П. Явуз*
Компьютерная верстка *С. Н. Егоровой*
Корректор *И. В. Сазонова*

Подписано в печать 30.12.2022. Формат 60×84/16.
Бумага офсетная. Печать цифровая. Усл. печ. л. 8,83.
Уч.-изд. л. 10,10. Тираж 95 экз. Заказ 535.

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости, 4, 220030, Минск.

Республиканское унитарное предприятие
«Издательский центр Белорусского государственного университета».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 2/63 от 19.03.2014.
Ул. Красноармейская, 6, 220030, Минск.