БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СБОРНИК ЗАДАЧ ПО ТЕОРИИ АЛГОРИТМОВ

Рекомендовано
Учебно-методическим объединением
по естественнонаучному образованию в качестве
учебно-методического пособия для студентов учреждений
высшего образования, обучающихся по специальностям
1-31 03 03 «Прикладная математика (по направлениям)»,
1-31 03 04 «Информатика», 1-31 03 05 «Актуарная математика»,
1-31 03 06 «Экономическая кибернетика (по направлениям)»,
направление специальности 1-31 03 06-01 «Экономическая кибернетика
(математические методы и компьютерное моделирование в экономике)»,
1-31 03 07 «Прикладная информатика (по направлениям)»,
направление специальности 1-31 03 07-02 «Прикладная информатика
(информационные технологии телекоммуникационных систем)»

МИНСК
БГУ
2017

УДК 510.51(075.8)
ББК 22.12я73-1
С23

Авторы:
В. М. Котов, Ю. Л. Орлович,
Е. П. Соболевская, С. А. Соболь

Рецензенты:
кафедра информатики и методики преподавания информатики
Белорусского государственного педагогического
университета имени Максима Танка
(заведующий кафедрой кандидат педагогических наук,
доцент *С. И. Зенько*);
доктор педагогических наук, доцент *О. И. Мельников*

Сборник задач по теории алгоритмов : учеб.-метод. пособие /
С23 В. М. Котов [и др.]. — Минск : БГУ, 2017. — 183 с.
ISBN 978-985-566-412-4.

Учебно-методическое пособие кроме теоретического материала включает задачи для самостоятельного решения, большинство из которых имеют творческий характер и предлагались на международных олимпиадах по информатике, а также указания к ним.

Предназначено для студентов учреждений высшего образования, обучающихся по специальностям 1-31 03 03 «Прикладная математика (по направлениям)», 1-31 03 04 «Информатика», 1-31 03 05 «Актуарная математика», 1-31 03 06 «Экономическая кибернетика (по направлениям)», направление специальности 1-31 03 06-01 «Экономическая кибернетика (математические методы и компьютерное моделирование в экономике)», 1-31 03 07 «Прикладная информатика (по направлениям)», направление специальности 1-31 03 07-02 «Прикладная информатика (информационные технологии телекоммуникационных систем)».

УДК 510.51(075.8)
ББК 22.12я73-1

ISBN 978-985-566-412-4

ПРЕДИСЛОВИЕ

Дисциплины по теории алгоритмов кроме изучения теоретического материала предусматривают решение практических заданий по построению математической модели для решаемой задачи, разработке эффективного (с точки зрения трудоёмкости) алгоритма с последующим высоким уровнем его реализации на некотором языке программирования. Для проверки работоспособности программ на факультете прикладной математики и информатики БГУ применяется система автоматического тестирования. Ее функциональность позволяет организовать дистанционное обучение, самостоятельную и контролируемую работу студентов. Используя в своей работе систему автоматического тестирования, преподаватель освобождается от рутинной работы, связанной с проверкой решений, получая возможность уделять больше внимания вопросам алгоритмизации.

Учебно-методическое пособие состоит из двух частей: «Алгоритмы на графах» и «Бинарные поисковые деревья». Первая часть содержит практические задачи, которые могут быть сформулированы в графовой постановке. Далее для их решения применяются соответствующие алгоритмы, например алгоритм построения максимального потока, кратчайшего пути и др. Во второй части рассматриваются вопросы организации поиска некоторого элемента в совокупности элементов. Известно, что данная задача может быть решена с использованием списковых структур, хеширования, а также путём построения для последовательности элементов поисковых деревьев (АВЛ-деревьев, 2-3-деревьев и др.). Авторы предлагают построить по последовательности ключей бинарное поисковое дерево, выполнить, используя соответствующий способ обхода вершин дерева, определённые действия с его вершинами и затем удалить

вершину, которая удовлетворяет требуемым свойствам. К каждой части приведены указания к решению задач. Отметим, что многие задачи, рассмотренные в сборнике, предлагались на международных олимпиадах по программированию, что, несомненно, является свидетельством их высокого уровня.

Учебно-методическое пособие будет интересно всем, кто стремится углубить свои знания в области алгоритмики.

Часть 1

АЛГОРИТМЫ НА ГРАФАХ

Графы являются весьма популярными комбинаторными объектами, получившими широкое применение в самых разнообразных областях науки и практики: экономике, физике, химии, молекулярной биологии, программировании, логистике, сетевом и календарном планировании, проектировании интегральных схем и сетей оптико-волоконной связи, распределении ресурсов и многих других.

Язык теории графов позволяет упростить формулировки достаточно сложных задач и предоставляет эффективный инструмент для их исследования.

На практике чаще всего возникают задачи о специальных размещениях и укладках графов на различных геометрических объектах (об оптимальной укладке графа на линии), поиске в графах подграфов с заданным свойством (о наибольшем паросочетании в графе), нахождении в графах специальных маршрутов, обладающих какой-либо экстремальной характеристикой (о кратчайшей цепи между заданной парой вершин графа), а также потоковые задачи на графах (о максимальном потоке минимальной стоимости). Многие из этих задач хорошо изучены, и для решения некоторых из них разработаны эффективные алгоритмы.

1.1. ГРАФЫ

1.1.1. Основные понятия и определения

Поскольку в теории графов нет устоявшейся терминологии, то приведём основные определения и базовые факты, примыкающие к понятию «граф», которые используются в данном учебно-методическом пособии.

Неориентированный граф (или просто *граф*) — упорядоченная пара (V, E), где V — непустое конечное множество, элементы которого

называются *вершинами графа*, а E — подмножество неупорядоченных пар различных элементов из V, т. е. $E \subseteq V^{(2)}$, где

$$V^{(2)} = \{U \subseteq V : |U| = 2\}.$$

Неупорядоченная пара $\{u, v\} \in E$ вершин u и v графа называется *ребром* этого графа (часто такое ребро обозначают uv и говорят, что оно *соединяет* вершины u и v). Если $e = uv$ — ребро, то вершины u и v называют его *концевыми вершинами* и говорят, что они *смежны*. Два ребра *смежны*, если они имеют общую концевую вершину. Ребро и вершина *инцидентны*, если вершина является концевой вершиной данного ребра. Множество вершин графа G обозначают $V(G)$, а множество его рёбер — $E(G)$. Если $|V(G)| = n$ и $|E(G)| = m$, то граф G называют (n, m)-*графом* (пример $(7, 8)$-графа приведён на рис. 1.1). Число вершин графа называется его *порядком*. Граф порядка один — *тривиальный*. Если граф не содержит рёбер, то он называется *пустым*. Граф *полный*, если любые две его вершины смежны (полный граф порядка n обозначается K_n).

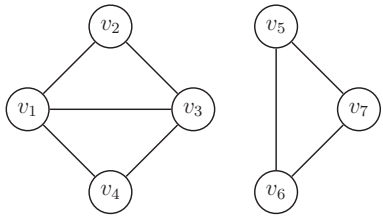


Рис. 1.1. (7,8)-граф

Из определения графа следует, что в нём нет рёбер, соединяющих вершину с ней самой (*петель*), и нет *кратных рёбер*, соединяющих одну и ту же пару вершин. Обобщением графа является *мультиграф*, в котором допускается наличие кратных рёбер. Дальнейшим обобщением графа служит *псевдограф*, в котором могут присутствовать как кратные рёбра, так и петли.

Граф H называется *подграфом* графа G, если выполняются включения $V(H) \subseteq V(G)$ и $E(H) \subseteq E(G)$ (если H — подграф графа G, то говорят, что H *содержится* в G). Всякий подграф может быть получен из графа удалением некоторых вершин (вместе с вершинами удаляются и все инцидентные им рёбра) или рёбер.

Подграф H графа G называется *подграфом, порождённым множеством вершин* $U \subseteq V(G)$, если $V(H) = U$ и $E(H)$ состоит из всех тех рёбер графа G, обе концевые вершины которых принадлежат U.

Можно определить также *подграф* графа G, *порождённый множеством рёбер* $F \subseteq E(G)$. Это подграф $H = (V', F)$, где V' состоит из всех вершин графа G, инцидентных рёбрам из F.

Если H *содержится* в G и $V(H) = V(G)$, то граф H называют *остовным подграфом* или *фактором* графа G.

Для графа, приведённого на рис. 1.1, один из его остовных подграфов H изображён на рис. 1.2.

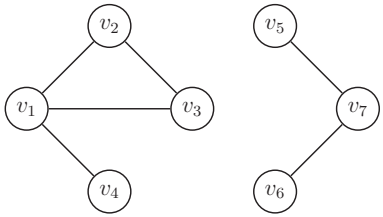


Рис. 1.2. Остовный подграф графа G, изображённого на рис. 1.1

Пусть $G = (V, E)$ — граф и $w\colon E \to \mathbb{R}$ — функция, ставящая в соответствие каждому ребру $e \in E$ некоторое действительное число $w(e)$, которое называется *весом* или *длиной* ребра e. Пара (G, w) называется *взвешенным графом*. Под *весом* (*длиной*) $w(H)$ любого *подграфа* H графа (G, w) будем понимать сумму весов его рёбер, т. е.

$$w(H) = \sum_{e \in E(H)} w(e).$$

Множество всех вершин графа G, смежных с данной вершиной v, называется *окружением* вершины v и обозначается через $N(v)$. *Степень* вершины v (обозначается $\deg(v)$) — число рёбер, инцидентных вершине v ($\deg(v) = |N(v)|$). Заметим, что сумма степеней всех вершин произвольного графа равна удвоенному числу его рёбер, и, следовательно, в любом графе число вершин нечётной степени чётно.

Граф порядка n назовём *помеченным*, если всем его вершинам присвоены метки, например целые числа $1, 2, \ldots, n$. Говорят, что два помеченных графа G и H, заданных на одном и том же множестве $\{1, 2, \ldots, n\}$ вершин, *равны*, если выполняется равенство $E(G) = E(H)$.

1.1.2. Структуры данных для представления графов

В памяти компьютера для задания помеченного (n, m)-графа G наиболее распространёнными способами являются *матрица смежности*, *матрица инцидентности*, *списки смежности*. Выбор того или иного способа задания графа зависит от задачи, которую необходимо решить.

1. *Матрица смежности* — двумерный массив $\boldsymbol{A}_G$ размера $n \times n$, в котором $\boldsymbol{A}_G[i, j] = 1$, если $\{i, j\} \in E(G)$, и $\boldsymbol{A}_G[i, j] = 0$ в противном случае (для взвешенного графа (G, w) в матрице $\boldsymbol{A}_{(G,w)}$ можно хранить вес ребра, и такую матрицу называют *матрицей весов*). Для мультиграфа и псевдографа значение $\boldsymbol{A}_G[i, j]$ равно числу рёбер, соединяющих вершины i и j (при этом петле в матрице соответствует число 2). Такой способ представления (n, m)-графа требует $\Theta(n^2)$ ячеек памяти.

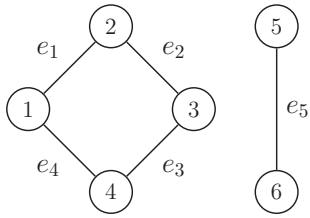


Рис. 1.3. Помеченный (6,5)-граф

Для графа, приведённого на рис. 1.3, его матрица смежности $\boldsymbol{A}_G$ будет иметь следующий вид:

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

2. *Матрица инцидентности* — матрица $\boldsymbol{I}_G$ размера $n \times m$, где $\boldsymbol{I}_G[i, j] = 1$, если вершина i инцидентна ребру e_j, и $\boldsymbol{I}_G[i, j] = 0$ в противном случае (здесь рёбра графа G занумерованы как $e_1, e_2, \ldots, e_m$). Такой способ представления (n, m)-графа требует $\Theta(nm)$ ячеек памяти.

Для графа, приведённого на рис. 1.3, его матрица инцидентности $\boldsymbol{I}_G$ будет иметь следующий вид:

	e_1	e_2	e_3	e_4	e_5
1	1	0	0	1	0
2	1	1	0	0	0
3	0	1	1	0	0
4	0	0	1	1	0
5	0	0	0	0	1
6	0	0	0	0	1

3. *Списки смежности.* При таком способе задания графа каждой вершине i ставится в соответствие список $N(i)$ вершин, смежных с вершиной i (т. е. окружение вершины i). Мощность каждого списка $N(i)$ равна $\deg(i)$ (для взвешенного графа (G, w) элемент списка помимо номера вершины может содержать и вес соответствующего ребра). Кроме того, формируется список $\boldsymbol{L}_G$ размера n, где $\boldsymbol{L}_G[i]$ — ссылка на ячейку, в которой хранится первый элемент списка $N(i)$. Такой способ представления (n, m)-графа требует $\Theta(n + m)$ ячеек памяти.

Для графа, приведённого на рис. 1.3, задание списками смежности будет иметь вид, который схематично представлен на рис. 1.4.

1	→	2	→	4
2	→	1	→	3
3	→	2	→	4
4	→	1	→	3
5	→	6		
6	→	5		

Рис. 1.4. Задание графа, приведённого на рис. 1.3, списками смежности

Использование матриц смежности и инцидентности позволяет применять методы линейной алгебры при исследовании свойств графов.

1.1.3. Маршруты

Маршрутом в графе между заданной парой вершин v_1 и v_{k+1} (или просто (v_1, v_{k+1})-*маршрутом*) называется чередующаяся последовательность вершин и рёбер вида

$$v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1},$$

где $e_i = v_i v_{i+1}$ для всех $i = 1, \ldots, k$ (маршрут может проходить по некоторым рёбрам несколько раз). Такой способ задания маршрутов

удобно использовать при работе с мультиграфами, так как он позволяет уточнить, по какому из рёбер, соединяющему две вершины, проходит этот маршрут. Если в графе нет кратных рёбер, маршрут можно задать перечислением его вершин:

$$v_1, v_2, \ldots, v_k, v_{k+1}.$$

Число k рёбер в маршруте называется его *длиной*. Для взвешенного графа (G, w) вес (v_1, v_{k+1})-*маршрута* определяется как сумма весов рёбер, входящих в этот маршрут (если ребро встречается в маршруте несколько раз, то столько же раз его вес учитывается при определении веса маршрута).

Если $v_1 = v_{k+1}$, то маршрут называется *замкнутым*, в противном случае — *открытым*.

Для подсчёта в графе G числа попарно различных (i, j)-маршрутов длины l можно использовать матрицу смежности этого графа $\boldsymbol{A}_G$ (если возвести матрицу смежности в степень l, то элемент матрицы $\boldsymbol{A}^l_G[i, j]$ равен числу (i, j)-маршрутов длины l).

Цепь в графе — маршрут, в котором каждое ребро встречается не более одного раза (цепь может проходить через некоторые вершины несколько раз и быть как открытой, так и закрытой).

Простая цепь в графе — цепь, в которой каждая вершина встречается не более одного раза.

Серия алгоритмов для нахождения маршрутов, обладающих заданными свойствами, приведена в [4] и [6].

1.1.4. Связность

Граф называется *связным*, если любые две его несовпадающие вершины соединены маршрутом. Граф, приведённый на рис. 1.5, является связным, а на рис. 1.1 — нет. Всякий максимальный по включению связный подграф графа G (т. е. не содержащийся в связном подграфе с большим числом элементов) называется *связной компонентой* графа G (граф, приведённый на рис. 1.1, содержит две связные компоненты).

Для выделения связных компонент графа можно использовать такие алгоритмы обхода графа, как *поиск в ширину* или *поиск в глубину* [4].

Пусть G — произвольный граф. Вершина $v \in V(G)$ называется *точкой сочленения*, если граф $G - v$, полученный в результате удаления из графа G вершины v, имеет больше компонент связности, чем G.

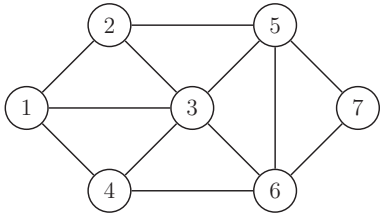


Рис. 1.5. Связный граф G порядка 7

Ребро, при удалении которого увеличивается число компонент связности, называется *мостом*. Для выделения точек сочленения и мостов можно использовать алгоритм, основанный на поиске в глубину, приведённый в [2].

1.1.5. Циклы

Цикл в графе — замкнутый маршрут, в котором каждое ребро встречается не более одного раза (т. е. замкнутая цепь). Граф без циклов называется *ациклическим* графом.

Для связного графа *эйлеров цикл* — это цикл, который содержит все рёбра графа (связный граф, обладающий эйлеровым циклом, называется *эйлеровым графом*). Необходимое и достаточное условие существования в графе эйлерова цикла даёт теорема Л. Эйлера.

Теорема 1.1 (*Л. Эйлер, 1736*). *Нетривиальный связный граф содержит эйлеров цикл тогда и только тогда, когда степени всех его вершин чётны.*

Связный граф, приведённый на рис. 1.5, не содержит эйлерова цикла, так как в нём имеются вершины нечётной степени.

Для построения эйлерова цикла в эйлеровом графе можно использовать такие структуры данных, как *стек* (LIFO) и *очередь* (FIFO) [4]. Алгоритм построения эйлерова цикла используется при решении задачи о покрытии графа наименьшим числом рёберно-непересекающихся цепей (можно говорить, что набор рёберно-непересекающихся цепей *покрывает* граф G, если каждое ребро графа входит в одну из этих цепей) [5].

Простой цикл в графе — замкнутый маршрут, в котором каждая вершина, кроме первой и последней, встречается не более одного раза (т. е. замкнутая простая цепь).

Все простые циклы в графе можно получить, используя, например,

поиск в глубину. Во время поиска в глубину для каждой компоненты связности строится корневое дерево поиска в глубину, т. е. все рёбра графа получают ориентацию и помечаются как древесные или обратные. Каждая обратная дуга корневого дерева поиска в глубину порождает простой цикл графа [4].

Га́мильтоновым циклом в графе называется простой цикл, который содержит каждую вершину этого графа (граф, обладающий гамильтоновым циклом, называется *гамильтоновым графом*). Задача распознавания гамильтоновости графа является NP-полной. Достаточные условия существования в графе гамильтонова цикла дают, например, следующие две классические теоремы.

Теорема 1.2 (*Г. Дира́к, 1952*). *Если для любой вершины v графа G порядка $n \geqslant 3$ выполняется неравенство* $\deg(v) \geqslant n/2$*, то граф G — гамильтонов.*

Теорема 1.3 (*О. О́ре, 1960*). *Пусть G — граф порядка $n \geqslant 3$. Если для любой пары u и v несмежных вершин графа G выполняется неравенство* $\deg(u) + \deg(v) \geqslant n$*, то граф G — гамильтонов.*

Для графа, приведённого на рис. 1.5, не для всех вершин выполняется неравенство из теоремы 1.2. Для этого же графа не для любой пары несмежных вершин выполняется неравенство из теоремы 1.3. Поскольку теоремы 1.2 и 1.3 дают лишь достаточное условие существования гамильтонова цикла, то на основе этих теорем нельзя ответить на вопрос о гамильтоновости данного графа. Легко заметить, что последовательность вершин $(1, 2, 3, 5, 7, 6, 4, 1)$ является гамильтоновым циклом данного графа.

Наряду с достаточными известны и необходимые условия существования гамильтонова цикла. Если необходимое условие не выполняется, то из этого следует, что граф не является гамильтоновым.

Теорема 1.4. *Если G — гамильтонов граф, то для любого непустого множества $X \subseteq V(G)$ число компонент связности графа $G - X$ не больше, чем $|X|$.*

Необходимое условие из теоремы 1.4 не является достаточным для гамильтоновости графа G, т. е. из его истинности не следует, что граф G — гамильтонов.

На рис. 1.6 приведён пример графа G, у которого для любого непустого множества $X \subseteq V(G)$ число компонент связности графа $G - X$ не больше, чем $|X|$, но при этом граф G не гамильтонов. Действительно, на рис. 1.6 выделены рёбра, которые обязательно вошли бы в любой

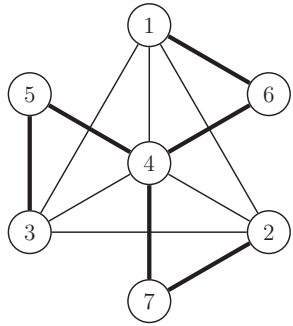


Рис. 1.6. Граф G, который не является гамильтоновым

гамильтонов цикл (иначе через вершины 5, 6, 7 не пройти). Вершина 4 инцидентна трём рёбрам $\{5,4\}$, $\{6,4\}$, $\{7,4\}$, и, значит, эту вершину нужно посетить более одного раза, что невозможно по определению гамильтонова цикла.

На основе теоремы 1.4 нетрудно установить справедливость следующего следствия.

Следствие. *Если G — гамильтонов граф порядка $n \geqslant 3$, то наибольшее число попарно несмежных вершин в графе G, называемое числом независимости графа G, не превосходит $n/2$.*

Очевидный алгоритм построения гамильтонова цикла в связном графе основан на полном переборе, который может быть реализован на основе поиска в глубину. Во время поиска в глубину, начинающего свою работу из некоторой произвольным образом выбранной вершины u, при удалении вершины v из стека положим её непосещённой, а в момент занесения вершины v в стек, используя корневое дерево поиска в глубину, проверим одновременное выполнение двух условий (если оба условия выполнены, то гамильтонов цикл найден и алгоритм можно завершить):

1) (u, v)-цепь содержит все вершины графа;
2) вершины u и v смежны.

Таким образом, алгоритм заключается в построении всех простых цепей с началом в произвольным образом выбранной вершине u. Алгоритм продолжает свою работу до тех пор, пока не обнаружится гамильтонов цикл либо не будут исследованы все простые цепи. Время работы данного алгоритма построения гамильтонова цикла имеет порядок $(n-1)!$.

1.1.6. Деревья

Деревом называется связный граф без циклов. Напомним, что граф без циклов (при этом он может быть несвязным) часто называют *ациклическим* или *лесом*. Вершины дерева, степень которых равна 1, называются *листьями дерева*.

Некоторые альтернативные определения дерева отражены в следующей теореме.

Теорема 1.5. *Для (n, m)-графа G следующие утверждения эквивалентны:*

1) *G — дерево*;

2) *G — связный граф и $m = n - 1$*;

3) *G — ациклический граф и $m = n - 1$*;

4) *любые две несовпадающие вершины u и v графа G соединяет единственная простая (u, v)-цепь*;

5) *G — такой ациклический граф, что добавление к нему ребра между любыми двумя его несмежными вершинами приводит к графу, содержащему единственный цикл.*

1.1.7. Остовные деревья

Пусть H — остовный подграф графа G. Если на каждой компоненте связности графа G множеством рёбер подграфа H порождается дерево, то H называют *остовом* или *каркасом* графа G. Очевидно, что в каждом графе существует остов, который может быть легко получен при помощи алгоритмов поиска в ширину или глубину (удалим из графа рёбра, соответствующие обратным дугам корневого дерева поиска в глубину или ширину, разрушая тем самым циклы графа). Для графа, приведённого на рис. 1.1, один из его остовов H изображён на рис. 1.7.

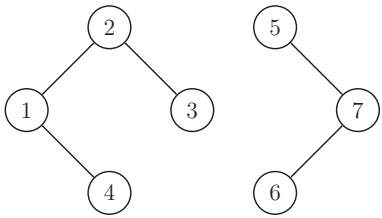


Рис. 1.7. Остов H для графа G, изображённого на рис. 1.1

Остовный подграф, который является деревом, называют *остовным деревом*. Для подсчёта числа остовных деревьев связного графа G порядка $n \geqslant 2$ можно воспользоваться матричной теоремой Г. Кирхгофа о деревьях. *Матрица Кирхгофа* — двумерный массив $\boldsymbol{K}_G$ размера $n \times n$, в котором $\boldsymbol{K}_G[i,j] = \deg(i)$, если $i = j$, $\boldsymbol{K}_G[i,j] = -1$, если $\{i,j\} \in E(G)$, и $\boldsymbol{A}_G[i,j] = 0$, если $i \neq j$ и $\{i,j\} \notin E(G)$.

Другими словами,

$$\boldsymbol{K}_G = \operatorname{diag}[\deg(1), \deg(2), \ldots, \deg(n)] - \boldsymbol{A}_G, \tag{1.1}$$

где $\operatorname{diag}[\deg(1), \deg(2), \ldots, \deg(n)]$ — диагональная $n \times n$ матрица с элементами $\deg(1), \deg(2), \ldots, \deg(n)$, стоящими на главной диагонали.

У матрицы $\boldsymbol{K}_G$ алгебраические дополнения всех элементов одинаковы и их общее значение равно числу остовных деревьев графа G.

Для связного взвешенного графа (G, w) задача о *минимальном остовном дереве* заключается в построении остова минимального веса.

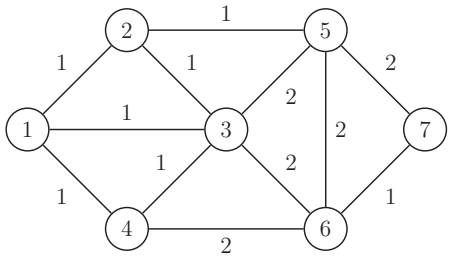


Рис. 1.8. Связный взвешенный граф (G, w)

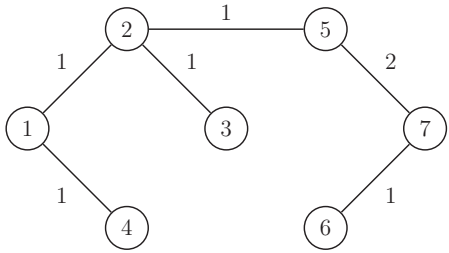


Рис. 1.9. Минимальное остовное дерево T графа G, изображённого на рис. 1.8

В [4] приведён алгоритм Дж. Кра́скала (1956) построения минимального остовного дерева. Если на итерациях алгоритма Краскала для определения цикла использовать такую структуру данных, как *система непересекающихся множеств*, которую в памяти компьютера зададим семейством корневых деревьев, а при объединении двух множеств станем придерживаться правила, что дерево с меньшим числом вершин нужно присоединить к дереву с большим числом вершин, то время работы алгоритма Краскала составит $O(m \log m + m \log n)$.

Для взвешенного графа (G, w), изображённого на рис. 1.8, одно из его минимальных остовных деревьев T (с весом 7) приведено на рис. 1.9.

1.1.8. Двудольные графы

Граф называется *двудольным*, если множество его вершин можно разбить на два подмножества (*доли*) таким образом, чтобы концевые вершины каждого ребра принадлежали разным подмножествам (в данном случае предположим, что одно из подмножеств разбиения может быть пустым). Двудольный граф $G = (V, E)$, для которого первая доля задаётся подмножеством X, а вторая — подмножеством Y $(X \cup Y = V)$, обозначают $G(X, Y, E)$. Необходимое и достаточное условие двудольности графа даёт известная теорема Д. Кёнига.

Теорема 1.6 (*Д. Кёниг, 1936*). *Для графа G эквивалентны следующие условия:*

1) *G — двудольный*;
2) *G не содержит циклов нечётных длин*;
3) *G не содержит простых циклов нечётных длин*;
4) *G не содержит порождённых простых циклов нечётных длин.*

Граф, приведённый на рис. 1.5, не является двудольным.

Граф называется *полным двудольным*, если каждая вершина одной доли смежна с каждой вершиной другой доли (полный двудольный граф, доли которого содержат p и q вершин, обозначают $K_{p,q}$).

Если граф несвязный, то для того, чтобы он был двудольным, необходимо и достаточно, чтобы каждая компонента связности этого графа была двудольным графом (граф, приведённый на рис. 1.3, является двудольным). Для определения двудольности графа можно использовать такие способы обхода графа, как поиск в ширину или поиск в глубину. Так, во время поиска в ширину вершинам будем присваивать номер доли 1 или 2. При удалении из очереди вершины v все смежные с ней непомеченные вершины отнесём к противоположной доле, а для

смежных помеченных вершин проверим, чтобы присвоенный им номер не совпадал с номером доли вершины v (если номера долей у смежных вершин совпали, то граф не является двудольным). Если для определения двудольности графа используется поиск в глубину, то в процессе его работы вершинам будем также присваивать номер доли 1 или 2 и строить корневое дерево поиска в глубину. Вершине, которая заносится в стек из вершины v, присвоим номер доли, противоположный тому, который был у вершины v. Граф не является двудольным, если концы обратных дуг корневого дерева поиска в глубину имеют одинаковые номера долей. Поскольку поиски в ширину и глубину работают за время $O(n + m)$, то такое же время будет потрачено на определение двудольности графа [4].

1.1.9. Паросочетания

Паросочетанием в графе G называется произвольное подмножество его попарно несмежных рёбер.

Паросочетание является *максимальным*, если оно не содержится в паросочетании с бóльшим числом рёбер, и *наибольшим*, если число рёбер в нём наибольшее среди всех возможных паросочетаний графа G. Максимальное паросочетание не всегда наибольшее. Паросочетание M называется *совершенным*, если каждая вершина графа G инцидентна некоторому ребру из M.

Для графа G на рис. 1.10 паросочетание $M_1 = \{\{2, 3\}\}$ — максимальное, но не наибольшее. В то же время паросочетание $M_2 = \{\{1, 2\}, \{3, 4\}\}$ является наибольшим, а так как каждая вершина графа инцидентна одному из рёбер паросочетания M_2, то это паросочетание совершенное.

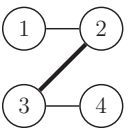


Рис. 1.10. Паросочетание в графе

Серия эффективных алгоритмов для нахождения паросочетаний, обладающих заданными свойствами, приведена в [6].

1.1.10. Кратчайшие маршруты

Кратчайшим маршрутом во взвешенном графе (G, w) между заданной парой вершин u и v называется (u, v)-*маршрут* минимального веса. В [4] и [6] приведён ряд алгоритмов, которые можно использовать для поиска кратчайших маршрутов, обладающих заданными свойствами.

Для решения задачи построения кратчайших маршрутов применяют, как правило, следующие классические алгоритмы (каждый из них в качестве маршрута минимального веса выдаёт простую цепь).

Алгоритм Э. Дейкстры (1959) находит кратчайшие маршруты между вершиной u и всеми вершинами, достижимыми из неё. Алгоритм работает только для графов без рёбер отрицательного веса. Время работы алгоритма Дейкстры для взвешенного (n, m)-графа зависит от выбранной структуры данных:

- массив — $O(n^2 + m)$;
- бинарная куча (в кучу добавляются рёбра при выполнении релаксации по ребру) — $O(m \log m)$;
- бинарная куча (в кучу добавляются вершины; на итерации алгоритма при релаксации по ребру (u, v) из вершины u выполняется операция модификации (уменьшения) приоритета вершины v в куче за время $O(\log n)$; для каждой вершины u хранится указатель на её местоположение в куче) — $O(n \log n + m \log n)$;
- куча Фибоначчи (в кучу добавляются вершины; на итерации алгоритма при релаксации по ребру (u, v) из вершины u выполняется операция модификации (уменьшения) приоритета вершины v в куче за время $O(1)$ (усреднённо); для каждой вершины u хранится указатель на её местоположение в куче) — $O(m + n \log n)$.

Алгоритм Беллмана — Форда (1969) находит кратчайшие маршруты между заданной вершиной u и всеми вершинами, достижимыми из неё. Алгоритм допускает наличие в графе рёбер отрицательного веса. Если в графе (G, w) есть циклы отрицательного веса, достижимые из вершины u, то после выполнения n итераций алгоритм Беллмана — Форда найдёт один из циклов отрицательного веса, а задача построения кратчайших маршрутов в этом случае останется без решения (так как, проходя по циклу, будем каждый раз получать маршрут меньшего веса). Если же в графе отсутствуют циклы отрицательного веса, то задача построения кратчайших маршрутов решится за время $O(nm)$.

Алгоритм Флойда — Уоршелла (1962) находит кратчайшие маршруты между всеми парами вершин. Алгоритм корректно работает также в предположении, что граф содержит рёбра отрицательного веса, но при этом не имеет циклов отрицательного веса. Время работы алгоритма — $O(n^3)$.

Отметим, что в общем случае, когда в графе допускаются циклы

отрицательного веса, задача нахождения кратчайшей простой цепи между заданной парой вершин остаётся корректной, но становится NP-трудной (она не менее трудна, чем известная задача о гамильтоновой (u, v)-цепи в графе). Действительно, пусть G — граф порядка $n \geqslant 2$ и u и v — две его различные вершины. Положив $w(e) = -1$ для каждого ребра $e \in E(G)$, получим взвешенный граф $G' = (G, w)$. Ясно, что в графе G' существует простая (u, v)-цепь длины $1 - n$ тогда и только тогда, когда граф G имеет гамильтонову (u, v)-цепь. Поскольку для заданного графа G и различных вершин $u, v \in V(G)$ задача о гамильтоновой цепи является NP-полной и граф G' может быть построен из G за полиномиальное время, то рассмотренная задача является NP-трудной.

1.2. ОРГРАФЫ

1.2.1. Основные понятия и определения

Напомним, что декартово произведение двух непустых множеств V_1 и V_2 определяется следующим образом:

$$V_1 \times V_2 = \{(u, v) : u \in V_1, v \in V_2\}.$$

Иначе говоря, декартово произведение двух непустых множеств — множество всех упорядоченных пар элементов, где первый элемент выбирается из первого множества, а второй — из второго множества. Декартов квадрат непустого множества V определяется как

$$V^2 = V \times V = \{(u, v) : u, v \in V\}.$$

Ориентированный граф D (*орграф*)— это упорядоченная пара (V, E), где V — непустое конечное множество, элементы которого называются *вершинами*, а $E \subseteq V^2$. Упорядоченная пара вершин $(u, v) \in E$ называется *дугой*. Множество вершин орграфа D обозначают $V(D)$, а множество его дуг — $E(D)$. Если $|V(D)| = n$ и $|E(D)| = m$, то орграф D называют (n, m)-*орграфом*. По аналогии с графом число вершин орграфа называют его *порядком*, а орграф порядка n — *помеченным*, если всем его вершинам присвоены метки, например целые числа $1, 2, \ldots, n$.

Пусть (u, v) — дуга орграфа. Тогда вершины u и v называют *началом* и *концом* дуги соответственно. Дуга (u, v) исходит из вершины u и заходит в вершину v, а вершины u и v *смежны*. Про дугу (u, v) говорят, что она *инцидентна* каждой из своих концевых вершин. Две дуги

называют *смежными*, если они имеют общую концевую вершину. Из определения орграфа следует, что в нём могут быть петли (дуга с совпадающими началом и концом), но отсутствуют кратные (одинаково направленные) дуги.

Для вершины $v \in V$ обозначим через $N^-(v)$ множество дуг с концом в этой вершине ($|N^-(v)|$ — *полустепень захода* для вершины v), а через $N^+(v)$ — множество дуг с началом в этой вершине ($|N^+(v)|$ — *полустепень исхода* для вершины v). *Степень вершины* v орграфа определяется как $\deg(v) = |N^-(v)| + |N^+(v)|$.

Проиллюстрируем некоторые из введённых понятий для орграфа, представленного на рис. 1.11. Дуги орграфа $e_2 = (2, 3)$ и $e_5 = (3, 5)$ являются смежными. Дуга $e_2 = (2, 3)$ инцидентна своим концевым вершинам 2 (начало дуги e_2) и 3 (конец дуги e_2). Вершины 2 и 3 смежны. Степень вершины 3 равна $\deg(3) = |N^-(3)| + |N^+(3)| = 1 + 3 = 4$.

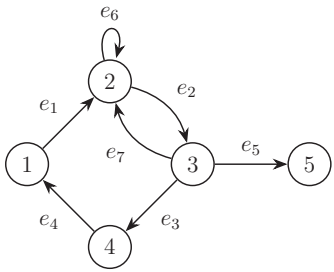


Рис. 1.11. Граф D порядка 5

Очевидно, что любой граф можно рассматривать как орграф, если каждое ребро uv графа заменить двумя противоположно направленными дугами (u, v) и (v, u), поэтому орграф — обобщение понятия «граф». Дальнейшее обобщение понятия «орграф» — *ориентированный мультиграф*, в котором допускается наличие кратных дуг.

Если в графе допускаются рёбра и дуги, то такой граф называют *смешанным графом*.

1.2.2. Структуры данных для представления орграфа

В памяти компьютера для задания помеченного (n, m)-орграфа наиболее распространёнными являются *матрица смежности*, *матрица инцидентности*, *списки смежности* и *списки дуг*.

1. *Матрица смежности* — двумерный массив $\boldsymbol{A}_D$ размера $n \times n$, в котором $\boldsymbol{A}_D[i,j] = 1$, если $(i,j) \in E(D)$ и $i \neq j$; $\boldsymbol{A}_D[i,j] = 2$, если $(i,j) \in E(D)$ и $i = j$; $\boldsymbol{A}_D[i,j] = 0$ в противном случае. Для взвешенного орграфа (D,w) в матрице $\boldsymbol{A}_{(D,w)}$ можно хранить вес дуги. Для ориентированного мультиграфа значение $\boldsymbol{A}_D[i,j]$ равно числу дуг, выходящих из вершины i и входящих в вершину j. Такой способ представления (n,m)-орграфа требует $\Theta(n^2)$ ячеек памяти.

Для орграфа, приведённого на рис. 1.11, его матрица смежности $\boldsymbol{A}_D$ будет иметь следующий вид:

	1	**2**	**3**	**4**	**5**
1	0	1	0	0	0
2	0	2	1	0	0
3	0	1	0	1	1
4	1	0	0	0	0
5	0	0	0	0	0

2. *Матрица инцидентности* — матрица $\boldsymbol{I}_D$ размера $n \times m$, где $\boldsymbol{I}_D[i,j] = 1$, если вершина i инцидентна дуге e_j и является её началом, и $\boldsymbol{I}_D[i,j] = -1$, если вершина i инцидентна дуге e_j и является её концом, и $\boldsymbol{I}_D[i,j] = 0$, если вершина i не инцидентна дуге e_j (здесь дуги орграфа D занумерованы $e_1, e_2, \ldots, e_m$). Петлям в матрице $\boldsymbol{I}_D$ будет соответствовать число 2. Такой способ представления (n,m)-орграфа требует $\Theta(nm)$ ячеек памяти.

Для орграфа, приведённого на рис. 1.11, его матрица инцидентности $\boldsymbol{I}_D$ будет иметь следующий вид:

	$\boldsymbol{e_1}$	$\boldsymbol{e_2}$	$\boldsymbol{e_3}$	$\boldsymbol{e_4}$	$\boldsymbol{e_5}$	$\boldsymbol{e_6}$	$\boldsymbol{e_7}$
1	1	0	0	−1	0	0	0
2	−1	1	0	0	0	2	−1
3	0	−1	1	0	1	0	1
4	0	0	−1	1	0	0	0
5	0	0	0	0	−1	0	0

Если в графе G каждому ребру придать некоторую ориентацию, то полученный орграф D будет называться *одной из ориентаций графа* G. Для графа G существует связь между матрицей Кирхгофа $\boldsymbol{K}_G$ и $\boldsymbol{I}_D$ —

соответствующей матрицей инцидентности некоторой его ориентации D:

$$\boldsymbol{K}_G = \boldsymbol{I}_D \cdot \boldsymbol{I}_D^T, \tag{1.2}$$

где $\boldsymbol{I}_D^T$ — матрица, транспонированная к матрице $\boldsymbol{I}_D$.

Поскольку для графа G матрица Кирхгофа $\boldsymbol{K}_G$ вычисляется через матрицу смежности $\boldsymbol{A}_G$ (см. соотношение (1.1)), то формула (1.2) связывает матрицу смежности графа G с матрицей инцидентности $\boldsymbol{I}_D$ некоторой его ориентации D.

3. *Списки смежности.* При таком способе задания орграфа каждой вершине i ставится в соответствие список $L(i)$ вершин, которые являются концевыми вершинами дуг, выходящих из вершины i. Размер каждого списка $L(i)$ равен $|N^+(i)|$ (для взвешенного орграфа (D, w) элемент списка помимо номера вершины может содержать и вес соответствующей дуги). Кроме того, формируется список $\boldsymbol{L}_D$ размера n, где $\boldsymbol{L}_D[i]$ — ссылка на ячейку, в которой хранится первый элемент списка $L(i)$. Такой способ представления (n, m)-орграфа требует $\Theta(n + m)$ ячеек памяти.

Для орграфа, изображённого на рис. 1.11, задание списками смежности будет иметь вид, который схематично представлен на рис. 1.12.

1	→ [2]
2	→ [3] → [2]
3	→ [4] → [5] → [2]
4	→ [1]
5	→ ∅

Рис. 1.12. Задание орграфа, приведённого на рис. 1.11, списками смежности

4. *Списки дуг.* Предположим, что задан взвешенный орграф и дуги пронумерованы в порядке $e_1, e_2, \ldots, e_m$. Данная структура представляет одномерный массив ***List*** размера n и таблицу ***ArcList***, которая имеет размер $3 \times m$ (в случае невзвешенного орграфа размер таблицы будет $2 \times m$). В массиве ***List*** элемент по индексу i содержит индекс столбца таблицы ***ArcList***, в котором находится информация о дуге, выходящей из вершины i и поступившей самой последней среди всех дуг, выходящих из i. Таблица ***ArcList*** содержит информацию о всех дугах орграфа: ***ArcList***$[1, j]$ — концевая вершина дуги e_j, ***ArcList***$[2, j]$ — вес дуги e_j, ***ArcList***$[3, j]$ — номер столбца таблицы ***ArcList***, в котором

находится информация о дуге, выходящей из той же вершины, что и дуга e_j, и которая поступила раньше. Такой способ представления (n, m)-орграфа требует $\Theta(n + m)$ (в отличие от списков смежности не нужно выделять память под указатели при организации списков).

Для $(5, 7)$-орграфа, приведённого на рис. 1.11, в котором дуги поступали в порядке $e_1, e_2, e_3, e_4, e_5, e_6, e_7$, одномерный массив ***List*** и таблица ***ArcList*** будут иметь следующий вид:

i	**1**	**2**	**3**	**4**	**5**
$List(i)$	1	6	7	4	0

	$\boldsymbol{e_1}$	$\boldsymbol{e_2}$	$\boldsymbol{e_3}$	$\boldsymbol{e_4}$	$\boldsymbol{e_5}$	$\boldsymbol{e_6}$	$\boldsymbol{e_7}$
1	2	3	4	1	5	2	2
2	$w(e_1)$	$w(e_2)$	$w(e_3)$	$w(e_4)$	$w(e_5)$	$w(e_6)$	$w(e_7)$
3	0	0	0	0	3	2	5

1.2.3. Маршруты. Сильная связность орграфа. Двудольные орграфы

Для ориентированного графа понятия *ориентированного маршрута*, *замкнутого ориентированнного маршрута*, *ориентированной цепи* и *ориентированного цикла* вводятся идентично тому, как это было сделано для графа. Вместо понятий *простая цепь* и *простой цикл* вводят понятия *путь* и *контур*. *Путь* в орграфе — ориентированный маршрут, в котором каждая вершина встречается не более одного раза. *Контур* в орграфе — замкнутый ориентированный маршрут, в котором каждая вершина, кроме первой и последней, встречается не более одного раза. Для определения кратчайших маршрутов во взвешенном орграфе можно использовать те же алгоритмы, что и для взвешенного графа.

Если в орграфе существует (u, v)-маршрут, то говорят, что вершина v *достижима* из вершины u (считают, что любая вершина достижима из себя самой). Орграф называют *сильносвязным*, если любые две его вершины достижимы друг из друга. *Сильносвязной компонентой орграфа* называется любой его максимальный по включению сильносвязный подграф (подграфы орграфа определяются так же, как и для графа). Для выделения сильносвязных компонент орграфа можно использовать алгоритм из [4], основанный на двукратном запуске поиска в глубину.

Если в орграфе D снять ориентацию с дуг (т. е. каждую дугу орграфа заменить ребром), то полученный в результате мультиграф

(если в орграфе были петли, то — псевдограф) называют *основанием орграфа* D и обозначают D_b. Орграф называется *связным*, если его основание — связный псевдограф.

Орграф D, приведённый на рис. 1.11, не является сильносвязным, так как из вершины 5 не достижима ни одна другая вершина; орграф D содержит две сильносвязные компоненты с множествами вершин $\{5\}$ и $\{1, 2, 3, 4\}$ и является связным.

Будем считать, что орграф без петель — двудольный, если его основание есть двудольный мультиграф. Таким образом, для определения двудольности орграфа D нужно сначала заменить дуги рёбрами, получив мультиграф D_b, а затем использовать алгоритмы для определения двудольности D_b. Орграф, приведённый на рис. 1.13, — двудольный.

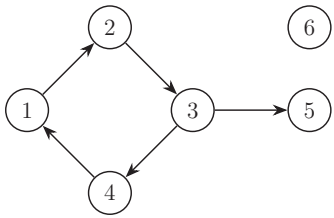


Рис. 1.13. Двудольный орграф

Определения эйлеровых и гамильтоновых орграфов сходны с аналогичными определениями для графов. Ориентированный связный граф является эйлеровым тогда и только тогда, когда для каждой вершины орграфа v выполняется равенство $|N^-(v)| = |N^+(v)|$. Орграф, приведённый на рис. 1.11, не является эйлеровым орграфом.

Достаточное условие гамильтоновости орграфа даёт следующая теорема М. Мейниела.

Теорема 1.7 (*М. Мейниел, 1973*). *Пусть D — сильносвязный орграф порядка $n \geqslant 2$ без петель и кратных дуг. Тогда, если для любой пары u и v его несовпадающих несмежных вершин справедливо равенство* $\deg(u) + \deg(v) \geqslant 2n - 1$, *то орграф D — гамильтонов.*

Понятие паросочетания ориентированного графа сходно с аналогичным понятием для графа.

1.2.4. Топологическая сортировка вершин орграфа

Топологической сортировкой вершин орграфа, который не содержит контуров, называется такая перенумерация его вершин, что у любой

дуги (u, v) номер её начала u меньше, чем номер её конца v (либо наоборот). Для выполнения топологической сортировки вершин ациклического орграфа можно использовать, например, алгоритмы А. Кана (1962) и Р. Тарьяна (1976).

Алгоритм Р. Тарьяна. Выполним поиск в глубину в орграфе. Во время поиска в глубину вершине v присвоим новый номер в момент её удаления из стека, т. е. когда все вершины, в которые ведут дуги из вершины v, уже были посещены (новая нумерация начинается с единицы). Время работы алгоритма Р. Тарьяна топологической сортировки вершин ациклического орграфа равно $O(n)$, при этом у любой дуги (u, v) номер её начала u больше, чем номер её конца v.

Если в орграфе заранее неизвестно о наличии ко́нтуров, то для выполнения топологической сортировки можно использовать *алгоритм А. Кана*, приведённый в [4]. В данном алгоритме, пока существуют вершины без входящих дуг, выполняют следующие действия: находят любую вершину v без входящих дуг и присваивают ей новый номер (новая нумерация начинается с единицы), а затем удаляют из орграфа эту вершину v вместе с выходящими из неё дугами и повторяют алгоритм. Если на некотором шаге алгоритма больше нет вершин без входящих дуг, но при этом не все вершины орграфа получили новые номера, то найден контур, а следовательно, топологическая сортировка не может быть выполнена. Реализация алгоритма А. Кана при задании орграфа матрицей смежности может быть выполнена за время $O(n^2)$, а его реализация при задании орграфа списками смежности — за время $O(n + m)$, при этом у любой дуги (u, v) номер её начала u меньше, чем номер её конца v.

Для орграфа D, изображённого на рис. 1.14, выполнена топологическая сортировка: возле каждой вершины в скобках указан её новый номер (очевидно, что топологическая сортировка неоднозначна).

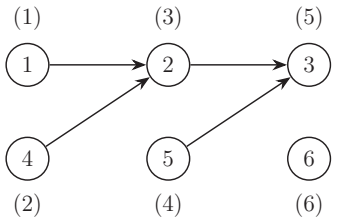


Рис. 1.14. Топологическая сортировка вершин орграфа

Орграф D, приведённый на рис. 1.13, содержит контуры, поэтому топологическая сортировка для него выполнена быть не может.

1.3. СЕТИ. МАКСИМАЛЬНЫЙ ПОТОК В СЕТИ

Пусть $f : E \to \mathbb{R}$ — некоторая функция, заданная на множестве дуг E орграфа D. *Дивергенция функции* f в вершине v определяется как разность сумм её значений на выходящих и входящих дугах:

$$\operatorname{div}_f(v) = \sum_{e \in N^+(v)} f(e) - \sum_{e \in N^-(v)} f(e).$$

Если в орграфе D выделены некоторые вершины, то он называется *сетью*, выделенные вершины — *полюсами*, а остальные вершины — *внутренними* (дуги, не инцидентные полюсам, также называют *внутренними дугами сети*). В классической *задаче о максимальном потоке* в сети D выделяют два полюса s (*источник*) и t (*сток*).

Потоком в сети называют функцию f, дивергенция которой на внутренних вершинах равна нулю. Если сложить дивергенцию для всех вершин орграфа, то полученное значение будет равно нулю, откуда следует, что

$$\operatorname{div}_f(s) = -\operatorname{div}_f(t).$$

Величину $M(f) = \operatorname{div}_f(s) = -\operatorname{div}_f(t)$ называют величиной потока f. Поток, равный нулю на всех дугах сети, — *нулевой поток*. Поток, равный одному и тому же числу на дугах некоторого пути (или вдоль некоторого цикла), а на остальных дугах равный нулю, называют *элементарным потоком*. Поток называют *положительным*, если $f(e) > 0$ для любой дуги $e \in E$.

Если на дугах сети D для потока f задаются ограничения

$$d(e) \leqslant f(e) \leqslant c(e), \text{ для любой дуги } e \in E,$$

то говорят о *сети с ограничениями*. Верхнее ограничение $c(e)$ называется *пропускной способностью дуги* e. В классической задаче о максимальном потоке все нижние ограничения $d(e)$ равны нулю.

Классическая задача о максимальном потоке может быть сформулирована следующим образом: для сети D с полюсами s и t требуется найти поток f максимальной величины, удовлетворяющий ограничениям $0 \leqslant f(e) \leqslant c(e)$ для любой дуги $e \in E$. Поток, величина которого максимальна, называется *максимальным потоком*.

На рис. 1.15 приведён пример потока f в сети D с ограничениями. Возле каждой дуги сначала указана величина потока по дуге, затем её пропускная способность (в формате «$f(e)/c(e)$»). Величина потока $M(f) = 4$. Несложно увидеть, что данный поток не является максимальным (величина максимального потока равна 7).

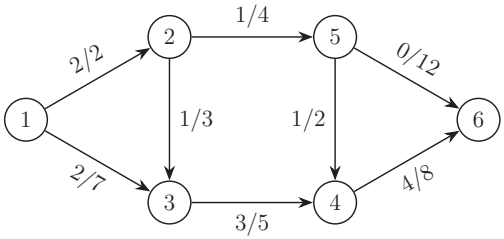


Рис. 1.15. Поток f в сети

Одним из подходов к построению максимального потока является метод Л. Форда и Д. Фалкерсона (1955). Метод Форда — Фалкерсона — итерационный процесс. На каждой итерации рассматривается некоторый поток f. На первой итерации в качестве такого потока f берётся нулевой поток и по нему строится *сеть остаточных пропускных способностей* D_f, например, для каждой дуги $e = (u, v)$ исходного орграфа D введём в соответствие противоположно направленную дугу $\overline{e} = (v, u)$ с равной нулю остаточной пропускной способностью $c_f(\overline{e})$, а остаточную пропускную способность дуг e исходного орграфа положим равной их пропускной способности в исходной сети, т. е. $c_f(e) = c(e)$. Затем в сети остаточных пропускных способностей D_f найдём (s, t)-путь L по дугам, остаточная пропускная способность которых больше нуля. Если (s, t)-путь L найден, то среди дуг этого пути найдём дугу с минимальной остаточной пропускной способностью $c_f^{\min}$ и увеличим величину текущего потока f на число $c_f^{\min}$. Перестроим сеть D_f для нового потока f по следующему правилу: для каждой дуги $e \in L$ уменьшим её остаточную пропускную способность на величину $c_f^{\min}$, а по соответствующей ей обратной дуге $\overline{e}$ — увеличим на $c_f^{\min}$ (остаточные пропускные способности дуг сети, которые не принадлежат пути L, оставим без изменения). Затем снова для текущего потока f в соответствующей ему сети D_f осуществим поиск (s, t)-пути. Если на некоторой итерации для текущего потока f в сети D_f нет увеличивающего (s, t)-пути, то, как будет показано далее, текущий поток f является максимальным (при

этом величину максимального потока f для каждой дуги $e \in E$ можно вычислить по формуле $f(e) = c(e) - c_f(e)$).

Если в методе Форда — Фалкерсона для построения увеличивающего пути использовать поиск в глубину, то в предположении, что пропускные способности дуг являются целыми числами, получим псевдополиномиальный алгоритм со временем работы $O(M(f)(n+m))$, где $M(f)$ — величина максимального потока. Если для построения увеличивающего пути использовать поиск в ширину (в этом случае в качестве (s,t)-пути в сети остаточных пропускных способностей D_f всегда выбирается (s,t)-путь наименьшей длины), то такую реализацию метода Форда — Фалкерсона называют алгоритмом Эдмондса — Карпа (впервые опубликован в 1972 г.), который является полиномиальным и работает за время $O(nm^2)$. Данная оценка получается исходя из того факта, что максимальный поток строится с использованием не более $mn/2$ путей (см. [1]).

Для доказательства корректности метода Форда — Фалкерсона рассмотрим связь максимального потока в сети и её минимального разреза.

Разрезом $R = (X, \overline{X})$ *сети* D с полюсами s и t называется разбиение множества её вершин V на непересекающиеся между собой подмножества X и $\overline{X}$, причём $s \in X$, $t \in \overline{X}$. Сложим дивергенцию по всем вершинам множества X (эта величина называется *дивергенцией потока* f *на разрезе* R и обозначается $\mathrm{div}_f(R)$). Сумма равна разности сумм потока по дугам, выходящим из множества X и входящим в множество $\overline{X}$ (множество этих дуг называют *выходящими из* X и обозначают $A^+(R)$), и дугам, выходящим из множества $\overline{X}$ и входящим в множество X (множество этих дуг называют *входящими в* X и обозначают $A^-(R)$). С другой стороны, сумма равна величине потока, так как дивергенция по всем внутренним вершинам сети D равна нулю:

$$\begin{aligned} \mathrm{div}_f(R) &= \sum_{v \in X} \mathrm{div}_f(v) = \sum_{e \in A^+(R)} f(e) - \sum_{e \in A^-(R)} f(e) = \\ &= \mathrm{div}_f(s) = M(f). \end{aligned} \tag{1.3}$$

Таким образом, для потока f и любого разреза R справедливо утверждение: дивергенция потока f на разрезе R равна величине потока (на рис. 1.15 несложно увидеть, что для заданного потока f и любого разреза R дивергенция потока f на разрезе R равна величине потока $M(f) = 4$).

Пропускная способность $c(R)$ *разреза* $R = (X, \overline{X})$ — это сумма пропускных способностей дуг из множества $A^+(R)$:

$$c(R) = \sum_{e \in A^+(R)} c(e). \tag{1.4}$$

Разрез, пропускная способность которого минимальна, называется *минимальным разрезом*. Для сети, приведённой на рис. 1.16, минимальным разрезом является $R = (X, \overline{X})$, где $X = \{1, 3\}$, пропускная способность минимального разреза $c(R) = 7$.

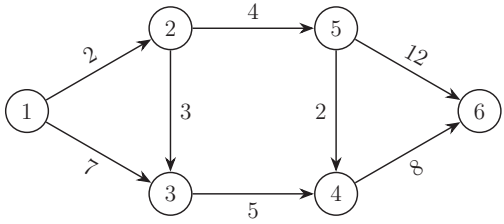


Рис. 1.16. Сеть с полюсами $s = 1$ и $t = 6$

Учитывая, что для каждой дуги $f(e) \leqslant c(e)$, из (1.3) и (1.4) получим следующее соотношение:

$$M(f) = \sum_{e \in A^+(R)} f(e) - \sum_{e \in A^-(R)} f(e) \leqslant \sum_{e \in A^+(R)} c(e) = c(R). \tag{1.5}$$

Таким образом, для любого потока f и любого разреза R справедливо утверждение: величина любого потока не превосходит пропускной способности любого разреза, а значит, и величина максимального потока не превосходит пропускной способности минимального разреза. Поэтому если для некоторого потока f и разреза R будет выполнено равенство $c(R) = M(f)$, то это будет означать, что текущий поток f — максимальный. Несложно показать, что подобная ситуация наступает, когда для текущего потока f в сети остаточных пропускных способностей D_f на итерациях метода Форда — Фалкерсона не существует увеличивающего (s, t)-пути, а множество вершин X минимального разреза $R = (X, \overline{X})$ в этом случае порождается множеством вершин, достижимых из источника s в сети D_f.

Действительно, из (1.5) следует, что

$$M(f) = \sum_{e \in A^+(R)} f(e) - \sum_{e \in A^-(R)} f(e) = \sum_{e \in A^+(R)} c(e) = c(R). \qquad (1.6)$$

Теорема 1.8 (*Форд — Фалкерсон, 1956, о максимальном потоке и минимальном разрезе*). *Величина максимального потока $M(f)$ равна пропускной способности $c(R)$ минимального разреза.*

В качестве примера рассмотрим сеть D, приведённую на рис. 1.16, с полюсами $s = 1$ и $t = 6$ и ограничениями $c(e)$ на пропускные способности дуг $e \in E$. Требуется найти поток f максимальной величины, удовлетворяющий ограничениям $0 \leqslant f(e) \leqslant c(e)$ для любой дуги $e \in E$.

На первой итерации в качестве текущего потока f берём нулевой поток, т. е. $M(f) = 0$ (рис. 1.17), и строим сеть остаточных пропускных способностей D_f (на рис. 1.17 для сети D_f дуги $e \in E$ показаны сплошными линиями, а соответствующие им обратные дуги $\overline{e}$ — пунктирными линиями). В сети остаточных пропускных способностей D_f, используя алгоритм поиска в ширину (движение осуществляется по дугам, остаточная пропускная способность которых больше нуля), построим увеличивающий $(1, 6)$-путь L (на рис. 1.17 дуги увеличивающего пути L выделены полужирными линиями). Среди дуг пути L найдём дугу с минимальной остаточной пропускной способностью $c_f^{\min} = 2$ и увеличим величину текущего потока f на число $c_f^{\min}$, т. е. $M(f) = 2$.

Перейдём ко второй итерации алгоритма. Перестроим по текущему потоку f сеть остаточных пропускных способностей D_f, вычитая из остаточных пропускных способностей дуг увеличивающего пути L, найденного на первой итерации, число $c_f^{\min}$ и добавляя это число к соответствующим им обратным дугам (остаточные пропускные способности дуг, которые не принадлежат пути L, оставим без изменения). Затем снова осуществим поиск увеличивающего $(1, 6)$-пути L, используя алгоритм поиска в ширину (на рис. 1.18 дуги увеличивающего пути L выделены полужирными линиями). Среди дуг пути L найдём дугу с минимальной остаточной пропускной способностью $c_f^{\min} = 5$ и увеличим величину текущего потока f на число $c_f^{\min}$, т. е. $M(f) = 7$.

Перейдём к третьей итерации алгоритма. Перестроим по текущему потоку f сеть остаточных пропускных способностей D_f и осуществим поиск увеличивающего $(1, 6)$-пути L, используя алгоритм поиска в ширину (рис. 1.19). Поскольку на третьей итерации из источника $s = 1$

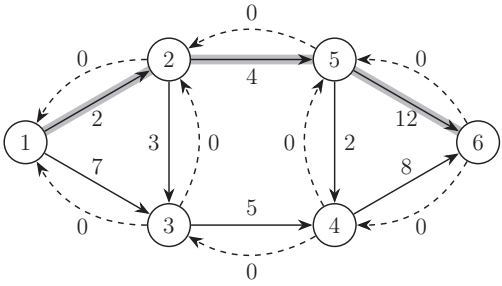


Рис. 1.17. Сеть остаточных пропускных способностей D_f на первой итерации и увеличивающий $(1, 6)$-путь L

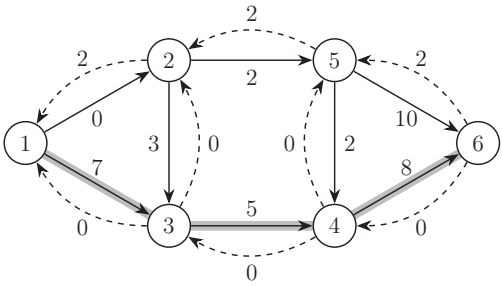


Рис. 1.18. Сеть остаточных пропускных способностей D_f на второй итерации алгоритма и увеличивающий $(1, 6)$-путь L

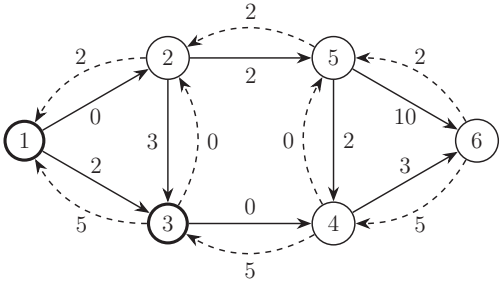


Рис. 1.19. Сеть остаточных пропускных способностей на третьей итерации алгоритма

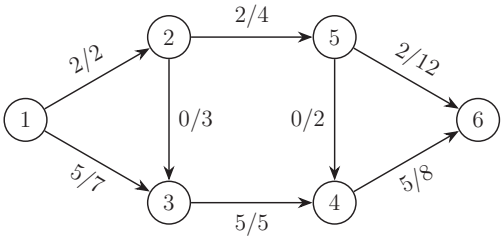


Рис. 1.20. Максимальный поток в сети

удаётся пометить только вершину с номером 3, то на основании теоремы Форда — Фалкерсона о максимальном потоке и минимальном разрезе текущий поток f является максимальным (величина максимального потока $M(f) = 7$). Таким образом, алгоритм Эдмондса — Карпа построения максимального потока в сети завершает свою работу.

Максимальный поток для сети, приведённой на рис. 1.16, показан на рис. 1.20 (возле каждой дуги сначала указана величина потока по ней, затем её пропускная способность). Множество вершин X минимального разреза $R = (X, \overline{X})$ определяется множеством вершин, достижимых из источника $s = 1$ в сети остаточных пропускных способностей D_f, построенной на третьей итерации алгоритма (на рис. 1.19 вершины множества $X = \{1, 3\}$ выделены). Пропускная способность минимального разреза

$$c(R) = c(1, 2) + c(3, 4) = 7$$

равна величине максимального потока $M(f) = 7$.

Подчеркнём, что в общем случае увеличивающий путь может проходить как по дугам исходной сети, так и по обратным дугам. На рис. 1.21, *а* показан пример сети с единичными пропускными способностями. На первой итерации (рис. 1.21, *б*) единица потока пропускается по кратчайшему пути 1 – 2 – 3 – 4. На второй итерации алгоритм находит увеличивающий путь 1 – 5 – 6 – 3 – 2 – 7 – 8 – 4 (рис. 1.21, *в*), где дуга $(3, 2)$ является обратной. Фактически проталкивание единицы потока вдоль этого пути отменяет ранее назначенный прямой дуге $(2, 3)$ поток. В результате второй итерации остаточная пропускная способность дуги прямой дуги $(2, 3)$ восстанавливается до значения 1, а обратной дуги $(3, 2)$ — до значения 0.

В том случае, когда в сети ограничения на верхнюю пропускную способность всех дуг равны единице, величина максимального потока равна наибольшему числу (s, t)-путей, которые попарно не пересекаются по дугам (дуги сети, поток по которым равен единице, задают искомое множеству дуг (s, t)-путей). Для сети, приведённой на рис. 1.21, *а*, наибольшее число $(1, 4)$-путей, которые не пересекаются по дугам, равно 2 (на рис. 1.21, *г* дуги искомых путей выделены).

В некоторых задачах для сети D, кроме пропускной способности потока по дуге $c(e) > 0$, задаётся *удельная стоимость потока* в дуге $p(e) \geqslant 0$. Необходимо найти максимальный поток минимальной

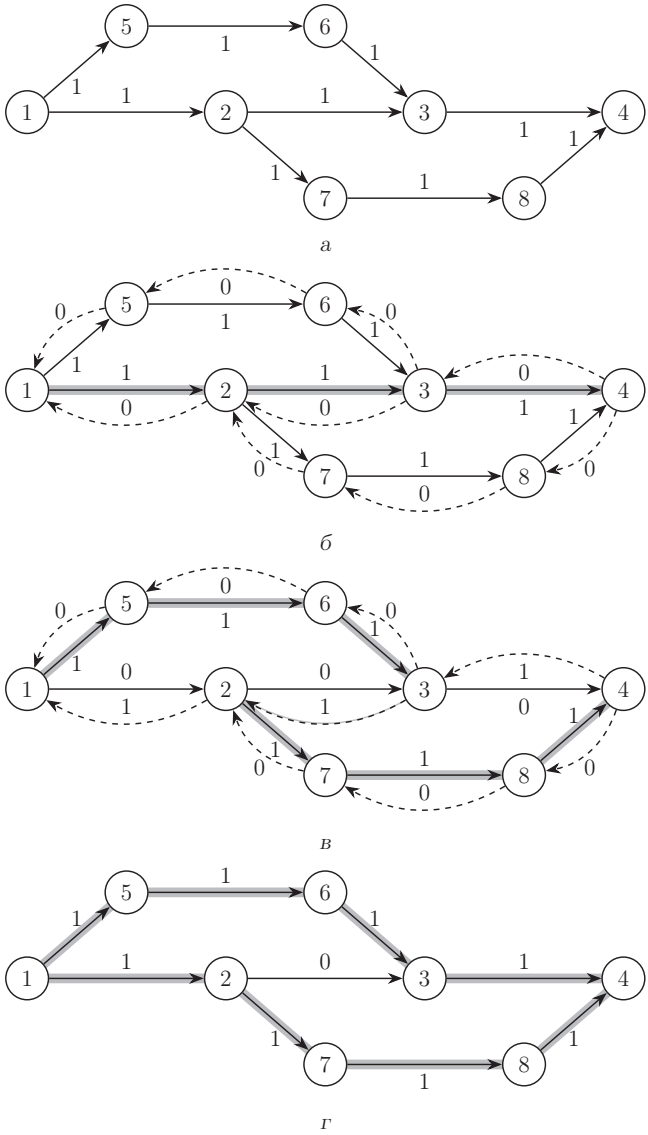


Рис. 1.21. Пример работы алгоритма Эдмондса — Карпа:
а — исходная сеть;
б, *в* — итерации алгоритма;
г — полученный максимальный поток

стоимости, где под стоимостью потока f в сети D понимается величина

$$P(f) = \sum_{e \in E} f(e) \cdot p(e).$$

В [4] приведён один из алгоритмов решения данной задачи. В нём сначала в сети D строится максимальный поток f^* любым из алгоритмов построения максимального потока. Затем в сети остаточных пропускных способностей D_{f^*} прямым дугам e поставим в соответствие удельную стоимость $p(e)$, а обратным дугам $\overline{e}$ — удельную стоимость, равную $-p(e)$ (будем считать, что дуга в сети существует, если $c_{f^*}(e) > 0$). Пока в сети остаточных пропускных способностей для перестраиваемого максимального потока существует контур отрицательной удельной стоимости, перераспределим поток вдоль этого контура, получая каждый раз поток той же мощности, но меньшей удельной стоимости. Если на некотором шаге в сети остаточных пропускных способностей нет контура отрицательной удельной стоимости, то построенный максимальный поток является максимальным потоком минимальной стоимости.

Ещё одним из алгоритмов решения данной задачи, который хорошо зарекомендовал себя на практике, является *метод минимальных путей* (алгоритм Басакера — Гоуэна) [1]. Как и в методе Форда — Фалкерсона, на каждой итерации по текущему потоку f строится сеть остаточных пропускных способностей D_f (считаем, что дуга в сети D_f существует, если $c_f(e) > 0$). Поставим прямым дугам e сети D_f в соответствие удельную стоимость $p(e)$, а обратным дугам $\overline{e}$ — удельную стоимость, равную $-p(e)$. Теперь на каждой итерации текущий максимальный поток будем наращивать вдоль увеличивающего пути минимальной удельной стоимости. Отличительные особенности этого метода в том, что, во-первых, на любой итерации построенный поток будет иметь наименьшую удельную стоимость среди потоков той же мощности. Во-вторых, в сети D_f, несмотря на наличие дуг с отрицательной удельной стоимостью, любой контур имеет неотрицательную стоимость, поэтому для поиска пути минимальной стоимости можно использовать алгоритм Форда — Беллмана.

1.4. ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

Решение задач этого раздела предполагает использование известных и разработку новых алгоритмов на графах. Базовые алгоритмы не следует как-то модифицировать, вместо этого лучше грамотно построить граф-модель задачи, чтобы использовать их без изменений.

Приемлемое время работы алгоритма определяется исходя из ограничений на входные данные в каждой задаче. Полученная для алгоритма оценка времени работы должна гарантировать, что решение уложится в отведённое время на *любом* наборе входных данных, соответствующем ограничениям, указанным в условии задачи.

Задача 1. Дерево

Задан неориентированный граф без петель и кратных рёбер. Определить, является ли этот граф деревом.

Формат входных данных

В первой строке входного файла записано целое число N — количество вершин в графе ($1 \leqslant N \leqslant 100$).

Далее записана матрица смежности графа: N строк по N чисел, каждое из которых — 0 или 1. Гарантируется, что матрица симметрическая и на главной диагонали все элементы нулевые.

Формат выходных данных

Выведите `Yes`, если граф является деревом по одному из эквивалентных определений, и `No` — в противном случае.

Примеры

входной файл	*выходной файл*	*пояснение*
3 0 1 0 1 0 1 0 1 0	Yes	рис. 1.22, *а*
3 0 1 1 1 0 1 1 1 0	No	рис. 1.22, *б*

а *б*

Рис. 1.22

Задача 2. Остовное дерево

Задан неориентированный граф без петель и кратных рёбер. Требуется построить какое-либо остовное дерево этого графа или сообщить, что его не существует.

Формат входных данных

В первой строке входного файла записано целое число N — количество вершин в графе ($1 \leqslant N \leqslant 100$).

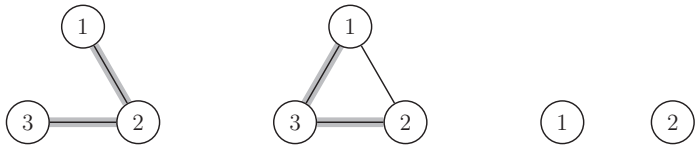
Далее записана матрица смежности графа: N строк по N чисел, каждое из которых — 0 или 1. Числа разделяются одиночными пробелами. Гарантируется, что матрица симметрическая, все элементы на главной диагонали нулевые.

Формат выходных данных

Если остовное дерево построить невозможно, выведите одно число -1. В противном случае в первой строке выведите число M — количество рёбер в остовном дереве, в последующих M строках напечатайте сами рёбра, т. е. укажите для каждого ребра пару вершин, которые это ребро соединяет. Вершины графа нумеруются числами $1, 2, \ldots, N$. Порядок вывода рёбер значения не имеет. Если остовных деревьев несколько, выведите любое.

Примеры

входной файл	*выходной файл*	*пояснение*
3 0 1 0 1 0 1 0 1 0	2 1 2 3 2	рис. 1.23, *а*
3 0 1 1 1 0 1 1 1 0	2 3 2 3 1	рис. 1.23, *б*
2 0 0 0 0	-1	рис. 1.23, *в*



а *б* *в*

Рис. 1.23

Задача 3. Наибольшее число маршрутов (не пересекающихся по рёбрам)

Задан граф. Необходимо найти наибольшее число не пересекающихся по рёбрам маршрутов между заданными вершинами v и w графа.

Формат входных данных

Первая строка содержит число N вершин и число M рёбер графа ($1 \leqslant N \leqslant 100$, $0 \leqslant M \leqslant 4950$). Вершины графа пронумерованы целыми числами от 1 до N.

Каждая из следующих N строк файла задаёт множество вершин графа, смежных с очередной вершиной. Так, $(i + 1)$-я строка файла содержит вершины, смежные с вершиной i, последним в строке идёт число 0 (если нет вершин, смежных с некоторой вершиной графа, то соответствующая строка содержит только число 0).

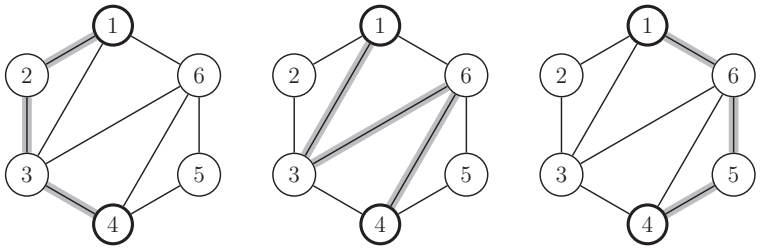
В последней строке файла находятся вершины v и w. Все числа внутри одной строки разделены одним или несколькими пробелами.

Формат выходных данных

В единственной строке выведите наибольшее число не пересекающихся по рёбрам маршрутов между вершинами v и w.

Пример

входной файл	*выходной файл*	*пояснение*
6 9 2 3 6 0 1 3 0 1 2 4 6 0 3 5 6 0 4 6 0 1 3 4 5 0 1 4	3	рис. 1.24



а *б* *в*

Рис. 1.24

Задача 4. Лабиринт (без пересечений по дорогам)

Лабиринт задаётся матрицей смежности C размера $N \times N$, где $C_{i,j} = 1$, если комната i связана с комнатой j посредством дороги (i и j — целые числа от 1 до N). Часть комнат является входами, часть — выходами. Входы и выходы задаются наборами комнат $\{ X_1, \ldots, X_p \}$ и $\{ Y_1, \ldots, Y_k \}$ соответственно. Необходимо найти наибольшее число людей, которых можно провести от входов до выходов таким образом, чтобы их пути не пересекались по дорогам, но могли пересекаться по комнатам.

Формат входных данных

Первая строка содержит размер лабиринта N ($1 \leqslant N \leqslant 300$), число входов p и число выходов k. Ни одна из комнат не является одновременно и входом, и выходом.

Затем идут N строк, которые содержат матрицу смежности C (каждой строке матрицы соответствует строка входного файла).

Следующая строка содержит p различных чисел — номера входов.

Последняя строка содержит k различных чисел — номера выходов. Комнаты нумеруются числами от 1 до N.

Формат выходных данных

Единственная строка этого файла содержит наибольшее число людей, которых можно провести.

Пример

входной файл	*выходной файл*	*пояснение*
6 1 3 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 4 5 6	3	рис. 1.25

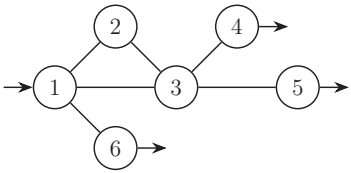


Рис. 1.25

Задача 5. Шестерёнки

Каждой из N шестерёнок присвоен номер: число от 1 до N. Задано M соединений пар шестерёнок в виде $\{ i, j \}$ (шестерёнка с номером i находится в зацеплении с шестерёнкой с номером j).

Необходимо определить, какое наименьшее число шестерёнок нужно удалить, чтобы первая пришла в движение. Если таких вариантов несколько, то нужно выбрать тот, при котором вращаться будет наибольшее число шестерёнок.

Формат входных данных

Первая строка содержит два числа, записанных через пробел: число N шестерёнок и число M соединений пар ($0 \leqslant N \leqslant 10$, $0 \leqslant M \leqslant 45$).

Затем идут M строк файла, каждая из которых задаёт пару $\{ i, j \}$ соединённых шестерёнок ($1 \leqslant i < j \leqslant N$).

Формат выходных данных

Если первая шестерёнка пришла в движение без удаления других шестерёнок, то в первой строке выведите `Yes`, а через пробел укажите число шестерёнок, которые пришли при этом в движение; во второй строке укажите (через пробел) номера шестерёнок, которые пришли в движение (в порядке возрастания их номеров).

Если для вращения первой шестерёнки необходимо удаление других шестерёнок, то в первой строке выведите `No`, а во второй — число шестерёнок, которые нужно убрать, и через двоеточие — их номера (если такой набор не один, то выдаётся лексикографически наименьший; в наборе шестерёнки перечисляются в порядке возрастания их номеров); в третьей строке — число шестерёнок, которые пришли в движение, и через двоеточие — их номера (в порядке возрастания).

Пример

входной файл	*выходной файл*
6 10 1 2 2 3 3 4 4 5 5 6 6 1 1 4 2 5 3 6 2 6	No 1:2 5:1 3 4 5 6

Задача 6. Открытки и конверты

Имеется N прямоугольных конвертов и N прямоугольных открыток различных размеров.

Необходимо определить, можно ли разложить все открытки по конвертам, чтобы в каждом конверте было по одной открытке. Открытки нельзя складывать, сгибать и т. п., но можно помещать в конверт под углом. Например, открытка с размерами сторон 5×1 помещается в конверты с размерами 5×1, 6×3, $4{,}3 \times 4{,}3$, но не входит в конверты с размерами 4×1, $10 \times 0{,}5$, $4{,}2 \times 4{,}2$.

Формат входных данных

Первая строка содержит число N — число открыток (конвертов) ($1 \leqslant N \leqslant 500$). Затем идут N строк, каждая из которых содержит два действительных числа, задающих размеры очередной открытки. Затем следуют N строк файла, каждая из которых содержит два действительных числа, задающих размеры очередного конверта. Все числа положительны, не превосходят 1000 и записаны с не более чем одним знаком после точки.

Формат выходных данных

Если все открытки можно разложить, то единственная строка файла должна содержать сообщение `YES`.

Если все открытки нельзя разложить, то первая строка выходного файла должна содержать сообщение `NO`, а вторая — наибольшее число открыток, которые можно разложить по конвертам.

Пример

входной файл	*выходной файл*
2 7 1.4 6.5 5 7 7 6 6	YES
1 5 20 16 19	YES
3 7 1.4 6.5 5 4 4 7 7 6 6 5 3	NO 2

Задача 7. Разбиение на команды по численности

Необходимо определить, можно ли разбить N студентов на две команды, численность которых отличается не более чем в два раза, если известно, что в каждой команде любых два студента должны быть знакомы друг с другом. Круг знакомств задаётся матрицей A размера $N \times N$ с элементом A_{ij}, равным 1, если i-й студент знаком со студентом j, и элементом A_{ij}, равным 0, если i-й студент не знаком со студентом j.

Формат входных данных

Первая строка содержит число N студентов ($1 \leqslant N \leqslant 1000$).

Затем идут N строк, которые задают матрицу A знакомств (каждой строке матрицы соответствует отдельная строка входного файла, числа в строках разделены пробелами).

Формат выходных данных

Если можно разбить студентов на две команды, численность которых отличается не более чем в два раза, то в первой строке выведите `YES`, а во второй — номера (в порядке возрастания) студентов, которые попали в команду с наибольшей численностью.

Если нельзя разбить студентов на две команды, численность которых отличается не более чем в два раза, то единственная строка файла должна содержать сообщение `NO`.

Примеры

входной файл	*выходной файл*
7 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0	NO
6 0 1 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0	YES 1 2 3 4
2 0 0 0 0	YES 2

Задача 8. Разбиение на группы незнакомых

Имеется N человек и матрица A размера $N \times N$. Элемент A_{ij} матрицы равен 1, если человек i знаком с человеком j (если i-й человек знает j-го, то считаем, что и j-й человек знает i-го), и элемент A_{ij} матрицы равен 0, если i-й человек не знаком с человеком j. Можно ли разбить людей на две группы, чтобы в каждой группе были только незнакомые люди (в каждой группе должен быть хотя бы один человек)?

Формат входных данных

Первая строка содержит число N людей ($2 \leqslant N \leqslant 500$).

Затем идут N строк файла, которые задают матрицу A знакомств (каждой строке матрицы соответствует отдельная строка входного файла, числа в строках разделены пробелами).

Формат выходных данных

Если можно разбить людей на две группы, чтобы в каждой группе были только незнакомые люди, то выведите в первой строке `YES`, а во второй — номера людей (люди нумеруются от 1 до N), которые попали в одну из групп. Числа в строке упорядочены по возрастанию.

Если разбить нужным образом нельзя, то выведите `NO`.

Примеры

входной файл	*выходной файл*
4 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 0	NO
4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	YES 1 2

Задача 9. Олимпиада

На олимпиаду прибыли N человек. Некоторые из них знакомы между собой. Круг знакомств задан матрицей A размера $N \times N$. Элемент A_{ij} матрицы равен 1, если человек i знаком с человеком j (если i-й человек знает j-го, то это значит, что j-й человек знает i-го), и элемент A_{ij} матрицы равен 0, если i-й человек не знаком с человеком j.

Необходимо определить, можно ли опосредованно (незнакомые люди могут познакомиться только через общего знакомого) перезнакомить всех между собой.

Формат входных данных

Первая строка содержит число N людей ($2 \leqslant N \leqslant 500$).

Затем идут N строк файла, которые задают матрицу знакомств A (каждой строке матрицы соответствует отдельная строка входного файла).

Формат выходных данных

Если можно опосредованно перезнакомить всех прибывших на олимпиаду между собой, то первая строка файла будет содержать сообщение YES; в противном случае строка файла должна содержать сообщение NO.

Пример

входной файл	*выходной файл*
5 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 0 0 1 0 0 0 1 0 0 0	YES
4 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0	NO
2 0 1 1 0	YES

Задача 10. Книга

Группа состоит из N человек. В ней каждый имеет ровно $N/2$ друзей. Отношение дружбы симметрично (если A — друг $Б$, то и $Б$ — друг A). У одного человека в группе (его номер X) есть книга, которую все хотели бы прочитать и обсудить.

Необходимо определить способ передачи книги, при котором она побывала бы у каждого в точности один раз, переходя только от друга к другу, и возвратилась к своему владельцу.

Формат входных данных

Первая строка файла содержит три числа N, K, X, где N — число человек в группе, $K = N/2$ — число друзей у каждого человека, X — номер обладателя книги ($2 \leqslant N \leqslant 1000$, $1 \leqslant X \leqslant N$).

Следующие N строк содержат по K различных чисел — номера друзей для 1-го, 2-го, ..., N-го человека соответственно. Гарантируется, что в списке друзей i-го человека он сам (число i) не упоминается.

Формат выходных данных

Выведите последовательность номеров людей в том порядке, в котором может осуществляться передача книги (первый и последний элементы этой строки должны соответствовать владельцу книги).

Пример

входной файл	*выходной файл*
6 3 1 2 3 5 1 4 6 1 4 6 2 3 5 1 4 6 2 3 5	1 5 6 2 4 3 1

Задача 11. Станки

Конвейер состоит из N различных станков. Есть N рабочих. Известна матрица C размера $N \times N$, где элемент c_{ij} задаёт производительность i-го рабочего на j-м станке.

Необходимо определить, каким должно быть распределение рабочих по станкам (каждый рабочий может быть назначен только на один станок, на каждом станке может работать только один рабочий), чтобы производительность всего конвейера была максимальной. Производительность конвейера при некотором распределении рабочих по станкам равна минимальной производительности рабочих на назначенных им на конвейере станках.

Если решение не единственно, вывести решение, первое в лексикографическом порядке среди всех решений.

Формат входных данных

Первая строка содержит число рабочих (станков) N ($1 \leqslant N \leqslant 500$).

Затем идут N строк файла, которые задают матрицу C производительностей ($1 \leqslant c_{ij} \leqslant 1000$).

Формат выходных данных

В первой строке выведите максимальную возможную производительность конвейера. Во второй строке — номера станков, на которые должны быть распределены рабочие $1, 2, \ldots, N$ соответственно.

Пример

входной файл	*выходной файл*
4 4 2 6 1 5 2 3 3 4 7 1 4 4 3 2 5	5 3 1 2 4

Задача 12. Встреча

В городе N домиков и M дорог. В каждом домике живёт по одному человеку. Необходимо найти точку (место встречи всех людей), от которой суммарное расстояние по дорогам до всех домиков будет минимальным. Место встречи следует искать среди точек, в которых расположены домики, а также точек, лежащих на дороге и отстоящих от домиков на целое число единиц длины.

Формат входных данных

Первая строка содержит число N домиков ($1 \leqslant N \leqslant 100$) и число M дорог ($0 \leqslant M \leqslant N \times (N-1)/2$). Домики пронумерованы числами от 1 до N.

Затем идут M строк файла, по три целых числа в каждой, задающие дороги: номера u и v домиков ($1 \leqslant u, v \leqslant N$), которые являются концами дороги, и длина w дороги ($1 \leqslant w \leqslant 100\,000$).

Гарантируется, что никакая дорога не соединяет домик с самим собой. Между любой парой домиков имеется не более одной дороги. По дорогам можно двигаться в обе стороны.

Формат выходных данных

Если точка встречи лежит на дороге, то выведите три целых числа: номера конечных домиков дороги и расстояние от первого из них до этой точки.

Если точка совпадает с домом, то выведите его номер и суммарное расстояние от всех домиков до него.

Гарантируется, что решение существует. Если решений несколько, выведите любое.

Пример

входной файл	*выходной файл*
6 9 1 2 2 2 3 3 3 4 15 4 5 5 5 6 6 6 1 20 1 3 7 3 6 5 4 6 8	3 37
2 1 1 2 10	1 2 5

Замечание

В первом примере ответ «6 37» также будет считаться верным.

Задача 13. Янка

Янка положил на стол N выпуклых K-гранников и N различных типов наклеек по K штук каждая. Ночью кто-то наклеил наклейки на грани, по одной на грань (на одном и том же многограннике могло оказаться несколько наклеек одного типа). Янке необходимо расставить многогранники так, чтобы наклейка каждого типа была видна ровно $K - 1$ раз (наклейка не видна, если она расположена на грани, на которой стоит многогранник). Предполагается, что такая расстановка всегда существует.

Формат входных данных

Первая строка содержит два целых числа: N ($1 < N \leqslant 500$) и K ($1 < K \leqslant 100$).

Затем идут N строк файла по K чисел в каждой. Каждая строка соответствует одному K-граннику и содержит номера наклеек, которые наклеены на 1-ю, 2-ю, ..., K-ю грани данного многогранника соответственно.

Формат выходных данных

Выведите в одну строку искомую расстановку: i-й элемент этой строки — номер наклейки на i-м многограннике, на которую его поставили (в результате в этой строке каждый тип из N типов наклеек встретится ровно один раз). Числа в строке разделены одним пробелом.

Пример

входной файл	*выходной файл*	*пояснение*
4 4 1 2 1 1 2 3 3 3 3 2 1 2 4 4 4 4	2 3 1 4	рис. 1.26

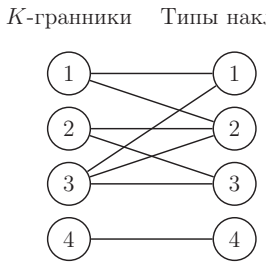


Рис. 1.26

Задача 14. Второй по длине маршрут

Задано N городов с номерами от 1 до N и сеть из M дорог с односторонним движением между ними. Каждая дорога определяется тройкой (i, j, k) натуральных чисел, где i — номер города, в котором дорога начинается, j — номер города, в котором дорога заканчивается, а k — её длина. Дороги друг с другом могут пересекаться только в конечных городах.

Все ориентированные маршруты между двумя указанными городами A и B можно упорядочить в список по неубыванию их длин. Необходимо найти один из маршрутов, который является вторым в этом упорядоченном списке (между городами A и B существует по крайней мере два ориентированных маршрута). Вывести его длину L и города, через которые он проходит (искомый маршрут по дугам может проходить несколько раз).

Формат входных данных

Первая строка содержит два целых числа N и M ($1 \leqslant N \leqslant 10\,000$, $1 \leqslant M \leqslant 100\,000$).

Затем идут M строк файла по три числа в каждой: i, j, k.

Последняя строка содержит номера городов A и B.

Формат выходных данных

Первая строка должна содержать длину второго по длине маршрута между городами A и B.

Вторая строка — это номера городов, через которые проходит второй по длине маршрут.

В случае если вторых по длине маршрутов несколько, выдать в выходной файл информацию о любом из них.

Пример

входной файл	*выходной файл*
3 3 1 2 1 2 1 1 2 3 1 1 3	4 1 2 1 2 3
3 4 1 3 10 2 1 10 2 3 20 3 2 5 2 3	20 2 3

Задача 15. Чётно-нечётная магистраль

Имеется страна с N городами. Между некоторыми парами городов построены магистрали. Длина любого прямого соединения равна единице. Движение по магистрали возможно в обе стороны. Систему магистралей будем называть *чётно-нечётной*, если каждая пара различных городов соединена маршрутом как чётной длины, так и нечётной (причём маршрут может проходить через один город несколько раз).

Необходимо определить, является ли система магистралей чётно-нечётной. Если ответ на вопрос отрицательный, то нужно найти одно из подмножеств множества городов, в котором наибольшее число элементов и которое удовлетворяет следующему условию: если какие-либо два различных города из этого подмножества соединены маршрутом, то его длина чётна.

Формат входных данных

Первая строка содержит число N городов ($1 \leqslant N \leqslant 300$).

Следующие N строк файла задают магистрали: $(i + 1)$-я строка содержит номера городов, которые связаны напрямую с городом i (если таких городов нет, то строка содержит нуль; числа в строке разделены пробелами).

Формат выходных данных

Если система магистралей является чётно-нечётной, то выведите единственную строку `YES`.

В противном случае в первой строке выведите `NO`, а во второй — мощность описанного наибольшего подмножества.

Примеры

входной файл	*выходной файл*
5 2 5 3 1 2 4 3 5 4 1	YES
3 2 1 0	NO 2
5 2 3 1 3 1 2 5 4	NO 2

Задача 16. Пересекающиеся дороги

Сеть дорог определяется следующим образом. Имеется N перекрёстков и K дорог, связывающих перекрёстки. Каждая дорога определяется тройкой чисел: двумя номерами перекрёстков и временем, требующимся на проезд по этой дороге.

Необходимо найти кратчайший маршрут машины от перекрёстка I до перекрёстка J, если на перекрёстке машина должна выждать время, равное числу пересекающихся дорог (для начального и конечного перекрёстков маршрута ожидание не требуется). Движение по дороге возможно в обоих направлениях.

Формат входных данных

В первой строке находятся два числа: N ($1 \leqslant N \leqslant 10\,000$) и K ($1 \leqslant K \leqslant 100\,000$).

Во второй строке заданы номера I и J перекрёстков, между которыми надо найти кратчайший маршрут.

Каждая из следующих K строк содержит по три числа (через пробел): номера перекрёстков, задающих очередную дорогу, и время, требующееся на проезд по этой дороге (это число является неотрицательным и не превосходит 100 000). Между любыми двумя перекрёстками — не более одной дороги.

Формат выходных данных

Выведите в первой строке минимальное время, а во второй — номера перекрёстков при движении от перекрёстка I до перекрёстка J, соответствующие найденному кратчайшему маршруту. Гарантируется, что маршрут между заданными перекрёстками существует.

Пример

входной файл	*выходной файл*
7 11 1 6 1 2 1 2 3 1 3 4 1 4 5 1 5 6 1 1 7 6 2 7 6 3 7 6 4 7 6 5 7 6 6 7 6	17 1 2 3 4 5 6

Задача 17. Домики

На прямоугольном участке расположены N домиков. Левый нижний угол участка имеет координаты $(0, 0)$, а правый верхний — $(100, 100)$. Местоположение каждого домика задаётся целочисленными координатами его нижнего левого угла. Каждый домик имеет размер 5×5 м. Стороны домиков параллельны сторонам участка. Домики отстоят друг от друга не менее чем на 1 м. Необходимо найти один из кратчайших путей от точки $(0, 0)$ до точки $(100, 100)$. При движении можно затронуть только стены домиков. Найденный путь представить в виде координат концов прямолинейных отрезков, составляющих этот путь.

Формат входных данных

Первая строка содержит число N домиков ($0 \leqslant N \leqslant 30$). Следующие N строк содержат координаты домиков: $(i + 1)$-я строка задаёт координаты левого нижнего угла i-го домика.

Формат выходных данных

В первой строке необходимо вывести длину кратчайшего пути (с абсолютной погрешностью не более 0,1), а во второй — координаты концов прямолинейных отрезков этого пути (см. пример).

Пример

входной файл	*выходной файл*	*пояснение*
6 5 5 15 13 15 20 30 20 35 40 50 50	142.2 (0;0) (10;5) (20;13) (30;25) (55;50) (100;100)	рис. 1.27

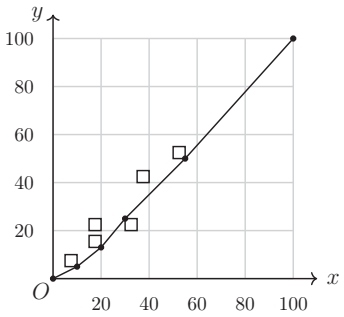


Рис. 1.27

Задача 18. Хакеры

Винни-Пух и Пятачок решили защитить компьютерную сеть от хакеров, которые выкачивали оттуда секретную информацию. Компьютерная сеть Винни-Пуха и Пятачка состояла из соединённых между собой больших ЭВМ, к каждой из которых подключалось несколько терминалов. Подключение к одной из больших ЭВМ позволяет получить информацию, содержащуюся в памяти этой ЭВМ, а также всю доступную информацию, которую может запросить данная ЭВМ.

Хакеры и раньше нападали на подобные компьютерные сети, и их тактика была известна. Поэтому Винни-Пух и Пятачок разработали специальную программу, которая помогла принять меры против нападения. Тактика хакеров: при нападении они всегда получают доступ к информации всех ЭВМ в сети. Добиваются они этого, захватывая некоторые ЭВМ в сети, чтобы от них можно было запросить информацию, которая имеется у оставшихся ЭВМ. Вариантов захвата существует множество, например захватить все ЭВМ. Однако хакеры всегда выбирают такой вариант, при котором суммарное число терминалов у захваченных ЭВМ минимально.

Необходимо определить список номеров ЭВМ, которые могут быть выбраны хакерами для захвата сети согласно их тактике.

Формат входных данных

Первая строка содержит число N ЭВМ в сети ($1 \leqslant N \leqslant 100\,000$).

Каждая из следующих N строк содержит число T_i терминалов у машины с номером i ($1 \leqslant T_i \leqslant 1000$).

Далее идут K строк (список прав на запрос) по два числа в каждой: A_j и B_j — номера машин, соединённых между собой. Каждая пара (A_j, B_j) означает, что ЭВМ с номером A_j имеет право запрашивать информацию, которая имеется у ЭВМ с номером B_j ($A_j \neq B_j$). Пары не повторяются. Завершает файл строка, содержащая два нуля.

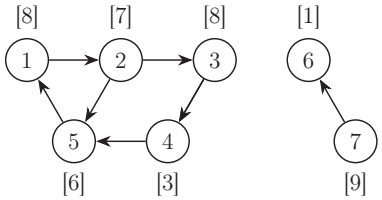


Рис. 1.28

Формат выходных данных

В первой строке выведите число захватываемых машин, во второй — их номера в произвольном порядке.

Если решений несколько, выведите любое из них.

Пример

входной файл	*выходной файл*	*пояснение*
7 8 7 8 3 6 1 9 1 2 2 3 3 4 4 5 5 1 2 5 3 4 7 6 0 0	2 4 7	рис. 1.28

Задача 19. Пирамида Хеопса

Внутри пирамиды Хеопса имеется N комнат, в которых установлены $2M$ модулей, составляющих M устройств. Каждое устройство включает два модуля, располагающихся в разных комнатах, и предназначено для перемещения между парой комнат, в которых установлены эти модули. Перемещение происходит за 0,5 условных единиц времени. В начальный момент времени модули всех устройств переходят в подготовительный режим. Каждый из модулей имеет некоторый свой целочисленный период времени, в течение которого он находится в подготовительном режиме. По истечении этого времени модуль мгновенно включается, после чего опять переходит в подготовительный режим. Устройством можно воспользоваться только в тот момент, когда одновременно включаются оба его модуля. Индиана Джонс сумел проникнуть в гробницу фараона. Обследовав её, он включил устройства и собрался уходить, но в этот момент проснулся хранитель гробницы. Теперь Джонсу необходимо как можно быстрее попасть в комнату с номером N, в которой находится выход из пирамиды. При этом из комнаты в комнату он может попадать только при помощи устройств, так как проснувшийся хранитель закрыл все двери в комнатах пирамиды.

Необходимо написать программу, которая получает на входе описание расположения устройств, их характеристик и выдаёт значение оптимального времени и последовательность устройств, которыми надо воспользоваться, чтобы попасть из комнаты 1 в комнату N за это время.

Формат входных данных

В первой строке содержится число N комнат ($2 \leqslant N \leqslant 10\,000$). Во второй строке — число M устройств ($0 \leqslant M \leqslant 100\,000$). В следующих M строках находится информация об устройствах. Каждая строка содержит числа $R_{1,i}$, $T_{1,i}$, $R_{2,i}$, $T_{2,i}$ — информацию об устройстве i:

- $R_{1,i}$ и $R_{2,i}$ — номера комнат, в которых располагаются модули устройства i;
- $T_{1,i}$, $T_{2,i}$ — периоды времени, через которые включаются соответствующие модули ($R_{1,i}$ — в момент времени $T_{1,i}$ и $R_{2,i}$ — в момент времени $T_{2,i}$) ($1 \leqslant T_{1,i}, T_{2,i} \leqslant 100\,000$, целые числа).

Формат выходных данных

Первая строка выходного файла содержит оптимальное время (с абсолютной погрешностью не более 0,1).

Во второй строке указана последовательность устройств, которыми надо воспользоваться, чтобы попасть из комнаты 1 в комнату N за минимальное время. Если решений несколько, выведите любое. Гарантируется, что решение существует.

Пример

входной файл	*выходной файл*
5 5 1 6 2 4 2 1 3 7 3 1 4 1 4 2 5 8 2 2 4 9	16.5 1 2 3 4

Задача 20. Без левых поворотов

План города представляет собой множество перекрёстков, которые соединены дорогами (движение по дороге возможно в обоих направлениях). Перекрёстки обозначаются точками на плоскости с координатами (x_1, y_1), (x_2, y_2), …, (x_n, y_n), а дорогам соответствуют пары $\{i, j\}$, где i и j обозначают номера перекрёстков, которые эта дорога соединяет. Каждая дорога является прямолинейным отрезком между соответствующими перекрёсткам точками.

Необходимо определить, можно ли проехать от перекрёстка s до перекрёстка f, не совершая левых поворотов, т. е. двигаясь только прямо или вправо. Если ответ положительный, то укажите последовательность перекрёстков.

Замечания

1. Если координаты стартовой точки (x, y), то направление движения в эту точку происходило из точки с координатами $(x, y-1)$.
2. Разворачиваться на 180° в точке запрещено.
3. Дороги пересекаются не обязательно под прямым углом. Например, какая-нибудь дорога может соединять перекрёстки с координатами $(1, 0)$ и $(2, 2)$, а другая — перекрёстки с координатами $(1, 0)$ и $(1, 1)$.

Формат входных данных

В первой строке входного файла находятся числа N и M — число перекрёстков и дорог на плане соответственно ($1 \leqslant N, M \leqslant 10\,000$). В каждой из следующих M строк содержится информация о дороге (шесть целых чисел от -10^9 до 10^9 через пробел): сначала указываются координаты перекрёстков, которые соединяет дорога, а затем — номера этих перекрёстков. В последней строке находятся номера перекрёстков s и f.

Формат выходных данных

Если проехать нельзя, то выходной файл должен содержать единственную строку с сообщением `No`. В противном случае в первой строке файла нужно вывести сообщение `Yes`, а во второй — номера перекрёстков (через пробел) в той последовательности, в которой по ним проезжали, не совершая левых поворотов (начинается строка с перекрёстка s и заканчивается перекрёстком f).

Примеры

входной файл	*выходной файл*
8 9 4 2 4 3 1 2 4 3 4 5 2 3 4 5 6 5 3 4 6 5 6 3 4 5 6 3 4 3 5 2 4 3 2 3 2 6 2 3 2 5 6 7 4 5 2 5 3 7 2 3 1 3 6 8 1 7	Yes 1 2 3 4 5 2 6 7
2 1 1 2 1 1 1 2 1 2	No

Задача 21. Кратчайший маршрут с дополнительным временем преодоления перекрёстка

План города представляет собой множество перекрёстков, соединённых дорогами (по дороге движение разрешено в обоих направлениях).

Каждая дорога задаётся номерами перекрёстков, которые она соединяет, и временем движения по ней. Между двумя различными перекрёстками может быть не более одной дороги. Дорога не может соединять перекрёсток сам с собой. Кроме того, время преодоления перекрёстка i равно qk_i, где q — заданная константа, а k_i — число дорог, подходящих к перекрёстку i.

Необходимо найти кратчайший по времени маршрут от перекрёстка s до перекрёстка f. Конечный перекрёсток f преодолевать не надо, а стартовая вершина s маршрута преодолевается как перекрёсток.

Формат входных данных

В первой строке находятся числа N и M — число перекрёстков ($1 \leqslant N \leqslant 11\,000$) и число дорог на плане города ($0 \leqslant M \leqslant 100\,000$) соответственно.

В каждой из следующих M строк файла сначала расположены номера перекрёстков, которые связывает очередная дорога, а затем время движения по ней (от 0 до 1000).

В последней строке находятся номера перекрёстков s и f ($1 \leqslant s, f \leqslant N$), а также константа q ($1 \leqslant q \leqslant 100$).

Формат выходных данных

Если проехать от перекрёстка s до перекрёстка f можно, в первой строке выведите сообщение `Yes`, а во второй строке — минимальное время движения. Если проехать нельзя, то в единственной строке выведите сообщение `No`.

Пример

входной файл	*выходной файл*
6 9 1 2 2 1 3 1 2 4 1 2 5 1 3 4 3 3 5 1 4 5 2 4 6 1 5 6 1 1 6 1	Yes 12

Задача 22. Железные и шоссейные дороги

Имеется N городов, пронумерованых числами от 1 до N. Некоторые из них соединены двусторонними дорогами, пересекающимися только в городах. Имеется два типа дорог: *шоссейные* и *железные*. Для каждой дороги известна базовая стоимость проезда по ней. Необходимо проехать из города A в город B, уплатив минимальную сумму за проезд. Стоимость проезда зависит от набора проезжаемых дорог и от способа проезда. Так, если вы подъехали к городу C по шоссейной (железной) дороге $X \to C$ и хотите ехать дальше по дороге $C \to Y$ того же типа, то должны уплатить только базовую стоимость проезда по дороге $C \to Y$. Если тип дороги $C \to Y$ отличен от типа дороги $X \to C$, то нужно уплатить базовую стоимость проезда по дороге $C \to Y$ плюс 10 % от базовой стоимости проезда по этой дороге (страховой взнос). При выезде из города A страховой взнос платится всегда.

Необходимо найти стоимость самого дешёвого маршрута, если он существует.

Формат входных данных

В первой строке находится целое число N городов ($1 \leqslant N \leqslant 1000$).

Во второй строке задано целое число M дорог ($0 \leqslant M \leqslant 1\,000\,000$).

В каждой из следующих M строк находятся четыре числа x, y, t, p (разделённые пробелом), где x и y — номера городов, которые соединяет дорога, t — тип дороги (0 — шоссейная, 1 — железная), p — базовая стоимость проезда по ней (p — действительное число, $0 < p \leqslant 1000$).

В последней строке задаются номера A и B начального и конечного городов.

Формат выходных данных

Если маршрута не существует, то в единственной строке выведите `No`.

Если же маршрут существует, то в первой строке выведите `Yes`, а во второй — стоимость проезда по самому дешёвому маршруту (с точностью до двух знаков после точки).

Пример

входной файл	*выходной файл*
5 5 1 5 1 10 1 3 1 10 1 4 0 30 1 2 0 1000 4 3 0 10 5 2	Yes 1062.00

Задача 23. Кратчайший путь в зависимости от направления поворота

Заданы декартовы координаты N перекрёстков города, которые пронумерованы числами от 1 до N. На каждом перекрёстке имеется светофор. Некоторые перекрёстки соединены дорогами с двусторонним (правосторонним) движением, пересекающимися только на перекрёстках. Для каждой из M дорог известно неотрицательное время, требуемое для проезда по ней от одного перекрёстка до другого.

Необходимо проехать от перекрёстка A до перекрёстка B за минимальное время.

Время проезда зависит от набора проезжаемых дорог и от времени ожидания на перекрёстках. Так, если вы подъехали от перекрёстка X к перекрёстку C по дороге $X \to C$ и хотите ехать дальше по дороге $C \to Y$, то время ожидания на перекрёстке C зависит от того, поворачиваете вы налево или нет. Если вы поворачиваете налево, то время ожидания равно DK, где число D — количество дорог, пересекающихся на перекрёстке C, а число K — некоторая константа. Если вы не поворачиваете налево, то время ожидания на перекрёстке равно нулю. Разворот на перекрёстке считается поворотом налево.

Замечание

Если первоначально машина находится на перекрёстке с координатами (x, y), то предполагается, что она приехала в эту точку из точки с координатами $(x, y - 1)$.

Формат входных данных

В первой строке входного файла находятся натуральные числа N, M и K ($N \leqslant 1000$, $M \leqslant 1000$, $K \leqslant 1000$).

В каждой из следующих N строк — пара целых чисел x и y (разделённых пробелом), которые задают координаты перекрёстка города ($|x|, |y| < 1000$).

В каждой из M следующих строк — по три числа p, r и t (разделённые пробелами), где p и r — номера перекрёстков, которые соединяет дорога, а натуральное число t ($t \leqslant 1000$) — время проезда по ней.

В последней строке файла находятся номера A и B начального и конечного перекрёстков соответственно.

Формат выходных данных

Если пути не существует, то единственная строка выходного файла должна содержать сообщение No.

Если путь существует, то в первой строке нужно вывести сообщение `Yes`, а во второй — натуральное число t — время проезда по самому быстрому маршруту.

Пример

входной файл	*выходной файл*
6 5 5 4 2 4 0 2 4 4 6 6 4 0 4 1 2 0 3 4 1 1 5 1 1 3 1 6 3 1 2 6	Yes 16

Задача 24. Уличная гонка

План улицы для гонки задан в виде точек, помеченных числами от 0 (старт) до $N-1$ (финиш), и стрелок, которые их соединяют. Стрелками представлены улицы с односторонним движением. Участники гонки передвигаются от точки к точке по улицам в направлении стрелок. В каждой точке участник может выбрать любую из исходящих стрелок.

Назовём план улицы *хорошим*, если он обладает следующими свойствами:

- каждая точка плана может быть достигнута со старта;
- финиш может быть достигнут из любой точки плана;
- у финиша нет исходящих стрелок.

Для достижения финиша участник не обязательно должен пройти через все точки. Однако некоторые точки невозможно обойти. Назовём их *неизбежными*.

Предположим, что гонка проводится на протяжении двух последовательных дней. Для этой цели план должен быть разбит на два «хороших» плана, по одному на каждый день. В первый день стартом является точка 0, а финишем служит некоторая «точка разбиения». Во второй день старт находится в этой «точке разбиения», а финиш — в точке N.

Точка S является «точкой разбиения» для «хорошего» плана C, если:

- S отличается от старта и финиша плана C;

• план разбивается ею на два «хороших» плана без общих стрелок (т. е. между планами нет стрелок, их соединяющих) и с единственной общей точкой S.

Для заданного «хорошего» плана необходимо:

1) определить множество «неизбежных точек», которые должны посетить все участники гонки (за исключением старта и финиша);

2) определить множество всех возможных «точек разбиения».

Формат входных данных

В первой строке задано число N точек ($0 \leqslant N \leqslant 50$).

Следующие N строк содержат конечные точки стрелок, исходящих соответственно из точек с номерами от 0 до $N-1$. Каждая из этих строк заканчивается числом -2.

Формат выходных данных

В первой строке выведите число «неизбежных точек» в заданном плане, затем — их номера в порядке возрастания.

Во второй строке — число «точек разбиения» в заданном плане, затем — их номера в порядке возрастания.

Если каких-либо точек нет, то соответствующая строка должна содержать число 0.

Пример

входной файл	*выходной файл*	*пояснение*
10 1 2 -2 3 -2 3 -2 5 4 -2 6 4 -2 6 -2 7 8 -2 9 -2 5 9 -2 -2	2 3 6 1 3	рис. 1.29

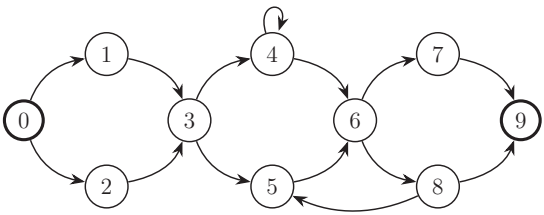


Рис. 1.29

Задача 25. Прогулка

Хозяин вышел на прогулку с собакой. Известно, что путь хозяина представляет собой ломаную линию, координаты отрезков ломаной заданы: (X_i, Y_i), $i = 1, \ldots, N$. Точка (X_1, Y_1) — начальная точка ломаной линии, а точка (X_N, Y_N) — конечная точка ломаной. Хозяин во время прогулки двигается от стартовой точки по отрезкам ломаной линии и заканчивает путь в конечной точке ломаной. У собаки есть свои любимые места (координаты любимых мест заданы: (X'_j, Y'_j), $j = 1, \ldots, M$), которые собака хотела бы посетить. В то время как хозяин проходит один отрезок ломаной, собака может посетить только одно из своих любимых мест. В начальной и конечной точке ломаной, а также когда хозяин проходит каждую точку изгиба ломаной, собака обязана подбежать к хозяину. Известно, что скорость собаки в два раза выше скорости хозяина.

Необходимо определить, какое наибольшее число своих любимых мест сможет посетить собака за время прогулки.

Формат входных данных

Первая строка содержит два числа: N — число точек ломаной (включая начальную и конечную точки пути) и M — число любимых мест собаки ($1 \leqslant N \leqslant 100$, $0 \leqslant M \leqslant 100$).

В следующих N строках идут координаты точек ломаной: сначала координата x, а затем через пробел — координата y и т. д. (координаты точек ломаной следуют в той последовательности, в которой они соединяются отрезками, начиная от начальной точки и заканчивая конечной).

В следующих M строках — координаты любимых мест собаки: сначала координата x, а затем через пробел — координата y и т. д.

Координаты — целые числа, по модулю не превосходящие 10 000. Все точки попарно различны.

Формат выходных данных

Выведите в единственной строке два числа (через пробел). Первое — наибольшее количество мест, которые сможет посетить собака за время прогулки (это и все точки ломаной, и некоторые любимые места собаки),

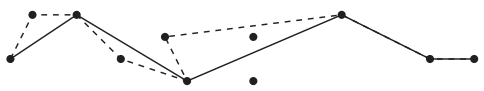


Рис. 1.30

второе — наибольшее число любимых мест собаки, которые она смогла посетить.

Пример

входной файл	*выходной файл*	*пояснение*
6 5 1 2 4 4 9 1 16 4 20 2 22 2 2 4 6 2 8 3 12 3 12 1	9 3	рис. 1.30

Задача 26. Сборка прибора

Прибор спроектирован таким образом, что он будет собираться из отдельных узлов, причём каждый узел уникален. Сами узлы, в свою очередь, могут требовать предварительной сборки. Для сборки каждого узла необходимо, чтобы все узлы, комплектующие его, уже были собраны. Узлы, не требующие сборки, обязательно тестируются на работоспособность. Собираемые узлы тестировать не требуется. На сборку одного узла или на его тестирование тратится один день. Готовый узел должен быть помещён на склад и может быть взят со склада только тогда, когда он необходим для сборки очередного узла или самого прибора. Хранение узла на складе в течение одного дня требует оплаты в размере одной условной денежной единицы.

Необходимо организовать сборку таким образом, чтобы плата за аренду склада была минимальной.

Формат входных данных

Первая строка содержит общее число N узлов (узлы пронумерованы от 1 до N, $N \leqslant 50$).

Вторая строка — номер узла, который надо собрать.

Каждая из следующих N строк файла содержит информацию об отдельном узле: L — его номер, C — количество его комплектующих. Затем в этой же строке идут C чисел, которые задают номера комплектующих узлов для узла L. Разделителем чисел в пределах строки является символ «:» (двоеточие).

Формат выходных данных

Выведите в первой строке плату за аренду, а во второй — после-

довательность, в которой собираются узлы (если вариантов сборки несколько, нужно вывести ту, которая лексикографически меньше).

Пример

входной файл	*выходной файл*
6 5 1:1:6 2:0 3:2:1:4 4:0 5:2:2:3 6:0	7 6 1 4 3 2 5

Задача 27. Скрудж Мак-Дак

Скрудж Мак-Дак решил сделать прибор для управления самолётом. Как известно, положение штурвала зависит от состояния входных датчиков, но эта функция довольно сложна. Его механик делает устройство, вычисляющее эту функцию за несколько этапов с использованием промежуточной памяти и вспомогательных функций. Для вычисления каждой из функций требуется, чтобы в ячейках памяти уже находились вычисленные параметры (которые являются значениями вычисленных функций), необходимые для её вычисления. Вычисление функции без параметров может производиться в любое время. После вычисления функции ячейки могут быть использованы повторно (хотя бы для записи результата вычисленной функции). Структура вызова функций такова, что каждая функция вычисляется не более одного раза и любой параметр используется не более одного раза. Любой параметр есть имя функции. Поскольку Скрудж не хочет тратить лишних денег на микросхемы, он поставил задачу минимизировать память прибора.

По заданной структуре вызовов функций нужно определить минимально возможный размер памяти прибора и указать последовательность вычисления функций.

Первоначально необходимо вычислить функции, которые принимают на вход наибольшее число параметров. При неоднозначности выбора функции требуется вычислять в лексикографическом порядке.

Формат входных данных

Первая строка содержит общее число N функций ($1 \leqslant N \leqslant 10\,000$). Вторая строка — имя функции, которую необходимо вычислить (имя функции есть натуральное число от 1 до N). Каждая из следующих N строк содержит имя функции, число параметров и список имён параметров.

Формат выходных данных

В первой строке выведите минимальное число ячеек памяти, во второй — последовательность вычисления функций.

Пример

входной файл	*выходной файл*
5 2 2:1:4 1:1:3 5:1:1 3:0 4:1:5	1 3 1 5 4 2

Задача 28. Заправочные станции

Имеется N городов, соединённых M двусторонними дорогами. Для каждой дороги задана её протяжённость в километрах. Машина может поворачивать (изменять направление движения) только в городах. Машина имеет бак вместимостью Z литров бензина, и для неё задан расход бензина X литров на один километр.

В некоторых городах имеются заправочные станции. У каждой заправочной станции задана своя стоимость дозаправки S_i до полного бака (т. е. вне зависимости от того, сколько бензина осталось в баке машины, на заправке доливается бензин до полного бака, и за это платится фиксированная сумма денег). Машина может (но не обязана) заправиться только в том случае, если её бак заполнен менее чем наполовину.

Необходимо определить самый дешёвый маршрут из города A в город B, если первоначально бак машины полон.

Формат входных данных

В первой строке записаны два целых числа N ($2 \leqslant N \leqslant 300$) и M ($1 \leqslant M \leqslant 5000$).

Во второй строке заданы два целых числа Z ($1 \leqslant Z \leqslant 10\,000$) и X ($1 \leqslant X \leqslant 100$).

В третьей строке указаны числа A и B ($1 \leqslant A, B \leqslant N$ и $A \neq B$).

Далее строка из N неотрицательных целых чисел, i-е число — стоимость S_i дозаправки ($1 \leqslant S_i \leqslant 10\,000$) или число 0, если в этом городе заправки нет.

Далее M строк занимают описания каждой дороги: первые два числа — номера городов, связанных двусторонней дорогой, и третье число — её протяжённость в километрах (целое число от 1 до 10 000).

Формат выходных данных

Первая строка выходного файла должна содержать слово `Yes`, если решение существует, или слово `No`, если оно отсутствует. Если решение найдено, то во второй строке должны следовать номера городов в порядке их посещения, начиная с номера города A и заканчивая номером города B. Если вы заправляетесь в каком-либо городе, в котором есть заправка, то номер этого города нужно вывести со знаком минус. Если решение неоднозначно, выведите любое из них.

Пример

входной файл	*выходной файл*
4 5 10 2 1 4 9 0 7 0 1 3 2 1 2 2 2 3 1 3 4 5 1 4 7	Yes 1 2 -3 4

Задача 29. Слова

Задана последовательность слов. Игра заключается в том, что участвующие по очереди называют слова из заданной последовательности. Правило, по которому называется слово: если названо некоторое слово, то следующий игрок может назвать слово, начинающееся с буквы, на которую заканчивается предыдущее слово и которое ещё не было названо.

Необходимо определить, можно ли в процессе игры выстроить цепочку из всех слов, причём последнее слово должно заканчиваться на ту букву, с которой начиналось первое.

Если выстроить цепочку возможно, необходимо указать один из вариантов.

Например, для последовательности слов

`terminator`, `department`, `epic`, `cat`, `rapid`, `toothpaste`

требуемая цепочка слов имеет вид

`cat`, `terminator`, `rapid`, `department`, `toothpaste`, `epic`.

Формат входных данных

В первой строке находится число N слов (от 1 до 131 000).

Следующие N строк содержат слова (по одному слову в каждой строке).

Слова состоят из строчных букв английского алфавита.

Размер входного файла не превосходит одного мебибайта.

Формат выходных данных

Если цепочку составить нельзя, то выведите сообщение `No`.

Если цепочку можно составить из всех слов, то в первой строке выведите сообщение `Yes`, а далее — непосредственно саму цепочку (по одному слову в строке).

Пример

входной файл	*выходной файл*
6 terminator department epic cat rapid toothpaste	Yes cat terminator rapid department toothpaste epic
5 cat orb ear pool mask	No

Задача 30. Цепочка из домино

На столе выложены доминошки. Доминошка состоит из двух частей, и на каждой части изображено некоторое число точек (от 0 до 6). Доминошки могут выкладываться в ряд одна за другой по следующему правилу: число точек на примыкающих частях соседних в ряду доминошек должно совпадать.

Необходимо определить, можно ли выстроить цепочку, в которой каждая доминошка встречается ровно один раз, причём число точек на свободных концах доминошек, которые являются крайними в цепочке, должно совпадать.

Формат входных данных

В первой строке находится число N доминошек ($2 \leqslant N \leqslant 10\,000$).

Следующие N строк содержат данные о доминошках (два числа через пробел): число точек на верхней и нижней частях доминошки.

Формат выходных данных

Если можно составить цепочку из всех доминошек, то первая строка файла должна содержать сообщение `Yes`.

Если цепочку составить нельзя, то в единственной строке выходного файла нужно вывести сообщение No.

Пример

входной файл	*выходной файл*
4 3 5 2 3 5 1 1 2	Yes
2 0 0 0 1	No

Задача 31. Степенная последовательность

Назовём степенной последовательностью неориентированного графа $G = (V, E)$ упорядоченный по невозрастанию список степеней его вершин.

В рамках теории графов изучаются разнообразные свойства степенных последовательностей. Простейшее свойство — сумма элементов степенной последовательности всегда чётна, так как любое ребро соединяет две вершины и поэтому учитывается в сумме дважды.

По заданному графу постройте его степенную последовательность.

Формат входных данных

В первой строке записаны два целых числа N и M — число вершин ($1 \leqslant N \leqslant 100\,000$) и число рёбер графа ($1 \leqslant M \leqslant 100\,000$).

В последующих M строках задаются рёбра путём указания для каждого ребра пары номеров вершин, которые инцидентны этому ребру.

Вершины пронумерованы числами от 1 до N. Гарантируется, что в графе нет петель и кратных рёбер.

Формат выходных данных

Выведите степенную последовательность графа, разделяя числа пробелами.

Пример

входной файл	*выходной файл*
5 4 1 2 3 4 3 2 2 4	3 2 2 1 0

Задача 32. Отрезки

Пусть на плоскости с евклидовой метрикой задано N красных и N синих точек (красные точки имеют номера от 1 до N, а синие — от $N+1$ до $2N$). Существует N отрезков, соединяющих эти точки таким образом, что каждый отрезок соединяет точки различного цвета и каждая точка — концевая только для одного отрезка.

Необходимо определить, является ли заданное соединение минимальным по длине (сумма длин всех отрезков) среди всех возможных соединений, удовлетворяющих заданным свойствам.

Формат входных данных

В первой строке находится натуральное число N ($1 \leqslant N \leqslant 100$).

Следующие N строк файла содержат по два целых числа: координаты красных точек (в первой строке — координаты красной точки с номером 1, во второй — с номером 2 и т. д.).

Затем идут N строк файла, в каждой из которых указаны два целых числа: координаты синих точек (первая из этих строк — координаты синей точки с номером $N+1$, вторая — с номером $N+2$ и т. д.).

Каждая из следующих N строк файла задаёт отрезки соединения и содержит по два числа, определяющих начальные и конечные номера точек отрезков.

Координаты красных и синих точек по модулю не превосходят 1000.

Формат выходных данных

Выведите в единственной строке сообщение `Yes`, если заданное соединение является минимальным по длине. Если оно таковым не является, то выведите сообщение `No`.

Пример

входной файл	*выходной файл*
3 0 1 0 2 0 3 1 3 1 2 1 1 1 6 2 4 3 5	No
1 100 100 200 200 1 2	Yes

Задача 33. Трубопроводы

Имеется сеть трубопроводов по перекачке нефти, при этом у участков трубопроводов есть пропускная способность (тонн/час) и стоимость транспортировки (транзита) тонны нефти по участку.

Необходимо организовать перекачку максимального количества тонн нефти при минимальных суммарных затратах на транзит из пункта A в пункт B. Если два пункта X и Y связаны трубопроводом, то по этому трубопроводу можно перекачивать нефть как из X в Y, так и из Y в X.

Формат входных данных

В первой строке находятся четыре натуральных числа, задающих сеть трубопроводов: число N пунктов, число M трубопроводов, номера пунктов A и B ($1 < N \leqslant 100$, $0 \leqslant M \leqslant N(N-1)/2$).

Каждая из следующих M строк содержит по четыре целых числа, задающих информацию о некотором трубопроводе: два номера пунктов, связанных трубопроводом, его пропускная способность P_i и стоимость S_i транспортировки по нему ($0 \leqslant P_i \leqslant 100$, $0 \leqslant S_i \leqslant 1000$).

Формат выходных данных

В первой строке выведите максимальное количество тонн нефти, которое можно перекачать из пункта A в пункт B.

Во второй строке — минимальные затраты на транзит.

Пример

входной файл	*выходной файл*
6 10 1 6 1 2 4 2 1 3 6 3 2 3 1 5 2 4 3 1 2 5 1 4 3 5 3 2 3 4 2 3 4 5 1 1 4 6 5 3 5 6 6 1	9 60

Задача 34. Таксист

Имеется N городов, связанных M дорогами (нумерация дорог и городов начинается с единицы). Движение по дорогам осуществляется только в одном направлении, и дороги пересекаются только в городах. Известна длина каждой дороги.

Необходимо найти все дороги, по которым потенциально может проехать таксист таким образом, чтобы длина его маршрута отличалась

не более чем на величину K от длины кратчайшего маршрута из 1-го города в N-й.

Формат входных данных

Первая строка файла содержит числа N, M и K ($2 \leqslant N \leqslant 10\,000$, $1 \leqslant M \leqslant 100\,000$, $0 \leqslant K \leqslant 10\,000$). Каждая из последующих M строк описывает дорогу: начальный город, конечный город и её длина (целое число не более 10 000).

Формат выходных данных

Первая строка содержит число L дорог, которые таксист потенциально может использовать.

Каждая из следующих L строк — одно число: номер потенциальной дороги. Номера дорогам присваиваются в соответствии с тем, в какой последовательности они были введены, а сами дороги следуют в порядке возрастания их номеров.

Пример

входной файл	*выходной файл*
4 5 1 1 2 1 1 3 4 2 3 1 2 4 3 3 4 1	4 1 3 4 5

Задача 35. Стрельба

На соревнованиях по стрельбе каждый участник будет стрелять в цель, которая представляет собой прямоугольник, разделённый на квадраты. Цель содержит $R \cdot C$ квадратов, расположенных в R строках и C столбцах. Квадраты выкрашены в белый или чёрный цвет. В каждом столбце находится ровно 2 белых и $R - 2$ чёрных квадрата. Строки пронумерованы от 1 до R сверху вниз, а столбцы — от 1 до C слева направо. Стрелок имеет ровно C стрел. Последовательность из C выстрелов называется *корректной*, если в каждом столбце поражён ровно один белый квадрат, а в каждой строке — не менее одного белого квадрата. Необходимо проверить, существует ли корректная последовательность выстрелов, и если да, то найти одну из них.

Формат входных данных

Первая строка содержит два целых числа R и C ($2 \leqslant R \leqslant C \leqslant 1000$). Эти числа определяют количество строк и столбцов. Каждая из следующих C строк в блоке содержит два натуральных числа, разделённых

пробелом. Числа в $(i+1)$-й строке определяют номера строк, где расположены белые квадраты в i-м столбце.

Формат выходных данных

Выведите последовательность из C чисел, соответствующих корректной последовательности выстрелов: i-й элемент последовательности — номер строки, белую клетку которой поразил выстрел i. Если такой последовательности не существует, то выведите сообщение No.

Пример

входной файл	*выходной файл*
4 4 2 4 3 4 1 3 1 4	2 3 1 4
3 3 2 3 2 3 2 3	No

Задача 36. Инкассаторы

Когда Макс Крейзи закончил финансовый колледж, он стал управляющим городского банка. Уже с первых дней работы Макс столкнулся с одной неразрешимой для него проблемой. В стране, где живёт Макс, есть N городов. Некоторые из них связаны двусторонними дорогами, пересекающимися только в городах. Раз в месяц инкассаторы Макса должны доставить деньги в K сберкасс. Все K сберкасс находятся в разных городах. Банк Макса небогат и имеет одну машину для перевозки денег. Вам необходимо помочь Максу составить маршрут минимальной длины, который начинается в городе L (в этом городе находится банк Макса), проходит по всем K городам, где располагаются нужные сберкассы, и заканчивается также в городе L.

Формат входных данных

В первой строке через пробел указаны три числа: N, M и K, где N — общее число городов в стране, M — общее число двусторонних дорог, K — число сберкасс, которые должны посетить инкассаторы ($1 < N \leqslant 100$, $0 < M \leqslant N(N-1)/2$, $0 < K < 18$, $K < N$).

Во второй строке через пробел идут K чисел — номера городов, где находятся сберкассы.

Следующие M строк содержат описание дорог. В каждой строке тремя целыми числами X, Y и S описывается одна дорога, причём X

и Y — номера городов, связанных дорогой, и S — длина этой дороги ($0 < S \leqslant 50\,000$, $1 \leqslant X, Y \leqslant N$).

В последней строке дано число L — номер города, где находится банк Макса ($1 \leqslant L \leqslant N$).

Формат выходных данных

Единственная строка должна содержать набор чисел — номеров городов искомого маршрута в порядке обхода.

В случае когда такого маршрута не существует, выведите одно слово — NO.

Примеры

входной файл	*выходной файл*
4 6 3 2 4 3 1 2 10 2 4 3 1 3 5 1 4 4 3 2 7 4 3 8 1	1 3 2 4 1
3 1 1 3 1 2 100 2	NO

Задача 37. Город *C*

Задана система дорог, определяемая набором пар городов. Каждая такая пара $\{ i, j \}$ указывает, что из города i можно проехать в город j и в обратном направлении.

Необходимо определить, можно ли проехать из заданного города A в заданный город B таким образом, чтобы посетить город C, не проезжать ни по какой дороге более одного раза и не заезжать ни в какой город более одного раза.

Формат входных данных

В первой строке находится число N городов (города нумеруются от 1 до N, $1 \leqslant N \leqslant 5000$).

Во второй строке — целое число M дорог ($0 \leqslant M \leqslant 10\,000$).

Далее в каждой из M строк — пара номеров городов, соединённых некоторой дорогой.

В последней, $(M + 3)$-й, строке находятся номера городов A, B и C.

Формат выходных данных

В единственной строке выведите сообщение `Yes`, если искомая цепь существует, или `No` в противном случае.

Примеры

входной файл	*выходной файл*
3 2 1 2 2 3 1 3 2	Yes
3 2 1 3 2 3 1 3 2	No
6 6 1 3 2 3 3 5 3 4 5 6 4 6 1 2 6	No

Задача 38. Расселение

Руководство фирмы ООО «Вектор», состоящей из двух отделов, решило на празднование годовщины основания организовать поездку всех сотрудников в санаторий. Санаторий располагает одноместными и двухместными номерами.

Для укрепления корпоративного духа было принято решение селить в двухместные номера сотрудников из разных отделов. Если в одном из отделов больше людей, чем в другом, то тех, кому не хватило пары, расселяют в одноместные номера. В силу разного возраста сотрудников вводится показатель недовольства, равный разности в возрасте заселяемых в один номер (вычисляется для каждого номера, а не сотрудника). Для тех, кто попадает в одноместный номер, он равен половине возраста.

Формат входных данных

В первой строке задаются возрасты сотрудников первого отдела, разделённые пробелами.

Во второй строке — возрасты сотрудников второго отдела.

Возраст сотрудника должен лежать в интервале от 18 до 60.

Формат выходных данных

Единственная строка должна содержать наименьший суммарный показатель недовольства (вычисляется как сумма показателей всех номеров). Число должно иметь одну цифру после точки.

Пример

входной файл	*выходной файл*
20 24 18 42	20.0

Задача 39. Сеть дорог

В Байтленде было решено построить сеть автомобильных дорог с односторонним движением. Каждая дорога должна соединять только два города. Известно, какие дороги возможно построить, а также пропускная способность для каждой из них и стоимость одной единицы пропускной способности по конкретной дороге (удельная стоимость). Задача строителей — построить дороги так, чтобы максимизировать пропускную способность между городами X и Y и затратить на это минимальную сумму денег. Первоначально предложен некоторый вариант строительства дорог.

Определить, оптимален ли данный выбор.

Формат входных данных

В первой строке находятся четыре целых числа, задающих сеть дорог: N — число пунктов, M — число дорог, начальный и конечный города X и Y. Каждая из следующих M строк содержит по четыре числа, задающих информацию о дороге, которую можно построить: номера начального и конечного городов дороги, её пропускная способность и стоимость одной единицы пропускной способности по данной дороге. Остальные строки содержат информацию об одном из возможных вариантов сети дорог: номеров начального и конечного городов, которые соединяет дорога.

Формат выходных данных

Если предложенная сеть дорог является оптимальной, то в первой строке выведите `Yes`; во второй — максимальную пропускную способность между городами X и Y; в третьей — минимальную стоимость, которая будет заплачена за провоз полученного значения пропускной способности.

Если предложенная сеть дорог не является оптимальной, то в первой строке выведите `No`; во второй — максимальную пропускную способность

между городами X и Y в оптимальной сети дорог; в третьей — минимальную стоимость, которая будет заплачена за провоз полученного значения пропускной способности для оптимальной сети дорог; в четвёртой — максимальную пропускную способность между городами X и Y в предложенной сети дорог; в пятой строке — минимальную стоимость, которая будет заплачена за провоз полученного значения пропускной способности для предложенной сети дорог.

Пример

входной файл	*выходной файл*
6 8 1 6 1 2 14 10 2 3 11 6 2 4 5 20 2 6 3 15 3 4 5 9 3 5 6 9 4 6 6 8 5 6 6 9 1 2 2 3 2 4 2 6 3 5 4 6 5 6	No 14 444 14 469

Задача 40. Боеприпасы

На складе хранится N единиц боеприпасов. Было решено их уничтожить в течение N дней по единице в день. Для каждой единицы боеприпаса известно, в какие дни она может быть уничтожена, и задана степень риска при уничтожении этой единицы в указанный день. Требуется уничтожить все боеприпасы так, чтобы суммарный риск был минимальным.

Формат входных данных

В первой строке задаётся число дней N, равное числу боеприпасов ($1 \leqslant N \leqslant 200$).

В следующих N строках — данные о риске уничтожения боеприпасов: в каждой строке сначала задаётся число K_i ($1 \leqslant K_i \leqslant N$) — число дней, в которые соответствующая единица боеприпасов может быть уничтожена. Затем идут $2K_i$ чисел, описывающих риск уничтожения этой единицы боеприпасов: на нечётных позициях — номера дней от 1 до N, а на чётных — соответствующие им степени риска (целые числа в пределах от 0 до 200).

Формат выходных данных

В первой строке выведите минимальную суммарную степень риска уничтожения всех боеприпасов.

Во второй строке — N чисел: i-е число означает, какой боеприпас в i-й день должен быть уничтожен.

Если же все боеприпасы уничтожить невозможно, то в единственной строке выведите -1.

Пример

входной файл	*выходной файл*
3 2 2 1 3 4 2 1 7 3 10 2 1 1 2 4	12 3 1 2

Замечание

Во второй день с риском 1 или в третий день с риском 4 можно уничтожить боеприпас 1. В первый день с риском 7 или в третий день с риском 10 можно уничтожить боеприпас 2. В первый день с риском 1 или во второй день с риском 4 можно уничтожить боеприпас 3. Безопаснее всего уничтожать боеприпасы в порядке 3, 1, 2 с суммарным риском 12.

Задача 41. Граф? Мультиграф? Псевдограф?

Из классического определения неориентированного графа следует, что в нём нет *петель* (рёбер, соединяющих вершину с ней самой) и *кратных рёбер* (рёбер, соединяющих одну и ту же пару вершин).

Обобщённое понятие графа — *мультиграф*, в котором допускается наличие кратных рёбер. Дальнейшее обобщение понятия графа — *псевдограф*, в котором могут присутствовать как кратные рёбра, так и петли.

Таким образом, исходя из определений любой граф является мультиграфом, а любой мультиграф — псевдографом.

Проверьте, является ли заданная структура из N вершин и M рёбер графом, мультиграфом или псевдографом.

Формат входных данных

В первой строке записаны два целых числа N и M — количество вершин ($1 \leqslant N \leqslant 500$) и количество рёбер ($0 \leqslant M \leqslant 100\,000$). Вершины пронумерованы целыми числами от 1 до N.

В следующих M строках описаны рёбра: в каждой строке задаётся через пробел пара вершин u и v ($1 \leqslant u, v \leqslant N$), соединённых ребром.

Формат выходных данных

Выведите три строки с ответами на три вопроса соответственно: верно ли, что на входе представлен:

- граф;
- мультиграф;
- псевдограф?

В случае утвердительного ответа на вопрос выведите `Yes`, иначе — `No`.

Пример

входной файл	*выходной файл*	*пояснение*
3 3 1 2 2 3 1 2	No Yes Yes	рис. 1.31

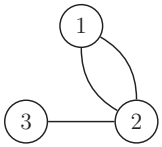


Рис. 1.31

Задача 42. Экскурсия

Экскурсия по Национальному парку Байтленда заключается в следующем: имеются острова, соединённые мостами. Группа собирается в начальном пункте, а затем передвигается по мостам до конечного пункта, причём каждый человек может идти по индивидуальному маршруту. Поскольку разные люди при этом осматривают различные достопримечательности, то стоимость экскурсии также отличается. По каждому из мостов принято ходить только в одном направлении, чтобы посетители не мешали друг другу. Известна некоторая стоимость прохождения каждого моста для одного человека, а стоимость экскурсии для человека равна сумме стоимостей для мостов, которые он собирается пройти. Группа является очень дисциплинированной, поэтому если кто-то попадает на остров, то он ждёт, пока не соберутся все, кто должен пройти через этот остров. Для каждого моста также известно максимальное число человек, которое он может одновременно принять по технике безопасности.

Если мост соединяет острова A и B, то можно выбрать любое из направлений с A на B или с B на A, но только одно. Так, если хотя бы

один из посетителей прошёл по мосту в направлении с A на B, то и все остальные, кто будет проходить по этому мосту, должны проходить этот мост в этом направлении.

Составьте такой план экскурсии из начального пункта в конечный, при котором группа была бы максимальной, а общая стоимость экскурсии для всей группы — минимальной из всех возможных при данном размере группы.

Формат входных данных

Первая строка содержит три целых числа, включающих общую информацию о системе мостиков: N — число островов ($2 \leqslant N \leqslant 30$), S — номер острова, откуда начинаются все экскурсии ($1 \leqslant S \leqslant N$), F — номер острова, на котором группа должна собраться после экскурсии ($1 \leqslant F \leqslant N$). Гарантируется, что начальный и конечный острова не совпадают.

Каждая из следующих строк содержит по четыре целых числа, задающих информацию о каждом мостике: номера соединённых этим мостиком островов (от 1 до N), максимальное число M человек, которые могут пройти по данному мосту ($0 \leqslant M \leqslant 50$), и стоимость C его прохождения ($0 \leqslant C \leqslant 1000$). Между парой островов существует не более одного моста. Никакой мост не соединяет остров с самим собой.

Формат выходных данных

Первая строка должна содержать максимальное число человек, которые смогут пройти по парку. Вторая строка — минимальную стоимость этой экскурсии.

Примеры

входной файл	*выходной файл*
7 1 7 1 2 10 43 1 3 7 34 1 4 20 35 2 3 14 42 2 5 14 36 2 6 18 39 3 4 9 45 3 5 12 29 4 6 20 48 4 7 19 40 5 6 17 47 6 7 37 39	37 3800
3 1 3 2 1 1 1 2 3 1 1 1 3 1 1	2 3

Задача 43. Машина времени

Между N населёнными пунктами совершаются пассажирские рейсы на машинах времени. В момент времени 0 вы находитесь в пункте A. Вам дано расписание рейсов. Требуется оказаться в пункте B как можно раньше (т. е. в наименьший возможный момент времени). При этом разрешается делать пересадки с одного рейса на другой. Если вы прибываете в некоторый пункт в момент времени T, то можете уехать из него любым рейсом, который отправляется из этого пункта в момент времени T или позднее (но не раньше).

Формат входных данных

Первая строка содержит целое число N ($1 \leqslant N \leqslant 1000$) населённых пунктов. Вторая строка — два числа A и B — номера начального и конечного пунктов. Третья строка — число K рейсов ($0 \leqslant K \leqslant 1000$).

Следующие K строк содержат описания рейсов, по одному на строке. Каждое описание представляет собой четвёрку целых чисел. Первое число каждой четвёрки задаёт номер пункта отправления, второе — время отправления, третье — пункт назначения, четвёртое — время прибытия. Номера пунктов — натуральные числа в диапазоне от 1 до N. Пункт назначения и пункт отправления могут совпадать. Время измеряется в некоторых абсолютных единицах и задаётся целым числом, по модулю не превышающим 10^9.

Поскольку рейсы совершаются на машинах времени, то время прибытия может быть как больше времени отправления, так и меньше или равным ему. Гарантируются такие входные данные, что добраться из пункта A в пункт B всегда можно.

Формат выходных данных

Выведите минимальное время, в которое вы сможете оказаться в пункте B.

Пример

входной файл	*выходной файл*
2 1 1 2 1 1 2 10 1 10 1 9	0
1 1 1 3 1 3 1 -5 1 -5 1 -3 1 -4 1 -10	-10

Задача 44. Максимальное число коней

Задана шахматная доска размера $N \times M$, часть клеток которой могут занимать чёрные фигуры. Необходимо на свободные клетки доски расставить максимальное число белых коней (количество белых коней не ограничено), чтобы они не били друг друга.

Формат входных данных

В первой строке находится число N строк доски, во второй строке — число M столбцов ($1 \leqslant N, M \leqslant 200$). В каждой из последующих строк — по два числа x и y, разделённых пробелом, которые являются координатами чёрной фигуры (x — номер строки, y — столбца). Координаты верхнего левого угла доски — $(1, 1)$.

Формат выходных данных

Выведите максимальное число белых коней.

Пример

входной файл	*выходной файл*
7 7 1 1 2 2 4 4 5 6	23

Задача 45. Военный поход

В одном феодальном государстве средневековой Европы было N городов и M дорог, каждая из которых соединяла некоторые два города. Каждая дорога принадлежала правителю одного из городов (не обязательно из тех, которые она соединяла). Однажды правитель города S решил объявить войну правителю города T. Перед ним сразу же встала нелёгкая задача: как довести армию до города T.

Правитель каждого города требует плату за проход войск через его город. Кроме того, правитель города S может перемещать войска только по дорогам, которые принадлежат ему. При этом он может купить любую дорогу у её владельца за определённую плату (даже если владелец — правитель города T). К сожалению, все деньги из казны города S были потрачены на предыдущий неудачный военный поход, поэтому сначала правителю придётся продать некоторые свои дороги (разумеется, после этого он не сможет провести по ним войска).

Помогите правителю выяснить, какие дороги следует продать, а какие купить, чтобы довести войска от города S до города T и оплатить проход через все промежуточные города. Все операции продажи и

покупки дорог надо осуществить до начала похода, пока правители других городов не догадались о воинственных намерениях правителя города S.

Формат входных данных

Первая строка содержит целые числа N и M — количество городов и дорог соответственно ($2 \leqslant N \leqslant 2000$, $1 \leqslant M \leqslant 50\,000$). Города нумеруются от 1 до N, города S и T имеют номера 1 и N соответственно.

В каждой из следующих N строк находится по одному целому числу R_i — плата за проезд через город i ($0 \leqslant R_i \leqslant 10\,000$, $R_1 = R_N = 0$).

Следующие M строк содержат описания дорог. Дорога описывается четырьмя целыми числами: A_i, B_i, P_i и C_i, где A_i и B_i — номера городов, которые соединяет дорога ($A_i \neq B_i$); P_i — номер города, правителю которого она принадлежит; C_i — её стоимость ($1 \leqslant C_i \leqslant 10\,000$).

По дороге можно перемещаться в обоих направлениях. Любые два города соединены не более чем одной дорогой.

Формат выходных данных

В первой строке выведите список дорог, необходимых для продажи, в следующем формате: сначала число дорог, а затем их номера. Дороги нумеруются с единицы в том порядке, в котором они заданы на входе.

Во второй строке выведите список дорог, необходимых для покупки, в том же формате.

В третьей строке выведите маршрут похода — номера городов в порядке следования войска.

Если решений несколько, выведите любое. Если решения нет, выведите число -1.

Пример

входной файл	*выходной файл*
3 3 0 1 0 1 2 1 10 2 3 1 10 3 1 2 2	1 1 1 3 1 3

Задача 46. План эвакуации

В городе было некоторое число муниципальных зданий и убежищ, построенных специально для укрытия муниципальных работников в случае ядерной войны. Каждое убежище имеет ограниченную вместимость. В идеале все рабочие из муниципального здания должны бежать

в ближайшее убежище. Однако это приведёт к переполнению некоторых убежищ, в то время как другие будут заполнены частично. Для решения проблемы городской совет разработал специальный план эвакуации.

Муниципальным зданиям выделили убежища, указав число рабочих из каждого здания, которые должны использовать данное убежище, и оставили задачу индивидуального назначения руководству здания. План учитывает число рабочих в каждом здании — все они назначены в убежища, а также ограниченную вместимость каждого убежища — в каждое убежище назначено не больше рабочих, чем оно может вместить.

Городской совет заявляет, что их план оптимален, так как минимизирует суммарное время, необходимое рабочим, чтобы добраться до назначенных им убежищ.

Мэр города, известный своей постоянной враждой с городским советом, нанял вас как независимого консультанта проверить план эвакуации. Ваша задача — либо убедиться, что план действительно оптимален, либо доказать обратное, предоставив другой план с меньшим общим временем достижения убежищ, ясно показав таким образом некомпетентность совета.

Собирая начальные сведения по своей задаче, вы выяснили, что город представляет собой прямоугольную сетку. Положение муниципальных зданий и убежищ определяется двумя целыми числами, а время перехода между муниципальным зданием в точке (X_i, Y_i) и убежищем в точке (P_j, Q_j) равно $D_{ij} = |X_i - P_j| + |Y_i - Q_j| + 1$ мин.

Формат входных данных

Первая строка содержит два числа N и M — число муниципальных зданий в городе (все они пронумерованы от 1 до N) и число убежищ (все они пронумерованы от 1 до M) соответственно ($1 \leqslant N, M \leqslant 100$).

Следующие N строк описывают муниципальные здания. Каждая строка содержит целые числа X_i, Y_i и B_i, где X_i, Y_i — координаты зданий ($-1000 \leqslant X_i, Y_i \leqslant 1000$), а B_i — число рабочих в соответствующем здании ($1 \leqslant B_i \leqslant 1000$).

Следующие M строк описывают убежища. Каждая строка содержит целые числа P_j, Q_j и C_j, где P_j, Q_j — координаты убежищ ($-1000 \leqslant P_j$, $Q_j \leqslant 1000$), а C_j — вместимость убежища ($1 \leqslant C_j \leqslant 1000$).

Затем следуют N строк. Каждая строка описывает план эвакуации одного здания в том порядке, в котором они даны в описании города. План эвакуации i-го муниципального здания состоит из M целых чисел E_{ij}, где E_{ij} — число рабочих, которые должны эвакуироваться из здания

i в убежище j ($0 \leqslant E_{ij} \leqslant 1000$). Гарантируется, что план корректен, т. е. он позволяет эвакуировать из каждого здания ровно столько рабочих, сколько их там работает, и не превышает вместимость какого-либо убежища.

Формат выходных данных

Если план оптимален, выведите слово `OPTIMAL`. В противном случае выведите слово `SUBOPTIMAL` в первой строке, а в следующих N строках опишите свой план в том же формате, что и на входе. Вашему плану не обязательно быть самому по себе оптимальным, но он должен быть корректным и лучше, чем план городского совета.

Примеры

входной файл	*выходной файл*
3 4 -3 3 5 -2 -2 6 2 2 5 -1 1 3 1 1 4 -2 -2 7 0 -1 3 3 1 1 0 0 0 6 0 0 3 0 2	SUBOPTIMAL 3 0 1 1 0 0 6 0 0 4 0 1
1 1 0 0 1 0 0 1 1	OPTIMAL

Замечание

В первом примере план городского совета обеспечивает суммарное время эвакуации 56, а ваш план — 54, поэтому он более оптимален.

Во втором примере суммарное время равно 1, план оптимален.

Задача 47. Королевское задание

Жил король и имел N сыновей. И было в королевстве N красавиц, и король знал, какая из них нравится каждому его сыну. Сыновья были молодые и легкомысленные, так что вполне вероятно, что одному сыну могли нравиться несколько девушек.

Король попросил волшебника найти для каждого сына девушку для женитьбы на ней. Волшебник так и сделал — для каждого сына была выбрана девушка, которая ему нравилась и, конечно же, каждая красавица должна была выйти замуж только за одного из сыновей.

Однако король посмотрел на список и сказал: «Мне нравится список, что ты сделал, но я не удовлетворён. Для каждого сына я хотел бы знать

всех девушек, на которых он может жениться. Конечно, после того как он женится на любой из этих девушек, любой другой сын должен иметь возможность жениться на девушке, которая ему нравится».

Задача для волшебника оказалась слишком сложной. Вы должны спасти голову волшебника, решив эту задачу.

Формат входных данных

В первой строке содержится число N сыновей ($1 \leqslant N \leqslant 2000$).

Следующие N строк для каждого из сыновей содержат список девушек, которые ему нравятся: вначале число K_i девушек, а затем K_i различных целых чисел от 1 до N, определяющих девушек. Сумма K_i не превосходит 200 000.

Последняя строка содержит список, сделанный волшебником, из N различных целых чисел — для каждого из сыновей номер девушки, на которой он мог бы жениться в соответствии со списком.

Гарантируется, что список корректен, т. е. каждому сыну нравится девушка, на которой он должен жениться.

Формат выходных данных

Выведите N строк. Для каждого сына вначале выведите число L_i различных девушек, которые ему нравятся, а затем — L_i различных целых чисел в произвольном порядке, которые соответствуют понравившимся ему девушкам, женившись на любой из которых, он оставляет возможность всем остальным сыновьям жениться на девушках, которые им нравятся.

Пример

входной файл	*выходной файл*
4 2 1 2 2 1 2 2 2 3 2 3 4 1 2 3 4	2 1 2 2 1 2 1 3 1 4
1 1 1 1	1 1

Задача 48. Платформы

Известная на весь Могилёв компания «Headache» выпустила игру, для которой необходима конструкция, состоящая из маленьких платформ и труб. Платформы разделяются на стартовые (их N_1 штук), финишные (N_3 штук) и промежуточные (N_2 штук).

Все стартовые и финишные платформы находятся на одинаковой высоте. Все высоты промежуточных платформ различны. Они меньше высоты стартовых, но больше высоты финишных. Каждой платформе соответствует уникальный номер от 1 до $N_1 + N_2 + N_3$. Нумерация следующая: сначала все стартовые платформы, затем промежуточные и, наконец, финишные.

Все промежуточные платформы пронумерованы по убыванию высоты, т. е. если номер промежуточной платформы A меньше номера платформы B, то высота A больше высоты B.

На каждой из стартовых платформ находится шарик. Шарик может скатиться с платформы A на платформу B, если они соединены трубой и высота A больше высоты B. На каждой из финишных платформ может оказаться не более одного шарика. Если шарик находится на некоторой платформе, то игрок может выбрать направление дальнейшего пути шарика, т. е. выбрать платформу, на которую шарик скатится. Для каждой промежуточной платформы задано число C, равное максимальному количеству шариков, которые могут прокатиться по ней за время игры.

Цель игры заключается в том, чтобы на финишных платформах оказалось как можно больше шариков.

Вам нужно узнать, какое максимальное число шариков может оказаться на финишных платформах в результате игры.

Формат входных данных

В первой строке записаны целые числа N_1, N_2, N_3 — соответственно количество стартовых, промежуточных и финишных платформ, где $0 < N_1, N_3 \leqslant 50$, $1 < N_1 + N_2 + N_3 \leqslant 200$.

В последующих N_2 строках для каждой промежуточной платформы j $(N_1 + 1 \leqslant j \leqslant N_1 + N_2)$ указано максимальное число шариков C_j, которые могут прокатиться по платформе за всё время игры $(0 \leqslant C_j \leqslant 50)$.

В каждой из следующих $N_1 + N_2$ строк для платформы с номером i $(1 \leqslant i \leqslant N_1 + N_2)$ сначала указывается число K_i — количество труб, выходящих из платформы, а затем перечислены K_i номеров платформ, на которые может скатиться шарик с платформы i.

Гарантируется, что не существует труб как между стартовыми платформами, так и между финишными.

Формат выходных данных

В первой строке должно находиться единственное число, равное

максимальному числу шариков, которые могут оказаться на финишных платформах в результате игры.

Пример

входной файл	*выходной файл*
3 4 3 3 2 1 2 1 4 1 4 1 4 2 5 6 1 7 1 7 3 8 9 10	2

Задача 49. Безопасные пути

В некотором далёком царстве есть система дорог, представляющая собой неориентированный граф. Каждая дорога соединяет два города. Между двумя городами — не более одной дороги. Известно, что с течением времени каждая дорога в этом царстве приобрела один из трёх статусов. Первый — статус дороги, на которой стоит царская охрана (далее именуемой R_1), второй — статус дороги, на которой регулярно происходят грабежи проезжающих дилижансов (далее именуемой R_2), и третий — статус нейтральной дороги (R_3). Отсюда видно, что население страны делится на два слоя — добропорядочных граждан и бандитов. Ясно, что обычные граждане ходят только по дорогам типа R_1 и R_3, а бандиты — по R_2 и R_3. Поскольку царская казна тратит много денег на содержание данных дорог, то царь решил убрать максимальное число дорог так, чтобы можно было пройти из любого города в любой другой как добропорядочному гражданину, так и грабителю (иначе в стране поднялся бы бунт — взбунтуются или мирные граждане, или бандиты). Необходимо найти максимальное число удаляемых дорог и их номера в соответствии с требованиями царя.

Формат входных данных

В первой строке записано сначала число N городов ($1 \leqslant N \leqslant 500$), затем число M дорог в царстве ($1 \leqslant M \leqslant 10\,000$).

Затем идут M троек чисел, описывающих дороги: первые два задают номера городов, которые она соединяет, а третье — статус: 1 — R_1, 2 — R_2, 3 — R_3. Между любыми двумя городами существует не более одной дороги.

Формат выходных данных

Выведите сначала число удаляемых дорог, а затем их номера (дороги нумеруются в том порядке, в каком они были заданы во входном файле). Если решения не существует, выдать -1.

Пример

входной файл	*выходной файл*
5 7 1 2 3 2 3 3 3 4 3 5 3 2 5 4 1 5 2 2 1 5 1	2 4 7

Задача 50. Светофоры

Дорожное движение в городе Дингилвилле устроено необычным образом. В городе есть перекрёстки и дороги, которыми перекрёстки связаны между собой. Любые два перекрёстка могут быть связаны не более чем одной дорогой. Не существует дорог, соединяющих один и тот же перекрёсток с самим собой. Время проезда по дороге в обоих направлениях одинаково. На каждом перекрёстке находится один светофор, сигнал которого в каждый момент времени может быть либо голубым, либо розовым. Сигнал каждого светофора изменяется периодически: в течение некоторого интервала времени он голубой, а затем в течение другого интервала — розовый. Движение по дороге между любыми двумя перекрёстками разрешено тогда и только тогда, когда светофоры на обоих перекрёстках этой дороги имеют один и тот же сигнал в момент въезда на эту дорогу. Если транспортное средство прибывает на перекрёсток в момент переключения светофора, то его поведение будет определяться новым сигналом светофора. Транспортные средства могут находиться в состоянии ожидания на перекрёстках.

У вас есть карта города, которая показывает время прохождения каждой дороги (целые числа); длительность каждого сигнала для каждого светофора (целые числа); начальный сигнал и оставшееся время действия этого сигнала (целые числа) для каждого светофора.

Необходимо определить путь между двумя заданными перекрёстками, позволяющий транспортному средству проехать от начального перекрёстка к конечному за минимальное время с момента старта. Если существуют несколько таких путей, то вы можете вывести любой.

Формат входных данных

Первая строка содержит два числа: номер начального и номер конечного перекрёстков.

Во второй строке записано число перекрёстков N ($2 \leqslant N \leqslant 300$) и число дорог M ($1 \leqslant M \leqslant 14\,000$). Перекрёстки нумеруются от 1 до N.

Следующие N строк содержат информацию об N перекрёстках. Отдельная строка содержит информацию о перекрёстке с номером i: C_i, R_i, T_i^B, T_i^P. Здесь прописная латинская буква C_i обозначает начальный свет светофора на перекрёстке (`B` — голубой, `P` — розовый). Далее R_i — оставшееся время действия начального сигнала ($1 \leqslant R_i$). Числа T_i^B и T_i^P — длительность действия голубого и розового сигнала светофора соответственно на перекрёстке i ($1 \leqslant T_i^B, T_i^P \leqslant 100$). Число R_i не превосходит длительности действия соответствующего сигнала C_i.

Следующие M строк описывают M дорог. Каждая строка имеет вид i, j, L_{ij}, где i и j — номера перекрёстков, которые связывает эта дорога, L_{ij} — время, необходимое для перемещения по дороге ($1 \leqslant L_{ij} \leqslant 100$).

Формат выходных данных

Если путь существует, то в первой строке выведите минимальное время, необходимое для достижения конечного перекрёстка из начального, а во второй — список перекрёстков, который соответствует найденному минимальному пути. Перекрёстки должны быть записаны в порядке их посещения. Первое число должно быть номером начального перекрёстка, последнее — конечного. Если пути нет, то выведите число 0.

Пример

входной файл	*выходной файл*
1 4 4 5 B 2 16 99 P 6 32 13 P 2 87 4 P 38 96 49 1 2 4 1 3 40 2 3 75 2 4 76 3 4 77	127 1 2 4

Задача 51. Достроить дороги

В некотором государстве есть N городов, которые пронумерованы целыми числами от 1 до N. Между городами построено M дорог. Каждая дорога соединяет два города. По дорогам можно путешествовать

в обоих направлениях. Определите, какое минимальное число новых дорог нужно достроить, чтобы из каждого города можно было проехать в любой другой.

Формат входных данных

В первой строке записано целое число N — количество городов ($1 \leqslant N \leqslant 100\,000$). Во второй строке — целое число M — количество уже существующих дорог ($0 \leqslant M \leqslant 100\,000$). В последующих M строках описаны дороги: пара чисел u и v, записанных через пробел, что свидетельствует о том, что между городами u и v построена дорога ($1 \leqslant u, v \leqslant N$ и $u \neq v$).

Формат выходных данных

Выведите одно число — минимальное количество дорог, добавление которых обеспечит связность дорожной сети в стране.

Примеры

входной файл	*выходной файл*
3 4 1 2 2 3 1 2 3 1	0
3 0	2

Задача 52. Минимальная плата за проезд

В одной далёкой стране есть очень интересная система налогообложения автомобилистов. За проезд по дороге из города i в город j необходимо уплатить налог C_{ij}. Система дорог в этой стране также очень интересная, она удовлетворяет трём правилам:

- из каждого города можно проехать в любой другой;
- если по дороге можно проехать из города i в город j, то по ней можно проехать и обратно;
- между любой парой городов существует ровно один путь.

Вы работаете в налоговом бюро, и от вас требуется написать программу, которая бы рассчитала, какую сумму нужно заплатить, чтобы проехать из города A в город B.

Формат входных данных

В первой строке входного файла находится натуральное число N ($N \leqslant 50\,000$) — количество городов (города нумеруются от 1 до N).

Далее в $N-1$ строках — описание дороги: номера городов, которые она соединяет, и налог (натуральное число, не превосходящее 10 000), который необходимо заплатить для проезда по ней.

В $N+1$ строке находится натуральное число M ($M \leqslant 50\,000$) — количество запросов в вашей программе.

Далее в M строках файла — пара номеров городов, для которых необходимо рассчитать суммарный налог для проезда из одного города в другой.

Формат выходных данных

В выходном файле должно быть ровно M строк, ответы на запросы в том порядке, в котором они идут во входном файле.

Пример

входной файл	*выходной файл*
6 1 6 5 2 6 5 6 5 3 4 5 4 3 5 6 4 1 2 1 4 4 6 1 3	10 12 7 14

Задача 53. Максимальное ребро

В одной далёкой стране существует очень интересная система налогообложения автомобилистов. Чтобы проехать из города i в соседний город j, необходимо уплатить «местный» налог C_{ij}. Для того чтобы проехать из города A в город B, необходимо уплатить налог, равный максимальному из всех «местных» налогов пути из A в B. Система дорог в этой стране также очень интересная, она удовлетворяет трём правилам:

- из каждого города можно проехать в любой другой;
- если по дороге можно проехать из города i в город j, то по ней можно проехать и обратно;
- между любой парой городов существует ровно один путь.

Вы работаете в налоговом бюро, и от вас требуется написать программу, которая бы рассчитала, какую сумму нужно заплатить, чтобы проехать из города A в город B.

Формат входных данных

В первой строке находится число N городов ($1 \leqslant N \leqslant 50\,000$).

В каждой из следующих $N-1$ строк — описание одной дороги: номера городов, которые она соединяет, и местный налог, необходимый для проезда по ней.

В $(N+1)$-й строке находится целое число M — количество запросов $(0 \leqslant M \leqslant 50\,000)$.

В каждой из следующих M строк — пара номеров городов, для которых необходимо рассчитать налог.

Формат выходных данных

Выведите ровно M строк: ответы на запросы в том порядке, в котором они идут во входном файле.

Пример

входной файл	*выходной файл*
6 1 6 5 2 6 5 6 5 3 4 5 4 3 5 6 4 1 2 1 4 4 6 1 3	5 5 4 6

Задача 54. Ровно *K*

Вася — большой любитель настольных игр. Недавно он приобрёл новую игру для одного игрока с интригующим названием «Ровно K» — это продолжение популярной серии игр «Ровно 1», «Ровно 2», «Ровно 3»...

На игровом поле отмечено N позиций, пронумерованых натуральными числами от 1 до N. Фишка игрока первоначально располагается на первой позиции. Некоторые пары позиций соединены линиями. По правилам игры за один ход фишку нужно обязательно переместить из текущей позиции в другую, соединённую с текущей. Если это сделать невозможно, то игра заканчивается и игроку засчитывается поражение. Между двумя позициями может быть проведено не более одной линии. Однако бывают линии, которые ведут из позиции в неё же, тогда фишка в течение одного хода может оставаться неподвижной. Все соединения действуют в обе стороны: если позиция i соединена с позицией j, то фишку можно передвигать как из позиции i в позицию j, так и обратно.

Игра состоит из нескольких независимых партий. Каждая партия описывается двумя числами: V и K. Чтобы выиграть партию, игроку

нужно переместить фишку из позиции 1 в позицию V, сделав ровно K ходов.

Вася распаковал коробку, разложил на столе игровое поле, извлёк брошюру с правилами игры и списком из Q пар чисел — это числа V и K для каждой партии.

Какие партии Вася сможет выиграть при оптимальной игре, а какие выиграть принципиально невозможно?

Формат входных данных

В первой строке входного файла записаны два целых числа N ($1 < N \leqslant 5000$) и M ($0 \leqslant M \leqslant 500\,000$) — количество позиций и соединительных линий на игровом поле соответственно.

Последующие M строк описывают соединения: линия задаётся парой номеров позиций, которые она соединяет.

Затем в отдельной строке записано число партий Q ($1 \leqslant Q \leqslant 5000$), которые предстоит сыграть Васе.

Затем в Q строках описаны партии: каждая пара чисел V и K ($1 \leqslant V \leqslant N$, $0 \leqslant K \leqslant 1\,000\,000\,000$) задана в отдельной строке.

Формат выходных данных

Выходной файл должен содержать ровно Q строк, по одной для каждой партии.

В случае если существует путь фишки из позиции 1 в позицию V, содержащий ровно K ходов, выведите в соответствующей строке `Yes`, иначе выведите `No`.

Пример

входной файл	*выходной файл*
6 8 1 2 1 4 3 4 2 5 4 6 5 6 4 5 3 6 3 3 2 6 5 6 1	Yes Yes No

Задача 55. Надёжная сеть

Акула хочет быть в курсе всех дел Квадратного Бизнесмена. Для этого он поручил Фицджеральду новое задание: связать в сеть компью-

теры Акулы и Квадратного Бизнесмена, используя для этого, возможно, промежуточные компьютеры, стоящие в цехах «Сыр Индастриз». Акуле также известно, что Квадратный Бизнесмен имеет достаточно могущества, чтобы отключить от сети одновременно до K промежуточных компьютеров, но на отключение $(K+1)$-го компьютера его власти уже недостаточно. Квадратный Бизнесмен никогда не выключает свой компьютер и не может отключить компьютер Акулы, ведь они в деле. Акула хочет, чтобы сеть была надёжной, т. е. чтобы Квадратный Бизнесмен не смог прервать связь между компьютерами Акулы и Квадратного Бизнесмена, даже если бы очень этого захотел.

Все компьютеры пронумерованы уникальными натуральными числами от 1 до N. Компьютер Акулы имеет номер A, а компьютер Квадратного Бизнесмена — номер B.

Фиц уже определил, какие пары компьютеров можно связать между собой непосредственно и сколько метров провода понадобится для этого. Но теперь он решил, что ему нужно зайти в бар к Роде, чтобы немного освежиться. А вы как его Друган должны посчитать, можно ли построить надёжную сеть. Если это возможно, то узнайте наименьшую суммарную длину проводов (в метрах), которая понадобится для этого, если нет — определите минимальное количество компьютеров, которое нужно отключить Квадратному Бизнесмену, чтобы Акула не мог быть в курсе его дел.

Формат входных данных

В первой строке входного файла содержатся два целых числа N и M ($1 \leqslant N, M \leqslant 10\,000$) — соответственно количество компьютеров и количество пар компьютеров, которые возможно соединить.

У каждой из следующих M строк есть по три целых числа X, Y, Z ($1 \leqslant X, Y \leqslant N$, $1 \leqslant Z \leqslant 20\,000$), означающих, что компьютеры X и Y можно соединить непосредственно и на это уйдёт Z метров провода.

Последняя строка содержит три целых числа: A, B и K ($1 \leqslant A$, $B \leqslant N$, $0 \leqslant K \leqslant N$).

Формат выходных данных

Первая строка выходного файла должна содержать сообщение `Yes`, если можно построить надёжную сеть, в противном случае — `No`.

Если ответ положительный, то во второй строке выведите одно целое число, равное наименьшей суммарной длине проводов (в метрах), необходимой для построения сети.

Иначе во второй строке выведите одно целое число — минималь-

ное количество компьютеров, которое нужно отключить Квадратному Бизнесмену, чтобы Акула не мог быть в курсе его дел.

Примеры

входной файл	*выходной файл*
8 11 1 2 1 1 3 1 2 3 7 2 4 6 3 7 3 3 8 4 4 5 9 4 8 1 5 6 5 6 7 3 6 8 2 2 6 2	No 2

Задача 56. Острова

Одно небольшое малоизвестное государство размещается на N островах, омываемых бескрайними водами Тихого океана. На трёх островах расположены три больших города — крупные промышленные узлы государства. Правительство разрабатывает проект строительства автомобильных мостов между некоторыми парами островов, чтобы обеспечить надёжную транспортную связь между городами. Цель — сделать возможным беспрепятственный проезд на автомобиле из каждого города в любой другой.

Инженеры разработали M проектов мостов между парами островов и оценили стоимость реализации всех проектов (каждая пара островов рассматривалась не более одного раза, причём строительство кольцевых мостов из острова в него же никого не интересует). Мосты предполагают двустороннее движение транспорта по ним.

Строить все мосты дорого, да и нет необходимости: достаточно связать лишь три острова, на которых располагаются города, можно не напрямую, а через другие острова. Какой минимальный объём финансирования потребуется для этого?

Формат входных данных

В первой строке входного файла записаны через пробел целые числа N ($3 \leqslant N \leqslant 100\,000$) и M ($1 \leqslant M \leqslant 100\,000$) — соответственно количество островов и количество мостов, которые можно построить. Острова имеют номера от 0 до $N - 1$.

В последующих M строках описываются мосты: каждая строка

содержит три числа A, B ($0 \leqslant A, B < N$) и K ($1 \leqslant K \leqslant 1\,000\,000$), которые свидетельствуют, что строительство моста между островами A и B стоит K у. е.

В последней строке записано три различных числа X, Y и Z ($0 \leqslant X$, Y, $Z < N$) — номера островов, между которыми следует организовать сухопутные перемещения.

Формат выходных данных

Выведите одно число — минимальную сумму (в у. е.), которую потребуется потратить на строительство мостов. Гарантируется, что решение существует.

Примеры

входной файл	*выходной файл*
3 2 0 1 1 1 2 1 0 1 2	2
4 4 0 1 4 1 2 5 1 3 1 3 2 2 0 1 2	7

Задача 57. Снести

Издавна сложилось, что большинство сельского населения страны Байтландии селилось на хуторах, состоящих из небольшого числа домов. Постройку и поддержание в надлежащем состоянии дорог, соединяющих некоторые из этих хуторов, взяло на себя государство. Однако затея оказалась слишком дорогостоящей.

К настоящему времени между некоторыми парами хуторов может не существовать пути, проходящего по государственным дорогам. Будем считать, что такие хутора находятся в различных изолированных областях. Денег на постройку новых дорог в казне нет. Поэтому королевские советники решили объявить некоторые хутора бесперспективными и снести их, предоставив обитателям благоустроенное жильё в других населённых пунктах.

На первом этапе реализации этой программы чиновники предлагают объявить бесперспективными два хутора, расположенных в разных изолированных областях. Однако король перед утверждением этого плана потребовал (наверное, по своей прихоти), чтобы число изолированных областей в Байтландии не изменилось после сноса бесперспективных

хуторов. И теперь советники задались вопросом: какую же пару хуторов снести? Помогите им, рассчитав число вариантов выбора такой пары.

Формат входных данных

В первой строке заданы число N хуторов и число M дорог в Байтландии ($3 \leqslant N \leqslant 100\,000$, $0 \leqslant M \leqslant 100\,000$).

Каждая из последующих M строк описывает одну дорогу и содержит номера хуторов, которые соединяет эта дорога. Нумерация хуторов начинается с единицы. Между любой парой хуторов может быть не более одной дороги. Никакая дорога не соединяет хутор сам с собой.

Формат выходных данных

Выведите единственное число — искомое число способов выделения пары бесперспективных хуторов.

Примеры

входной файл	*выходной файл*
4 2 1 2 3 4	4
7 6 1 2 2 3 3 1 4 5 5 6 5 7	9

Задача 58. Сталкер: туда и обратно

Спасаясь от монстров в заброшенных подземельях НИИ «Агропром», вы юркнули в неприметную дверь — и она вдруг закрылась, оставив вас в лабиринте со множеством комнат и приоткрытыми дверями, ведущими из одной комнаты в другую. К счастью, ваш верный лэптоп, как всегда, не подвёл — вы сумели скачать карту лабиринта.

Лабиринт, оказывается, состоит из квадратных комнат, образующих прямоугольник. Некоторые комнаты замурованы (там, видно, хранится что-то ценное, но у вас не хватает опыта, чтобы туда проникнуть). Из каждой комнаты можно перейти в комнаты, которые имеют общую сторону (соседние комнаты), естественно, не выходя за пределы лабиринта и не проникая в замурованные комнаты. Единственный выход из лабиринта находится в той комнате, куда вы так неудачно попали (эта комната — крайняя на плане лабиринта). Ключ от двери находится в другой комнате, так что вам предстоит добраться до ключа и вернуться обратно. Одно радует — монстров в этом лабиринте нет.

К сожалению, в каждом переходе между любыми двумя комнатами расположены датчики движения, которые намертво закрывают дверь между этими комнатами, если по переходу кто-то прошёл, так что каждым переходом можно воспользоваться не более одного раза.

Переход из одной комнаты в соседнюю занимает одну единицу времени, а ресурс вашего противорадиационного костюма не бесконечен. Именно поэтому вы должны выбраться из лабиринта как можно скорее.

Формат входных данных

В первой строке входного файла записаны два числа M и N — количество строк и столбцов в плане лабиринта ($1 \leqslant M, N \leqslant 50$).

Далее следуют M строк, каждая из которых содержит N символов — план лабиринта.

В этих строках допускаются следующие символы:

`.` — пустая комната;

`*` — замурованная комната;

`M` — комната, в которой находится вход и выход из лабиринта;

`G` — комната, в которой находится ключ.

Формат выходных данных

В выходной файл заносится единственное число — минимальное время, необходимое вам для путешествия по переходам лабиринта, чтобы открыть выходную дверь.

Если решение задачи по какой-либо причине невозможно, выведите число -1.

Примеры

входной файл	*выходной файл*
4 6 M..... .****. .*.... ...*.G	18
4 6 M..... .****. .*.**. ...*.G	-1

Задача 59. Полёт с пересадками

У Квадратного Бизнесмена очень много дел в разных городах. Поэтому нередко ему приходится перемещаться из одного города в другой. Сегодня как раз такой случай. Квадратный Бизнесмен предпочитает передвигаться на самолёте, потому что это очень быстро и удобно.

Поскольку Квадратный Бизнесмен летает часто, он пронумеровал все города, в которых приходится бывать, целыми числами от 0 до $N-1$. Сегодня Квадратный Бизнесмен находится в городе S, но на днях он должен оказаться на важном совещании, которое проходит в городе T.

Зная цены билетов на все авиарейсы, найдите наилучший по стоимости перелёт из S в T, совершающий не более Q пересадок (Квадратный Бизнесмен, как и автор задачи, считает очень утомительным большое число пересадок при авиаперелёте). Для примера, если совершается полёт из города 0 в город 4, а затем из города 4 в город 3, то использовано два рейса и сделана одна пересадка.

Формат входных данных

В первой строке содержатся два целых числа N и M — число городов и число рейсов соответственно ($1 \leqslant N, M \leqslant 10^5$).

Каждая из следующих M строк имеет три целых числа X, Y и Z, означающих, что из города X в город Y есть авиарейс стоимостью Z ($0 \leqslant X, Y < N, 1 \leqslant Z \leqslant 1000$). Между одной и той же парой городов может быть несколько авиарейсов, возможно, в различных направлениях. Однако нет авиарейса, который ведёт из города в этот же город.

Последняя строка содержит три числа S, T и Q ($0 \leqslant S, T, Q < N$). Города S и T не совпадают. Известно также, что $N \times Q \leqslant 5 \cdot 10^6$, $M \times Q \leqslant 5 \cdot 10^6$.

Формат выходных данных

В первой строке выведите `Yes`, если можно удовлетворить требования Квадратного Бизнесмена, в противном случае — `No`. Если ответ положительный, то во второй строке выведите два целых числа: стоимость C и число R пересадок в наилучшем по стоимости перелёте из тех, которые требуют не более Q пересадок, а в третьей строке через пробел $R+2$ целых числа — номера городов, которые посетит Квадратный Бизнесмен на этом пути (в порядке посещения).

Примеры

входной файл	*выходной файл*
5 5 0 1 2 1 2 5 2 3 3 0 4 5 4 3 10 0 3 1	Yes 15 1 0 4 3

Задача 60. Нефть

В некоторой стране имеются N нефтяных вышек, i-я из которых способна выкачать до a_i тысяч баррелей нефти в сутки, и M нефтеперерабатывающих заводов, каждый из которых, в свою очередь, способен переработать до b_j тысяч баррелей нефти в сутки.

Заводы и вышки связаны трубопроводами; ввиду различных причин стоимость транспортировки одной тысячи баррелей от i-й вышки к j-му заводу составляет c_{ij} денежных единиц; объём перекачки не ограничен.

Учитывая, что вышки и заводы должны находиться в производственном балансе, т. е. вся выкачиваемая нефть должна быть вовремя переработана (излишки нигде не должны сливаться или накапливаться), какое максимальное количество нефти в сутки может перерабатывать данная инфраструктура? На данный вопрос ответить относительно легко, поэтому нас интересует лишь величина минимальных возможных затрат среди всех максимальных по объёму перерабатываемой нефти решений.

Формат входных данных

В первой строке входного файла находятся целые числа N и M ($1 \leqslant N, M \leqslant 300$) — число вышек и заводов соответственно.

Во второй строке через пробел перечислены N целых чисел — значения a_i ($1 \leqslant a_i \leqslant 30\,000$).

В третьей строке через пробел даны M целых чисел — значения b_j ($1 \leqslant b_j \leqslant 30\,000$). Сумма всех a_i равна сумме всех b_j.

В последующих N строках по M целых чисел записаны стоимости транспортировки одной тысячи баррелей нефти от соответствующей вышки к заводу; j-е число в $(i+2)$-й строке задаёт c_{ij} ($1 \leqslant c_{ij} \leqslant 10\,000$).

Формат выходных данных

В единственной строке выведите одно целое число — минимальную величину затрат на транспортировку при условии максимизации объёма переработки.

Гарантируется, что ответ можно поместить в знаковое тридцатидвухбитное целое.

Пример

входной файл	*выходной файл*
3 4 3 6 7 2 5 1 8 1 2 3 4 8 7 6 5 9 12 10 11	110

1.5. УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ

1. Дерево. Для решения задачи можно воспользоваться эквивалентными определениями дерева. Например, граф является деревом, если он связный и число его рёбер $m = n - 1$. Для проверки связности графа можно воспользоваться алгоритмами поиска в ширину или глубину.

2. Остовное дерево. Для построения остовного дерева графа можно воспользоваться алгоритмом Крускала (строит минимальное остовное дерево связного графа), в котором рёбра графа нужно перебирать в произвольном порядке. Если окажется, что все вершины графа принадлежат одному множеству, то остовное дерево построено, в противном случае остовного дерева не существует (граф несвязный).

Другой способ решения задачи — выполнить поиск в ширину (глубину) и вывести рёбра, которым соответствуют дуги корневого дерева поиска в ширину (глубину). Если граф оказался несвязным, сообщить, что остовного дерева не существует.

3. Наибольшее число маршрутов (не пересекающихся по рёбрам). Построить сеть, выбирая в качестве полюсов заданные вершины v и w, а каждое ребро графа заменить двумя противоположно направленными дугами с пропускной способностью, равной единице. Воспользоваться алгоритмом нахождения максимального потока в сети. Если в задаче потребуется, чтобы маршруты не пересекались по вершинам, то необходимо дополнительно каждую вершину графа превратить в дугу с единичной пропускной способностью.

4. Лабиринт (без пересечений по дорогам). По графу построим сеть. Для этого введём два полюса: источник s, который соединим дугами со всеми входами, и сток t, с которым соединим дугами все выходы; каждое ребро заменим двумя противоположно направленными дугами; пропускные способности всех дуг положим равными 1. Теперь воспользуемся алгоритмом построения максимального потока в сети.

5. Шестерёнки. Построим граф, в котором вершины соответствуют шестерёнкам, а ребро соединяет шестерёнки, сцепленные друг с другом. Теперь задача сводится к определению двудольности связной компоненты графа, которой принадлежит шестерёнка с номером 1. Случай удаления шестерёнок решается алгоритмом полного перебора

(переберём все возможные варианты выбрасывания по одной, а в случае неудачи — по две, три, ..., $N-1$ шестерёнки).

6. Открытки и конверты. Можно сформировать двудольный граф: первая доля — открытки, вторая — конверты, причём две вершины соединены ребром, если одна соответствует открытке, другая — конверту, при этом открытка входит в конверт. В построенном двудольном графе найдём наибольшее паросочетание (для этого можно использовать любую из реализаций алгоритма Куна, приведённую в [6]). Если наибольшее паросочетание является совершенным, то открытки можно разложить по конвертам.

7. Разбиение на команды по численности. Построим граф, в котором вершины сопоставляются со студентами, а ребро соединяет две вершины, если соответствующие студенты не знают друг друга. Теперь необходимо сначала решить задачу определения двудольности построенного графа, а затем в случае его двудольности определить правило, которого надо придерживаться в попытке разбить студентов на две команды, численность которых отличается не более чем в два раза.

8. Разбиение на группы незнакомых. Построим граф, в котором вершины соответствуют людям, а ребро соединяет две вершины, если эти люди знакомы друг с другом. Теперь задача сводится к определению двудольности графа.

9. Олимпиада. Постоим граф, в котором вершины будут соответствовать участникам олимпиады, а ребро соединяет две вершины, если соответствующие участники знакомы друг с другом. Теперь задача сводится к определению связности графа.

10. Книга. С одной стороны, задача заключается в построении гамильтонова цикла и является NP-трудной задачей, но исходя из ограничений задачи граф является графом Дирака, что позволяет эффективно построить гамильтонов цикл. Предположим, что построен некоторый путь $x_1, x_2, \ldots, x_k$ (первоначально начнём с пути длины 1). Множество вершин, вошедших в путь, будем считать пройденными, остальные — непройденными. Возможны три ситуации.

1) Одна из вершин x_1 или x_k смежна с одной из непройденных вершин, поэтому путь можно удлинить на одну вершину.

2) Ни одна из вершин x_1, x_k не смежна с одной из непройденных вершин, а вершины x_1 и x_k смежны. Очевидно, что $k > N/2$, так как по условию x_1 и x_k смежны с $N/2$ вершинами. Следовательно, количество непройденных вершин не больше $N/2$. Поэтому любая вершина y из них смежна с одной из пройденных вершин, например x_i. Но тогда можно получить более длинный путь:

$$y, x_i, x_{i+1}, \ldots, x_k, x_1, x_2, \ldots, x_{i-1}.$$

3) Ни одна из вершин x_1, x_k не смежна с одной из непройденных вершин и вершины x_1 и x_k не смежны. В этом случае нетрудно показать (в силу условия, что степень любой вершины равна $N/2$), что в пути $x_1, x_2, \ldots, x_k$ существует такой индекс i, что вершина x_1 смежна с x_i, а x_{i-1} смежна x_k. Следовательно, рассмотрев путь

$$x_i, x_{i+1}, \ldots, x_k, x_{i-1}, x_{i-2}, \ldots, x_1,$$

мы имеем ситуацию 2), поэтому можно получить более длинный путь.

11. Станки. Поскольку производительность всего конвейера определяется минимальной производительностью рабочего на конвейере, то для решения задачи достаточно определить максимальную производительность P, при которой возможно распределение всех рабочих по станкам, причём производительность у каждого рабочего не ниже производительности P. Для определения максимальной производительности P можно воспользоваться методом дихотомии, отсортировав все элементы матрицы C. Сформируем двудольный граф G: вершины первой доли соответствуют рабочим, второй доли — станкам; вершина первой доли соединена ребром с вершиной второй доли, если рабочий может быть назначен на станок (при этом вес ребра равен производительности рабочего на станке). Предположим, что P' — некоторая производительность. В двудольном графе G оставим рёбра, вес которых не меньше P', обозначим полученный граф через $G_{P'}$. Если в двудольном графе $G_{P'}$ существует совершенное паросочетание, то конвейер работает с производительностью P' (для построения наибольшего паросочетания можно воспользоваться одной из реализаций алгоритма Куна) и величина возможной производительности конвейера P' увеличивается. Если нет, то величина возможной производительности конвейера P' уменьшается.

Для построения лексикографически меньшего совершенного паросочетания можно применить алгоритм, предложенный в [6], суть которого в том, что в двудольном графе возможны циклы только чётной длины. Распределение рабочих по станкам определяется структурой совершенного паросочетания.

12. Встреча. Покажем, что достаточно ограничиться случаем, когда искомая точка совпадает с одним из домиков [3]. Предположим, что искомая точка z лежит на дороге, соединяющей домики u и v на расстоянии d от домика u (предположим, что расстояние между этими домиками равно l). Разобьём все домики на два подмножества V и U. Во множество V войдут все домики, для которых расстояние от них до дома v меньше, чем до u (число этих домиков обозначим K_v), а все остальные домики отнесём ко множеству U (число этих домиков обозначим K_u). Пусть сначала $K_v > K_u$. Обозначим через $R(z, d)$ суммарное пройденное расстояние от всех домиков до искомой точки z, находящейся на расстоянии d от домика u.

Перенесём искомую точку z по направлению к дому v на расстояние p, $p < (l - d)$. Тогда

$$R(z, d + p) = R(z, d) + K_u p - K_v p = R(z, d) + (K_u - K_v)p < R(z, d).$$

Таким образом, первоначальная установка точки z была неверной. Аналогично рассматривается случай, когда $K_v < K_u$, а искомая точка z сдвигается по направлению к дому u. Получим, что в этих двух случаях искомая точка z не может быть расположена на дороге.

Рассмотрим теперь последний случай, когда $K_v = K_u$. Тогда

$$R(z, d + p) = R(z, d),$$

что свидетельствует о том, что искомая точка может быть на дороге в любом месте, в том числе располагаться и в концевых домиках этой дороги.

Таким образом, ограничимся рассмотрением случая, когда встреча произойдёт в одном из домиков. Воспользуемся алгоритмом Флойда для нахождения кратчайших расстояний между всеми парами домиков и выберем тот, для которого суммарное расстояние до всех домиков минимально.

13. Янка. Задача эквивалентна поиску такой расстановки на плоскости, чтобы на гранях, которыми многогранники соприкасаются с

плоскостью, были различные наклейки. Сформируем двудольный граф из $2N$ вершин: вершины первой доли соответствуют многогранникам, а второй — типам наклеек. Ребро соединяет многогранник с типом наклейки на его гранях. Теперь в двудольном графе нужно найти совершенное паросочетание (по условию задачи предполагается, что оно всегда существует). Рёбра совершенного паросочетания указывают, на какую наклейку нужно поставить соответствующий многогранник.

Для нахождения совершенного паросочетания в двудольном графе можно использовать одну из реализаций алгоритма Куна, приведённую в [6].

14. Второй по длине маршрут. Поскольку по условию задачи допускается, чтобы искомый маршрут по некоторым дугам проходил несколько раз, то для решения задачи можно применить алгоритм Дейкстры с использованием приоритетной очереди. В данном алгоритме вершину необходимо считать просмотренной лишь после её второго удаления из приоритетной очереди [6].

15. Чётно-нечётная магистраль. Предположим, что граф, задающий систему магистралей, связный, но не двудольный. В таком случае по критерию двудольности Кёнига в этом графе есть цикл нечётной длины, используя который можно изменить чётность длины маршрута между любой парой вершин.

Предположим, что граф, задающий систему магистралей, — несвязный. В этом случае система магистралей не является чётно-нечётной, а искомое подмножество городов (в котором максимальное число элементов и которое удовлетворяет следующему условию: если какие-либо два различных города относятся к этому подмножеству и соединены маршрутом, то его длина чётна) может быть сформировано на основе анализа двудольности компонент связности графа.

16. Пересекающиеся дороги. Построим граф: вершины соответствуют перекрёсткам, ребро соединяет две вершины, если они соединены дорогой. По входным данным для каждой вершины определим её степень, которая в дальнейшем будет использоваться для вычисления времени проезда через вершину. Для решения задачи нахождения простой цепи между заданной парой вершин можно использовать алгоритм Дейкстры. При реализации алгоритма Дейкстры метка вершины v равна времени, затраченному на маршрут из стартовой вершины I до

вершины v, без учёта времени ожидания, необходимого для преодоления вершины v. Время ожидания, равное степени вершины v, учитывается при релаксации.

17. Домики. Будем формировать граф по следующему правилу. Соединим точки с координатами $(0, 0)$ и $(100, 100)$, получим отрезок $[A, B]$. У каждого домика определим его главную диагональ как отрезок, соединяющий левый верхний и правый нижний углы домика. Предположим, что существуют домики, главные диагонали которых отрезок $[A, B]$ пересёк (если нет пересечений, то ответом задачи является отрезок $[A, B]$). Если есть пересечения, то выберем из домиков тот, у которого одна из координат x или y наименьшая. Пусть у этого домика главная диагональ, которую пересёк отрезок $[A, B]$, равна $[C, D]$. Тогда заменим отрезок $[A, B]$ на четыре отрезка: $[A, C]$, $[C, B]$, $[A, D]$ и $[D, B]$. Затем с каждым из отрезков поступим аналогичным образом: если он ничего не пересекает, то добавим его в граф, в противном случае заменим его четырьмя отрезками. Процесс продолжается до тех пор, пока не станет отрезков, которые что-то пересекают. Теперь в построенном графе нужно найти кратчайший путь между стартовой и финишной точками.

18. Хакеры. Построим ориентированный граф: вершины соответствуют компьютерам, дуга (i, j) означает, что компьютер i имеет право запрашивать информацию, которая имеется у компьютера j.

Очевидно, что если компьютеры принадлежат одной сильносвязной компоненте, то для захвата всех этих компьютеров хакерам достаточно захватить только один из них (по условию это будет компьютер данной компоненты, у которого минимальное число терминалов). Поэтому если все вершины каждой сильносвязной компоненты заменить одной «сжатой» вершиной (все дуги, которые входили в вершины компоненты, будут теперь входить в эту «сжатую» вершину, а дуги, которые выходили из вершин этой компоненты, будут теперь выходить из этой «сжатой» вершины), то для захвата всей сети достаточно захватить вершины, полустепень захода которых равна нулю. Для выделения сильносвязных компонент ориентированного графа можно использовать алгоритм, основанный на двух запусках поиска в глубину.

19. Пирамида Хеопса. Задача сводится к поиску кратчайшего пути в орграфе, который можно найти, используя одну из реализаций

алгоритма Дейкстры. Перемещение между комнатами в некоторый момент времени возможно, если в них располагаются части одного устройства и в данный момент времени обе части активны (для определения момента времени, когда обе части устройства i активны, нужно найти наименьшее общее кратное времени срабатывания модулей этого устройства, т. е. НОК$(T_{1,i}, T_{2,i})$).

20. Без левых поворотов. По входным данным задачи построим ориентированный граф, заменив каждую дорогу двумя противоположно направленными дугами. Поскольку искомый маршрут может проходить через некоторые вершины несколько раз, а по одной дуге проезжать несколько раз нецелесообразно, то для поиска искомого маршрута можно использовать поиск в глубину или поиск в ширину, в которых блокируются не пройденные вершины, а дуги.

В задаче запрещён поворот налево. Предположим, что на плоскости заданы три точки $A(x_1, y_1)$, $B(x_2, y_2)$ и $C(x_3, y_3)$ и что движение осуществлялось по прямолинейному отрезку из точки A в точку B, а затем в точку C. Тогда, если определитель

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

больше нуля, то обход точек в порядке A, B, C осуществлялся против часовой стрелки (поворот налево при движении от точки B в точку C). Если определитель меньше нуля, то обход точек в порядке A, B, C происходил по часовой стрелке (поворот направо при движении от точки B в точку C). Если определитель равен нулю, то точка C лежит на прямой, соединяющей точки A и B.

21. Кратчайший маршрут с дополнительным временем преодоления перекрёстка. По входным данным построим граф, не содержащий петель и кратных рёбер. Для поиска кратчайшего маршрута между заданной парой вершин можно использовать алгоритм Дейкстры, в котором при пересчёте метки вершины u из вершины v необходимо учитывать степень вершины v.

22. Железные и шоссейные дороги. По входным данным построим взвешенный ориентированный граф: вершинам будут соответствовать города, а каждому типу дорог, соединяющему два города, поставим в соответствие две противоположно направленные дуги. Таким

образом, если два города соединены железной и шоссейной дорогами, то в орграфе вершины, соответствующие этим городам, будут соединены четырьмя дугами. Поскольку в оптимальном решении через некоторый город можно проехать несколько раз, то при поиске кратчайшего маршрута можно блокировать дуги. Для построения кратчайшего маршрута можно применить алгоритм Дейкстры с использованием приоритетной очереди, в котором нужно учитывать время пересадки с одного типа дорог на другой.

23. Кратчайший путь в зависимости от направления поворота. По входным данным построим ориентированный взвешенный орграф. Поскольку в оптимальном маршруте через некоторую вершину орграфа можно проехать несколько раз, а по одной дуге — не более одного раза, то при выборе реализации алгоритма Дейкстры, использующей приоритетную очередь, блокировать нужно не вершины, а дуги. Для определения направления поворота можно применить правило, описанное при разборе задачи 20. Без левых поворотов.

24. Уличная гонка. Легко заметить, что точка является неизбежной тогда и только тогда, когда её удаление ведёт к невозможности добраться от старта до финиша. Все точки, в которые можно попасть после удаления некоторой заданной, можно найти с помощью поиска в глубину (ширину). Пусть A — множество точек, в которые можно попасть со старта, если удалить точку k ($0 < k < N$), B — множество всех остальных точек (включая точку k).

Точка k является неизбежной тогда и только тогда, когда финиш (т. е. точка N) лежит в B.

Точка k является точкой разбиения тогда и только тогда, когда выполняются оба условия:

1) нет стрелок от точек из B к точкам из A;

2) нет стрелок от точек из A к точкам из B, за исключением стрелок, ведущих в точку k. Стрелки от точки k к ней же самой допускаются, так как их можно включить в план для второго дня гонки.

Заметим, что точка разбиения всегда является неизбежной точкой.

25. Прогулка. Построим двудольный граф: вершины первой доли соответствуют отрезкам ломаной (каждый отрезок ломаной — одна вершина первой доли), вершины второй доли — любимым местам собаки; вершина первой доли i смежна с вершиной второй доли j, если собака

успевает посетить точку j, пока хозяин проходит отрезок ломаной, соответствующий вершине i. Теперь для решения задачи в двудольном графе нужно найти паросочетание наибольшей мощности (одно и то же любимое место собака могла бы посетить многократно, но это не увеличит общего числа посещённых ею точек). Для построения наибольшего паросочетания двудольный граф нужно достроить до сети: введём полюс s, который соединим дугами со всеми вершинами первой доли; введём полюс t, с которым соединим дугами все вершины второй доли; сориентируем рёбра двудольного графа дугами по направлению от первой доли ко второй; верхние пропускные способности всех дуг положим равными единице. Теперь для нахождения наибольшего паросочетания воспользуемся алгоритмом построения максимального потока в сети. Дуги двудольного графа, по которым поток равен единице, будут соответствовать рёбрам наибольшего паросочетания (ребро наибольшего паросочетания указывает, какое любимое место посещает собака, пока хозяин проходит соответствующий отрезок ломаной).

26. Сборка прибора. Одной из возможных математических моделей для данной задачи является оптимальная (по длине) укладка корневого дерева. Как показано в [6], для минимизации длины укладки корневого дерева T необходимо для любой внутренней вершины дерева сначала выполнить оптимально укладку её поддеревьев в порядке, когда первым укладывается поддерево с наибольшим количеством вершин, а последним — с наименьшим.

27. Скрудж Мак-Дак. Одной из возможных математических моделей для данной задачи является оптимальная (по ширине) укладка корневого дерева (подробнее см. в [6]).

28. Заправочные станции. Построим по входным данным взвешенный орграф (D, w) (стоимость дуги — протяжённость соответствующей дороги). Нетрудно заметить, что искомый маршрут из A в B может быть представлен в виде последовательности подмаршрутов. Каждый подмаршрут начинается с полного бака машины, следует без дозаправок через города (города могут повторяться и быть как с заправочными станциями, так и без них) и заканчивается городом с заправочной станцией, в котором выполняется дозаправка бака машины (за исключением конечного города B, в котором дозаправка не обязательна). Поскольку по условию задачи стоимость дозаправки не зависит от количества до-

ливаемого в бак бензина, то стоимость некоторого (u, v)-подмаршрута определим как стоимость дозаправки в его конечном городе v.

Поэтому в задаче можно ограничиться рассмотрением взвешенного орграфа (D', w'), вершины которого $V(D')$ соответствуют начальному городу A, конечному городу B и конечным городам подмаршрутов (города с заправочными станциями). Пусть $V''(D') = V'(D') \setminus \{A, B\}$. Соединим вершину A дугами со всеми вершинами из $V'(D') \setminus A$, все вершины множества $V''(D')$ соединим дугами с вершиной B, а каждую вершину из множества $V''(D')$ соединим дугами со всеми оставшимися вершинами этого множества. Дуга (u, v) орграфа (D', w') соответствует (u, v)-подмаршруту, а вес дуги $w'(u, v)$ — стоимость (u, v)-подмаршрута (понятно, что такой маршрут предполагает достаточное количества бензина).

Опишем правила вычисления стоимостей дуг орграфа (D', w'). Поскольку в городе B дозаправка не обязательна, то для определения стоимостей дуг, входящих в вершину B, достаточно найти кратчайшие маршруты из B в вершины множества $V'(D') \setminus B$ в графе (D, w). Затем для каждого из этих маршрутов определим, хватит ли на его преодоление бака бензина. В [4] показано, что для дуг (u, v), $v \neq B$, орграфа (D', w') справедливо следующее утверждение: если расход бензина на кратчайший (u, v)-маршрут в графе (D, w) составил более половины бака, то в этом случае по условию задачи можно заправиться в городе v; если расход бензина не превышает половины бака, то можно «сжечь» бензин, дублируя нужное число раз некоторую из частей кратчайшего (u, v)-маршрута, и затем заправиться в городе v; если расход бензина более половины бака, то стоимость соответствующей дуги положим равной ∞. Теперь задача сводится к поиску во взвешенном орграфе (D', w') кратчайшего по стоимости (A, B)-маршрута.

29. Слова. Для данной задачи важно правильно построить её математическую модель. Если по входным данным задачи построить орграф, в котором вершины соответствуют словам, а дуга соединяет два слова, которые можно назвать друг за другом, то задача сведётся к определению гамильтоновости орграфа. Данная задача является NP-трудной и для неё не существует эффективного алгоритма решения.

Если же в качестве вершин взять различные первые и последние буквы слов, а две буквы соединить дугой, если есть слово, начинающееся с первой буквы и заканчивающееся второй, то получится ориентирован-

ный мультиграф. Задача сведётся к определению эйлеровости ориентированного мультиграфа и может быть решена за полиномиальное время. Для существования эйлерова контура основание ориентированного мультиграфа должно быть связным и для каждой вершины полустепень захода должна равняться полустепени исхода.

Всего есть 26 строчных латинских букв, значит, в псевдоорграфе будет не более 26 вершин. Буквы можно пронумеровать подряд целыми числами от 1 до 26. Для хранения слов можно использовать матрицу размера 26×26, в которой элемент в i-й строке j-го столбца является списком слов, начинающихся с буквы номер i и заканчивающихся на букву номер j.

30. Цепочка из домино. Построим псевдограф (возможны кратные рёбра и петли). В качестве вершин возьмём различные количества точек на концах доминошек (количество вершин в графе не более семи). Вершины i и j соединим ребром $\{i, j\}$, если есть доминошка, у которой количество точек на концах равно соответственно i и j. Тогда наша задача сведётся к определению эйлеровости графа и может быть решена за полиномиальное время (для того чтобы в псевдографе существовал эйлеров цикл, псевдограф должен быть связный и степени всех его вершин должны быть чётными).

31. Степенная последовательность. Для каждой вершины подсчитать число инцидентных ей рёбер.

32. Отрезки. В графовой постановке — для взвешенного двудольного графа (вершины первой доли соответствуют красным точкам, второй — синим, вес ребра — расстояние между точками) — задано некоторое совершенное паросочетание. Необходимо определить, является ли это паросочетание минимальным по длине среди всех совершенных паросочетаний.

Для решения данной задачи заменим каждое ребро паросочетания дугой, ориентированной от первой доли ко второй, и присвоим ей вес, равный расстоянию между этими точками, взятый со знаком минус. Каждое ребро, не принадлежащее паросочетанию, заменим дугой, ориентированной от второй доли к первой, и присвоим ей вес, равный расстоянию между этими точками. Поскольку орграф двудольный, то если в нём и есть контуры, их длина чётна. Следовательно, если в графе существует контур, то половина дуг в нём соответствует рёбрам паросочетания, а половина — нет.

Используя, например, алгоритм Форда — Беллмана, определим, есть ли в орграфе контур отрицательного веса. Если контура отрицательного веса не существует, то текущее паросочетание является искомым.

33. Трубопроводы. В графовой постановке задача сводится к построению максимального потока минимальной стоимости в сети. Для построения сети из графа необходимо каждое ребро графа заменить двумя противоположно направленными дугами с одинаковой пропускной способностью и стоимостью транспортировки.

34. Таксист. Для решения задачи сначала определим для каждой вершины v две метки: $m_1[v]$ — длину кратчайшего пути из вершины 1 в вершину v и $m_2[v]$ — длину кратчайшего пути из вершины v в вершину N. Для формирования меток достаточно запустить два раза алгоритм Дейкстры (из вершины 1 и из вершины N). На основании меток $m_1[u]$ и $m_2[v]$ для дуги (u, v) можно ответить на вопрос, может ли её использовать таксист.

35. Стрельба. Цель на соревнованиях по стрельбе можно проинтерпретировать как матрицу инцидентности графа (белые квадраты заменить числом 1, а чёрные — 0). Теперь необходимо определить, можно ли придать всем рёбрам графа ориентацию так, чтобы в каждую вершину графа входило не менее одной дуги.

Для решения данной задачи нужно, чтобы в каждой компоненте связности графа был цикл, так как в противном случае не существует корректной последовательности выстрелов (будет существовать вершина с нулевой полустепенью захода). Для поиска циклов можно использовать поиск в глубину или ширину.

Предположим, что в каждой компоненте связности графа есть цикл. Тогда, стартуя из любой вершины цикла, можно сориентировать рёбра графа, используя, например, поиск в глубину (строится корневое дерево поиска в глубину).

Другое решение этой задачи основано на алгоритме поиска наибольшего паросочетания в двудольном графе. Вершины первой доли соответствуют столбцам (C штук), второй доли — строкам (R штук). Из каждой вершины первой доли выходят ровно две дуги, соответствующие номерам строк с белыми квадратами.

36. Инкассаторы. Поставим в соответствие системе дорог взвешенный граф A, вершины которого соответствуют городам, а рёбра —

дорогам. Назовём отмеченными вершины, соответствующие городам, в которых расположены банки, а также городу, в котором расположен банк Макса Крейзи. Построим полный взвешенный граф B, вершины которого соответствуют отмеченным вершинам графа A, а длина каждого ребра равна кратчайшему маршруту между соответствующими вершинами графа A (сами кратчайшие маршруты тоже нужно сохранять). Тогда легко показать, что в графе A не существует маршрута, проходящего по всем отмеченным вершинам и имеющего длину меньшую, чем длина кратчайшего из гамильтоновых циклов графа B.

Таким образом, задача сводится к поиску в полном графе B гамильтонова цикла наименьшей длины. Поскольку граф полный, то эта задача эквивалентна поиску перестановки, для которой значение некоторой функции должно быть минимально. Для решения данной задачи удобно использовать динамическое программирование.

37. Город *C*. Данная задача имеет достаточно простое решение с учётом того, что по условию задан граф (а не орграф). Построим по графу сеть: введём источник s, соединим его дугами с городами A и B; в качестве стока t возьмём город C; каждое ребро заменим двумя противоположно направленными дугами; каждую вершину v (за исключением A, B и C) заменим на две v' и v'', соединив их дугой (v', v''); все дуги, входящие в вершину v, направим в v', а дуги, выходящие из вершины v, направим из v''; пропускные способности всех дуг положим равными единице.

Далее нужно применить алгоритм нахождения максимального потока в сети (если в сети существует поток мощности два, то можно проехать из города A в город B таким образом, чтобы посетить город C, не проезжать ни по какой дороге более одного раза и не заезжать ни в какой город более одного раза).

38. Расселение. Построим двудольный граф с долями X и Y: каждому сотруднику первого отдела будет соответствовать вершина из множества X, а второго — из Y; в случае если в одном из отделов оказалось сотрудников меньше, чем в другом, добавим в долю меньшей мощности фиктивные вершины; соединим каждую вершину первой доли рёбрами со всеми вершинами второй доли и присвоим стоимость, равную «показателю недовольства».

Теперь в полном двудольном графе нужно найти наибольшее паросочетание минимальной стоимости, а поскольку двудольный граф

является полным, то в качестве начального паросочетания можно взять совершенное паросочетание, соединив каждую вершину первой доли с произвольной вершиной второй доли, которая ещё не покрыта паросочетанием [4].

39. Сеть дорог. В сети, построенной по входным данным, нужно найти максимальный поток минимальной стоимости. Поскольку по условию задан максимальный поток f, то по этому потоку строим сеть остаточных пропускных способностей G_f, в которой прямым дугам (u, w) сети G_f поставим удельную стоимость $p(u, w)$, а обратным дугам (w, u) — удельную стоимость, равную $-p(u, w)$.

Пока в сети G_f существует контур отрицательной удельной стоимости, перераспределим поток вдоль этого контура, получая каждый раз поток f той же мощности, но меньшей удельной стоимости. Если на некотором шаге в сети остаточных пропускных способностей G_f нет контура отрицательной удельной стоимости, то текущий поток f является максимальным потоком минимальной стоимости.

40. Боеприпасы. По входным данным построим двудольный граф: вершины первой доли будут соответствовать определённой единице боеприпасов, вершины второй доли — различным дням; две вершины соединены ребром, если соответствующую единицу боеприпасов можно уничтожить в соответствующий день, а вес этого ребра — степень риска.

Далее применим алгоритм нахождения наибольшего паросочетания минимальной стоимости в двудольном графе: сначала в графе найдём некоторое наибольшее паросочетание (любая модификация алгоритма Куна) и, если оно не является совершенным, уничтожить все боеприпасы нельзя.

Если в графе найдено совершенное паросочетание, то построим по нему специальный взвешенный орграф, в котором при наличии контура отрицательного веса перестроим текущее паросочетание вдоль этого контура, получая паросочетание той же мощности, но меньшей стоимости.

41. Граф? Мультиграф? Псевдограф? По определению заданный на входе объект является графом тогда и только тогда, когда в нём нет ни петель, ни кратных рёбер (первый вопрос). Объект является мультиграфом тогда и только тогда, когда в нём нет петель (второй вопрос). Псевдографом объект является всегда (третий вопрос).

42. Экскурсия. По входным данным задачи строится сеть, в которой нужно найти максимальный поток минимальной стоимости.

43. Машина времени. Построим вспомогательный орграф, где в качестве вершин возьмём рейсы, а дуга (u,v) в орграфе возможна в том случае, если конечный пункт рейса u и начальный пункт рейса v совпадают, а время отправления рейса v не меньше времени прибытия рейса u. При построении орграфа будем учитывать некоторый фиктивный рейс s, прибывающий в начальный пункт A в момент времени нуль.

Теперь поиском в ширину (или глубину) найдём все достижимые из s вершины, а затем выберем среди них ту, которой соответствует рейс с минимальным временем прибытия в пункт B.

44. Максимальное число коней. Построим граф, в котором вершинами являются клетки шахматной доски, а рёбра соединяют пары клеток, между которыми возможен ход конём (буквой «Г»). Нетрудно заметить, что этот граф является двудольным (один ход коня меняет чётность суммы координат клетки, поэтому в первую долю можно отнести клетки доски, у которых сумма номера строки и столбца нечётна, во вторую долю — у которых сумма чётна). Клетки, занятые чёрными фигурами, в граф не включаются.

Суть задачи — в полученном графе найти наибольшее независимое множество (выбрать подмножество вершин максимального размера, чтобы никакие две вершины не были соединены ребром).

В общем случае эта задача NP-полна, но для двудольных графов она может быть решена эффективно (см. теорему Кёнига и алгоритм поиска наибольшего паросочетания).

45. Военный поход. По входным данным задачи построим взвешенный граф. Очевидно, что можно без ограничения общности сразу продать все имеющиеся у нас дороги, а затем купить их заново, если необходимо.

В графе найдём кратчайший по стоимости путь P между городами 1 и N, используя, например, алгоритм Дейкстры (при проходе через вершину учитываем стоимость прохода через неё). Если суммы денег после продажи всех дорог хватает на приобретение дорог пути P, то задача имеет решение.

46. План эвакуации. По входным данным задачи построим сеть. Введём фиктивную вершину s, которую соединим дугами со всеми

вершинами, соответствующими муниципальным зданиям; пропускные способности этих дуг положим равными числу людей в соответствующем здании, а удельную стоимость по дуге — равной нулю.

Введём фиктивную вершину t, с которой соединим дугами все вершины, соответствующие убежищам; пропускные способности этих дуг положим равными числу людей, которое может принять данное убежище, а удельную стоимость по дуге — равной нулю.

Все муниципальные здания соединим дугами с убежищами, пропускную способность этих дуг положим равной большому числу (бóльшему, чем общее число рабочих во всех зданиях), а удельную стоимость по дуге — расстоянию между соответствующим зданием и убежищем.

По условию задачи в сети задан некоторый максимальный поток и необходимо проверить, существует ли поток той же мощности, но меньшей удельной стоимости. Для решения данной задачи по текущему максимальному потоку достаточно построить сеть остаточных пропускных способностей, в которой если существует контур отрицательной удельной стоимости, то текущий план эвакуации не является оптимальным.

Используя контур отрицательной удельной стоимости, можно перераспределить вдоль него максимальный поток, получая поток той же мощности, но меньшей стоимости.

Для поиска контура отрицательной удельной стоимости можно использовать, например, алгоритм Форда — Беллмана. Если для текущего максимального потока в сети остаточных пропускных способностей нет контуров отрицательного удельного веса, то заданный по условию план эвакуации является оптимальным.

47. Королевское задание. По входным данным построим двудольный граф из $2N$ вершин: N вершин первой доли соответствуют сыновьям, а N вершин второй доли — девушкам, ребро соединяет вершину первой доли с вершиной второй доли, если соответствующий сын выражает симпатию к данной девушке. По условию задачи в двудольном графе задано совершенное паросочетание. Если ориентировать рёбра, входящие в паросочетание, от сыновей к девушкам, а остальные рёбра — от девушек к сыновьям, то, чтобы сохранить совершенное паросочетание и при этом назначить некоторому сыну другую девушку, необходимо и достаточно найти контур в построенном орграфе, проходящий через дугу, назначенную этому сыну, и дугу, которую мы хотим

ему назначить. Таким образом, чтобы для вершины x (некоторого сына) проверить вершину y (может ли он жениться на девушке, соответствующей y), достаточно проверить, существует ли контур, проходящий через (x,y), что равносильно тому, что x и y принадлежат одной компоненте сильной связности. Для выделения сильносвязных компонент орграфа можно использовать алгоритм Касарайю — Шарира, основанный на двух запусках поиска в глубину.

48. Платформы. По входным данным задачи построим сеть. Введём два полюса s и t. Каждой из стартовых и финишных платформ будет соответствовать по одной вершине орграфа. Соединим полюс s дугами со всеми вершинами, соответствующими стартовым платформам; пропускную способность этих дуг положим равными единице (так как на каждой стартовой платформе по условию лежит только один шарик). Соединим с полюсом t дугами все вершины, соответствующие финишным платформам; пропускную способность этих дуг положим равными единице (так как на каждой финишной платформе может оказаться не больше одного шарика). Каждой промежуточной платформе будут соответствовать две вершины сети, соединённые дугой с пропускной способностью, равной числу шариков, которые допустимо пропустить через эту платформу (первая из них будет соответствовать началу платформы, вторая — её концу). Трубам, которые по условию задачи соединяют платформы между собой и по которым может скатиться шарик, в сети соответствуют следующие дуги: ведущие от начальных платформ к началам промежуточных; ведущие от концов промежуточных платформ к финишным платформам; ведущие от концов промежуточных платформ к началам промежуточных платформ; пропускную способность всех этих дуг положим равными большому числу (бóльшему, чем число N). Теперь для нахождения ответа достаточно в построенной сети найти максимальный поток из s в t, воспользовавшись для этого методом Форда — Фалкерсона.

49. Безопасные пути. По входным данным построим граф G, рёбрам которого приписаны метки 1, 2 или 3 — статус дороги.

На первом этапе выполним поиск в глубину в графе G, осуществляя движение только по дорогам со статусом 3 (нейтральные дороги, по которым могут ездить все). Во время поиска в глубину будет построен набор корневых деревьев T_i^3. Стянем каждое корневое дерево T_i^3 в одну вершину. Обозначим полученное множество вершин через V'.

На втором этапе построим граф G^1: соединим попарно вершины из множества V' ребром, если существует дорога со статусом 1 (охраняемая дорога), соединяющая в исходном графе G вершины, принадлежащие соответствующим корневым деревьям.

На третьем этапе построим граф G^2: соединим попарно вершины из множества V' ребром, если существует дорога со статусом 2 (дорога бандитов), соединяющая в исходном графе G вершины, принадлежащие соответствующим корневым деревьям.

На заключительном этапе выполним поиск в глубину для графов G^1 и G^2. Если хотя бы один из них несвязный, то решения исходной задачи не существует. Если графы связные, то построенные корневые деревья поиска в глубину обозначим через T^1 и T^2 соответственно. Объединим рёбра деревьев T^1, T^2 и семейства корневых деревьев T_i^3 — это дороги, которые нужно оставить, а все оставшиеся дороги можно удалить.

50. Светофоры. Поскольку на каждый перекрёсток необходимо прибыть в максимально раннее время, то для решения задачи можно применить алгоритм Дейкстры.

51. Достроить дороги. Задача сводится к определению числа компонент связности в мультиграфе. Для решения данной подзадачи можно использовать алгоритмы поиска в ширину или глубину.

52. Минимальная плата за проезд. По входным данным построим граф, вершинам которого будут соответствовать города, а рёбрам — дороги между соответствующими городами. Исходя из правил, которым удовлетворяет система дорог, можно утверждать, что рассматриваемый граф является деревом. Каждому ребру графа сопоставим вес — стоимость проезда по соответствующей дороге. Тогда исходная задача формулируется в терминах теории графов следующим образом: для каждой пары вершин v и w (запросы) найти стоимость (v, w)-маршрута.

Рассмотрим эффективный алгоритм решения данной задачи. Выберем произвольную вершину r и, используя поиск в глубину, построим корневое дерево поиска с корнем в вершине r. В каждую вершину дерева существует единственный путь из корня r (путь зададим последовательностью входящих в него вершин). Вершину i будем называть *предком* вершины j, если вершина i лежит на пути из корня дерева r в вершину j. Рассмотрим некоторую пару вершин v и w. Пути из корня r к каждой из них имеют непустое пересечение (во всяком случае, в него

входит корень). Вершины из такого пересечения путей будем называть *общими предками* пары вершин v и w. Назовём *самым низким общим предком* (*Lowest Common Ancestor*, *LCA*) пары вершин v и w их общего предка, находящегося на наибольшей глубине (т. е. на наибольшем удалении от корня дерева r). Предположим, что мы научимся для пары вершин v и w находить их наименьшего общего предка p. Путь между парой вершин v и w (а он, как указано, единственный) можно получить как объединение путей от p до v и от p до w. Теперь если найти стоимости путей из корня дерева r до всех вершин дерева, то стоимость (v, w)-маршрута может быть найдена как сумма стоимостей маршрутов (s, v) и (s, w) минус удвоенная стоимость (s, p)-маршрута.

Существует большое количество алгоритмов поиска *LCA* для заданных пар вершин. Некоторые из них имеют линейные относительно количества вершин и количества запросов временные оценки, однако константа при таких оценках бывает весьма значительной, а сами алгоритмы — достаточно сложны. Решим задачу, исходя из уже известных пар запросов, для которых нужно вычислить наименьшего общего предка (офлайн-задача). Для решения задачи воспользуемся алгоритмом Р. Тарьяна, который имеет чуть худшую, чем линейная, оценку времени работы [2]. Будем считать вершину v посещённой при её занесении в стек и просмотренной при её удалении из стека.

Алгоритм Р. Тарьяна
(поиск LCA)

1. Выберем в качестве корня произвольную вершину дерева r.

2. Выполним обход в глубину из выбранного корня r. Вершины, которые поиск в глубину когда-либо посещал, объединяются во множества (образующие систему непересекающихся множеств) таким образом, что вершины одного множества имеют одного и того же наименьшего общего предка (имя множества определяется как наименьший общий предок всех элементов этого множества).

Для построения описанной системы непересекающихся множеств выполним следующую последовательность шагов:

- при входе в вершину v (занесении её в стек) создаётся множество, состоящее из единственного элемента v, а имя множества полагается равным v;
- после обхода очередного потомка w вершины v (w удаляется из стека и считается просмотренной) объединяются два множества,

соответствующие вершине v и её потомку w, а объединённому множеству присваивается имя v;

• если вершина w становится просмотренной, то просматриваются все запросы, в которые входит вершина w; и если другая вершина u из этого запроса уже была просмотрена, то ответом на запрос является имя множества, которому принадлежит вершина u.

Следует отметить, что можно сразу раздать запросам номера и записывать ответ по известному номеру. Для этого при чтении исходных данных каждой вершине ставится в соответствие список запросов, в которых она участвует, с указанием второй вершины и номера запроса, а когда становится известен ответ на запрос, он записывается в массив ответов в позицию, соответствующую номеру запроса, и таким образом список ответов формируется сразу в правильном порядке.

При реализации множеств в виде семейства корневых деревьев с эвристиками сжатия путей и объединения по рангам алгоритм Тарьяна работает за $O((n+m)\cdot\alpha(n+m,n))$, где n — количество вершин дерева; m — количество запросов; $\alpha(i,j)$ — очень медленно растущая обратная функция Аккермана (её значение меньше пяти даже для оценки числа атомов в наблюдаемой части Вселенной — числа порядка 10^{80}).

53. Максимальное ребро. Предположим, что p — наименьший общий предок для вершин v и w. Путь между парой вершин v и w (а он, как указано, единственный) можно получить как объединение путей от p до v и от p до w. Если известен вес максимального ребра на каждом из этих двух путей, то ответом на вопрос задачи, очевидно, будет являться максимум из этих весов.

Таким образом, возникают две подзадачи: эффективное вычисление наименьшего общего предка для заданных пар вершин (для решения подзадачи можно воспользоваться алгоритмом Р. Тарьяна [2], который подробно описан в указаниях к решению на предыдущую задачу) и нахождение веса максимального ребра на пути от вершины к её предку. Во время поиска в глубину при входе в вершину вычисляются её предки, отстоящие от неё на расстояние 2^0, 2^1, …, $2^{\lfloor\log n\rfloor+1}$ рёбер, а если некоторого предка не существует, то соответствующее значение полагается равным корню дерева поиска в глубину). Параллельно вычисляется вес максимального ребра, лежащего на пути от соответствующего предка к данной вершине.

Для нахождения ответа на запрос производится подъём от каждой

вершины из пары к наименьшему общему предку шагами, составляющими степени 2, и окончательное вычисление ответа.

54. Ровно K. Для решения задачи можно использовать алгоритм, основанный на алгоритме поиска в ширину в графе (поиск в ширину выполняется из вершины 1). Поскольку по условию задачи искомый маршрут не является цепью, то если для каждой вершины v мы найдём длину наименьшей (s, v)-цепи как чётной, так и нечётной длины, то затем сможем за константное время ответить на каждый запрос о возможности партии (по ребру можно пройти два раза, увеличив длину некоторого маршрута на два). Для решения этой подзадачи достаточно во время поиска в ширину отдельно выполнить блокировку вершины v при её первом занесении в очередь как по чётной, так и по нечётной длине (s, v)-маршрута.

Другой подход к решению задачи заключается в построении нового графа путём раздвоения вершин: для каждой вершины в новом графе создаются две вершины: «нечётная» и «чётная». Пусть в исходном графе есть вершины u и v, соединённые ребром $\{u, v\}$. В новом графе вершинам u и v соответствуют четыре вершины u', u'', v', v'', между которыми проходят два ребра $\{u', v''\}$ и $\{u'', v'\}$ (при движении фишки по ребру исходного графа меняется чётность длины маршрута). Затем выполним поиск в ширину из «чётной» копии вершины 1.

55. Надёжная сеть. Для решения задачи необходимо определить, существует ли в графе между заданной парой вершин $k + 1$ простых цепей, которые попарно не пересекаются по вершинам и суммарная длина которых минимальна. Данная задача может быть сведена к поиску в сети максимального потока минимальной стоимости. Для решения задачи также можно применить алгоритм, приведённый в [6].

56. Острова. Для того чтобы искомая сеть дорог была оптимальна, она должна представлять из себя дерево, у которого в качестве листьев могут быть только главные острова. В дереве возможны два или три листа. Каждая вершина этого дерева, за исключением листьев, имеет степень 2, возможно, кроме одной вершины со степенью 3 (эта вершина соединена простыми цепями с каждым из трёх главных островов). Если в дереве два листа, то дерево вырождается в простую цепь, соединяющую некоторые два главных острова, а третий остров лежит на этой цепи (рис. 1.32).

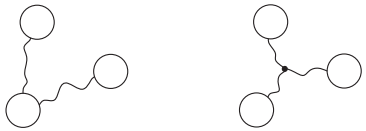


Рис. 1.32

Для решения задачи можно использовать алгоритм Дейкстры, запустив его для каждого главного острова ровно один раз. Теперь среди всех вершин графа необходимо выбрать ту, до которой суммарное расстояние от трёх главных островов минимально.

57. Снести. В задаче необходимо рассмотреть два варианта удаления вершин графа (хуторов).

1. Происходит удаление двух вершин графа из различных компонент связности, не являющихся изолированными вершинами. В этом случае допускается удаление только тех вершин, которые не являются точками сочленения.

2. Одна из удаляемых вершин — изолированная. Поскольку при удалении изолированной вершины количество компонент связности графа уменьшится на единицу, то тогда, чтобы выполнялось условие задачи, другой удаляемой вершиной должна быть точка сочленения, при удалении которой количество компонент связности возрастёт ровно на единицу.

Для поиска точек сочленения в графе можно использовать алгоритм, основанный на поиске в глубину. Во время поиска в глубину вершинам присваиваются две метки. Первая метка $m_1(v)$ присваивается вершине в момент её занесения в стек. Вторая метка окончательно формируется у вершины v в момент её удаления из стека. Вторая метка вершины $m_2(v)$ равна минимуму из следующих величин: первой метки самой вершины $m_1(v)$, первых меток $m_1(w)$, где (v, w) — обратная дуга в корневом дереве поиска в глубину, и вторых меток $m_2(u)$ непосредственных сыновей u вершины v в корневом дереве поиска в глубину. Если в момент удаления вершины v из стека верно неравенство

$$m_1(v) \leqslant m_2(v),$$

то вершина v — точка сочленения. Для корня дерева r проверка проходит по-другому: если у r не менее двух древесных дуг, то r — точка сочленения. Следует отметить, что алгоритм поиска точек сочленения

необходимо модифицировать так, чтобы он находил число, на которое увеличится количество компонент связности в исходном графе при удалении текущей точки сочленения.

58. Сталкер: туда и обратно. Построим сеть, выбрав в качестве полюсов точки M и G. Вершинам соответствуют комнаты, которые не замурованы. Рёбра (соединяют две комнаты, между которыми возможно сообщение) заменим двумя противоположно направленными дугами с единичной пропускной способностью и единичной удельной стоимостью. Теперь в сети найдём поток мощности два минимальной удельной стоимости.

59. Полёт с пересадками. Для построения кратчайшей простой цепи при условии, что в ней должно содержаться не более Q рёбер, можно использовать модификацию алгоритма Дейкстры [6].

60. Нефть. Введём два полюса s и t. Вершинам соответствуют вышки и заводы. Источник s соединим дугами со всеми вышками, поставим соответствующей дуге в качестве пропускной способности число, равное количеству тысяч баррелей нефти в сутки, которое может выкачать данная вышка, а удельную стоимость положим равной нулю. Со стоком t соединим дугами все перерабатывающие заводы, поставим соответствующей дуге в качестве пропускной способности число, равное количеству тысяч баррелей нефти в сутки, которое может переработать этот завод, а удельную стоимость дуги положим равной нулю. Соединим каждую вышку дугами со всеми заводами, поставив каждой такой дуге в качестве пропускной способности ∞, а удельную стоимость каждой дуги положим равной стоимости транспортировки одной тысячи баррелей нефти от соответствующей вышки к заводу. Теперь в сети найдём максимальный поток минимальной стоимости.

Часть 2

БИНАРНЫЕ ПОИСКОВЫЕ ДЕРЕВЬЯ

2.1. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

На практике часто приходится иметь дело со специальными видами деревьев. Наиболее распространённым среди них является корневое дерево (рис. 2.1).

Корневое дерево T — орграф, который удовлетворяет следующим условиям:

1) имеется в точности одна вершина, в которую не входит ни одна дуга (*корень* дерева T);

2) в каждую вершину, кроме корня, входит ровно одна дуга;

3) из корня дерева есть путь к каждой вершине дерева.

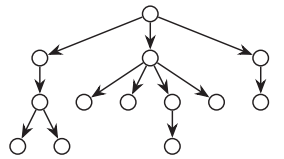


Рис. 2.1. Корневое дерево

Пусть (u, v) — дуга корневого дерева, тогда вершину u называют *отцом* вершины v, а вершину v — *сыном* вершины u. Если у вершины u нет сыновей, то говорят, что вершина u — *лист* корневого дерева (в литературе листья часто называют *висячими* или *терминальными* вершинами дерева). Вершины корневого дерева, отличные от листьев, называют *внутренними* вершинами.

Пусть u — некоторая вершина корневого дерева, тогда все вершины v, которые достижимы из вершины u (т. е. существует путь из u в v), называют *потомками* вершины u (u — *предок* v). Подграф, состоящий из всех потомков вершины u (включая вершину u) и исходящих из них дуг, называется *поддеревом с корнем в вершине* u и обозначается через T^u.

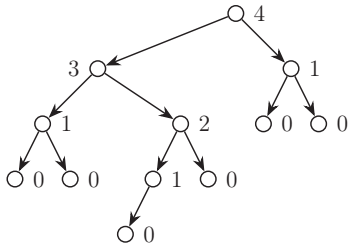


Рис. 2.2. Высоты вершин дерева

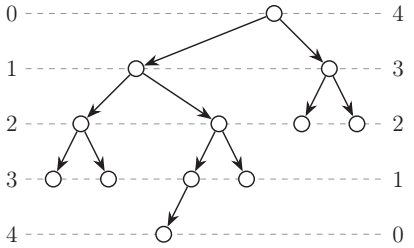


Рис. 2.3. Глубины (слева) и уровни (справа) вершин дерева

Высотой вершины v в корневом дереве T (рис. 2.2) называется длина наибольшего пути из вершины v до одного из его потомков (очевидно, что наибольший путь ведёт из вершины v в лист). Напомним, что длина пути определяется по количеству пройденных дуг.

Высота корневого дерева — высота корня. Высота дерева, состоящего из одной единственной вершины, полагается равной нулю.

Глубиной вершины v в корневом дереве T (рис. 2.3) называется длина пути (в дугах) из корня дерева в вершину v. Нетрудно увидеть, что этот путь единственный.

Уровнем вершины v (см. рис. 2.3) называется разность между высотой дерева T и глубиной вершины v.

Упорядоченное корневое дерево — это корневое дерево, у которого дуги, выходящие из каждой вершины, упорядочены (в дальнейшем будем считать, что они упорядочены слева направо).

Для корневых деревьев по разным критериям вводят *инварианты сбалансированности*, которые гарантируют, что все словарные операции (поиск элемента по заданному ключу, добавление элемента с заданным ключом, удаление элемента) выполняются за время $\mathrm{O}(\log n)$, где n — число вершин в дереве, и при этом поддержка инвариантов сбалансированности также выполняется за время $\mathrm{O}(\log n)$.

Корневое дерево называется *k-сбалансированным по высоте*, если для каждой вершины v высоты её максимального (по высоте) и минимального (по высоте) поддеревьев отличаются не более чем на k. Если $k = 1$, то будем говорить, что *дерево сбалансировано*.

Корневое дерево называется *k-сбалансированным по числу вершин*, если для каждой вершины v число вершин максимального (по числу вершин) и минимального (по числу вершин) поддеревьев отличаются

не более чем на k. Если $k = 1$, то будем говорить, что *дерево идеально сбалансировано*.

Каждое идеально сбалансированное дерево является сбалансированным, а вот обратное верно не всегда.

Бинарное дерево — это упорядоченное корневое дерево, у каждой вершины которого имеется не более двух сыновей. В бинарном дереве каждый сын произвольной вершины определяется как левый или правый.

Поддерево (если оно существует), корнем которого является левый сын вершины v, называется *левым поддеревом вершины* v. Аналогичным образом определяется *правое поддерево для вершины* v.

Предположим, что каждой вершине бинарного дерева соответствует некоторое ключевое значение (например, целое число). Бинарное дерево называется *деревом поиска* (*бинарным поисковым деревом*), если оно организовано так, что для каждой вершины v справедливо утверждение, что все ключи в левом поддереве вершины v меньше ключа вершины v, а все ключи в правом поддереве — больше.

В дальнейшем будем предполагать, что в бинарном поисковом дереве нет двух вершин с одинаковыми ключевыми значениями, если не оговорено иное.

Высота h бинарного поискового дерева из n вершин равна $\mathrm{O}(n)$ (максимальная высота достигается, когда вершины дерева вытянуты в линейный список). Высота h сбалансированного бинарного поискового дерева из n вершин равна $\mathrm{O}(\log n)$.

Рассмотрим бинарное поисковое дерево T, приведённое на рис. 2.4. Дерево T имеет высоту $h = 3$ и корень 20. Листьями дерева T являются вершины 1, 5, 13, 41 и 43. Вершина с ключом 42 имеет высоту 1, глубину 2, уровень 1 (на первом уровне располагаются вершины с ключами 4, 14 и 42).

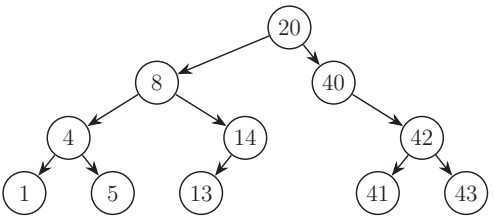


Рис. 2.4. Бинарное поисковое дерево

Дерево T, приведённое на рис. 2.4, не является сбалансированным, так как для вершины с ключом 40 не выполняется инвариант сбалансированности по высотам (для простоты высоту несуществующего поддерева полагаем равной -1).

Во многих задачах требуется найти *среднюю по значению из вершин* дерева. Средней по значению является та вершина, которой соответствует такое ключевое значение x, что количество вершин, имеющих ключевое значение строго меньше x, равно количеству вершин, имеющих ключевые значения строго больше x. В дереве, приведённом на рис. 2.5, средней по значению является вершина с ключом 14.

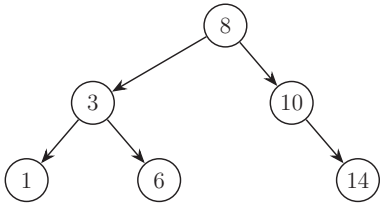


Рис. 2.5. Пример бинарного поискового дерева

Если количество вершин в дереве чётно, то будем считать, что средней по значению вершины не существует. Если же дерево состоит из единственной вершины, то будем полагать, что эта вершина является средней по значению. В дереве, приведённом на рис. 2.5, число вершин чётно, поэтому средней по значению вершины не существует.

2.2. ПРЕДСТАВЛЕНИЕ ДЕРЕВА В ПАМЯТИ КОМПЬЮТЕРА

Как правило, каждая вершина дерева хранится в памяти отдельно в виде структуры некоторого типа. Вершина ссылается на две другие вершины — на левого и правого сына. Если поддерево отсутствует, то соответствующая ссылка имеет нулевое значение.

Заметим, что если известна ссылка на корень, то с помощью прохода по ссылкам можно достичь всех вершин дерева. Поэтому, чтобы говорить о представлении дерева в памяти, не нужно создавать список или массив всех вершин дерева, а достаточно поддерживать только ссылку на корень. Остальные вершины доступны по связям от корня.

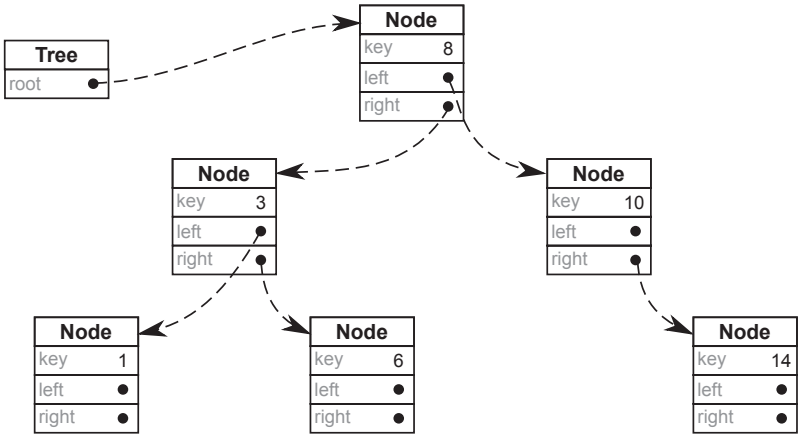


Рис. 2.6. Представление дерева 2.5 в памяти компьютера

Для бинарного поискового дерева, приведённого на рис. 2.5, его представление в памяти компьютера в виде списковой структуры продемонстрировано на рис. 2.6.

2.2.1. Класс вершины дерева

В программе на объектно ориентированном языке программирования каждую вершину дерева удобно представлять в виде объекта определённого класса. Будем именовать класс словом **Node** (англ. — узел, вершина).

В простейшем случае объект вершины содержит три поля:

1) ключ (число, которое хранится в вершине);

2) ссылка на левого сына (возможно, нулевая, если левое поддерево отсутствует);

3) ссылка на правого сына (возможно, нулевая, если правое поддерево отсутствует).

Если у вершины отсутствует левое или правое поддерево, соответствующую ссылку обычно устанавливают в нулевое значение, принятое в конкретном языке программирования (`None`, `null`, `NULL`, `nullptr`).

В зависимости от задачи можно добавлять те или иные поля в класс вершины. Например, это может быть целочисленное поле для хранения высоты вершины или для подсчёта числа вершин в поддереве с корнем в текущей вершине. Важно не забывать поддерживать актуальность этих

полей при изменении структуры дерева, если этого требует решаемая задача (например, после выполнения удаления изменилась высота).

2.2.2. Класс дерева

Кроме класса **Node** для вершины часто удобно создать класс **Tree**, представляющий всё дерево, который будет содержать единственное поле — ссылку на корень дерева. В зависимости от языка класс вершины для удобства можно объявлять внутри класса дерева (вложенным классом), потому что вершина обычно не используется сама по себе.

2.2.3. Пустое дерево

С точки зрения теории объект, в котором нуль вершин, не может называться деревом. Однако на практике при программировании хорошо и удобно, когда структура данных «бинарное поисковое дерево» имеет состояние по умолчанию, когда в дереве нет ни одной вершины (дерево представляет пустое множество ключей). Так, дерево можно создать изначально пустым, а затем по очереди добавить в него заданные ключи. Поддержка пустого состояния в классе дерева обычно требует некоторых усилий, что обусловлено удобством дальнейшей работы. Пустое дерево очевидным образом представляется объектом класса дерева, в котором поле вершины-корня нулевое.

2.2.4. Рекурсивные и нерекурсивные реализации операций

Алгоритмы для работы с бинарными поисковыми деревьями можно реализовывать рекурсивно или нерекурсивно (итеративно). Организация дерева имеет рекурсивный характер: у любой вершины бинарного поискового дерева её левое и правое поддеревья по отдельности являются бинарными поисковыми деревьями. При использовании рекурсии код, вероятно, получится немного короче, но может работать на практике несколько медленнее. На теоретическую трудоёмкость способ реализации не влияет (один и тот же алгоритм записывается разными способами).

2.2.5. Псевдокод

Для изложения алгоритмов мы будем использовать код на псевдоязыке, имеющем сходство с языком программирования Python. Действительно, исходный код программ на Python можно рассматривать как

псевдокод: обычно он прост и понятен даже тем, кто с языком не сталкивался. В псевдокоде оператор присваивания имеет вид `=`. Проверка двух величин на равенство выполняется оператором `==`. Ключевое слово `def` используется для объявления функций. Проверка условия осуществляется при помощи оператора `if`. Цикл с предусловием записывается через оператор `while`. Для группировки операторов используются горизонтальные отступы.

Пусть переменная `x` в коде хранит целое число x. Создание вершины v с ключом x будем записывать так: `v = Node(x)`. В результате выполнения такого кода переменная `v` будет содержать ссылку на объект, который представляет вершину v в памяти компьютера. Ключ вершины v будет доступен по имени `v.key`, левый сын будет обозначаться через `v.left`, правый сын — через `v.right`. Нулевая ссылка имеет специальное значение `None`.

Корень дерева обозначим `tree.root`.

2.3. ПОИСК КЛЮЧА В ДЕРЕВЕ

Отыскание в бинарном поисковом дереве заданного ключа x может быть реализовано так: выполняется спуск по дереву, при этом всякий раз направление дальнейшего движения (влево или вправо) выбирается исходя из того, меньше или больше искомый ключ текущего ключа в вершине. Начинаем с проверки корня дерева. Если дерево пустое, то ключ, который мы ищем, в дереве отсутствует. Иначе, если ключ совпадает с ключом корня, поиск завершился успешно и мы возвращаем вершину. Если искомый ключ меньше ключа корня, продолжаем поиск в левом поддереве. Если ключ больше ключа в корне, ищем в правом поддереве. Процесс повторяется, пока ключ не будет найден или мы не придём к нулевому (отсутствующему) поддереву (на рис. 2.7 путь, пройденный при поиске ключа 6, выделен).

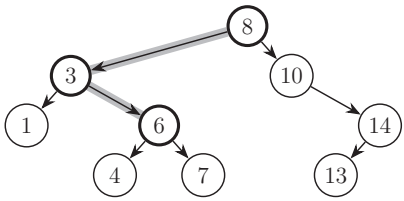


Рис. 2.7. Поиск ключа 6 в дереве

2.3.1. Рекурсивная реализация

Описанный алгоритм можно легко реализовать рекурсивно. Создаётся функция, принимающая на вход ссылку на вершину `v`, с которой будет начинаться поиск, и ключ `x`, который нужно найти. Поиск выполняется в поддереве, корнем которого является указанная стартовая вершина (возможно, поддерева не существует, если этот корень нулевой). В качестве результата функция возвращает ссылку на найденную вершину или `None`, если ключа в поддереве нет.

```
def SearchRecursively(v, x):
    if v is None or v.key == x:
        return v
    elif x < v.key:
        return SearchRecursively(v.left, x)
    else:  # x > v.key
        return SearchRecursively(v.right, x)
```

Для поиска ключа во всём дереве при первом вызове указанной функции следует передавать ей в качестве вершины `v` корень дерева:

```
result = SearchRecursively(tree.root, x)
```

2.3.2. Нерекурсивная реализация

Тот же алгоритм может быть реализован нерекурсивно при помощи цикла.

```
def SearchIteratively(x):
    v = root
    while v is not None:
        if v.key == x:
            return v
        elif x < v.key:
            v = v.left
        else:  # x > v.key:
            v = v.right
    return None
```

2.4. ДОБАВЛЕНИЕ КЛЮЧА В ДЕРЕВО

Рассмотрим, как можно реализовать добавление в дерево некоторого ключа x. Напомним, что по договорённости все ключи в дереве уникальны, поэтому при попытке добавить ключ, который уже есть, дерево не изменяется.

Как и прежде, выполняется спуск по дереву в поисках подходящего места для новой вершины. Спуск начинается с корня. Если новый ключ x меньше ключа текущей вершины, то спускаемся влево, если x больше, то вправо. В случае если x равен текущему ключу, то ничего делать не нужно, вставка не требуется.

2.4.1. Рекурсивная реализация

Используя рекурсию, операцию вставки часто реализуют так. Создаётся функция, параметрами которой являются корень поддерева `v` (возможно нулевой) и ключ `x`, который нужно добавить. Функция осуществляет вставку ключа в поддерево и возвращает корень поддерева после вставки (корень мог поменяться в единственном случае: поддерева изначально не существовало и была создана новая вершина, которая становится корнем; иначе функция просто возвращает тот же корень, который был ей передан).

```
def InsertRecursively(v, x):
    if v is None:
        return Node(x)
    if x < v.key:
        v.left = InsertRecursively(v.left, x)
    elif x > node.key:
        v.right = InsertRecursively(v.right, x)
    # v.key == x
    return v
```

Вызывать такую функцию нужно из корня дерева и при этом присваивать корню возвращаемое значение, чтобы был обработан случай изначально пустого дерева:

```
tree.root = InsertRecursively(tree.root, x)
```

2.4.2. Нерекурсивная реализация

Если не использовать рекурсию, то код получается более сложным.

Спустимся посредством цикла по дереву, начиная от корня, в поисках места для вставки. Чтобы, найдя пустое место, иметь возможность заменить его на новую вершину, в коде кроме ссылки на текущую вершину `v` поддержим ссылку на вершину-предка в переменной `parent`. Если вставляемый ключ меньше текущего, пойдём влево; если больше, то двинемся вправо; если равен, то остановимся и выйдем: ключ уже есть в дереве и вставлять его не нужно. Когда переменная `v` примет нулевое

значение None, место для вставки найдено. Создадим новую вершину и в предке parent исправим ссылку left или right в зависимости от того, влево или вправо мы двигались из parent до того, как попали в None. В отдельном случае, когда дерево изначально было пусто, нужно обновить переменную root (создан новый корень).

```
def InsertIteratively(x):
    parent = None
    v = tree.root
    while v is not None:
        parent = v
        if x < v.key:
            v = v.left
        elif x > v.key:
            v = v.right
        else:  # x == v.key
            return

    w = Node(x)

    if parent is None:
        tree.root = w
    elif x < parent.key:
        parent.left = w
    elif x > parent.key:
        parent.right = w
```

2.5. УДАЛЕНИЕ ИЗ ДЕРЕВА

При удалении некоторой вершины v отдельно рассматриваются три случая.

1. Вершина v не имеет поддеревьев, т. е. является листом дерева. В этом случае она просто уничтожается (вместе с дугой, ведущей в неё).

Пусть в программе переменная v — ссылка на вершину v, которую требуется удалить. Если у вершины v был предок parent, то соответствующая сылка parent.left или parent.right обнуляется (рис. 2.8); если же вершина v была корнем дерева, то обнуляется переменная root, которая хранит ссылку на корень.

Заметим, что для выполнения такой операции необходимо иметь ссылку на предка вершины. Как правило, хранить для этого в каждой вершине дополнительное поле parent со ссылкой на родителя неразумно. Как можно обойти эту трудность, будет рассмотрено позже.

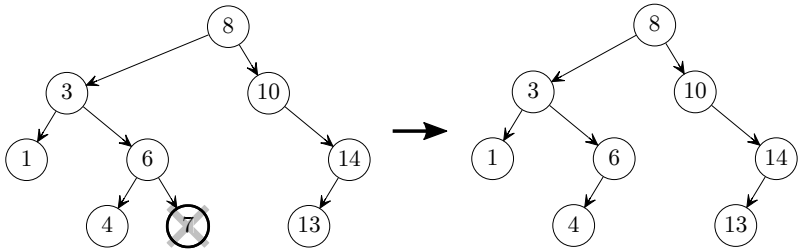


Рис. 2.8. Удаление висячей вершины с ключом 7

2. У вершины v есть только левое или только правое поддерево. В этом случае вершина v также убирается без перестроения дерева.

Если у вершины v есть вершина-предок, то дуга, ведущая в вершину v из предка, перенаправляется на того сына вершины v, который существует (рис. 2.9).

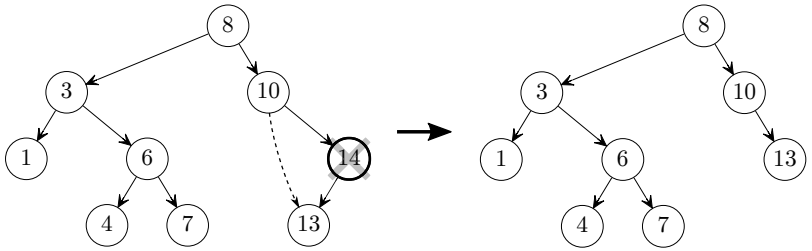


Рис. 2.9. Удаление вершины с ключом 14

На рис. 2.10 продемонстрирован отдельно случай удаления, когда удаляемая вершина v является корнем дерева и у неё есть только одно поддерево.

3. У вершины v есть оба поддерева. Это самый сложный случай, но он сводится к предыдущим. Здесь различают *левое* и *правое* удаление. Вначале отыскивается или в левом поддереве вершины v самая правая вершина (левое удаление), или в правом поддереве — самая левая вершина (правое удаление). Пусть это вершина w. Нетрудно понять, что вершина w будет иметь не более одного поддерева, поэтому она может быть удалена указанным выше методом. Затем ключ вершины v заменяется на ключ вершины w.

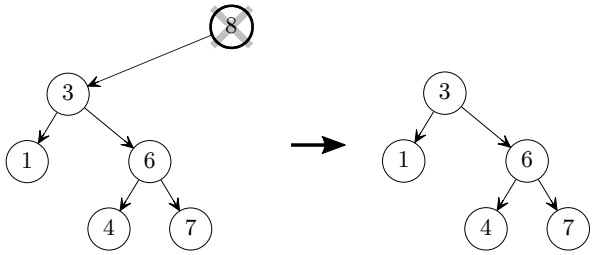


Рис. 2.10. Удаление корня дерева с ключом 8

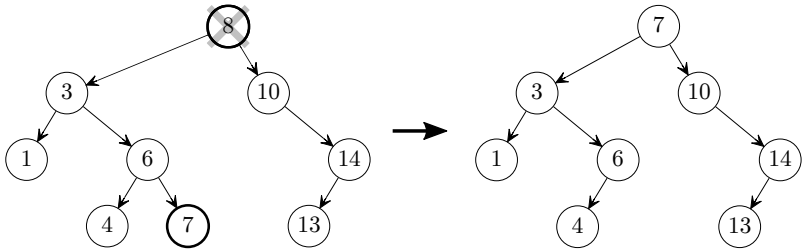


Рис. 2.11. Левое удаление вершины с ключом 8

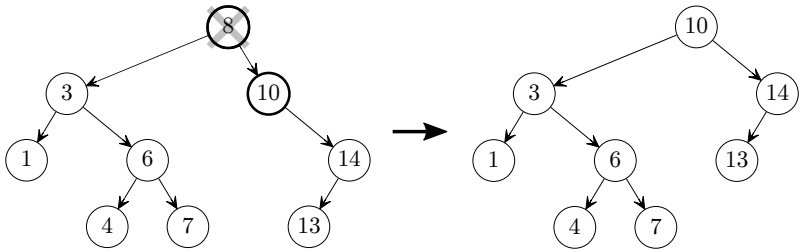


Рис. 2.12. Правое удаление вершины с ключом 8

Для вершины v с ключом 8, у которой есть оба поддерева, на рис. 2.11 продемонстрировано её левое удаление, а на рис. 2.12 — правое.

2.5.1. Рекурсивная реализация

При помощи рекурсии удаление из дерева вершины с заданным ключом обычно осуществляется так. Реализуется функция (будем на-

зывать её DeleteRecursively), которая принимает на вход ссылку на вершину v и ключ x. Функция выполняет удаление вершины с ключом x в поддереве с корнем v, если такой ключ там есть. Функция возвращает новый корень поддерева после удаления (корень v может поменяться, если он сам окажется удалённым).

Функцию нужно вызывать из корня дерева и при этом присваивать корню возвращаемое значение:

```
tree.root = DeleteRecursively(tree.root, x)
```

В коде функции вначале делаются проверки ключа и выполняется спуск по дереву в поисках x (если меньше — идём влево, если больше — движемся вправо). Когда функция DeleteRecursively оказывается вызванной для вершины v с ключом x, переходим непосредственно к удалению. Сначала проверяется число потомков вершины v. Если левое и/или правое поддерево отсутствует, это простой случай. Иначе, когда есть оба поддерева, действия отличаются в зависимости от того, выполняется удаление левое или правое. Рассмотрим случай правого удаления. Вызывается вспомогательная функция FindMin, которая ищет вершину с наименьшим ключом min_key в правом поддереве вершины v. Затем ключ в вершине v заменяется на найденный наименьший ключ min_key, а исходная вершина с ключом min_key удаляется через новый вызов функции DeleteRecursively.

```
def DeleteRecursively(v, x):
    if v is None:
        return None
    if x < v.key:
        v.left = DeleteRecursively(v.left, x)
        return v
    elif x > v.key:
        v.right = DeleteRecursively(v.right, x)
        return v
    # v.key == x
    if v.left is None:
        return v.right
    elif v.right is None:
        return v.left
    else:
        # both subtrees are present
        min_key = FindMin(v.right).key
        v.key = min_key
        v.right = DeleteRecursively(v.right, min_key)
        return v
```

Вспомогательная функция `FindMin` находит вершину с наименьшим ключом в поддереве с корнем `v`. Выполняется рекурсивный спуск по дереву всё время влево (там ключи меньше):

```
def FindMin(v):
    if v.left is not None:
        return FindMin(v.left)
    else:
        return v
```

2.5.2. Нерекурсивная реализация

Реализация удаления вершины с заданным ключом без использования рекурсии происходит немного сложнее.

Для начала определим вспомогательную функцию, которая у вершины `parent` заменит вершину-сына `old` на вершину `new` (если в качестве `parent` будет передано значение `None`, то заменён будет корень всего дерева):

```
def ReplaceChild(parent, old, new):
    if parent is None:
        tree.root = new
    elif parent.left == old:
        parent.left = new
    elif parent.right == old:
        parent.right = new
```

Для поиска вершины с ключом `x` можно было бы воспользоваться ранее разработанной функцией поиска, но она потребовала бы доработки: для удаления нужно знать не только ссылку на саму вершину `v` с ключом `x`, но и ссылку на её предка `parent`. Поэтому для определённости реализуем поиск ещё раз.

В целом логика удаления здесь та же, как и в рекурсивной реализации. Когда ищем вершину с наименьшим ключом в правом поддереве, мы также вынуждены поддерживать при спуске ссылку на предка, чтобы потом иметь возможность эту вершину удобно удалить (вершина `min_node_parent` является предком вершины `min_node`).

```
def DeleteIteratively(x):
    parent = None
    v = tree.root

    while True:
        if v is None:
            return
```

```
        if x < v.key:
            parent = v
            v = v.left
        elif x > v.key:
            parent = v
            v = v.right
        else:  # x == v.key
            break

    result = None

    if v.left is None:
        result = v.right
    elif v.right is None:
        result = v.left
    else:
        min_node_parent = v
        min_node = v.right
        while min_node.left is not None:
            min_node_parent = min_node
            min_node = min_node.left

        result = v
        v.key = min_node.key
        ReplaceChild(min_node_parent, min_node, min_node.right)

    ReplaceChild(parent, v, result)
```

2.6. ОБХОДЫ ВЕРШИН ДЕРЕВА

Многие алгоритмы, работая с бинарными корневыми деревьями, посещают в определённом порядке каждую вершину дерева. Существуют три наиболее распространённых способа обхода вершин бинарного дерева: прямой (префиксный), обратный (постфиксный) и внутренний (инфиксный). Каждый обход выполняется за время $O(n)$.

Прямой порядок обхода заключается в том, что корень некоторого дерева посещается раньше, чем его поддеревья. Если при прямом обходе после обработки корня осуществляется переход в его левое (правое) поддерево, то обход называется *прямым левым* (*правым*).

Ниже приведена функция `PreOrderTraversal` прямого левого обхода. В коде использована вспомогательная функция `Action`, в которой могут выполняться некоторые действия с вершиной `v`, например

пересчёт некоторой характеристики вершины на основании данных, полученных от её отца.

```
def PreOrderTraversal(v):
    if v is not None:
        Action(v)
        PreOrderTraversal(v.left)
        PreOrderTraversal(v.right)
```

Обратный порядок обхода заключается в том, что корень дерева посещается после его поддеревьев. Если при обратном обходе сначала осуществляется переход в левое (правое) поддерево корня, то обход называется *обратным левым* (*правым*).

Далее приведена функция `PostOrderTraversal` обратного левого обхода, где вызывается вспомогательная функция `Action`, в которой могут выполняться некоторые действия с вершиной `v`, например подсчёт меток вершины исходя из меток её сыновей.

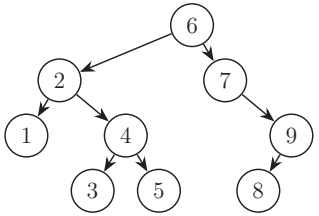
```
def PostOrderTraversal(v):
    if v is not None:
        PostOrderTraversal(v.left)
        PostOrderTraversal(v.right)
        Action(v)
```

Внутренний порядок обхода заключается в том, что корень дерева посещается после одного из его поддеревьев. Если корень посещается после обхода левого (правого) поддерева, то обход называется *внутренним левым* (*правым*).

Ниже приведена функция `InOrderTraversal` внутреннего левого обхода. Во вспомогательной функции `Action` могут выполняться необходимые действия с вершиной `v`, например печать ключа вершины.

```
def InOrderTraversal(v):
    if v is not None:
        InOrderTraversal(v.left)
        Action(v)
        InOrderTraversal(v.right)
```

Заметим, что внутренний левый (правый) обход проходит все вершины дерева в порядке возрастания (убывания) ключей вершин. Таким образом, если нам необходимо отсортировать последовательность чисел, то мы можем сначала по заданной последовательности построить сбалансированное поисковое дерево, а затем выполнить внутренний обход. Время работы алгоритма — $O(n \log n)$.



Прямой левый: 6, 2, 1, 4, 3, 5, 7, 9, 8
Прямой правый: 6, 7, 9, 8, 2, 4, 5, 3, 1

Обратный левый: 1, 3, 5, 4, 2, 8, 9, 7, 6
Обратный правый: 8, 9, 7, 5, 3, 4, 1, 2, 6

Внутренний левый: 1, 2, 3, 4, 5, 6, 7, 8, 9
Внутренний правый: 9, 8, 7, 6, 5, 4, 3, 2, 1

Рис. 2.13. Порядок посещения вершин при разных обходах

На рис. 2.13 для примера указан порядок посещения вершин конкретного бинарного поискового дерева при выполнении различных обходов.

2.7. НАИБОЛЬШИЕ ПОЛУПУТИ

2.7.1. Путь и полупуть

Путём в дереве называется чередующаяся последовательность вершин и дуг $v_0, (v_0, v_1), v_1, (v_1, v_2), v_2, \ldots, v_n$. Известно, что в пути дуги не могут повторяться, а длина пути определяется как количество дуг в нём.

Для бинарного поискового дерева, приведённого на рис. 2.14, *а*, последовательность

$$8, (8, 4), 4, (4, 5), 5 \tag{2.1}$$

является путём длины 2. Будем говорить, что данный путь соединяет вершины 8 и 5.

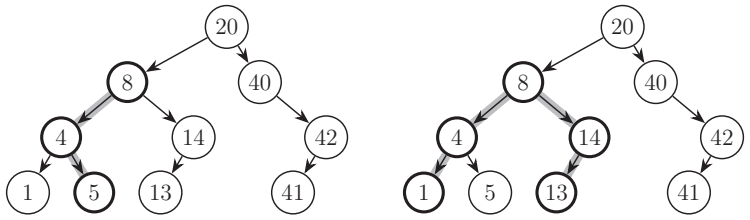


Рис. 2.14. Пример пути (*а*) и полупути (*б*) в дереве

Определим *центральную вершину некоторого пути* P как такую вершину v, что количество вершин в пути P до неё равно количеству

вершин в пути P после неё (если количество вершин в пути P чётно, то будем говорить, что центральной вершины для заданного пути P не существует). Определим *среднюю вершину некоторого пути* P как такую вершину v, что количество вершин пути P, ключ которых меньше ключа вершины v, равно количеству вершин в пути P, ключ которых больше ключа вершины v (если количество вершин в пути P чётно, то будем говорить, что средней вершины для заданного пути P не существует). Для пути (2.1) центральной является вершина 4, а средней по значению — 5.

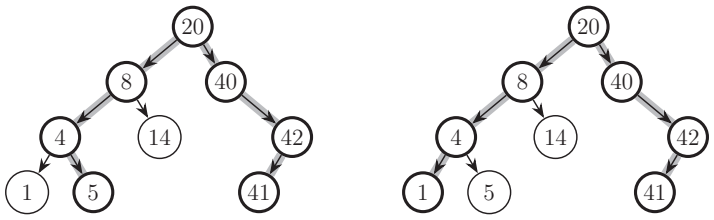
Для *полупути* снимается ограничение на направление дуг. Например, для дерева, приведённого на рис. 2.14, *б*, последовательность

$$1, (1, 4), 4, (4, 8), 8, (8, 14), 14, (14, 13), 13 \tag{2.2}$$

является полупутём длины 4, но не является путём. В полупути, как и в пути, дуги не могут повторяться. *Корнем полупути* будем называть вершину этого полупути, которая находится на наибольшей высоте. Центральная и средняя вершины полупути определяются так же, как и для пути. Для приведённого выше полупути (2.2) вершина 8 является и центральной, и средней, и корнем этого полупути (будем говорить, что данный полупуть соединяет вершины 1 и 13).

2.7.2. Наибольший полупуть

Наибольшим полупутём в дереве будем называть полупуть наибольшей длины. Не стоит путать наибольший полупуть с *максимальным полупутём* — полупутём, который нельзя продолжить (наибольший полупуть всегда является максимальным, а вот обратное не всегда верно). Полупуть (2.2) — максимальный, но не наибольший.



а *б*

Рис. 2.15. Наибольшие полупути в бинарном поисковом дереве

На рис. 2.15 выделены два наибольших полупути длины 6 в бинарном поисковом дереве. Первый наибольший полупуть соединяет вершины 5 и 41 (рис. 2.15, *а*):

$$5,\ (5,4),\ 4,\ (4,8),\ 8,\ (8,20),\ 20,\ (20,40),\ 40,\ (40,42),\ 42,\ (42,41),\ 41,$$

второй наибольший полупуть соединяет вершины 1 и 41 (рис. 2.15, *б*):

$$1,\ (1,4),\ 4,\ (4,8),\ 8,\ (8,20),\ 20,\ (20,40),\ 40,\ (40,42),\ 42,\ (42,41).$$

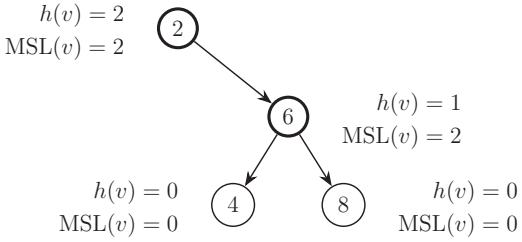


Рис. 2.16. Бинарное поисковое дерево с метками

Следует отметить, что наибольший полупуть в дереве не обязательно соединяет листья дерева. Так, если у корня дерева только одно поддерево, то наибольший полупуть может соединять корень дерева и лист, т. е. являться путём (см. дерево на рис. 2.16).

2.7.3. Определение длины наибольшего полупути

Для нахождения наибольшего полупути с корнем в вершине v достаточно знать для каждой вершины дерева её высоту. Для расстановки меток высот обойдём дерево обратным обходом (во время обратного обхода для каждой вершины v сначала посещаются её поддеревья). Во время обратного обхода сформируем метки высот по следующим правилам:

- если вершина v — лист, то полагаем её высоту равной 0;
- если у вершины v только одно поддерево, то полагаем её высоту равной высоте единственного поддерева плюс 1;
- если у вершины v есть оба поддерева, то полагаем её высоту равной максимуму из меток высот её поддеревьев плюс 1.

Для определения длины наибольшего полупути с корнем в вершине v, которую обозначим $\mathrm{MSL}(v)$ (maximum semipath length), необходимо найти сумму меток высот сыновей вершины v и увеличить это число на 1, если у вершины v только один сын, и на число 2, если у вершины v есть оба сына (для дерева, приведённого на рис. 2.16, возле каждой вершины указаны метки). Для определения длины наибольшего полупути для всего дерева, которую будем обозначать $\mathrm{MSL}(T)$, нужно выбрать вершины, для которых величина $\mathrm{MSL}(v)$ является максимальной.

Заметим, что в дереве может быть несколько корней полупутей длины $\mathrm{MSL}(T)$, а через один и тот же корень наибольшего полупути может проходить несколько попарно различных полупутей наибольшей длины. Для дерева T, приведённого на рис. 2.16, длина наибольшего полупути $\mathrm{MSL}(T) = 2$, а вершины с ключами 2 и 6 являются корнями полупутей наибольшей длины.

2.7.4. Подсчёт числа наибольших полупутей

Для подсчёта числа попарно различных полупутей наибольшей длины, которые проходят через v (данную величину будем обозначать через $c(v)$), можно использовать алгоритм, приведённый ниже.

Пусть u и w — сыновья вершины v, f — отец (если v не корень дерева), v' — брат (если есть) (рис. 2.17).

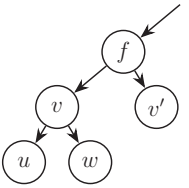


Рис. 2.17

На первом шаге алгоритма выполним обратный обход дерева, формируя для каждой вершины v две метки: высоту $h(v)$ и число листьев $l_h(v)$, лежащих на пути из v на расстоянии $h(v)$. Во время этого же обхода найдём длину наибольшего полупути $\mathrm{MSL}(T)$ и корни полупутей наибольшей длины (см. п. 2.7.3). Правило формирования метки $h(v)$ приведено в п. 2.7.3, а для вычисления метки $l_h(v)$ поступим следующим образом:

- если вершина v — лист, то $l_h(v) = 1$;
- если у вершины v есть только одно поддерево с корнем в вершине w, то $l_h(v) = l_h(w)$;
- если у вершины v есть оба поддерева с корнями u и w и $h(u) = h(w)$, то $l_h(v) = l_h(u) + l_h(w)$;
- если у вершины v есть оба поддерева и, например, $h(u) > h(w)$, то $l_h(v) = l_h(u)$.

Теперь для каждой вершины v определим число $b(v)$ попарно различных полупутей длины $\mathrm{MSL}(T)$, для каждого из которых данная вершина — корень наибольшего полупути (рис. 2.18, *а*). Для вычисления $b(v)$ поступим следующим образом:

- если вершина v — не корень наибольшего полупути, то $b(v) = 0$;
- если вершина v — корень наибольшего полупути, то полагаем $b(v) = l_h(u) \cdot l_h(w)$, при этом если у вершины v нет некоторого сына, то в формуле соответствующее значение полагается равным единице.

а *б*

Рис. 2.18. Наибольшие полупути через вершину v:
а — $b(v)$ штук; *б* — $a(v)$ штук

На втором шаге алгоритма для каждой вершины v дерева вычислим число $a(v)$ попарно различных полупутей длины $\mathrm{MSL}(T)$, которые проходят через v и при этом приходят в неё сверху от отца (рис. 2.18, *б*). Для этого выполним прямой обход дерева, формируя у каждой вершины v две метки: $h'(v)$ — длина наибольшего полупути из v в некоторую вершину, не принадлежащую поддеревьям вершины v; $l'(v)$ — число таких наибольших полупутей. Если f — корень дерева, то положим $h'(f) = 0$ и для простоты дальнейших вычислений — $l'(f) = 1$.

Пусть мы находимся в вершине f и для неё подсчитаны $h'(f)$ и $l'(f)$, тогда вычислим эти величины для сыновей v и v' вершины f. Сначала предположим, что у вершины f есть только один сын, например, v. Тогда справедливы следующие соотношения:

$$h'(v) = h'(f) + 1,$$
$$l'(v) = l'(f).$$

Теперь предположим, что у вершины f есть оба сына (см. рис. 2.17). Покажем, как вычислить величины h' и l' для одного из сыновей, например v (для другого сына вычисления выполняются аналогично):

- если $h'(f) = h(v') + 1$, тогда

$$h'(v) = h(v') + 2,$$
$$l'(v) = l'(f) + l_h(v');$$

- если $h'(f) < h(v') + 1$, тогда

$$h'(v) = h(v') + 2,$$
$$l'(v) = l_h(v');$$

- если $h'(f) > h(v') + 1$, тогда

$$h'(v) = h'(f) + 1,$$
$$l'(v) = l'(f).$$

Если f — корень дерева, то положим $a(f) = 0$.

Пусть v — некоторая некорневая вершины и выполняется равенство

$$h'(v) + h(v) = \mathrm{MSL}(T), \tag{2.3}$$

тогда положим

$$a(v) = l'(v) \cdot l_h(v), \tag{2.4}$$

в противном случае — $a(v) = 0$.

После того как для каждой вершины v сформированы две метки $a(v)$ и $b(v)$, число полупутей длины $\mathrm{MSL}(T)$, проходящих через эту вершину, складывается из двух величин: $c(v) = a(v) + b(v)$.

На втором шаге алгоритма для вычисления метки $a(v)$ можно использовать другой подход, который не предполагает поддержку меток $l'(v)$ и $h'(v)$, а величина $a(v)$ пересчитывается во время прямого обхода дерева на основании аналогичной метки её отца. Покажем, как это можно сделать.

Если v — корень дерева, то положим $a(v) = 0$. Если у вершины v есть только один сын, например u, то положим $a(u) = a(v) + b(v)$.

Теперь предположим, что у вершины v есть оба сына. Пусть в вершину v сверху пришло $a(v)$ полупутей длины $\mathrm{MSL}(T)$, тогда поступим следующим образом:

- если $h(u) > h(w)$, то $a(u) = a(v) + b(v)$, $a(w) = b(v)$;
- если $h(w) > h(u)$, то $a(w) = a(v) + b(v)$, $a(u) = b(v)$;

- если $h(u) = h(w)$, то с учётом формулы (2.4)

$$a(u) = b(v) + l_h(u) \cdot l'(v) = b(v) + l_h(u) \cdot \frac{a(v)}{l_h(v)},$$

$$a(w) = b(v) + l_h(w) \cdot l'(v) = b(v) + l_h(w) \cdot \frac{a(v)}{l_h(v)}.$$

Продемонстрируем описанный выше алгоритм для дерева, приведённого на рис. 2.19. Возле каждой вершины на рисунке указаны вычисленные метки. Для данного примера длина наибольшего полупути $\mathrm{MSL}(T) = 4$. Вершины с ключами 2 и 6 являются корнями полупутей наибольшей длины. Через вершину дерева с ключом 6 проходит наибольшее число полупутей наибольшей длины (8), из которых четыре наибольших полупути приходят в вершину 6 от её отца, а для четырёх полупутей наибольшей длины вершина 6 является корневой.

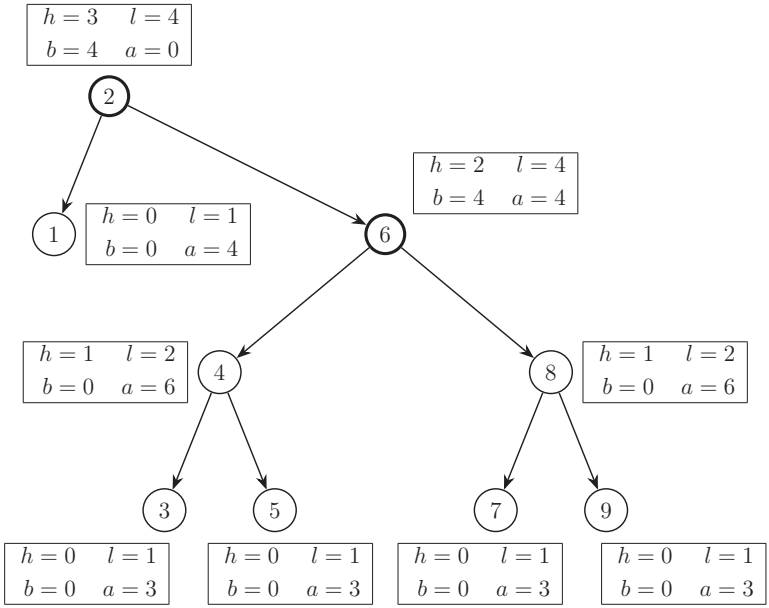


Рис. 2.19. Бинарное поисковое дерево с метками для подсчёта путей

2.8. ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

Каждая задача предполагает по последовательности чисел входного файла построение бинарного поискового дерева (предполагаем, что в дереве не может быть двух вершин с одинаковыми ключами). Затем в большинстве задач в дереве необходимо найти вершины, которые удовлетворяют заданным свойствам, удалить их и вывести итоговое дерево в выходной файл, используя указанный обход дерева.

Содержательная часть каждой задачи (за исключением построения дерева) должна решаться за линейное от числа вершин время. Исключение составляет задача 24, где допускается нелинейная зависимость от числа вершин. Кроме того, в задачах 23 и 35 допускается линейная зависимость от параметра k, т. е. общее время $O(nk)$, где n — число вершин дерева.

Задача 1

Найти среднюю по значению вершину из вершин дерева, у которых число потомков в левом поддереве не равно числу потомков в правом поддереве. Удалить её (правым удалением), если такая вершина существует. Выполнить прямой левый обход полученного дерева.

Если у вершины отсутствует некоторое поддерево, то число вершин этого поддерева полагаем равным 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
20 8 40 42 14 4 13 41 5 1	40 8 4 1 5 14 13 42 41

Задача 2

Найти среднюю по значению вершину из вершин дерева, у которых высота левого поддерева не равна высоте правого поддерева. Удалить её (правым удалением), если такая вершина существует. Выполнить прямой левый обход полученного дерева.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
2 20 11 18 25 22 7 48	20 11 7 18 25 22 48
6 4 8 2 5 7 9	6 4 2 5 8 7 9

Задача 3

Найти среднюю по значению из вершин дерева, у которых число потомков в левом поддереве отличается от числа потомков в правом поддереве на единицу. Удалить её (правым удалением), если такая вершина существует. Выполнить прямой левый обход полученного дерева.

Если у вершины нет некоторого поддерева, то число вершин этого поддерева полагаем равным 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
40 30 15 10 20 31 3 12 5 7	40 30 15 10 3 7 12 20 31

Задача 4

Найти высоту H дерева и удалить (правым удалением) среднюю по значению из вершин на уровне $\lfloor H/2 \rfloor$, у которых число потомков в левом поддереве больше числа потомков в правом поддереве. Выполнить прямой левый обход полученного дерева.

Если у вершины нет некоторого поддерева, то число вершин этого поддерева полагаем равным 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
20 10 22 15 13 12 14 5 6 16 4	20 10 5 4 6 16 13 12 14 22
1 2	1 2

Задача 5

Среди полупутей минимальной положительной длины между листьями выбрать тот, у которого сумма ключей вершин минимальна. Если такой полупуть существует, то удалить (правым удалением) центральную вершину этого полупути.

Если решение неоднозначно, то выбирать такой минимальный полупуть между листьями, корневая вершина которого имеет минимальное ключевое значение.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Примеры

входной файл	*выходной файл*
10 5 13 7 4 11 6 8 15 1	10 5 4 1 8 6 13 11 15
5 1 0 15 19 8 7 6 9 10	6 1 0 15 8 7 9 10 19

Задача 6

Найти наибольший полупуть между вершинами дерева, у которого сумма ключей вершин максимальна, и удалить (правым удалением) сначала центральную вершину этого полупути, если она существует, потом корень этого же полупути, если он не являлся одновременно и центральной вершиной этого полупути. Выполнить прямой левый обход полученного дерева.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 11 5 3 6 7 8 4	11 5 3 4 6 7 8

Задача 7

Найти высоту дерева H и удалить (правым удалением) среднюю по значению из вершин на уровне $\lfloor H/2 \rfloor$, у которых высота левого поддерева равна высоте правого. Выполнить прямой левый обход полученного дерева.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 5 15 7 13 17 3	10 5 3 7 15 13 17

Задача 8

Найти такой наибольший полупуть между вершинами дерева, у которого корневая вершина находится на наименьшей глубине. Удалить (правым удалением) эту корневую вершину.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
20 10 30 5 1 7 3 9 25 40 15 11 13 16 17	25 10 5 1 3 7 9 15 11 13 16 17 30 40
1 2 3 4 5	2 3 4 5

Задача 9

Найти вершины, через которые проходят наибольшие полупути, и удалить (правым удалением) ту из них, ключ которой является вторым по значению.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*	*пояснение*
10 30 180 181 20 70 60 50 40 130 176 177 178 179 120 110 100 90	0 30 180 70 60 50 40 130 120 110 100 90 176 177 178 179 181	рис. 2.20

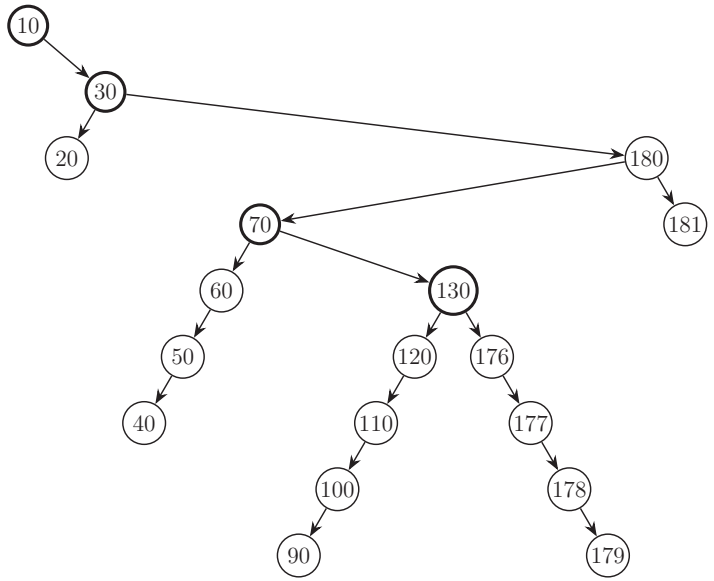


Рис. 2.20

Задача 10

Найти вершины, через которые проходит наибольшее число наибольших (по длине) полупутей, и удалить их (правым удалением) обратным левым обходом.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Замечания

1. Гарантируется, что хотя бы одна вершина не подлежит удалению.
2. Для удаления всех вершин, удовлетворяющих заданным требованиям, необходимо разработать алгоритм, работающий за время $O(n)$.

Примеры

входной файл	*выходной файл*
10 5 15 4 6 14 16 17 9	14 4
10 30 180 181 20 70 60 50 40 130 176 177 178 179 120 110 100 90	10 30 20 180 70 60 50 40 176 120 110 100 90 177 178 179 181

Задача 11

Найти такой наибольший полупуть, у которого сумма ключей крайних вершин минимальна. Удалить (правым удалением) корневую вершину этого полупути.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
50 40 60 30 45 27 35 46	50 45 30 27 35 46 60

Задача 12

Найти наибольший полупуть между вершинами с разным числом потомков с минимальной суммой ключей крайних вершин (длина искомого полупути должна быть больше 0).

Если таких полупутей несколько, то выбрать из них тот, у которого корневая вершина имеет минимальное ключевое значение. Удалить (правым удалением), если существует, среднюю по значению вершину этого полупути.

Если у вершины отсутствует некоторое поддерево, то число вершин этого поддерева полагаем равным 0.

Замечание

В случае неоднозначности выбора удаляемой вершины (например, несколько полупутей максимальной длины между вершинами с разным числом потомков и с минимальной суммой ключей крайних вершин имеют один и тот же корень, но средние по значению вершины этих полупутей не совпадают) ничего из дерева удалять не нужно.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
0 40 50 60 70 80 90 2 1	0 40 2 1 60 70 80 90
0 4 5 6 7 8 9 3 1	0 4 3 1 5 7 8 9

Задача 13

Необходимо найти наибольший полупуть между вершинами разной высоты с минимальной суммой ключей крайних вершин.

Если таких полупутей несколько, то выбрать тот из них, у которого корневая вершина имеет наименьшее ключевое значение.

Удалить (правым удалением), если существует, среднюю по значению вершину этого полупути.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1.

Замечание

В случае неоднозначности выбора удаляемой вершины (например, несколько наибольших полупутей между вершинами разной высоты и с минимальной суммой ключей крайних вершин имеют один и тот же корень, но средние по значению вершины этих полупутей не совпадают) ничего из дерева удалять не нужно.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
0 4 5 6 7 8 9 3 1	0 4 3 1 5 7 8 9

Задача 14

Необходимо найти пути минимальной длины между корнем и листьями и удалить (левым удалением) в порядке возрастания ключей средние по значению вершины этих путей, если они существуют.

Замечание

Если некоторая вершина является средней по значению для нескольких путей минимальной длины, то удаляется она только один раз.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 15 13 18 5 8 3	10 3 18
1 2	1 2

Задача 15

Определить, являются ли два дерева зеркальным отражением друг друга по структуре, если сначала удалить (правым удалением) корень каждого из деревьев.

Замечания

1. Если деревья были тривиальными (одна вершина в каждом дереве), то в результате удаления корней получаются два не графа, поэтому будем считать, что тривиальные деревья зеркально совпадают.

2. Если в результате удаления корней получается два тривиальных дерева, то будем считать, что они зеркально совпадают.

Формат входных данных

Входной файл содержит последовательности чисел — ключи вершин в порядке добавления в дерево. Деревья разделяются строкой с единственным символом «*».

Формат выходных данных

Выходной файл должен содержать в первой строке сообщение `YES` или `NO`. Если сообщение `YES`, то во второй строке вывести последовательность ключей вершин, полученную прямым левым обходом первого дерева, а в третьей строке — последовательность ключей вершин, полученную прямым левым обходом второго дерева.

Примеры

входной файл	*выходной файл*
2 1 3 * 2 1 3	NO
96 90 97 80 95 85 * 20 30 40 50 33 45	YES 97 90 80 85 95 30 40 33 50 4

Задача 16

Необходимо удалить (правым удалением) корень дерева, а затем для каждой вершины определить, сколько наибольших полупутей проходят через неё.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выполните прямой левый обход итогового дерева. Для каждой вершины в отдельной строке выведите через пробел её ключ и количество полупутей наибольшей длины, проходящих через неё.

Пример

входной файл	*выходной файл*
1 5 3 7 2 4 6 8	5 4 3 4 2 2 4 2 7 4 6 2 8 2
10 5 4 3 2 1 6 7 8 9	5 1 4 1 3 1 2 1 1 1 6 1 7 1 8 1 9 1

Задача 17

Найти и удалить (правым удалением), если существует, среднюю по значению из вершин дерева, у которых высота левого поддерева отличается от высоты правого наибольшим образом. Выполнить прямой левый обход полученного дерева.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 5 6 8 7 11 1 2 12 13 3	10 5 1 2 3 8 7 11 12 13
3 1 2 5 4 6 7	3 1 2 5 4 6 7

Задача 18

Найти вершины, через которые проходит чётное число наибольших полупутей, и удалить (правым удалением) ту из них, ключ которой наименьший (если нет вершин, удовлетворяющих нужному свойству, то ничего из дерева удалять не надо).

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
2 1 5 4 6 3 7	2 5 4 3 6 7

Задача 19

Найти вершины, через которые проходят наибольшие полупути, и удалить (правым удалением) самую высокую из них.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
20 21 10 1 15 14 13 19 16 17 11 12 18	20 10 1 16 14 13 11 12 19 17 18 21

Задача 20

Найти наибольший полупуть между вершинами разного уровня с минимальной суммой ключей крайних вершин и сделать центральную вершину этого полупути корнем дерева (методом поворотов).

Замечания

1. Исходное дерево содержит не менее двух вершин.

2. Если центральной вершины нет, то метод поворотов не выполняется. Выходной файл должен содержать сначала минимальную сумму ключей крайних вершин, а затем массив вершин, полученный прямым левым обходом исходного дерева.

3. Для выполнения метода поворотов (рис. 2.21, 2.22) нужно спуститься из корня дерева к центральной вершине и поднять её (после выполнения очередного поворота центральная вершина оказывается на меньшей глубине).

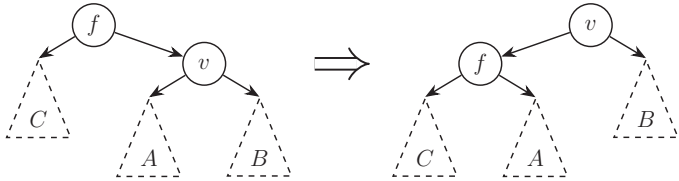


Рис. 2.21. Малый левый поворот

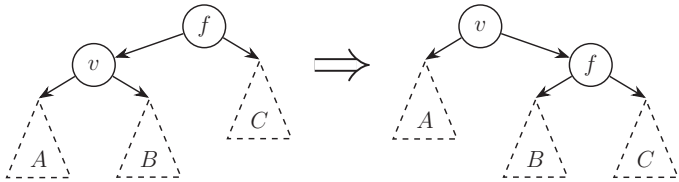


Рис. 2.22. Малый правый поворот

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать сначала сумму ключей крайних вершин найденного полупути, а затем последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 30 180 181 20 70 60 50 40 130 176 177 178 179 120 110 100 90	100 130 10 30 20 70 60 50 40 120 110 100 90 180 176 177 178 179 181

Задача 21

Найти среднюю по значению из вершин дерева, у которых высоты поддеревьев равны, а число потомков в правом и левом поддеревьях отличается. Удалить её (правым удалением), если такая вершина существует.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1, а число вершин этого поддерева — 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*	*пояснение*
3 2 1 5 4 6	4 2 1 5 6	рис. 2.23

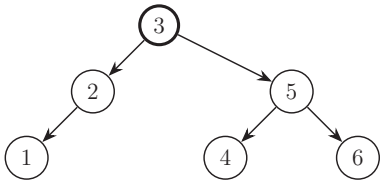


Рис. 2.23

Задача 22

Найти среднюю из вершин дерева, у которых высоты поддеревьев не равны, а число потомков в правом и левом поддеревьях равны. Удалить её (правым удалением), если такая вершина существует.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1, а число вершин этого поддерева — 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*	*пояснение*
10 8 12 4 9 11 13 3 6 2 5 7 1	10 8 5 3 2 1 6 7 9 12 11 13	рис. 2.24

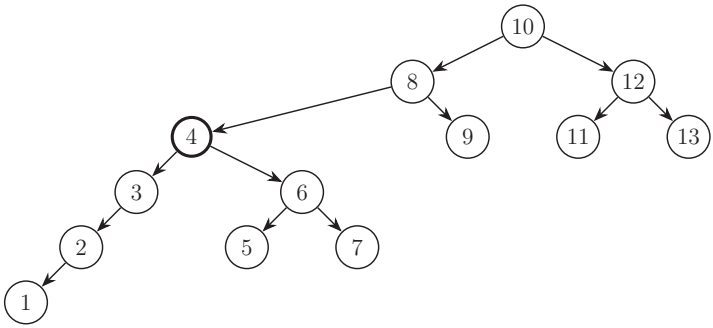


Рис. 2.24

Задача 23

Найти все вершины, через которые проходят полупути длины K между висячими вершинами. Выбрать те, которые находятся на наименьшей глубине, и удалить (левым удалением) из них среднюю по значению, если она существует.

Формат входных данных

Входной файл содержит в первой строке число K ($2 < K \leqslant 10^9$), а в последующих строках — последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*	*пояснение*
8 13 1 57 20 81 59 48 36 90 83 75 18 86 72 52 31 2 10 37 15 17 99 45 12 3	12 1 2 10 3 57 20 18 15 17 48 36 31 37 45 52 81 59 75 72 90 83 86 99	рис. 2.25

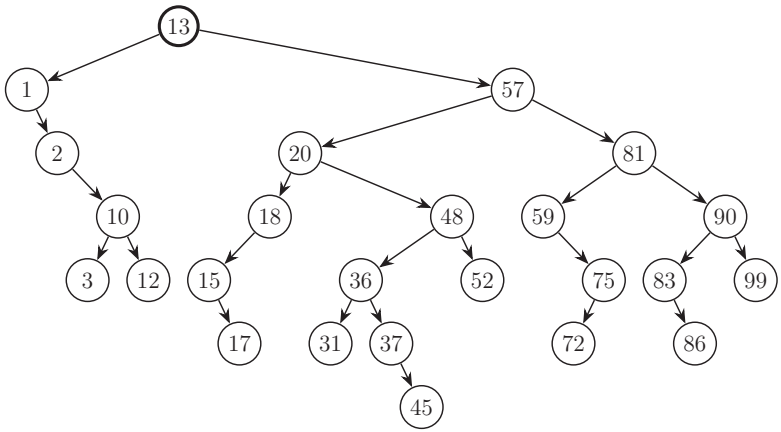


Рис. 2.25

Задача 24

Найти все полупути длины H между висячими вершинами, где H — высота дерева. Выбрать корни этих полупутей, расположенные на максимальной глубине, и удалить (правым удалением) тот из них, который является средним по значению.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 8 16 7 9 13 17 12 14	10 8 7 9 17 13 12 14
10 8 16 7 9 13 17 12 14	10 8 7 9 17 13 12 14

Задача 25

Найти и удалить (правым удалением), если существует, среднюю по значению из вершин дерева, у которых число потомков в левом поддереве отличается от числа потомков в правом наибольшим образом.

Если у вершины отсутствует некоторое поддерево, то число вершин этого поддерева полагаем равным 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
50 40 60 30 55 70 27 35 40 65 80 30	50 30 27 35 60 55 70 65 80

Задача 26

Найти и удалить (правым удалением), если существует, вершину с наибольшим ключом из вершин дерева, у которых число потомков в правом и левом поддеревьях отличается наибольшим образом.

Если у вершины отсутствует некоторое поддерево, то число вершин этого поддерева полагаем равным 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
50 40 60 30 45 55 70 27 35 46 65 62 68 30	50 40 30 27 35 45 46 60 55 65 62 68

Задача 27

Найти средний по значению, если существует, из листьев дерева и удалить (правым удалением) его отца.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
50 40 60 30 45 55 70 27 35 46 50 80 40 55 90	50 40 30 27 35 46 60 55 70 80 90
1 2	2

Задача 28

Найти в дереве среднюю по значению из вершин, у которых высоты поддеревьев равны. Удалить (правым удалением) данную вершину, если она существует. Листья дерева также рассматриваются в качестве вершин, у которых высоты поддеревьев равны.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
50 60 70 80	50 60 70

Задача 29

Найти в дереве среднюю по значению из вершин, у которых число потомков в левом поддереве отличается от числа потомков в правом на два. Удалить (левым удалением) данную вершину, если она существует. Если у вершины отсутствует некоторое поддерево, то число вершин этого поддерева полагаем равным 0.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*	*пояснение*
10 5 11 21 9 1 6	9 5 1 6 11 21	рис. 2.26

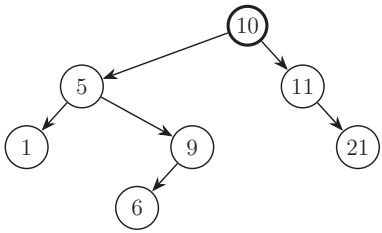


Рис. 2.26

Задача 30

Найти среднюю по значению вершину в дереве, у которой высота левого поддерева отличается от высоты правого на два. Удалить (левым удалением) данную вершину, если она существует. Выполнить прямой (левый) обход полученного дерева.

Если у вершины отсутствует некоторое поддерево, то его высоту полагаем равной -1.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 5 11 12 6 3 1 2	10 3 1 2 6 11 12
10 1 15 11 14	10 1 15 11 14

Задача 31

Найти вершины, через которые проходит нечётное число наибольших полупутей, и удалить (правым удалением) ту из них, ключ которой наибольший.

Если в дереве нет вершин, удовлетворяющих нужному свойству, то ничего удалять не требуется. Выполнить прямой (левый) обход полученного дерева.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
10 5 20 4 6 15 30 3 7 14 40 8 50 9 60	10 5 4 3 6 7 8 9 20 15 14 30 40 50

Задача 32

Найти среднюю по значению вершину дерева из вершин, через которые проходит чётное положительное число наибольших полупутей, и удалить её (правым удалением), если она существует. Если в дереве нет вершин, удовлетворяющих заданному свойству, то ничего из дерева удалять не надо.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
20 10 22 5 11 21 23	21 10 5 11 22 23

Задача 33

Найти наибольший полупуть между вершинами дерева, у которого сумма ключей крайних вершин минимальна. Удалить (правым удалением) среднюю по значению вершину этого полупути.

Если средней по значению вершины не существует (т. е. число вершин искомого полупути чётно), то ничего из дерева удалять не надо.

Формат входных данных

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево.

Гарантируется, что в дереве не менее двух вершин.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Пример

входной файл	*выходной файл*
50 40 60 30 45 27 35 46	50 45 30 27 35 46 60

Задача 34

Найти и удалить (если такая вершина есть, правым удалением) максимальную по значению среди вершин, для которых расстояние до любой другой вершины дерева строго меньше k и через которые не проходят полупути длины k (длина полупути и расстояние между двумя вершинами измеряются числом дуг).

Формат входных данных

В первой строке находится целое число k ($0 \leqslant k \leqslant 10^9$).

В последующих строках, в каждой по одному числу, находятся ключи вершин исходного дерева (от 0 до $2^{31} - 1$) в порядке добавления в дерево.

Число вершин произвольное — от 0 до 1000.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Если в результате дерево удалилось полностью, то необходимо вывести в выходной файл сообщение `Empty`.

Пример

входной файл	*выходной файл*
6 5 3 1 0 6 4 8 2 7 9	5 3 1 0 2 6 8 7 9

Задача 35

Найти максимальную по значению вершину, расстояние от которой до любой другой вершины дерева строго меньше k и через которую не проходят полупути длины k с висячими концевыми вершинами (длина полупути и расстояние между двумя вершинами измеряется в дугах). Если такая вершина есть, то удалить её правым удалением.

Формат входных данных

В первой строке находится целое число k ($0 \leqslant k \leqslant 10^9$). В последующих строках, в каждой по одному числу — ключи вершин исходного дерева, в порядке добавления в дерево (значения в пределах от 0 до $2^{31} - 1$). Число вершин не превосходит 1000.

Формат выходных данных

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева. Если в результате дерево удалилось полностью, то необходимо вывести в выходной файл сообщение `Empty`.

Пример

входной файл	*выходной файл*
6 5 3 1 0 6 4 8 2 7 9	5 3 1 0 2 6 8 7 9

2.9. УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ

1. С помощью обратного обхода для каждой вершины дерева v определим число вершин в дереве с корнем в v и подсчитаем количество вершин, удовлетворяющих нужным требованиям. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим. Выведем дерево.

2. С помощью обратного обхода для каждой вершины дерева v определим её высоту и подсчитаем количество вершин, удовлетворяющих нужным требованиям. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим. Выведем дерево.

3. С помощью обратного обхода для каждой вершины дерева v определим число вершин в дереве с корнем в v и подсчитаем количество вершин, удовлетворяющих нужным требованиям. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим. Выведем дерево.

4. С помощью обратного обхода для каждой вершины дерева v определим её высоту и число потомков в поддереве с корнем в этой вершине. Для спуска на нужный уровень можно использовать поиск в ширину из корня дерева. Поскольку дерево поисковое, то на одном уровне вершины упорядочены слева направо по возрастанию ключа. Другой способ — прямым обходом определим дополнительно для каждой вершины её глубину, затем, зная высоту корня и глубины вершин, любым обходом вычислим уровни вершин по определению. Подсчитаем количество вершин, удовлетворяющих нужным требованиям на заданном уровне, и если число нужных вершин нечётно, то найдём среднюю из них и удалим. Выведем дерево.

5. С помощью обратного обхода для каждой вершины v определим длину $h'(v)$ кратчайшего пути от вершины v до листа и вес самого лёгкого такого пути (вес пути — сумма ключей вершин этого пути). Найдём корень полупути минимальной положительной длины между листьями с минимальным весом. Для этого рассмотрим каждую вершину v, имеющую двух потомков l и r. Длину искомого полупути с корнем в v можно вычислить по формуле $h'(l) + h'(r) + 2$. Необходимо также учесть вес полупути и минимизировать его. В случае равенства

длин и весов полупутей выберем корень с меньшим ключом. Если длина искомого полупути чётна, то найдём его центральную вершину и удалим. Выведем дерево.

6. С помощью обратного обхода для каждой вершины v определим её высоту $h(v)$ и вес самого тяжёлого пути (вес пути — сумма ключей вершин этого пути), выходящего из v и имеющего длину $h(v)$. Найдём наибольшие полупути, а среди них в качестве искомого выберем самый тяжёлый и определим его корень w. Если длина найденного полупути чётна, то найдём центральную вершину этого пути и удалим её. Если вершина w не совпала с центральной вершиной, то удалим и её. Выведем дерево.

7. С помощью обратного обхода для каждой вершины дерева v определим её высоту. Для спуска на нужный уровень можно использовать поиск в ширину из корня дерева. Поскольку дерево поисковое, то на одном уровне вершины упорядочены слева направо по возрастанию ключа. Другой способ — прямым обходом определим дополнительно для каждой вершины её глубину, затем, зная высоту корня и глубины вершин, любым обходом вычислим уровни вершин по определению. Подсчитаем количество вершин, удовлетворяющих нужным требованиям на заданном уровне, и если число нужных вершин нечётно, то найдём среднюю из них и удалим её. Выведем дерево.

8. С помощью обратного обхода для каждой вершины дерева определим её высоту. Найдём наибольшие полупути и выберем из них тот, у которого корневая вершина находится на наименьшей глубине (для этого нужно исследовать взаимное расположение корней полупутей наибольшей длины). Удалим корневую вершину искомого наибольшего полупути. Выведем дерево.

9. С помощью обратного обхода для каждой вершины дерева определим её высоту. Найдём корни наибольших полупутей и выберем из них тот, которому будет принадлежать вторая по значению вершина, через которую проходят наибольшие полупути. Осуществим спуск из найденного корня к нужной вершине и удалим её. Выведем дерево.

10. С помощью обратного обхода для каждой вершины дерева определим высоту. Найдём корни наибольших полупутей и выберем среди

них тот, через который пройдёт наибольшее число полупутей наибольшей длины (для этого достаточно проанализировать взаимное расположение всех наибольших полупутей и выявить их общие фрагменты). Двигаясь из найденного корня, выделим все нужные вершины и удалим их (удаление одной вершины нужно проводить за константное время). Выведем дерево.

11. С помощью обратного обхода для каждой вершины дерева v определим её высоту $h(v)$. Вычислим корни наибольших полупутей и выберем среди них тот, у которого сумма ключей вершин, которые он соединяет, минимальна (для этого достаточно проанализировать взаимное расположение всех наибольших полупутей и выявить их общие фрагменты). Другим способом решения задачи является поддержка ещё одной метки у каждой вершины v: минимальное значение листа на расстоянии $h(v)$. Удалим корневую вершину найденного полупути. Выведем дерево.

12. С помощью обратного обхода для каждой вершины дерева v определим её высоту $h(v)$ и минимально возможные значения ключей на расстоянии $h(v)$ и $(h(v) - 1)$, если двигаться из вершины v. Среди полупутей, у которых крайние вершины имеют разное число потомков (т. е. полупути, соединяющие лист — лист, нам не подходят), выберем наибольшие по длине, а среди них в качестве искомых оставим те, у которых сумма ключей крайних вершин минимальна. В случае неоднозначности выбора оставим из выбранных полупутей те, у которых корневая вершина имеет минимальное ключевое значение. Если длина полупутей чётная и не возникает неоднозначности с выбором удаляемой вершины, то удалим среднюю по значению вершину. Выведем дерево.

13. С помощью обратного обхода для каждой вершины дерева v определим её высоту $h(v)$ и минимально возможные значения ключей на расстоянии $h(v)$ и $(h(v) - 1)$, если двигаться из вершины v. Среди полупутей, у которых крайние вершины имеют разную высоту (т. е. полупути, соединяющие лист — лист, нам не подходят), выберем наибольшие по длине, а среди них оставим те, у которых сумма ключей крайних вершин минимальна (учтём, что искомый полупуть может являться путём, соединяющим корень дерева и лист). В случае неоднозначности выбора оставим те полупути, у которых корневая вершина имеет минимальное ключевое значение. Если длина полупутей чётная

и не возникает неоднозначности с выбором удаляемой вершины, то удалим среднюю по значению вершину. Выведем дерево.

14. С помощью обратного обхода для каждой вершины дерева v определим её «минимальную» высоту: длину наименьшего пути из вершины v в лист. Найдём длину наименьшего пути между корнем и листьями. С помощью прямого обхода для каждой вершины дерева v определим число вершин, лежащих на пути от корня дерева к вершине v, ключи которых больше (меньше), чем ключ вершины v, и сделаем вывод о том, является ли вершина v средней по значению вершиной некоторого наименьшего пути. Удалим нужные вершины в порядке возрастания ключей. Выведем дерево.

15. Удалим корни деревьев. Выполним обход (любой) двух деревьев: если в первом поддереве двигаемся влево (вправо), то во втором поддереве будем двигаться вправо (влево). Если в некоторый момент окажется, что движение в одном поддереве удаётся сделать, а ассиметричное ему движение в другом поддереве не удаётся выполнить, то деревья не являются зеркальным отображением друг друга по структуре. Выведем соответствующие сообщения.

16. Удалим корень дерева. Затем с помощью обратного обхода для каждой вершины дерева v определим её высоту $h(v)$ и количество листьев $l_h(v)$, которые лежат на пути из v на расстоянии $h(v)$. Во время этого же обхода определим длину наибольшего полупути. Выполним прямой обход и определим для каждой вершины v число полупутей наибольшей длины, проходящих через данную вершину. Прямым левым обходом для каждой вершины выведем её ключ и количество полупутей наибольшей длины, проходящих через неё.

17. С помощью обратного обхода для каждой вершины дерева определим высоту вершины и разность между высотами её поддеревьев. Во время этого же обхода найдём вершины дерева, у которых высота левого поддерева отличается от высоты правого наибольшим образом. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим её. Выведем дерево.

18. С помощью обратного обхода для каждой вершины дерева v определим её высоту $h(v)$ и количество листьев $l_h(v)$, которые лежат на пути из v на расстоянии $h(v)$. Во время этого же обхода определим

длину наибольшего полупути. Выполним прямой обход и определим для каждой вершины число полупутей наибольшей длины, проходящих через данную вершину. Запомним вершины, через которые проходит чётное число наибольших полупутей, и, если такие вершины есть, то удалим ту из них, ключ которой наименьший. Выведем дерево.

19. С помощью обратного обхода для каждой вершины дерева определим высоту. Найдём наибольшие полупути и выберем из них тот, у которого корневая вершина имеет наибольшую метку высоты. Выведем дерево.

20. С помощью обратного обхода для каждой вершины дерева v определим высоту $h(v)$ и минимально возможные значения ключей на расстоянии $h(v)$ и $(h(v) - 1)$, если двигаться из вершины v. Среди полупутей, у которых крайние вершины имеют разный уровень, выберем наибольшие по длине, а среди них оставим те, у которых сумма ключей крайних вершин минимальна (учтём, что искомый полупуть может являться путём, соединяющим корень дерева и лист, а также претендентом может быть полупуть, не являющийся наибольшим). Если длина полупути чётная, то найдём центральную вершину этого полупути и по указанному правилу сделаем её корнем дерева: выполним серию поворотов, после каждого поворота найденная вершина поднимается на один уровень, пока не станет корнем дерева. Выведем сначала сумму ключей крайних вершин найденного полупути, а затем последовательность ключей полученного дерева.

21. С помощью обратного обхода для каждой вершины дерева v определим высоту и число вершин поддерева с корнем в v. Во время этого же обхода найдём вершины дерева, у которых высоты поддеревьев равны, а число потомков в правом и левом поддеревьях отличается. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим её. Выведем дерево.

22. С помощью обратного обхода для каждой вершины дерева v определим высоту и число вершин поддерева с корнем в v. Определим число вершин дерева, у которых высоты поддеревьев не равны, а число потомков в правом и левом поддеревьях равны. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим её. Выведем дерево.

23. С помощью обратного обхода для каждой вершины дерева v определим длины всевозможных путей из вершины v до листьев. Пометим вершины, через которые проходят полупути длины K между листьями. Выберем из помеченных вершин те, которые находятся на наименьшей глубине, и, если их число нечётно, удалим из них среднюю по значению (для этого можно использовать поиск в ширину из корня дерева, спускаясь по уровням до тех пор, пока не встретим вершину, через которую проходят полупути длины K между листьями). Выведем дерево.

24. С помощью обратного обхода для каждой вершины дерева v определим высоту и длины всевозможных путей из вершины v до листьев, а с помощью прямого обхода — её глубину. Пусть H — высота дерева. Среди вершин, через которые проходят полупути длины H между листьями, выберем те, которые находятся на наибольшей глубине, и, если их число нечётно, удалим из них среднюю по значению. Выведем дерево.

25. С помощью обратного обхода для каждой вершины дерева v определим число вершин в поддереве с корнем в v. Во время этого же обхода найдём вершины дерева, у которых число потомков в правом и левом поддеревьях отличается наибольшим образом. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим её. Выведем дерево.

26. С помощью обратного обхода для каждой вершины дерева v определим число вершин в поддереве с корнем в v. Во время этого же обхода найдём вершины дерева, у которых число потомков в правом и левом поддеревьях отличается наибольшим образом, а среди них выберем ту, ключ которой наибольший. Удалим найденную вершину. Выведем дерево.

27. Выполним любой обход, подсчитывая число листьев дерева. Если это число нечётно, то внутренним обходом найдём средний лист и удалим его отца. Выведем дерево.

28. С помощью обратного обхода для каждой вершины дерева определим её высоту и подсчитаем количество вершин, удовлетворяющих нужным требованиям. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим её. Выведем дерево.

29. С помощью обратного обхода для каждой вершины дерева v определим число вершин дерева с корнем в v и подсчитаем количество вершин, удовлетворяющих нужным требованиям. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим. Выведем дерево.

30. С помощью обратного обхода для каждой вершины дерева определим её высоту и подсчитаем количество вершин, удовлетворяющих нужным требованиям. Если число нужных вершин нечётно, то внутренним обходом найдём среднюю из них и удалим её. Выведем дерево.

31. С помощью обратного обхода для каждой вершины дерева v определим её высоту $h(v)$ и количество листьев $l_h(v)$, лежащих на пути из v на расстоянии $h(v)$. Во время этого же обхода определим длину наибольшего полупути. Выполним прямой обход и определим для каждой вершины число полупутей наибольшей длины, проходящих через данную вершину. Запомним вершины, через которые проходит нечётное число наибольших полупутей, и, если такие вершины есть, удалим ту из них, ключ которой наибольший. Выведем дерево.

32. С помощью обратного обхода для каждой вершины дерева v определим её высоту $h(v)$ и количество листьев $l_h(v)$, лежащих на пути из v на расстоянии $h(v)$. Во время этого же обхода определим длину наибольшего полупути. Выполним прямой обход и определим для каждой вершины число полупутей наибольшей длины, проходящих через данную вершину. Запомним вершины, через которые проходит чётное положительное число наибольших полупутей, и, если такие вершины есть и их количество нечётно, удалим среднюю из них. Выведем дерево.

33. С помощью обратного обхода для каждой вершины дерева определим её высоту. Вычислим наибольшие полупути и выберем из них тот, у которого сумма ключей конечных вершин минимальна (для этого нужно исследовать взаимное расположение корней полупутей наибольшей длины и сделать вывод о том, какой из наибольших полупутей станет искомым). Удалим среднюю по значению вершину искомого наибольшего полупути (для этого достаточно спуститься из корня полупути, анализируя число поворотов налево и направо). Выведем дерево.

34. Для каждой вершины v необходимо сначала найти длину наибольшего полупути, который через неё проходит. Эту величину можно

вычислить на основании трёх меток, поставленных в соответствие вершине v: длины наибольшего полупути, для которого вершина v является концевой (т. е. наибольшего полупути, идущего из вершины v в вершину, не принадлежащую поддеревьям вершины v), а также высоты левого и правого поддеревьев вершины v.

Предположим, что для вершины v длина наибольшего полупути, который через неё проходит, равна l и $l < k$. Тогда любой другой полупуть, проходящий через эту вершину, также имеет длину меньше, чем k, а значит, полупуть длины k через вершину v точно не пройдёт. Учитывая тот факт, что расстояние от v до любой вершины дерева строго меньше k, получим, что вершина v подходит нам в качестве претендента на удаление. Предположим, что для вершины v длина наибольшего полупути, который через неё проходит, равна l и $l \geqslant k$. Значит, существует и полупуть длины k, проходящий через v (отметим, что концевые вершины этого полупути не обязательно листья), т. е. вершина v не подходит нам для удаления, так как одно из условий задачи нарушается.

35. Как и в предыдущей задаче, для каждой вершины v нужно найти длину l наибольшего полупути, который через неё проходит.

Если $l < k$, то вершина v является претендентом на удаление.

Если $l \geqslant k$, то мы не можем сразу исключить вершину v из рассмотрения, как это было сделано в предыдущей задаче. Необходимо проверить, существует ли полупуть длины k с висячими концевыми вершинами, проходящий через вершину v, а также проанализировать расстояние от вершины v до всех остальных вершин дерева.

Рассмотрим, как понять, проходит ли через данную вершину v полупуть длины k с висячими концевыми вершинами. Пусть L — множество длин путей от вершины v до листьев левого поддерева вершины v. Аналогично определим множество R — множество расстояний до листьев правого поддерева. Введём множество U — множество расстояний от вершины v до всех остальных листьев дерева, не лежащих в поддереве с корнем в вершине v (см. пример на рис. 2.27: множество L включает расстояния до листьев 1, 3, 5, множество R — до листа 7, во множестве U хранятся расстояния до листьев 9 и 11).

Пусть (X, Y) — одна из пар множеств (L, R), (L, U), (R, U). Если существуют такие числа $x \in X$ и $y \in Y$, что $x + y = k$, то через вершину v проходит полупуть длины k, соединяющий два листа.

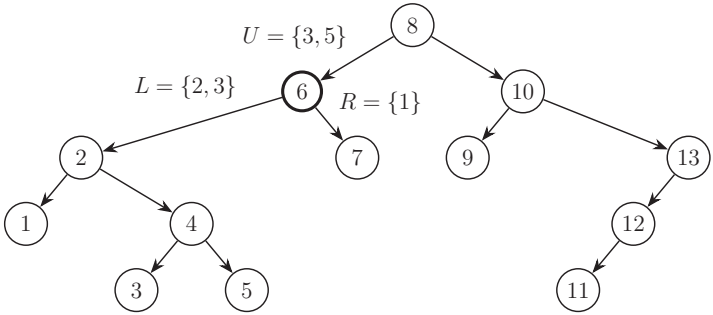


Рис. 2.27. Множества длин полупутей L, R, U от вершины 6 до листьев

Множества L, R считаются одним обратным обходом, затем при следующем обходе множество U динамически пересчитывается на основании имеющейся информации.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1. *Адельсон-Вельский Г. М.*, *Диниц А. В.*, *Карзанов А. В.* Потоковые алгоритмы. — М. : Наука, 1975. — 119 с.

2. Алгоритмы: построение и анализ / Т. Кормен [и др.]. — М. : Вильямс, 2005. — 1296 с.

3. *Котов В. М.*, *Мельников О. И.* Информатика. Методы алгоритмизации : учеб. пособие для 10 – 11 кл. общеобразоват. шк. с углубл. изучением информатики. — Минск : Нар. асвета, 2000. — 221 с.

4. *Котов В. М.*, *Соболевская Е. П.*, *Толстиков А. А.* Алгоритмы и структуры данных : учеб. пособие. — Минск : БГУ, 2011. — 267 с. — (Классическое университетское издание).

5. Лекции по теории графов / В. А. Емеличев [и др.]. — М. : Наука, 1990. — 384 с.

6. Теория алгоритмов : учеб. пособие / П. А. Иржавский [и др.]. — Минск : БГУ, 2013. — 159 с.

СОДЕРЖАНИЕ

Учебное издание

Котов Владимир Михайлович
Орлович Юрий Леонидович
Соболевская Елена Павловна и др.

СБОРНИК ЗАДАЧ ПО ТЕОРИИ АЛГОРИТМОВ

Учебно-методическое пособие

Редактор *Т. А. Белапко*
Художник обложки *С. А. Соболь*
Технический редактор *Т. К. Раманович*
Компьютерная вёрстка *С. А. Соболя*
Корректор *Е. В. Гордейко*

Подписано в печать 31.05.2017. Формат 60×84/16. Бумага офсетная.
Печать офсетная. Усл. печ. л. 10,69. Уч.-изд. л. 10,6.
Тираж 150 экз. Заказ 347.

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.