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

Сборник задач по теории алгоритмов:

организация перебора вариантов и приближенные алгоритмы

ОГЛАВЛЕНИЕ

Тема 1. Организация полного перебора

В том случае, когда для решения задачи не удается разработать эффективный алгоритм решения, используют алгоритм полного перебора.

1.1. Построение дерева решений

Основной принцип, на котором базируются методы полного перебора вариантов, состоит в разбиении начальной задачи P_0 на подзадачи $P_1, P_2, \ldots, P_k$ (в целом представляющих всю задачу P_0) с последующей попыткой разрешить каждую из этих подзадач. Это разбиение описывается деревом, представленным на рис. 1.1, причем вершины изображают подзадачи.

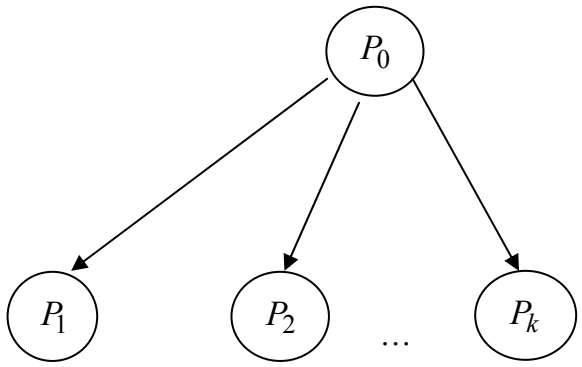


Рис. 1.1

Выражение "решить подзадачу" означает одну из следующих возможностей:

1) находится оптимальное решение,

2) показывается, что значение оптимального решения подзадачи хуже, чем лучшее из найденных решений,

3) показывается, что подзадача является недопустимой.

Смысл разбиения задачи P_0 на подзадачи состоит в том, что эти подзадачи проще решить, так как они имеют меньший размер, или обладают структурой, не присущей первоначальной задаче P_0. Может оказаться, что подзадачу P_i нельзя решить, и эта подзадача также разбивается на новые подзадачи $P_{i_1}, P_{i_2}, \ldots, P_{i_r}$, как это показано на рис. 1.2. Это разбиение (называемое также ветвлением) повторяется для каждой подзадачи, которая не может быть решена.

На любом этапе полное множество подзадач, требующих решения, представляется множеством концевых вершин всех путей, исходящих из

корня дерева решений (корень дерева – начальная задача P_0). Эти концевые вершины называются висячими вершинами, и на рис. 1.2 это

$$P_1,\ldots,P_{i_1},P_{i_2},\ldots,P_{i_r},\ldots,P_k .$$

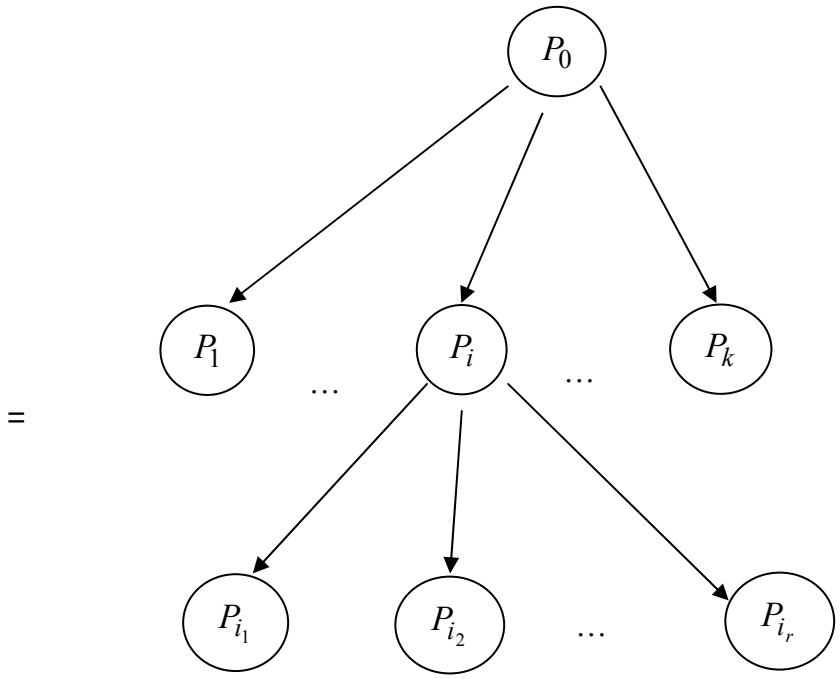


Рис. 1.2

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

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

где $\{P_i\}$ обозначает множество всех допустимых решений задачи P_i. Так как данное соотношение должно быть применено к каждому разбиению, то $\{P_0\}=\bigcup\{P_j\}$, где P_j – висячая вершина дерева.

Понятно, что было бы хорошо избежать дублирования построенных решений, т. е. разбивать задачу P_i на подзадачи $P_{i_1},P_{i_2},\ldots,P_{i_r}$ таким образом, чтобы

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

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

Хотя условие (1.1) не является необходимым для полноценного поиска с деревом решений, оно, тем не менее, имеет большие выгоды с вычислительной точки зрения, так как:

• для задачи оптимизации P_0 оптимальное решение является решением одной и только одной подзадачи, представляемой висячей вершиной;

• для задачи полного перебора объединение множеств решений подзадач, представляемых висячими вершинами, дает множество всех решений задачи P_0 без дублирования.

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

Способы ветвления

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

Покомпонентное ветвление

Возможно разбиение P_i на четыре подзадачи – $P_{i_1}, P_{i_2}, P_{i_3}, P_{i_4}$, причем для подзадачи P_{i_1} мы полагаем $\chi = a$; для P_{i_2} полагаем $\chi = b$; для для P_{i_3} полагаем $\chi = c$; и для для P_{i_4} полагаем $\chi = d$. Каждая из подзадач $P_{i_1}, P_{i_2}, P_{i_3}, P_{i_4}$, содержит $n-1$ переменных и, возможно, допускает более простое решение, чем задача P_i (рис. 1.3).

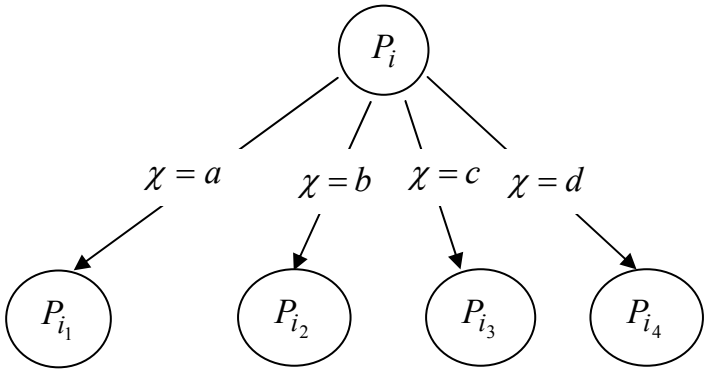


Рис. 1.3

Разбиение по некоторому признаку

Возможно другое разбиение P_i на две подзадачи – P_{i_1}, P_{i_2}, где для P_{i_1} мы полагаем $\chi = a$; а для P_{i_2} полагаем $\chi \neq a$, т. е. $\chi = b, c,$ или d (рис. 1.4).

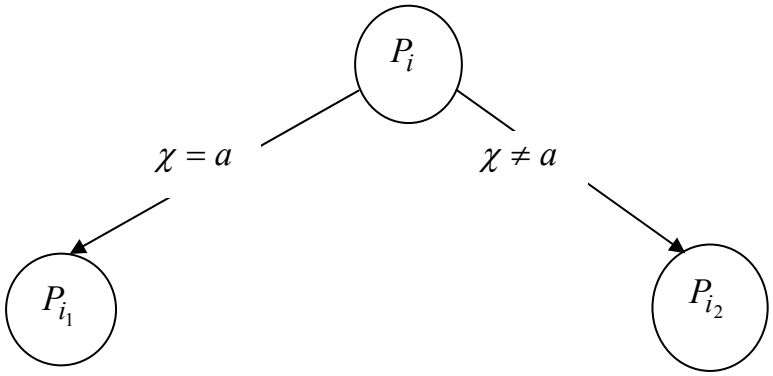


Рис. 1.4

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

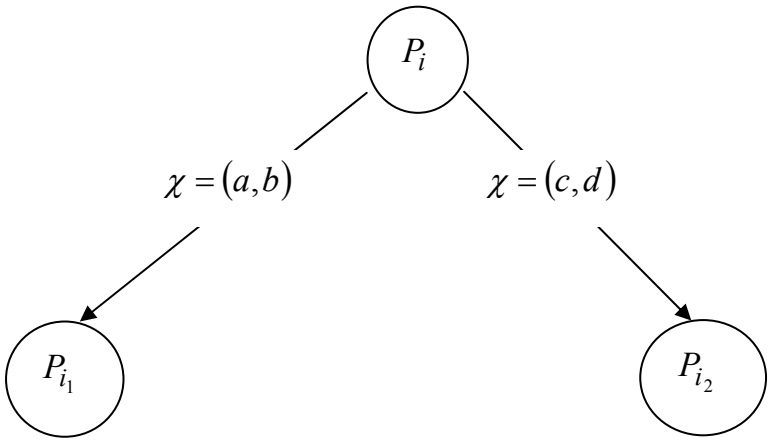


Рис. 1.5

Все ветвления являются допустимыми и удовлетворяют условию (1.1). Какому из них отдать предпочтение – зависит от природы решаемой задачи, причем возможности первых двух типов используются чаще остальных. Но при этом следует отметить, что подзадачи этих подзадач могут и не обладать свойством (1.1). Подзадача P_{i_1} оптимальна на подзадаче $\chi = a$ и подзадача P_{i_2} оптимальна на подзадаче $\chi = a$.

1.2. Способы обхода дерева решений

Из вышесказанного видно, что всякая подзадача, представляемая вершиной и не поддающаяся решению, может быть в любой момент разбита на подзадачи. Существует две основные стратегии в зависимости от того, как выбирается следующая висячая вершина для продолжения процесса ветвления.

Фронтальное ветвление

При такой стратегии ветвление выполняется для наиболее перспективной вершины так начальная задача P_0 разбивается на подзадачи $P_1, P_2, \ldots, P_k$, образуя фронт ветвления.

Из этих подзадач выбирается наиболее перспективная подзадача, и разбивается на подзадачи, увеличивая количество вершин.

Вид дерева решений при этом типе поиска показан на рис. 1.6.

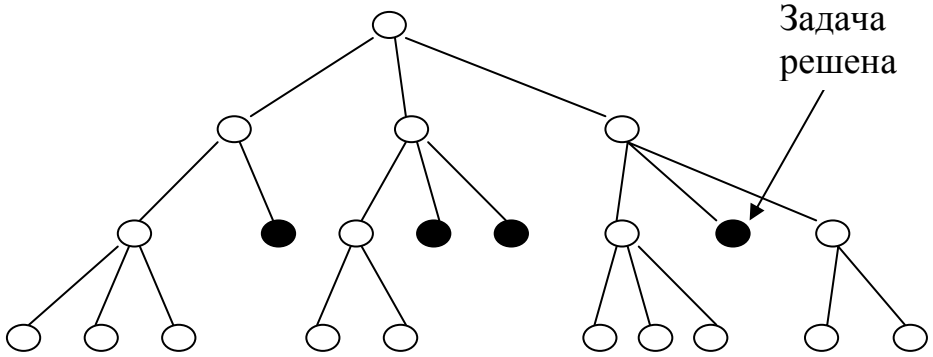


Рис. 1.6

Одностороннее ветвление

При этом типе поиска ветвление осуществляется в последней выбранной подзадаче и такой процесс ветвления продолжается до тех пор, пока не будет порождена подзадача, которую можно решить. В этом месте делается возврат, т. е. берется предпоследняя порожденная подзадача и ветвление продолжается в соответствующей вершине.

При этом типе поиска задачи, получаемые на каждом этапе, хранятся в стеке вместе с исходной задачей. Вновь получаемые задачи помещаются в стек, а когда подзадача разрешена, она удаляется из стека.

Вид дерева решений при этом типе поиска, когда разрешается первая подзадача, показан на рис. 1.7, где порядок приоритета исследования получаемых подзадач отражен соответствующей нумерацией.

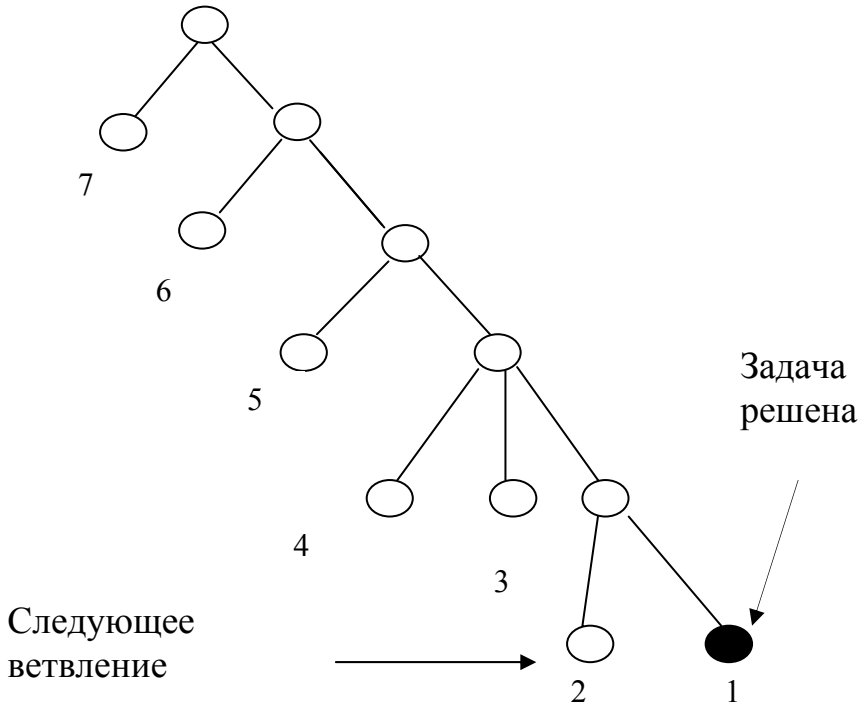


Рис. 1.7

Такая стратегия часто называется "*поиск с возвратом"*. Основными ее достоинствами является простота реализации и минимизация используемой памяти, так как в отличие от фронтального ветвления нет необходимости хранить формат вершин кандидатов для ветвления. Может показаться, что аналогичным образом ведет себя и *поиск в глубину* при обходе вершин графа. В поиске в глубину мы строили *глубинное дерево поиска* (т. е. расширяли некоторый частичный путь до тех пор, пока из конечной вершины этого частичного пути можно еще куда-нибудь пойти). Если же нам не удается расширить частичный путь, то происходит возврат по дереву на шаг назад и делается попытка движения в другом направлении. *Заметим, что при дальнейшем движении при поиске в глубину нам будут доступны лишь те вершины, которые ранее никогда не посещались.* Для алгоритма полного перебора вариантов при расширении частичного решения такого ограничения нет. Проиллюстрируем это на следующем примере.

Предположим, что для графа, приведенного на рис. 1.8, необходимо найти элементарный маршрут между 1-ой и 6-ой вершинами, используя алгоритм поиска в глубину.

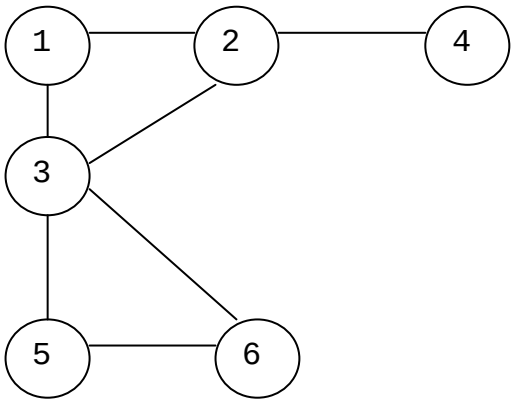


Рис. 1.8

В результате поиска получаем корневое дерево поиска в глубину, представленное на рис. 1.9. Несложно увидеть, что в корневом дереве поиска в глубину нет повторяющихся вершин.

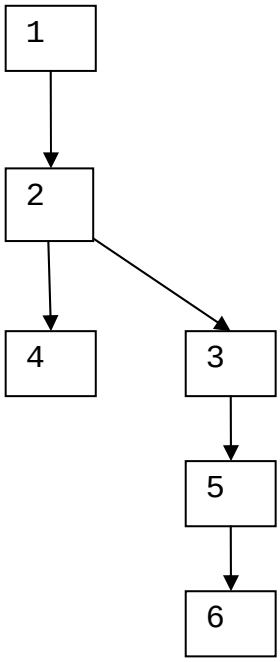


Рис. 1.9

Построим теперь для графа, приведенного на рис. 1.8, все элементарные маршруты между 1-ой и 6-ой вершинами, используя алгоритм полного перебора вариантов с односторонним ветвлением.

Ветвление будет начато в вершине с номером 1. Если выбраны некоторые вершины графа $v_{1,}v_2,\ldots,v_{k-1}$ в качестве частичного решения, то при выборе очередной вершины v_k множество возможных выборов представляет собой вершины графа, которые смежны с вершиной v_{k-1} и от-

личны от вершин $v_{1,}v_2,\ldots,v_{k-1}$. Ветвление в некоторой вершине v_k закончится, когда $v_k = 6$ (частичное решение является решением задачи) или текущее частичное решение из вершины v_k не может быть больше расширено.

Применяя данную стратегию для графа, изображенного на рис. 1.8, получим следующую древовидную структуру (рис. 1.10). Сгенерировано четыре различных пути из 1-ой вершины графа в 6-ую вершину $(1 \to 2 \to 3 \to 5 \to 6; 1 \to 2 \to 4; 1 \to 3 \to 5 \to 6;\ 1 \to 3 \to 6)$.

Несложно увидеть, что в дереве решений (рис. 1.10) некоторые вершины повторяются.

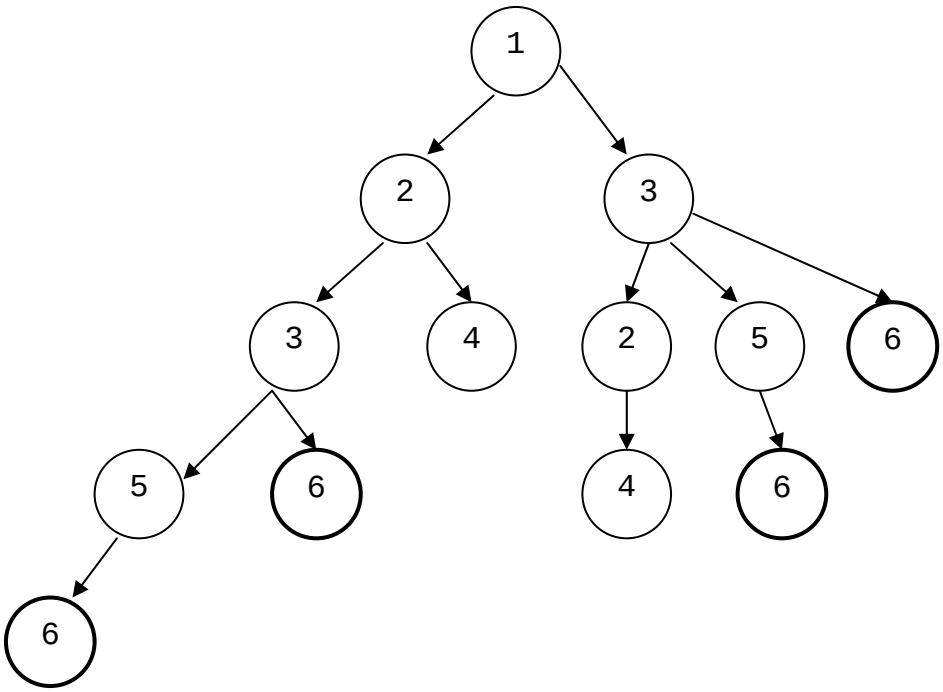


Рис. 1.10

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

Основной целью при организации перебора вариантов решения является сокращение количества решаемых подзадач. Рассмотрим три стратегии, которые предназначены для отсева неперспективных вариантов ветвления.

1. *Отсев по повторению* (исключение повторяющихся подзадач).

2. *Отсев по допустимости* (если когда-то пришли в некоторое множество и выяснили, что оно нам ничего допустимого не дает, то незачем в дальнейшем туда ходить).

3. *Отсев по рекорду* (если заранее можно определить, что некоторая подзадача не даст решения лучше, чем построенное алгоритмом ранее (рекорд), то незачем решать такую подзадачу).

Отсев по повторению

Кликой графа $G = (V, E)$ называется такое максимальное по включению подмножество вершин графа, что любые две вершины этого подмножества соединены ребром.

Для графа, приведенного на рис. 1.11, мы имеем три клики:

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

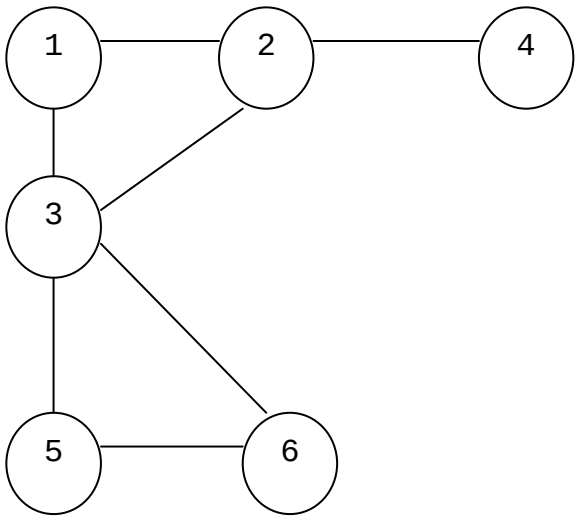


Рис. 1.11

Одним из алгоритмов построения *множества всех клик графа* является поиск с возвратом. Каждая вершина дерева поиска будет соответствовать полному подграфу графа (полный подграф задается множеством его вершин), а каждое ребро дерева поиска – вершине графа. Корень дерева– пустое множество вершин.

Предположим, что некоторой вершине дерева поиска поставлен в соответствие полный подграф графа с множеством вершин C. Пусть вершина w графа смежна с каждой из вершин множества C. Тогда вершина $C \cup \{w\}$ будет сыном вершины C, а ребро, идущее от C к $C \cup \{w\}$ будет соответствовать вершине w. Если не существует вершин смежных с каж-

дой из вершин множества C, то ветвление в соответствующей вершине дерева завершено и данная висячая вершина порождает клику графа.

Для графа, изображенного на рис. 1.11 дерево решений будет иметь следующий вид (рис. 1.12):

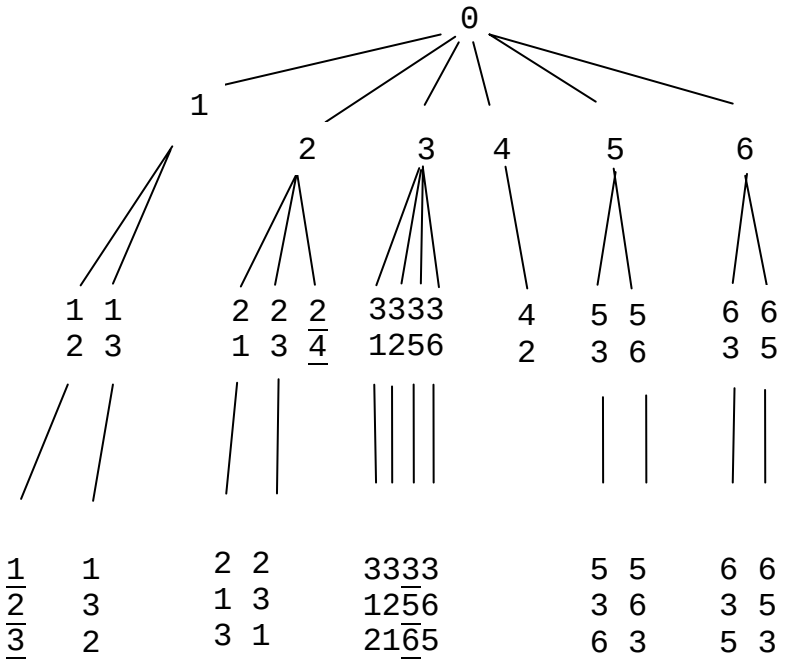


Рис. 1.12

На рис. 1.12 мы видим, что поиск с возвратом привел нас к тому, что мы породили 14 клик, но только три из них различны (выделены подчеркиванием).

Число клик в графе может расти экспоненциально относительно числа вершин. Поэтому важно при построении дерева решений *избегать повторений*. Для того, чтобы избежать повторений, будем использовать следующие две теоремы.

Теорема 1.1. Пусть C – некоторая вершина дерева поиска с возвратом, а $C \cup \{w\}$ – ее первый сын, который должен быть исследован. Предположим, что все поддеревья вершины $C \cup \{w\}$ уже исследованы, и при этом порождены все клики, включающие вершины множества $C \cup \{w\}$. Тогда необходимо исследовать только тех из оставшихся сыновей вершины C, которые не смежны с вершиной w графа (рис. 1.13).

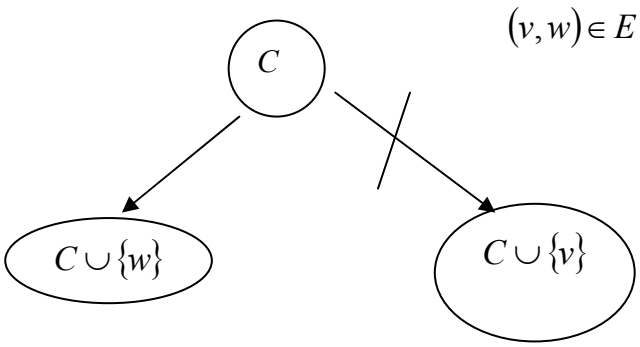


Рис. 1.13

Данная теорема применяется только к первому исследуемому сыну вершины поиска C и не применима к оставшимся его сыновьям.

Теорема 1.2. Пусть C – некоторая вершина дерева поиска с возвратом и $\underline{C}$– ее предок в дереве поиска с возвратом. Если все поддеревья вершины $\underline{C}\cup\{w\}$ уже исследованы, и при этом порождены все клики, включающие вершины множества $\underline{C}\cup\{w\}$, то все не исследованные поддеревья с корнями в вершине $C\cup\{w\}$ можно проигнорировать.

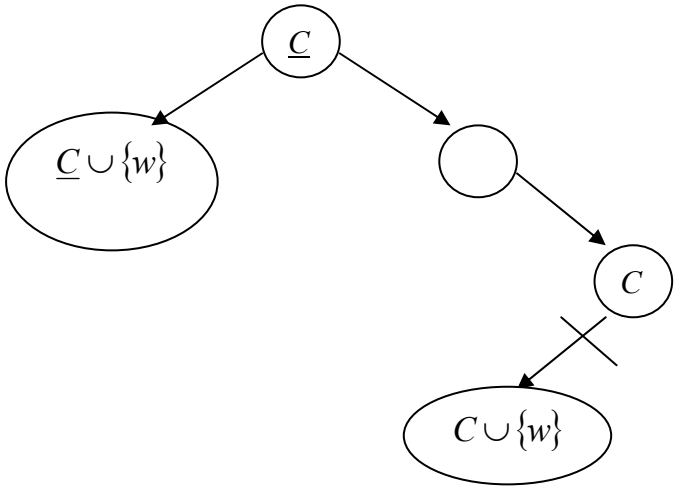


Рис. 1.14

С учетом приведенных теорем 1.1 и 1.2 (отсев по повторению) дерево решений поиска с возвратом для графа, приведенного на рис. 1.11, будет иметь следующий вид (рис. 1.15).

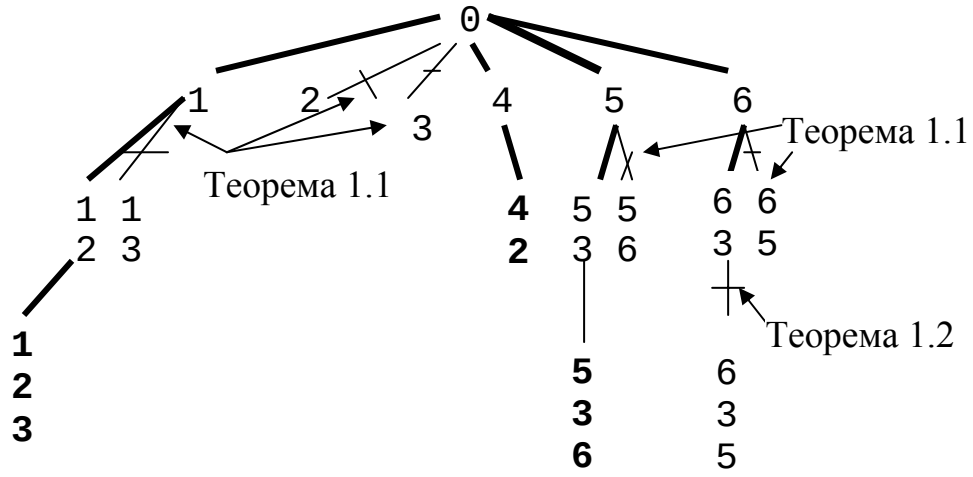


Рис. 1.15

Алгоритм можно улучшить, если более аккуратно выбирать корень первого исследуемого поддерева. В соответствии с теоремой 1.1, такой вершиной должна быть вершина, которая смежна с наибольшим количеством вершин, так как такой выбор позволит отсечь наибольшее число поддеревьев.

Одним из наиболее простых и используемых способов отсева по повторению является построение подзадач в лексикографическом порядке, который гарантирует однозначность решений. Однако такой подход не всегда сочетается с другими способами отсечения.

Отсев по допустимости

Рассмотрим задачу определения множество всех контуров ориентированного графа (орграфа) $G=(V,U),\quad |V|=n,|U|=m$.

Достаточно просто ответить на вопрос: содержит ли неориентированный граф (в дальнейшем просто граф) цикл (или орграф – контур)? Для этого достаточно для графа (орграфа) выполнить алгоритм поиска в глубину с пометкой ребер (дуг). Граф (орграф) содержит цикл (контур) тогда и только тогда, когда в нем будет существовать хотя бы одно обратное ребро (дуга).

Не сложной является и задача построения *фундаментального множества циклов графа* $G=(V,E)$ (такого минимального множества циклов, из которого могут быть построены все циклы графа). Для решения этой задачи мы выполняем поиск в глубину, помечая при этом все ребра графа как «древесные» или «обратные». Во время этого поиска будет построено глубинное дерево (дереву принадлежат все ребра графа, которые были помечены как древесные). Каждое обратное ребро при добавлении

к глубинному дереву поиска порождает ровно один цикл. Мощность фундаментального множества циклов графа равна $|E|-|V|-1$.

Проиллюстрируем построение фундаментального множества циклов для графа, изображенного на рис. 1.16.

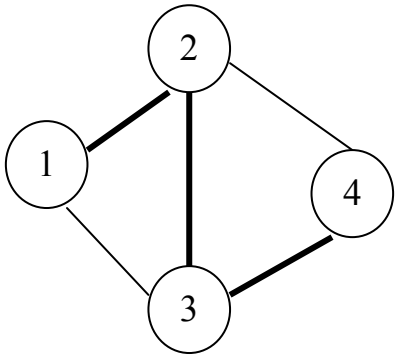


Рис. 1.16

Для заданного графа, начиная, например, с вершины 1, выполним алгоритм поиска в глубину. На рис. 1.16 ребра глубинного дерева поиска выделены жирными линиями. Обратными являются ребра $(1,3)$ и $(2,4)$. На рис. 1.17 показано фундаментальное множество циклов заданного графа.

Рис. 1.17

Более сложной является задача *построения множества всех контуров орграфа*. Очевидно, что если мы научимся решать эту задачу для орграфа, то мы решим ее и для графа (каждое ребро графа заменим двумя дугами с противоположными направлениями).

Заметим, что количество всех контуров полного орграфа с n вершинами может иметь порядок $(n-1)!$. Поэтому важной является задача отсечения повторяющихся контуров, а также отсев неперспективных ветв-

лений (т. е. таких путей, которые заведомо не дадут контура). Для отсева повторяющихся контуров введем понятие *корня контура*.

Корень контура – вершина с наименьшим номером среди всех вершин этого контура (для контура $3 \to 2 \to 6 \to 3$ корнем является вершина 2).

Алгоритм

Последовательно будем строить контуры с корнями в вершинах $1, 2, \ldots, n$.

Для построения всех контуров с корнем в вершине s поступаем следующим образом.

- Считаем, что все вершины орграфа не заблокированы, т. е. доступны.
- Поиском в глубину (идем в вершины, которые не заблокированы) строим ориентированный путь $(s, v_1, v_2, \ldots, v_k)$ такой, что $v_i > s \quad \forall 1 \le i \le k$ (*отсечение по повторению*). Как только вершина присоединяется к пути, она сразу же блокируется (за исключением вершины s). Блокирование вершин v_i, присоединяемых к пути, выполняется для того, чтобы не допустить прохождение контуров с корнями в вершинах v_i.
- Контур с корнем в вершине s построен, если на некотором шаге алгоритма $v_{k+1} = s$. После построения контура $(s, v_1, v_2, \ldots, v_k, s)$ исследуем следующую дугу, выходящую из вершины v_k, и если она ведет в не заблокированную вершину, то продолжаем поиск в глубину из этой вершины. Если же оказывается, что все дуги, выходящие из вершины v_k исследованы, то возвращаемся в вершину v_{k-1} и исследуем выходящие из нее пути. Если из некоторой вершины v_k был найден хотя бы один контур с корнем в стартовой вершине s, то при возврате из вершины v_k она должна быть разблокирована (в противном случае вершина остается заблокированной даже после возвращения из нее). В свою очередь если на некотором этапе произошло разблокирование некоторой вершины v_i, то при этом вершины, которые не принадлежат рассматриваемому пути $s \to v_1 \to \ldots \to v_i$ и являются предшественника вершины v_i (являются началами дуг, входящих в вершину v_i) также должны быть разблокированы. Таким образом, разблокирование некоторой вершины пути может привести к цепочке разблокирования других ранее заблокированных вершин (*отсечение по допустимости)*.

• Процедура построения контура с корнем в вершине *s* завершится, когда мы попытаемся вернуться во время поиска в глубину за вершину *s*. После того, как все контуры с корнем в вершине *s* будут построены, вершина *s* может быть удалена из орграфа вместе с инцидентными ей дугами.

Проиллюстрируем работу алгоритма на следующем примере. Для графа, изображенного на рис. 1.18, построить множество всех контуров с корнем в вершине 1, используя отсечения по повторению и допустимости.

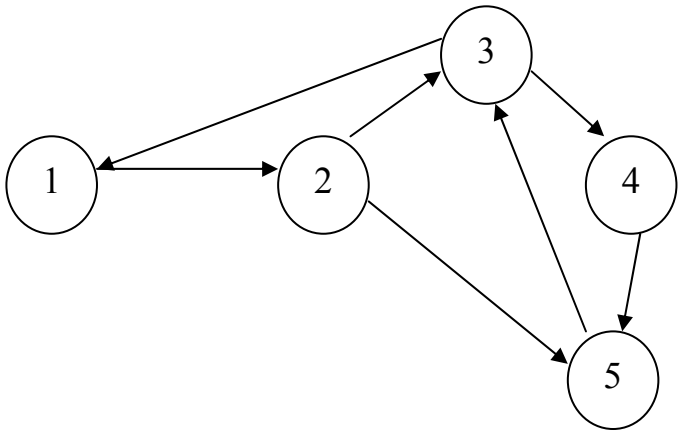


Рис. 1.18

При поиске контуров с корнем в вершине 1 мы построим путь $1 \to 2 \to 3 \to 4 \to 5$ (рис. 1.19).

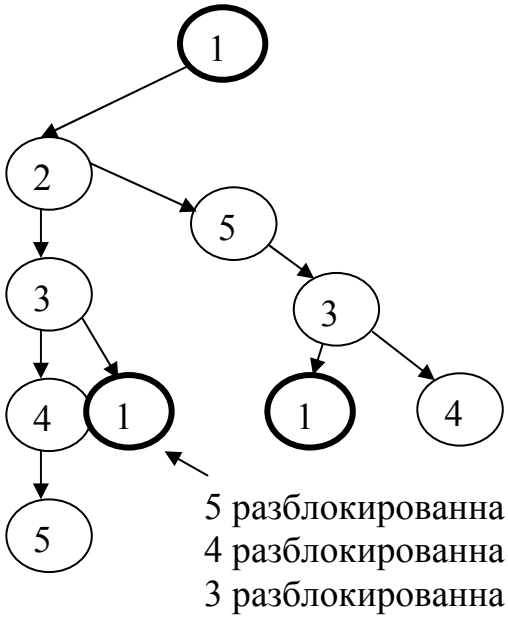


Рис. 1.19

После чего все вершины (за исключением вершины 1) станут заблокированными, а контур с корнем в вершине 1 не построен. Поэтому бу-

дет осуществлен возврат из вершины 5, но при этом она останется заблокированной. Аналогично произойдет и с вершиной 4.

Для пути $1 \to 2 \to 3$ существует еще одна не просмотренная дуга $3 \to 1$, приводящая к контуру. Поэтому после того, как будет осуществлен возврат из вершины 3, она станет разблокированной. Разблокирование вершины 3 повлечет за собой разблокирование вершины 5 (вершина 5 не принадлежит пути $1 \to 2 \to 3$ и является предшественником вершины 3 в орграфе). В свою очередь, разблокирование вершины приведет к разблокированию вершины 4.

Теперь при возврате из вершины 3 в вершину 2 у нас есть три разблокированные вершины: 3, 4 и 5. Продолжаем поиск в глубину из вершины 2 по ранее не исследованным дугам. Из вершины 2 идем по не просмотренной дуге $2 \to 5$ в незаблокированную вершину 5 (блокируем ее), а затем из вершины 5 в незаблокированную вершину 3 (блокируем ее) и наконец в вершину 1 (рис. 1.19). Получаем контур $1 \to 2 \to 5 \to 3 \to 1$. Исследуем далее дуги, выходящие из вершины 3. Двигаемся из вершины 3 по еще не просмотренной дуге в незаблокированную вершину 4 (блокируем), а далее пути нет так как все вершины заблокированы $(1 \to 2 \to 5 \to 3 \to 4)$. Возвращаемся из вершины 4, оставляя ее заблокированной. Осуществляем возврат из вершины 3, так как все ребра, выходящие из этой вершины обследованы. Вершина 3 становится разблокированной (из нее был контур в вершину 1). Осуществляем возврат из вершины 5 (разблокирована), что приводит к разблокировке вершины с номером 4. Затем осуществляем возврат и из вершины 2 (разблокирована) так как все дуги, выходящие из этой вершины исследованы. После чего осуществляется возврат из вершины 1 и алгоритм построения контуров с корнем в вершине 1 завершен (рис. 1.19).

В результате построены два контура с корнем в вершине 1: $1 \to 2 \to 3 \to 1$ и $1 \to 2 \to 5 \to 3 \to 1$.

Пример 1.1. Построить множество контуров, для орграфа, изображенного на рис. 1.20, используя отсечения по повторению и допустимости.

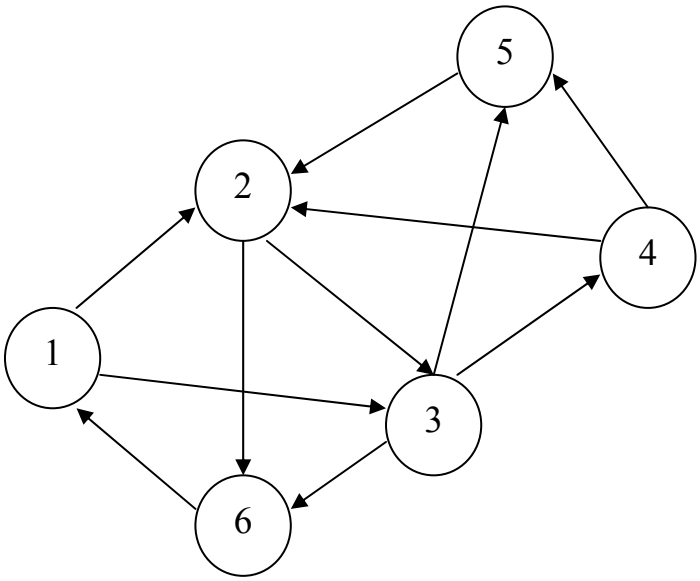


Рис. 1.20

Решение. Последовательность действий построения множества контуров с корнем в вершине 1 приведена на рис. 1.21.

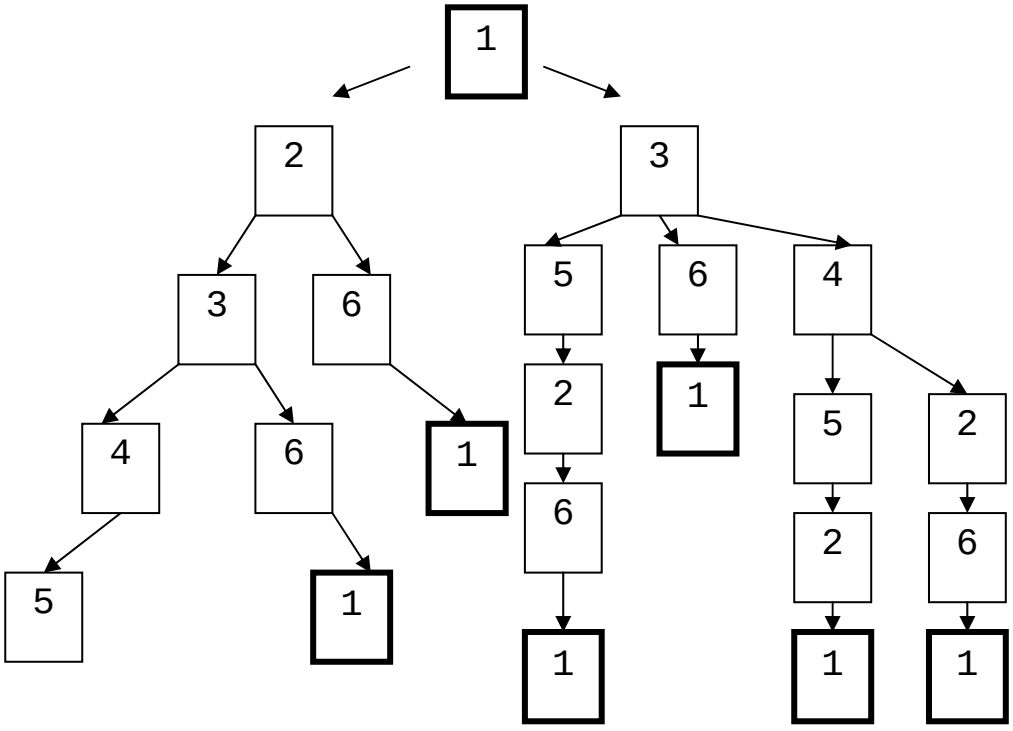


Рис. 1.21

Контуры с корнем в вершине 1 построены:

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

Удаляем из орграфа, приведенного на рис. 1.20 вершину 1 вместе с инцидентными ей дугами и для модифицированного орграфа (рис. 1.22) строим множество контуров с корнем в вершине 2.

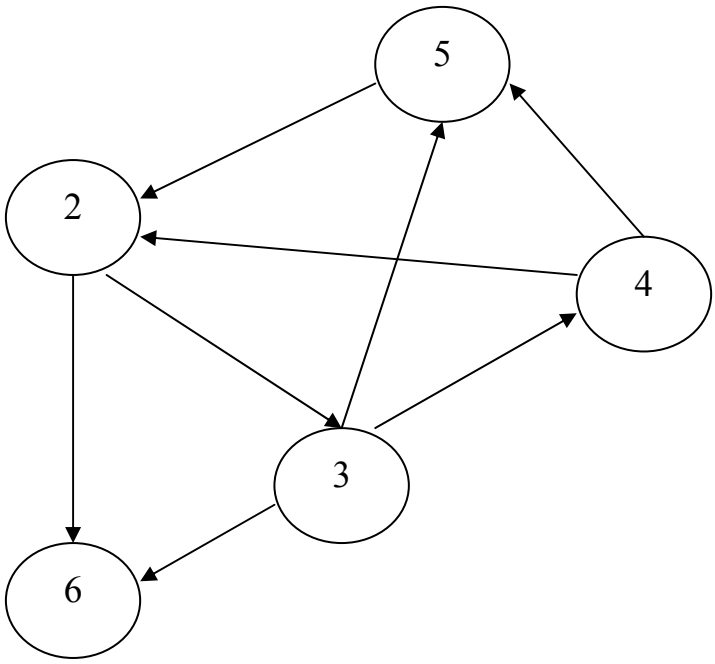


Рис. 1.22

Получаем следующее дерево ветвления на подзадачи (рис. 1.23).

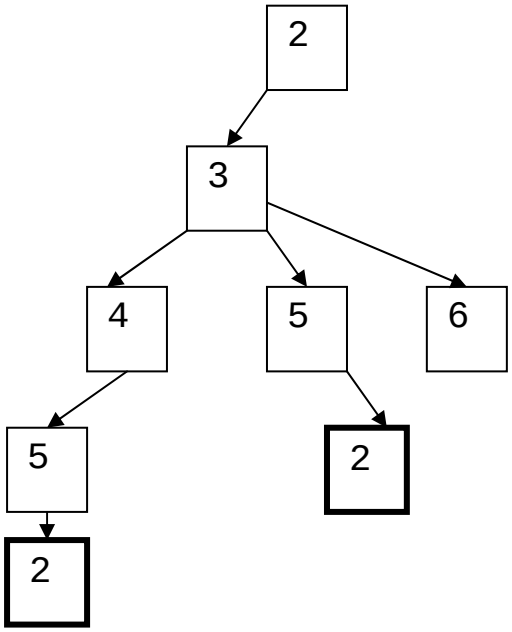


Рис. 1.23

Контуры с корнем в вершине 2 построены:

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

Удаляем из орграфа, приведенного на рис. 1.22 вершину 2 вместе с инцидентными ей дугами (рис. 1.24).

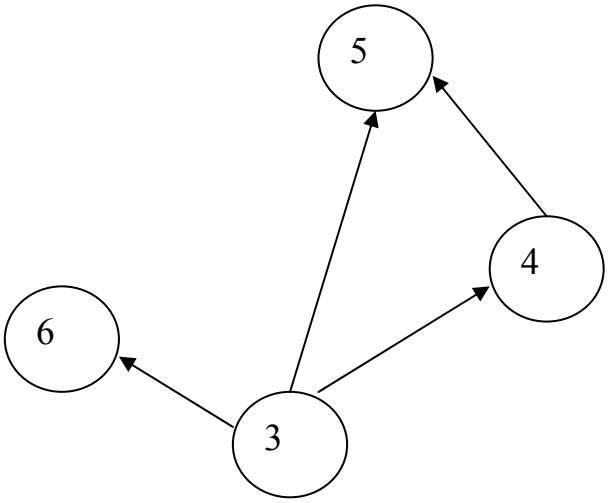


Рис. 1.24

В модифицированном орграфе контуров нет, и алгоритм заканчивает свою работу.

Отсев по рекорду
(применение оценок)

Предположим, что каждому решению (в том числе частичному) соответствует некоторая целевая функция $F(x_1,x_2,\ldots,x_k)$, причем

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

Во многих задачах дискретной оптимизации целевая функция удовлетворяет равенству:

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

где $f(x_k) \ge 0, \quad k = 1,\ldots,n.$ Такая функция является сепарабельной.

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

При организации перебора вариантов решения строится дерево вариантов, причем частичным решениям соответствуют внутренние вершины дерева, а решениям исходной задачи – листья. Целью организации поиска является такой обход древа, который позволяет построить все оптимальные решения, просмотрев минимальное число вершин дерева. Самым простым решением поставленной задачи является полный обход дерева. Однако существуют механизмы, позволяющие для многих задач существенно сократить число рассматриваемых вершин. Для этого достаточно определить, какие из подзадач целесообразно рассматривать (ветвить), а какие являются неперспективными и их можно игнорировать в силу того факта, что они заведомо не приведут к построению оптимального решения.

Для этого определим следующие основные понятия.

1. F^* – значение оптимального решения задачи.
2. F – рекорд, который является значением наилучшего известного решения. Рекорд характеризует приближение к оптимальному решению. Поэтому важно удачно выбрать начальное рекордное решение. Его находят обычно с помощью приближенных методов решения исходной задачи. Если не известно ни одного допустимого решения исходной задачи, то не остается ничего другого, как положить значение F равным бесконечности, для задачи минимизации и 0 для задачи максимизации (считая, что $F^* \ge 0$).
3. Вершине x дерева ветвления соответствует некоторое частичное решение, в котором часть переменных $x = (x_1,x_2,\ldots,x_k)$ зафиксирована, а ос-

тальные переменные $\bar{x} = (x_{k+1},\ldots,x_n)$ являются свободными, т.е. пока еще не определены. Значение частичного решения x определим как $F(x) = f(x_1) + f(x_2) + \cdots + f(x_k)$.

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

$$HO(\bar{x}) \le opt(\bar{x}) \le BO(\bar{x}),$$

где $opt(\bar{x})$ – значение оптимального решения подзадачи x (рис. 1.25).

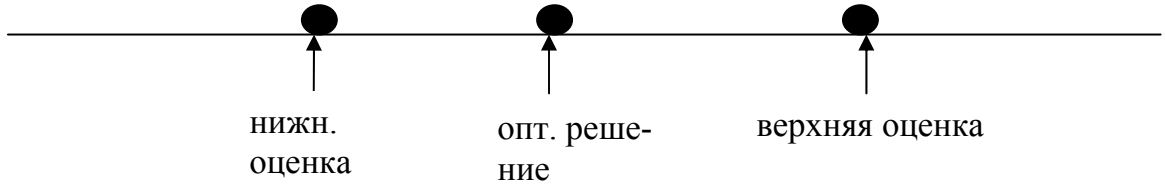


Рис. 1.25

Для вычисления оценки, существует несколько способов. Часто рассматривается более общая задача, которая легко решается. Для этого, например, от дискретных переменных переходят к непрерывным, либо убирают (ослабляют) некоторые ограничения, тем самым гарантируя, что значение оптимального решения более общей задачи не больше в задаче минимизации и не меньше в задаче максимизации, чем значение оптимального решения исходной задачи. Затем значение оптимальное решение более общей задачи берется в качестве оценки.

Если в задаче минимизации для вершины дерева x выполняется неравенство

$$F < F(x) + HO(\bar{x}),$$

то понятно, что

$$F < F(x) + opt(\bar{x})$$

поэтому частичное решение x не приведет нас к оптимальному решению. Следовательно, для вершины x можно производить отсечение по рекорду. Очевидно, что в качестве нижней оценки в задаче минимизации можно всегда взять $HO(\bar{x}) = 0$. В этом случае отсечение частичного решения по рекорду происходит только в случае, если значение целевой функции для частичного решения больше значения рекорда.

Аналогично, если в задаче максимизации для вершины дерева x выполняется неравенство

$$F > F(x) + BO(\bar{x}), \text{ то}$$
$$F > F(x) + opt(\bar{x}),$$

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

Метод организации перебора вариантов решения с отсечениями по рекорду часто называют методом «ветвей и границ».

Рассмотрим задачу минимизации.

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

$$HO(\bar{x}) = opt(\bar{x}),$$

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

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

$$HO(\bar{x}) = opt(\bar{x}) = F^*,$$

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

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

$$F^* = F(x) + opt(\bar{x})$$

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

Аналогично, для задачи максимизации справедлива следующая теорема.

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

$$BO(\bar{x}) = opt(\bar{x}),$$

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

Доказательство теоремы аналогично доказательству теоремы 1.3.

Следует заметить, что в случае совпадения оценки с оптимальным решением при генерации всех оптимальных решений трудоемкость будет зависеть от количества решений, трудоемкости вычисления оценки и высоты дерева перебора.

Пример 1.2. Проиллюстрируем отсев по рекорду на следующем примере. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлены фигуры. Необходимо найти все кратчайшие маршруты коня между двумя заданными позициями: *s* – стартовой и *t* – финальной.

Решение. Дерево перебора вариантов решения для данной задачи будет иметь следующий вид. Вершинам дерева соответствуют позиции шахматной доски. Корень дерева – стартовая позиция *s*.

Частичное решение для вершины дерева *x* – это путь из корня дерева в вершину *x* (т. е. последовательность ходов коня из позиции *s* шахматной доски в позицию *x*). Значение целевой функции на частичном решении ($F(x)$) определяется как длина пути из корня дерева в вершину *x* ($F(s) = 0$).

Сыновья вершины *x* – всевозможные пути, которые получаются путем добавления к частичному решению, соответствующему вершине *x*, одного хода конем.

$$F(v) = F(отец(v)) + 1.$$

Пусть *x* – некоторая вершина дерева. Тогда подзадача для данной вершины – это путь коня из позиции *x* шахматной доски в точку финиша *f*. Нижнюю оценку для подзадачи $HO(\bar{x})$ определим как длину кратчайшего пути коня на шахматной доске из позиции *x* в позицию *f*. Нетрудно заметить, что нижняя оценка подзадачи совпадает со значением ее оптимального решения:

$$HO(\bar{x}) = opt(\bar{x}).$$

Это обеспечивает просмотр только таких вершин дерева, которые соответствуют кратчайшему пути коня. В качестве рекорда возьмем длину кратчайшего пути коня из стартовой позиции шахматной доски в финишную позицию.

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

$$F^* = F(x) + HO(\bar{x}) = F(x) + opt(\bar{x})$$

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

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

Для эффективного вычисления рекорда F^* и нижних оценок подзадач $HO(\bar{x})$ сопоставим шахматной доске граф и выполним поиск в ширину (конем) из точки финиша *f* в точку старта *s*. При этом все достижимые из точки финиша *f* позиции шахматной доски получат метки, которые являются нижними оценками подзадач, соответствующих данным позициям. Метка, которая будет присвоена позиции, *s* есть рекорд F^*.

Трудоемкость описанного алгоритма – это трудоемкость вычисления рекорда и нижних оценок подзадач и трудоемкость построения кратчайших путей. Так как трудоёмкость поиска в ширину есть $O(N)$, где N – количество пустых клеток на доске, а трудоёмкость алгоритма восстановления путей есть $O(k \cdot l)$, где k – количество кратчайших путей и l – длина кратчайшего пути, то трудоёмкость всего алгоритма есть $O(N + k \cdot l)$.

Проиллюстрируем данный алгоритм на конкретном примере. Пусть

$$N = M = 4, \quad s = (1,1), \quad f = (4,4).$$

Построим все кратчайшие пути коня из позиции *s* шахматной доски в позицию *f*.

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

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

Рекорд $F^* = 2$. Дерево решений для данного примера представлено на рис. 1.26.

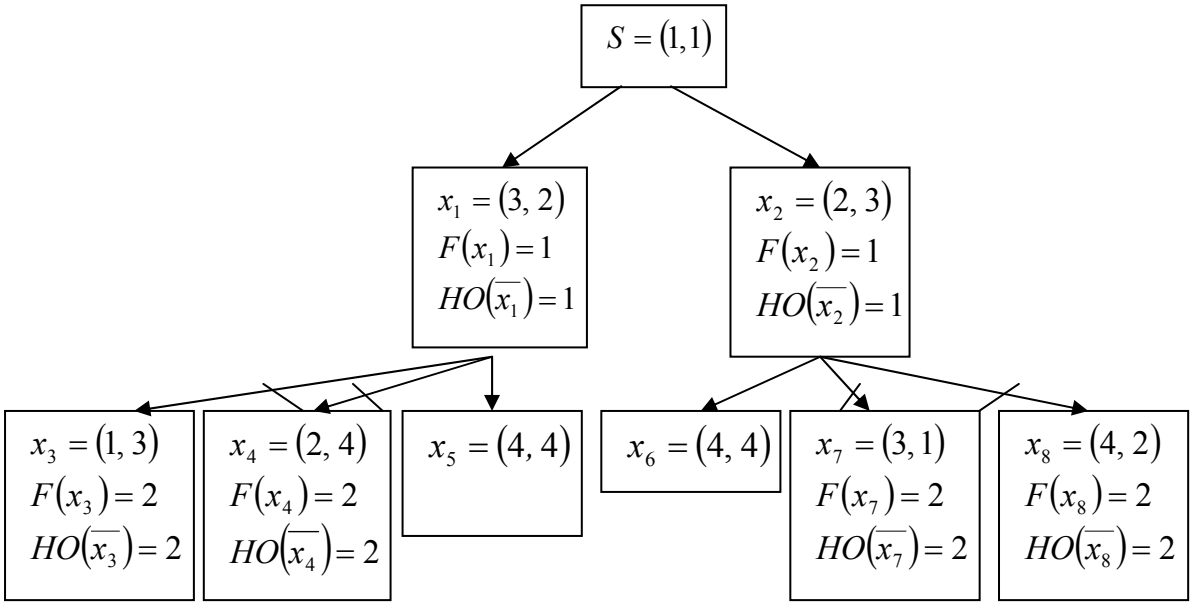


Рис. 1.26

Как видно из рис. 1.26, вершины x_1, x_2 являлись перспективными, а для вершин x_3, x_4, x_7, x_8 было выполнено отсечение по рекорду:

$$F^* = 2 = F(x_1) + HO(\overline{x_1}) = 1 + 1 = 2$$
$$F^* = 2 = F(x_2) + HO(\overline{x_2}) = 1 + 1 = 2$$
$$F^* = 2 < F(x_3) + HO(\overline{x_3}) = 2 + 2 = 4$$
$$F^* = 2 < F(x_4) + HO(\overline{x_4}) = 2 + 2 = 4$$
$$F^* = 2 < F(x_7) + HO(\overline{x_7}) = 2 + 2 = 4$$
$$F^* = 2 < F(x_8) + HO(\overline{x_8}) = 2 + 2 = 4.$$

Вершины x_5, x_6 порождают оптимальные решения задачи:

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

1.4. Функции ветвления

Как при поиске по глубине, так и при поиске по ширине выбор очередной вершины для ветвления не был полностью определен. При поиске по глубине, когда после ветвления задача P_i разбивается на подзадачи $P_{i_1}, P_{i_2}, \ldots, P_{i_r}$ очередное ветвление, как уже говорилось, производится в одной из этих только что порожденных подзадач. Но мы не указали в какой именно, и любая из них может рассматриваться как "последняя по-

рожденная". При поиске по ширине, как уже было сказано, все подзадачи данного уровня должны исследоваться до задач следующего уровня, но не был указан порядок их исследования.

Функция ветвления – это функция, которая позволяет "вычислить", какая из допустимых вершин должна использоваться при следующем ветвлении. Для вершины, соответствующей подзадаче P_j, эта функция является некоторой мерой вероятности того, что оптимальное решение всей задачи P_0 является решением для P_j. Совершенно очевидно, что вершина, соответствующая подзадаче с большими шансами на оптимальное решение, должна пользоваться правом преимущественного выбора при очередном ветвлении. Можно указать несколько эвристических мер этой вероятности, причем одна из полезных мер связана просто с вычислением для вершин нижних или верхних границ. Для такой меры вершина с более низкой нижней границей (для случая минимизации) считается имеющей большую вероятность.

После введения понятия функции ветвления сразу же возникает мысль о другом типе поиска с деревом решений (в дополнение к описанным ранее поискам по глубине и ширине). Можно использовать функцию ветвления и так, чтобы она полностью определяла выбор следующей для ветвления вершины. Например, если значениями функции ветвления являются границы вершин (нижние и верхние), как упоминалось выше, то всегда можно производить ветвление в той висячей вершине, нижняя граница которой наименьшая. Этот тип поиска является, вообще говоря, гибридом поисков по глубине и ширине, хотя в литературе он часто называется поиском по ширине.

Пример 1.3. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить минимальное число белых ладей, чтобы пробивались все свободные позиции.

Решение. Вершинам дерева перебора вариантов будут соответствовать некоторые расстановки ладей. В корне дерева перебора находится заданная расстановка черных фигур. Ветвление осуществляется следующим образом: пусть x – некоторая вершина дерева, тогда множеством её сыновей будет множество расстановок с добавленной новой ладьи к уже имеющейся расстановке, описываемой данной вершиной x. Таким образом, на глубине l дерева решений на шахматной доске будет находиться l ладей. Обход осуществляется в лексикографическом порядке. Такой способ обхода гарантирует отсутствие повторений и возможность получения любой целесообразной расстановки. Рекорд вычисляется после пер-

вого завершённого обхода дерева (построен первый «лист») и впоследствии может быть улучшен.

Возможные отсечения

1) По рекорду. Если на некотором шаге видно, что для частичного решения, соответствующего вершине x, количество ладей в конечной расстановке будет больше (либо даже не меньше), чем полученный рекорд, то выполняем отсечение вершины x по рекорду (здесь нижняя оценка подзадачи равна 0). Если для вершины x количество уже поставленных ладей плюс минимум из свободных вертикалей и горизонталей больше рекорда, то ветвить далее смысла нет, и мы выполняем отсечение вершины x.

2) По доминированию. Если клетка, в которую возможна постановка ладьи на некотором шаге, уже пробивается, то ставить ладью в эту клетку не имеет смысла, т.к., очевидно, что, поставив ладью в первую не пробивающуюся клетку, следующую в лексикографическом порядке за данной, мы получим большее множество пробивающихся клеток.

3) Построение нижней оценки. Построим двудольный граф. Вершины первой доли – вертикали шахматной доски. Вершины второй доли – горизонтали. Ребро между двумя вершинами двудольного графа проводится в том случае, если соответствующие вертикаль и горизонталь пересекаются.

Под *наибольшим паросочетанием* будем понимать такое паросочетание, которое не является подмножеством никакого другого наибольшего паросочетания (наибольшее по включению). Для двудольного графа, приведенного на рис. 1.27, мы имеем два наибольших паросочетания: $\{(1,1')\}$ *и* $\{(1,2'),(2,1')\}$.

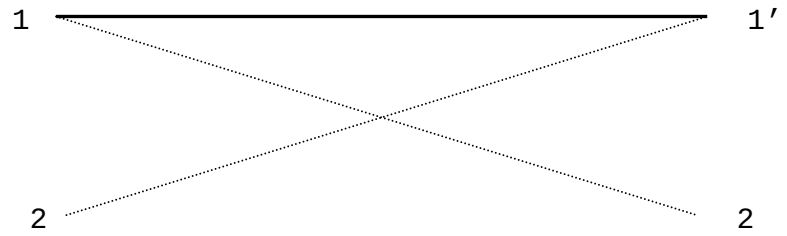


Рис. 1.27

Минимальное наибольшее паросочетание это паросочетание минимальной мощности среди наибольших паросочетаний. Для двудольного графа, приведенного на рис. 1.27 минимальное наибольшее паросочетание: $\{(1,1')\}$.

Точной оценкой задачи является мощность минимального наибольшего паросочетания в построенном двудольном графе. Более того, по тако-

му паросочетанию можно восстановить решение: расстановкой являются белые ладьи, соответсвующие рёбрам, входящим в минимальное наибольшее паросочетание (каждое ребро двудольного графа задает позицию шахматной доски). Изолированные точки отдельно не рассматриваются, т.к. они в любом случае обойдутся при обходе в лексикографическом порядке. В этом случае граф будет иметь столько компонент связности, сколько изолированных компонент будет иметь начальная расстановка. Оценка сначала применяется к каждой доле, после чего неравенства складываются почленно и получается оценка для всего графа.

Точная оценка позволяет строить только те узлы дерева, которые приведут к построению оптимального решения.

1.5. Задачи для самостоятельного решения

1. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлены черные фигуры, M – количество столбцов, а N – количество строк. Расставить минимальное число белых коней, чтобы пробивались все свободные позиции.

Рекомендация: минимальное вершинное покрытие графа, вершинами которого будут все свободные клетки, и будет существовать ребро (x,y), если при постановке коня в клетку x будет пробиваться клетка y.

2. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить минимальное число белых ладей, чтобы пробивались все свободные позиции.

Рекомендация: двудольный граф с вершинами первой доли – вертикали шахматной доски, второй доли – горизонтали, ребро между двумя вершинами двудольного графа проводится в том случае, если соответствующие вертикаль и горизонталь пересекаются.

3. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить минимальное число белых ферзей, чтобы пробивались все свободные позиции.

Рекомендация: минимальное вершинное покрытие графа, вершинами которого будут все свободные клетки.

4. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить минимальное число белых слонов, чтобы пробивались все свободные позиции.

Рекомендация: минимальное вершинное покрытие графа, вершинами которого будут все свободные клетки.

5. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить минималь-

ное количество коней, чтобы они пробивали все клетки доски не менее K раз.

Рекомендация: двудольный граф вершинам первой доли соответствуют диагонали «слева-направо», вершинам второй доли соответствуют диагонали «справа-налево»; в двудольном графе будем проводить ребро (a, b) в том и только в том случае, если a и b пересекаются.

6. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить минимальное количество ферзей, чтобы они пробивали все клетки доски не менее K раз.

Рекомендация: двудольный граф вершинам первой доли соответствуют диагонали «слева-направо», вершинам второй доли соответствуют диагонали «справа-налево»; в двудольном графе будем проводить ребро (a, b) в том и только в том случае, если a и b пересекаются.

7. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить максимальное число белых коней, чтобы они не били друг друга.

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

8. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить максимальное число белых ладей, чтобы они не били друг друга.

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

9. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить максимальное число белых ферзей, чтобы они не били друг друга.

Рекомендация: поиск максимального паросочетания в графе, ребрами которого являются вертикали, горизонтали и диагонали шахматной доски.

10. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлено K черных фигур. Необходимо расставить максимальное число белых слонов так, чтобы они не били друг друга.

Рекомендация: проведем на доске все диагонали справа налево и сверху вниз (множество A). Каждой диагонали сопоставим вершину гра-

фа. Аналогично проведем все диагонали слева направо и сверху вниз (множество B) и каждой диагонали сопоставим вершину графа. B полученном графе найдем максимальное парасочетание, которое и будет максимальным количеством расставленных слонов, которые не бьют друг друга.

11. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлены фигуры. Необходимо найти все кратчайшие маршруты коня между двумя заданными позициями: s – стартовой и t – финальной.

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

12. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлены фигуры. Необходимо найти все кратчайшие маршруты ладьи между двумя заданными позициями: s – стартовой и t – финальной.

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

13. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлены фигуры. Необходимо найти все кратчайшие маршруты ферзя между двумя заданными позициями: s – стартовой и t – финальной.

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

14. Найти все возможные различные разрезы доски размером $N \times N$ ($N=2K$– чётное) на две одинаковые по форме связные фигуры.

15. Составить из костяшек одного набора домино все магические квадраты размера 4×4. Костяшки можно класть только горизонтально, костяшка занимает 2 клетки по горизонтали и 1 по вертикали. Каждая костяшка из набора в каждом отдельно взятом квадрате используется не более одного раза. Магическим квадратом называется квадратная числовая матрица, у которой суммы элементов во всех строках, всех столбцах и на двух главных диагоналях совпадают.

Рекомендация: дерево решений представляет собой дерево, в котором у каждой вершины может быть до 7 потомков и каждое ребро имеет мет-

ку из множества {0, 1, 2, 3, 4, 5, 6}. Вершина дерева глубины k задает частично построенный квадрат.

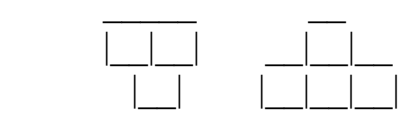
16. Составить из костяшек набора домино все возможные замкнутые цепочки прямоугольной формы.

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

17. Задаются 2 матрицы. Необходимо покрыть их с помощью набора фишек домино.

Рекомендация: дерево решений бинарное.

18. Имеется клеточное поле размера $M \times N$, в некоторых позициях которого расставлены черные фигуры. Необходимо выложить его фигурами вида



19. Раскрасить вершины графа в минимальное число цветов, смежные вершины должны иметь разные цвета.

Рекомендация: в дереве прербора: вершина – номер вершины в графе; ребро – номер цвета, в который выкрашиваем вершину. Каждая вершина имеет (k + 1) сыновей, где k – количество цветов уже использованных для раскраски графа.

20. Имеется клеточное поле размера $N \times M$, в некоторых позициях которого расставлены черные фигуры. Необходимо расставить на клеточном поле всеми возможными способами фишки таким образом, чтобы в каждой линии (горизонтальной, вертикальной, диагональной) располагалось четное число фишек.

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

21. Имеется n деталей и m станков. Каждая деталь характеризуется временем обработки (для каждой детали время обработки на всех станках одинаково). Станок в каждый момент времени обрабатывает только одну деталь. Детали на каждом станке обрабатываются последовательно.

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

Рекомендация: вершиной дерева перебора будет n – вектор. Тогда значение i-ой координаты этого вектора будет совпадать с номером станка, на который мы назначили i-ую деталь. В корень дерева мы положим 0-вектор.

22. Разрезать прямоугольник размера $X \times Y$, на детали прямоугольной формы размера $X_1 \times Y_1$, и $X_2 \times Y_2$, чтобы отходы были минимальны. Возможны только вертикальные и горизонтальные разрезы. Т. е. после первого разреза получается два прямоугольника, которые в последствии также можно разрезать.

Рекомендация: разрезание прямоугольника осуществим следующим образом: двигаясь из левого верхнего угла исходного прямоугольника, будем осуществлять горизонтальные и вертикальные разрезы шириной, равной одной из величин x_1, x_2, y_1, y_2. При этом полученный прямоугольник вышеуказанной ширины, именуемый в дальнейшем «полоса», разрежем последовательно на полосы ширины, соответственно y_1, y_2, x_1, x_2, пока это будет возможно. Далее аналогичным образом поступаем со вторым прямоугольником, полученным после первоначального разреза, и так далее, пока это будет возможно.

23. Петя и Вася очень любят решать головоломки. Однажды, воскресным утром они собрались вместе и долго думали какую же головоломку им порешать. Они уже разгадали не одну сотню японский кросвордов, судоку и других подобных задач. И тут Петя придумал новую задачу. Он достал из полки старую бумагу для вырезаний, которая осталась у него еще с детского садика и они с Васей захотели уложить их в прямоугольник минимальной площади. Они долго ломали голову как это сделать и к вечеру поняли, что лучше бы им упростить задачу. Для этого они установили в ней ограничение, что прямоугольники могут располагаться только параллельно осям координат (включая прямоугольник минимальной площади, в который они хотят вставить все эти прямоугольники). Дело было вечером и поэтому Вася и Петя разошлись по домам, но напоследок поспорили, кто же быстрее разложит 7 прямоугольников в прямоугольник минимальной площади. Они договорились о размерах этих 7 прямоугольниках. Помогите Пете решить эту непосильную задачу. Так же Петя понял, что возможно, что Вася захочет реванш, поэтому он хочет получить программу, которая находила бы расстановку для любых размеров, а не только тех, о которых он договорился с Васей в вос-

кресный вечер. Для заданных размеров прямоугольников необходимо определить минимальную площадь прямоугольника, в который влезут эти прямоугольники.

Рекомендация: каждой вершине дерева соответствует некоторое расположение прямоугольников и точек привязки для размещения последующих прямоугольников. Корнем дерева будет служить пустая раскладка. Для построения рекорда упорядочим стороны прямоугольников и расположим их в линию. Получим формулу $R = \left(\sum_{i=1}^{7} a_i \right) * b_j$, где a_i - одна из сторон исходных прямоугольников, а b_j - максимум из других сторон прямоугольников.

24. Построить все минимальные остовные деревья в графе.

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

Тема 2. Приближенные алгоритмы

2.1. Основные понятия

Задачи, принадлежащие классу NP практически неразрешимы. Однако такие задачи имеют много практических приложений, поэтому их решение достаточно важно. Поскольку точные полиномиальные алгоритмы решения этих задач неизвестны, то следует рассмотреть другие альтернативные возможности, дающие лишь достаточно хорошие (разумные) ответы. Из NP-трудности задачи следует необходимость построения для ее решения эвристических или приближенных алгоритмов, применения схем направленного перебора вариантов (метод ветвей и границ и динамическое программирование).

Предположим, что задана задача максимизации:

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

Пусть $x^{onm.}$ – дает максимум фунционалу, т. е.

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

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

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

Тогда возможно

$$F(x^{onm.}) \cong F(x^A).$$

Если алгоритм А – точный, то получим строгое равенство $F(x^{onm.}) = F(x^A)$.

Определение 2.1. Абсолютная погрешность алгоритма А на входном наборе I определяется как величина:

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

Определение 2.2. Относительная погрешность алгоритма А на входном наборе I определяется как величина:

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

Иногда для оценки качества алгоритма А рассматривают функцию:

$$\Delta_A = \frac{\inf\left|F(x^A)\right|}{\left|F(x^{onm.})\right|},$$

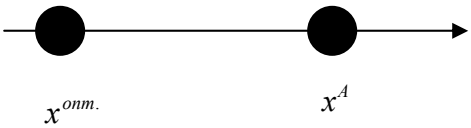
где inf берется по всем возможным наборам входных данных.

Определение 2.3. Под гарантированной оценкой точности решения, получаемого при помощи алгоритма *A*, понимаем:

- для задачи максимизации – нижнюю оценку (inf) величины Δ_A,

$$z \le \Delta_A = \frac{\left|F(x^A)\right|}{\left|F(x^{onm.})\right|} \le 1$$

- для задачи на минимум – верхнюю оценку (sup) величины Δ_A.



$$1 \le \Delta_A = \frac{\left|F(x^A)\right|}{\left|F(x^{onm.})\right|} \le z.$$

Определение 2.4. Алгоритм называется приближенным, если для него известны:

- либо абсолютная (относительная) погрешность, не зависящая от входных данных.
- либо гарантированная оценка точности;
- либо скорость сходимости.

Для приближенных алгоритмов существует следующая классификация:

- субоптимальный приближенный алгоритм (такой приближенный алгоритм, для которого величина Δ_A отлична от 0 (для задач максимизации) или от ∞ (для задач минимизации));
- ε-приближенный (PTAS) алгоритм (такой алгоритм, который для любого $\varepsilon > 0$ строит решение с относительной погрешностью ε, причем трудоемкость этого алгоритма есть функция от длины входа l и величины $1/\varepsilon$);
- быстрый ε- приближенный (FPTAS) алгоритм (такой алгоритм, который для любого $\varepsilon > 0$ строит решение с относительной погрешностью ε, причем трудоемкость этого алгоритма есть полином от длины входа l и величины $1/\varepsilon$).

Пример 2.1. Пусть $p_1, p_2, \ldots, p_n -$ длительности программ. Имеются два носителя одинаковой длины L. Необходимо вместить как можно больше программ на носители. Следует отметить, что в силу того, что значение $F(X^{onm})$ не известно, как правило оценивается величина $\frac{F(X^A)}{BO(I)}$ для задачи максимизации и $\frac{F(X^A)}{HO(I)}$ для задачи минимизации, где $BO(I)(HO(I))$ - верхняя (нижняя) оценка оптимального решения для индивидуальной задачи (на определенном наборе входных данных). Действительно, в этом случае $\frac{F(X^A)}{F(X^{onm})} \ge \frac{F(X^A)}{BO(I)}$ в силу $BO(I) \ge F(X^{onm})$ для задачи на максимум, $\frac{F(X^A)}{F(X^{onm})} \le \frac{F(X^A)}{HO(I)}$ в силу $HO(I) \le F(X^{onm})$ для задачи на минимум.

Таким образом, для вычисления гарантированной оценки, как правило требуется определить верхние (нижние) оценки оптимального решения, и только потом попытаться найти соотношение между этими оценками и решением, построенным предлагаемым алгоритмом. Понятно, что при таком подходе наиболее важно найти такие оценки оптимальных решений, которые наиболее близки к оптимальному решению.

Решение. Предположим, что существует такое *k*, что

$$\sum_{i=1}^{k} p_k \le 2L$$

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

Тогда $F(x^{onm.}) \le k$. Предположим, что существует некоторый алгоритма А, для которого $F(x^A) \ge k-1.$ Тогда А – приближенный алгоритм с абсолютной погрешностью $\left|F(x^{onm.}) - F(x^A)\right| \le \left|k-(k-1)\right| \le 1.$

Рассмотрим сейчас несколько приближенных алгоритмов для приведенных ранее задач. Все эти приближенные алгоритмы полиномиальны по времени.

2.2. Приближенный жадный алгоритм для задачи о коммивояжере

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

Алгоритм

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

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

2) добавляемое ребро после добавления к ранее выбранным ребрам не будет являться третьим ребром, выходящим из некоторой вершины.

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

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

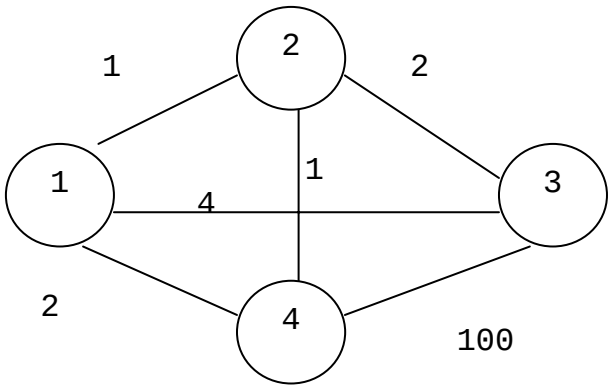


Рис. 2.1

Жадный приближенный алгоритм сначала выберет ребро (1,2), затем ребро (2, 4) и присоединит их к выбранному множеству ребер.

Следующим ребром минимального веса будет ребро (1,4), но оно будет отвергнуто, так как для него не выполняется пункт 1) из условий присоединения ребер и оно не является завершающим ребром пути (цикл $1 \to 2 \to 3 \to 1$ не проходит через все вершины графа).

Последующее ребро (2,3) также будет отвергнуто, так как для него не выполняется 2) пункт из условий присоединения, т. е. из вершины 2 в этом случае выходило бы три ребра.

Последующее ребро (1,3) успешно присоединяется.

И, наконец, присоединяется последнее ребро пути (4,3). Оно является завершающим ребром пути, и полученный цикл проходит через все вершины графа.

Данный алгоритм сгенерировал путь $1 \to 2 \to 4 \to 3 \to 1$ стоимости 106 (рис. 2.2).

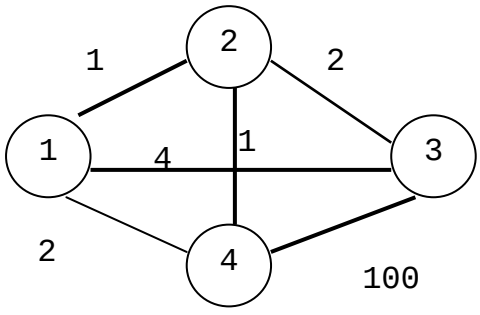


Рис. 2.2

Это решение не является оптимальным: есть, по крайней мере, один более короткий путь $1 \to 2 \to 3 \to 4 \to 1$ стоимости 104 (рис. 2.2).

2.3. Приближенный жадный алгоритм для задачи о рюкзаке

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

Алгоритм

Отсортируем список объектов по отношению их стоимости к объему:

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

и будем последовательно заполнять рюкзак объектами в соответствии с упорядочением. Если очередной объект не входит, то отбрасываем его и переходим к рассмотрению следующего объекта списка. Описанный процесс продолжаем до тех пор, пока рюкзак не заполниться или будет исчерпан упорядоченный список объектов.

Пример 2.3. Упаковать рюкзак объемом W=80 объектами, сведения о которых приведены в таблице 2.1.

Таблица 2.1

(a_i, c_i)	(25, 20)	(20,80)	(20,50)	(15, 45)	(30,105)	(35,35)	(20,10)	(10,45)
c_i / a_i	2	4	2.5	3	3.5	1	0.5	4.5
$\frac{c_i}{a_i} \geq \frac{c_{i+1}}{a_{i+1}}$	(10,45) 1	(20,80) 2	(30,105) 3	(15, 45) 4	(20,50) 5	(25, 20) 6	(35,35) 7	(20,10) 8

Так как емкость рюкзака 80, то мы сможем упаковать в соответствии с нашим алгоритмом первые 4 объекта отсортированного списка. Общий объем упакованных объектов – 75, стоимость – 275. Это решение не является оптимальным, так как существует лучшее решение, которое дают первые три объекта отсортированного списка и еще пятый объект этого списка: объем – 80, стоимость – 280.

2.4. Приближенный жадный алгоритм для задачи о суммах элементов подмножеств.

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

Алгоритм 1.

Для построения приближенного алгоритма будем использовать жадный алгоритм. Для этого, по аналогии с задачей о рюкзаке, отсортируем список предметов в порядке убывания их размеров, и будем последовательно дополнять предметы к пустому множеству в соответствии с упорядочением. Если очередной предмет при добавлении к множеству приводит к тому, что суммарный объем предметов, входящих в сгенерированное множество превышает L, то отбрасываем его и переходим к рассмотрению следующего предмета упорядоченного списка. Описанный процесс продолжаем до тех пор, пока суммарный размер предметов множества не станет равным L или будет исчерпан упорядоченный список объектов.

Пример 2.4. Объемы предметов a_i приведены в таблице 2.2. Предельная сумма объемов предметов множества равна L=55. Необходимо найти такой набор предметов, сумма размеров которых была бы наиболее близка к L.

Таблица 2.2

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

Так как емкость множества 55, то мы сможем добавить в него в соответствии с нашим алгоритмом 3 предмета с объемами: 27, 22 и 1. Общий объем сгенерированного множества – 50. Это решение не является оптимальным, так как существует лучшее решение, которое дают предметы с объемами 22, 14, 11, 7 и 1. В этом случае объем – 55, т. е. данное решение является оптимальным.

Алгоритм 2.

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

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

2. На втором проходе алгоритм начинает свою работу со всевозможных одноэлементных множеств и добавляет к ним, аналогично первому проходу, элементы из упорядоченного по объемам списка предметов.

3. На третьем проходе алгоритм аналогичным образом исследует все трехэлементные множества и т. д.

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

Пример 2.5. Для приведенного в таблице 2.2 примера (упорядоченность: 27, 22, 14, 11,7,1) уже на втором проходе будет построено оптимальное решение задачи (Таблица 2.3).

Таблица 2.3

Номер прохода	Размер начального подмножества	Добавляемые элементы	Сумма элементов сгенерированного множества
0	0	27,22,1	50
1	27	22 1	50
	22	27 1	50
	14	27 11 1	53
	11	27 14 1	53
	7	27 14 1	49
	1	27 22	50
2	27 22	1	50
	27 14	11 1	53
	27 11	14 1	53
	27 7	14 1	49
	27 1	22	50
	22 14	11 7 1	55 ! опт.
	22 11	14 7 1	55
	22 7	14 11 1	55
	22 1	27	50
	14 11	27 1	53
	14 7	27 1	49
	14 1	27 11	53
	11 7	27 1	46
	11 1	27 14	53
	7 1	27 14	49

2.5. Приближенный жадный алгоритм для задачи о раскраске графа

Требуется определить минимальное количество цветов *c*, в которые можно раскрасить вершины графа так, чтобы концы каждого ребра графа были окрашены в разные цвета.

Для данной задачи доказано, что число красок, даваемое лучшим приближенным полиномиальным алгоритмом, более чем вдвое превышает оптимальное. Рассмотрим простой алгоритм последовательной раскраски произвольного графа с N вершинами.

Алгоритм

Последовательно рассматриваем все вершины исходного графа, чтобы определить каким цветом их покрасить. Предположим, что мы хотим определить каким цветом должна быть покрашена вершина *i*.

1. Полагаем первоначально номер цвета для этой вершины $c = 1$.

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

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

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

Пример 2.6. Решить задачу раскраски графа, приведенного на рис. 2.3.

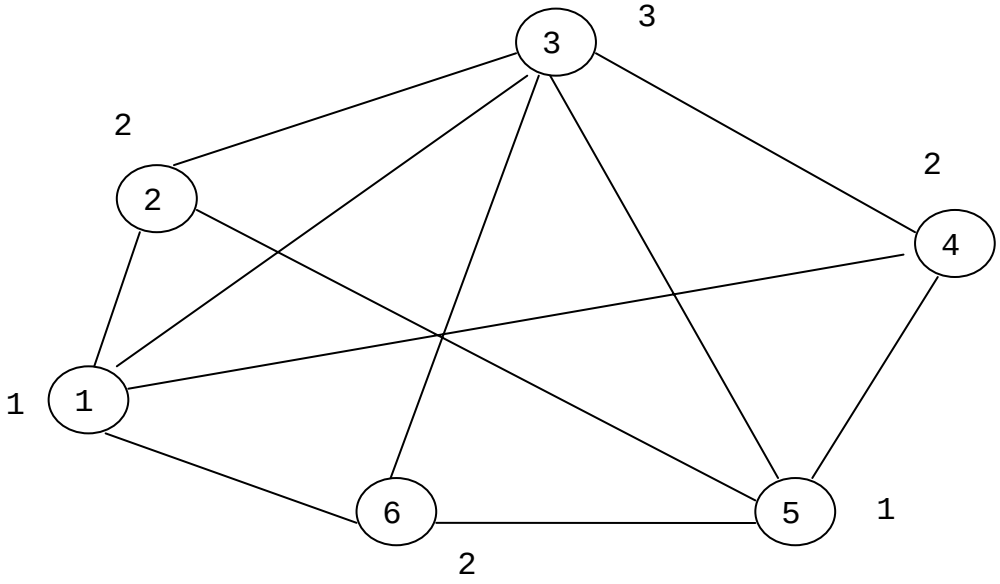


Рис. 2.3

Раскраска вершин проводилась в соответствии с приведенным алгоритмом. Вершины графа просматривались в порядке возрастания их номеров. Как видно из рис. 2.3 минимальное число потребовавшихся красок равно 3.

2.6. Приближенные алгоритмы с гарантированной оценкой точности.

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

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

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

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

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

Опишем приближенный алгоритм А решения задачи $P//\sum w_i C_i$, который состоит в следующем. Перенумеруем требования в порядке $SWPT$ так, что $p_1/w_1 \le \ldots \le p_n/w_n$.

Организуем m подмножеств $N_1, N_2, \ldots, N_m$, полагая вначале $N_1 = N_2 = \ldots = N_m = \varnothing$. Будем последовательно, начиная с первого, распределять требования $1,2,\ldots,n$ по подмножествам. Пусть требования $1,2,\ldots,k, k < n$ распределены, где 0 - фиктивное требование, соответствующее начальной ситуации $p_0 = 0$. Требование $k+1$ назначается следующим образом. Обозначим $P_l = \sum_{j \in N_l} p_j$, $l = 1,\ldots,m$. Находим такое подмножество N_l что $P_l = \min_{1 \le h \le m} P_h$. Полагаем $N_l = N_l \cup \{k+1\}$ и, если $k+1 \ne n$ переходим к назначению требования $k+2$.

Временная сложность алгоритма А равна $O(n \log n)$, так как значения $P_1, P_2, \ldots, P_m$ можно хранить в виде кучи или поискового 2 - 3 дерева.

Обозначим через F_1^* и F_n^* минимальное значение функционала $\sum w_i C_i$ при $m = 1$ и $m = n$ соответственно. Очевидно, что тогда $F_1^* = \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i$ и $F_n^* = \sum_{j=1}^{n} w_j p_j$.

Найдем верхнюю и нижнюю оценки значения F^*. Очевидно, что $F^* \le F^A$. Поскольку на итерации j алгоритма А выполняется соотношение $P_l = \min_{1 \le h \le n} \{P_h\} \le \frac{1}{m} \sum_{i=1}^{j-1} p_i$, то имеем

$$F^A \le \sum_{j=1}^{n} w_j \left(\frac{1}{m} \sum_{i=1}^{j-1} p_i + p_j\right) = \frac{1}{m} \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i + \left(1 - \frac{1}{m}\right) \sum_{j=1}^{n} w_j p_j = \frac{1}{m} F_1^* + \frac{m-1}{m} F_n^*.$$

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

Лемма 2.1. *Пусть* $b_1, b_2, \ldots, b_n$ *- действительные числа. Тогда* $\left(\sum_{j=1}^{n} b_j\right)^2 \le n \sum_{j=1}^{n} b_j^2$.

Лемма 2.2. *Если* $\frac{p_j}{w_j} = C$, $j = 1,2,\ldots,n$, *то* $F_1^* = \frac{1}{2C}\left(\sum_{j=1}^{n} p_j\right)^2 + \frac{1}{2} F_n^*$.

Лемма 2.3. *Если* $\frac{p_j}{w_j} = C,\ j = 1,2,\ldots,n$, *то* $F^* - \frac{1}{2}F_n^* \geq \frac{1}{m}(F_1^* - \frac{1}{2}F_n^*)$.

Теорема 2.1. Для любого расписания справедливо $\sum w_j C_j \geq \frac{1}{m}F_1^* + \frac{m-1}{2m}F_n^*$.

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

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

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

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

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

Из леммы 2.3 следует, что $m\overline{Y} \geq \overline{X}$ и $m\overline{Y}^q \geq \overline{X}^q$, а и равенств $\overline{w}_j = \gamma w_j$, $j = \overline{1,q}$, $X = \frac{1}{\gamma}\overline{X}^q + \delta(\overline{X})$ и $Y = \frac{1}{\gamma}\overline{Y}^q + \delta(\overline{Y})$. Возможны два случая: 1) $\delta(\overline{X}) \leq m\delta(\overline{Y})$ и 2) $\delta(\overline{X}) > m\delta(\overline{Y})$. В первом случае получаем $X \leq mY$, т. е. $\frac{1}{m}(F_1^* - \frac{1}{2}F_n^*) \leq \sum w_j C_j - \frac{1}{2}F_n^*$. Во втором случае из неравенства $m\overline{Y}^q \geq \overline{X}^q$ следует, что $m\delta(\overline{X})\overline{Y}^q > m\overline{X}^q\delta(Y)$.

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

этого неравенства $\overline{X}^q \overline{Y}^q + \gamma\delta(\overline{X})\delta(\overline{Y})$, получаем $(\overline{X}^q + \delta(\overline{X}))(\overline{Y} + \gamma\delta(\overline{Y})) > (\overline{X}^q + \gamma\delta(\overline{X}))(\overline{Y}^q + \delta(\overline{Y}))$.

Однако

$$\overline{X}^q + \delta(\overline{X}) = \overline{X},\ \overline{Y}^q + \delta(\overline{Y}) = \overline{Y},$$

$$\overline{Y}^q + \gamma\delta(\overline{Y}) = \gamma Y,\ \overline{X}^q + \gamma\delta(\overline{X}) = \gamma X.$$

Поэтому последнее неравенство принимает вид $\gamma\overline{X}Y > \gamma X\overline{Y}$. Из $m\overline{Y} \geq \overline{X}$ и последнего неравенства следует $\gamma m\overline{Y}Y > \gamma X\overline{Y}$. Таким образом, $mY > X$.

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

Сейчас нетрудно получить оценку сверху величины Δ_A:

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

2.7. Задача об упаковке в контейнеры

Рассмотрим NP-трудную в сильном смысле задачу об упаковке в контейнеры, которая в терминах теории расписаний может быть сформулирована следующим образом. В условиях задачи $P/d_j = d/C_j \leq d_j$ предполагается, что количество приборов равно числу требований $m = n$, общий директивный срок не меньше максимальной длительности обслуживания требований $d \geq \max_j p_j$, и требуется отыскать минимальное число приборов и соответствующее расписание, при котором все требования будут обслужены к директивному сроку d.

Приведем описание четырех приближенных алгоритмов решения задачи об упаковке в контейнеры. В алгоритмах рассматривается некоторая перестановка π требований, требования последовательно выбираются из этой перестановки и назначаются на приборы в соответствии с некоторыми правилами. Резервом времени прибора l, обозначаемым R_l, называется разница между d и суммарной длительностью обслуживания требований, назначенных на прибор l.

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

времени для завершения этого требования в срок. В алгоритме B_2 это требование назначается на прибор с наименьшим достаточным резервом времени.

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

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

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

Докажем ряд вспомогательных утверждений для алгоритмов B_1 и B_2. Введем в рассмотрение функцию $\psi(x), x \in [0,1]$.

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

Далее будем рассматривать расписания, построенные алгоритмом B_1 либо B_2.

Лемма 2.4. *Пусть на некоторый прибор назначены требования* $1,\ldots,j$. *Тогда* $\sum_{i=1}^{j}\psi(p_i) \le 17/10$.

Не ограничивая общности, предположим, что $p_1 \ge \ldots \ge p_j$. Если $p_1 \le 1/2$, то из определения $\psi(x)$ следует $\psi(p_i) \le \frac{3}{2}p_i$, $i = 1,\ldots,j$. Тогда $\sum_{i=1}^{j}\psi(p_i) \le \frac{3}{2}\sum_{i=1}^{j}p_i \le 3/2$. Если $p_1 > 1/2$, то $\sum_{i=2}^{j}p_i < 1/2$. В этом случае достаточно показать, что $\sum_{i=2}^{j}\psi(p_i) \le 7/10$. Возможны следующие варианты:

1) $p_2 \in (1/3, 1/2]$, $p_3 \in (0, 1/6]$,

2) $p_2 \in (1/6, 1/3]$, $p_3 \in (0, 1/6]$,

3) $p_2, p_3 \in (1/6, 1/3]$, $p_4 \in (0, 1/6]$,

4) $p_2 \in (0, 1/6]$.

При вариантах 1) и 2) имеем соответственно $\sum_{i=2}^{j} \psi(p_j) < \frac{6}{5} p_2 + \frac{1}{10} + \frac{6}{5}(\frac{1}{2} - p_2) = \frac{7}{10}$, и

$\sum_{i=2}^{j} \psi(p_i) < \frac{9}{5} p_2 - \frac{1}{10} + \frac{6}{5}(\frac{1}{2} - p_2) = \frac{1}{2} + \frac{3}{5} p_2 \le 7/10$. При варианте 3) имеем $\sum_{i=2}^{j} \psi(p_i) < \frac{9}{5}(p_2 + p_3) - \frac{2}{10} + \frac{6}{5}(\frac{1}{2} - p_2 - p_3) = \frac{2}{5} + \frac{3}{5}(p_2 + p_3) < 7/10$, поскольку $p_2 + p_3 < \frac{1}{2}$. При варианте 4) имеем $\sum_{i=2}^{j} \psi(p_i) < \frac{6}{5} \sum_{i=2}^{j} p_i < 6/10$.

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

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

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

Пусть $Q_l < 1/2$. Тогда $R_h < 1/2$, $h = 1, \ldots, l-1$ и длительность обслуживания первого назначенного на прибор l требования больше, чем Q_l. Обозначим эту длительность через τ. Если $\tau > 1/2$, то лемма доказана. Если $\tau \le 1/2$, то $R_l \ge 1/2$ и длительность второго назначаемого на прибор l требования больше, чем Q_l. Обозначим эту длительность через τ'. Если $\tau + \tau' \le 1/2$, то лемма 2.5 доказана. Если $\tau + \tau' > 1/2$, то длительность третьего назначаемого на прибор l требования больше чем Q_l. Повторяя приведенные рассуждения конечное число раз, получаем справедливость утверждения леммы.

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

Лемма 2.6. *Пусть на прибор* l *назначены требования* $1,\ldots,j$ *и* $Q_l < 1/2$. *Если* $\sum_{i=1}^{j} p_i \geq 1 - Q_l$, *то* $\sum_{i=1}^{j} \psi(p_i) \geq 1$.

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

Пусть $p_1 \leq 1/2$. Поскольку $\sum_{i=1}^{j} p_i \geq 1 - Q_l > 1/2$, имеем $l \geq 2$. Из леммы 2.5 и $p_1 \leq 1/2$ следует, что $p_2 > Q_l$. Возможны варианты: 1) $Q_l \in (0, 1/6]$, 2) $Q_l \in (1/6, 1/3]$, 3) $Q_l \in (1/3, 1/2]$.

В условиях первого варианта $\sum_{i=1}^{j} \psi(p_i) \geq \frac{6}{5} \sum_{i=1}^{j} p_i \geq \frac{6}{5}(1 - Q_l) \geq \frac{6}{5}\frac{5}{6} = 1$.

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

$$\psi(p_1) + \psi(p_2) = \frac{6}{5} p_1 + \frac{9}{5} p_2 = \frac{6}{5}(p_1 + p_2) + \frac{3}{5} p_2 > \frac{6}{5}(1 - Q_l) + \frac{3}{5} Q_l = \frac{6}{5} - \frac{3}{5} Q_l \geq 1,$$

поскольку $Q_l \leq \frac{1}{3}$.

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

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

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

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

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

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

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

Пусть $l \ge 2$. Поскольку $p_1 \le 1/2$, то в силу леммы 2.6 имеем $p_1 \ge p_2 > Q_l$, $\sum_{i=1}^{j} p_i = 1 - Q_l - \gamma$, где $\gamma > 0$. Поскольку $\sum_{i=1}^{j} p_i + \gamma < 1$, можно выбрать числа $\delta_1 \le 1/2$ и $\delta_2 \le 1/2$, такие, что $\delta_1 \ge p_1$, $\delta_2 \ge p_2$ и $\delta_1 + \delta_2 = p_1 + p_2 + \gamma$.

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

Для любых x и y $0 \le x < y \le 1/2$ выполняется $\psi(y) \le \psi(x) + \frac{9}{5}(y - x)$. Поэтому $\psi(\delta_1) + \psi(\delta_2) \le \psi(p_1) + \psi(p_2) + \frac{9}{5}\gamma$. Отсюда получаем $\sum_{i=1}^{j}\psi(p_i) + \frac{9}{5}\gamma \ge 1$. Следовательно, $\frac{9}{5}\gamma \ge \alpha$ и $\gamma \ge \frac{5}{9}\alpha$. Лемма 2.7 доказана.

Теорема 2.2. Справедливы неравенства $m_1 < \frac{17}{10}m^* + 2$ *и* $m_2 < \frac{17}{10}m^* + 2$.

Доказательство. Обозначим $\Psi = \sum_{i=1}^{n}\psi(p_i)$. Из леммы 2.5 следует, что $\frac{17}{10}m^* \ge \Psi$.

Пусть требования $1,\ldots,n$. распределены по приборам. Обозначим через N_l множество требований, назначенных на прибор l. Пусть $\psi(N_l) = \sum_{i\in N_l}\psi(p_i)$. Из последовательности приборов, на каждый из которых назначено хотя бы одно требование, выделим подпоследователь-

ность $l'_1, l'_2, \ldots, l'_k$ всех приборов, таких, что $\psi(N'_l) = 1 - \alpha_h$, $\alpha_h > 0$, $h = 1, \ldots, k$.

Для любого $i \in N_{l'_h}$, $h = 1, \ldots, k$, выполняется $p_i \le 1/2$, поскольку в противном случае $\psi(N_{l'_h}) \ge 1$. Отсюда следует, что $Q_{l'_h} < 1/2$, $h = 1, \ldots, k$. Покажем, что $Q_{l'_{h+1}} \ge Q_{l'_h} + \frac{5}{9}\alpha_h$, $h = 1, \ldots, k$. Очевидно, что $Q_{l'_1} = 0$ и в силу леммы 2.7 $\sum_{i \in N_{l'_1}} p_i \le 1 - \frac{5}{9}\alpha_1$. Тогда $Q_{l'_2} \ge \frac{5}{9}\alpha_1$. В силу леммы 2.7 имеем $\sum_{i \in N_{l'_2}} p_i \le 1 - \frac{5}{9}(\alpha_1 + \alpha_2)$. Поэтому $Q_{l'_3} \ge \frac{5}{9}(\alpha_1 + \alpha_2) = Q_{l'_2} + \frac{5}{9}\alpha_2$ и т.д.

Таким образом, $\sum_{h=1}^{k-1} \alpha_h \le \frac{9}{5} \sum_{h=1}^{k-1} (Q_{l'_{h+1}} - Q_{l'_h}) = \frac{9}{5}(Q_{l'_k} - Q_{l'_1}) < \frac{9}{10}$, поскольку $Q_{l'_h} < 1/2$, $h = 1, \ldots, k$. Отсюда, а также из $\alpha_k < 1$ получаем $\sum_{h=1}^{k} \alpha_h < 2$. Как нетрудно убедиться, $\psi(N_l) \ge 1$, если $\sum_{j \in N_l} p_j = 1$. Пусть Θ - множество всех таких приборов l. Что $\sum_{j \in N_l} p_j = 1$. Тогда $m_1 \le \sum_{l \in \Theta} \psi(N_l) + k \le \sum_{l \in \Theta} \psi(N_l) + \sum_{h=1}^{k} \psi(N_{l'_h}) + \sum_{h=1}^{k} \alpha_h < \Psi + 2 \le \frac{17}{10} m^* + 2$. Аналогично, $m_2 < \frac{17}{10} m^* + 2$. Теорема 2.2 доказана.

Из Теоремы 2.2 следует, что $\Delta_H \le \frac{17}{10} + \frac{2}{m^*}$ для алгоритма $H \in \{B_1, B_2\}$. Поскольку $m^* \ge \lceil P \rceil$, где $P = \sum_{j=1}^{n} p_j$, то $\Delta_H \le \frac{17}{10} + \frac{2}{\lceil P \rceil}$. Напомним, что здесь предполагается $d = 1$ и $p_j \le 1$, $j = 1, \ldots, n$.

Если d и $p_j \le 1$, $j = 1, \ldots, n$ произвольные числа, то $\Delta_H \le \frac{17}{10} + 2 / \lceil P/d \rceil$.

Перейдем к рассмотрению алгоритмов B_3 и B_4.

Теорема 2.3. Справедливы неравенства $m_3 \le \frac{11}{9} m^* + 4$ *и* $m_4 \le \frac{11}{9} m^* + 4$.

Доказательство. Схема доказательства теоремы 2.3 аналогична схеме доказательства теоремы 2.2 . Оно сводится к построению такой весовой

функции, аналогичной $\psi(x)$, что суммарный вес требований, назначенных на один и тот же прибор, не превосходит некоторой константы $C \geq 1$ и число m_3 или m_4 может превосходить суммарный вес всех требований не более чем на некоторую константу C'. В случае алгоритмов B_1 и B_2 получаем $C = 17/10$, $C' = 2$, а в случае алгоритмов B_3 и B_4 получаем $C = 9/11$, $C' = 4$.

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

2.8. Задача распределения работ на конечное число одинаковых процессоров.

Рассмотрим задачу распределения *n* работ на *m* одинаковых процессоров таким образом, чтобы минимизировать загрузку максимально загруженного процессора. Пусть p_i соответствует времени выполнения i-й работы на процессоре.

Попробуем оценить алгоритм, на каждом шаге которого работа загружается на минимально загруженный процессор.

Предположим, что k работ уже распределены, и происходит назначение $(k+1)$-й работы. Нетрудно понять, так как суммарная длительность уже распределенных работ равна $p_1 + \cdots + p_k$, то существует процессор, загрузка которого не превышает величину $Z_k = (p_1 + \cdots p_k)/m$. В этом случае все процессоры равнозагружены. Тогда после назначения $(k+1)$-й работы на этот процессор его загрузка не превысит величины $Z_k + p_{k+1} = Z_k + p_{k+1}/m + (m-1)p_{k+1}/m = Z_{k+1} + (m-1)p_{k+1}/m$. Следует отметить, что это соотношение справедливо на каждом шаге.

Теперь необходимо найти нижнюю границу оптимального решения. Первую границу можно взять как среднюю загрузку: $LB \geq (p_1 + \cdots p_n)/m$, причем $LB \geq Z_k$ для любого *k*.

Поэтому, учитывая предыдущие соотношения, на каждом шаге новая загрузка процессора, на который назначена очередная работа, не превышает величины $LB + (m-1)p_{k+1}/m$.

Теперь осталось воспользоваться тем свойством, что нижняя граница оптимального решения не меньше p_k для любого k, т. е. $LB \geq \max\{p_k\}$.

Следовательно, если положить $LB = \max\{p_k\}$, то алгоритм «в минимально загруженный» для любой последовательности $p_1, \cdots p_n$ строит

решение, в котором загрузка максимального процессора не превосходит величины $LB + (m-1)LB/m = (2-1/m)LB$.

2.9. Задачи для самостоятельного решения

1. Имеется n деталей и m станков. Каждая деталь характеризуется 3 параметрами: временем доставки, временем обработки, временем доставки на склад. Станок обрабатывает любую деталь сразу, все станки одинаковы. Определить порядок обработки деталей на станках, когда все детали будут на складе за минимальное время.

Рекомендация: $F^A \le 2F^*$.

2. Имеется n деталей и m станков. Каждая деталь характеризуется 2 параметрами: временем доставки, временем обработки. Станок обрабатывает любую деталь сразу, все станки одинаковы. Определить порядок обработки деталей на станках, когда все детали будут обработаны за минимальное время.

Рекомендация: $F^A \le 2F^*$.

3. Имеется n деталей и $2m$ станков 2-х типов. Деталь обрабатывается в 2 стадии: сначала на станке первого типа, затем на станке второго типа. Любая деталь характеризуется двумя параметрами: временем обработки на станке первого типа и временем обработки на станке второго типа. Станок обрабатывает каждую деталь сразу, станков разных типов одинаковое количество. Определить порядок обработки деталей на станках, когда все детали будут обработаны за минимальное время.

Рекомендация: $F^A \le 3F^*$.

4. Имеется N работ и M работников. Каждая работа характеризуется временем выполнения. Требуется распределить работы между работниками, чтобы минимально загруженный работник выполнял максимальное количество работы.

Рекомендация: $F^A \ge (\frac{2}{3} + \frac{1}{3m})F^*$.

5. Имеется n работ. Каждая работа характеризуется длительностью ее исполнения работником (все работники одинаковы). Необходимо так распределить работы среди работников, чтобы каждый работник был загружен не менее времени X, при этом количество работников было занято как можно больше.

Рекомендация: $F^A \ge \frac{3}{4} F^*$.

6. Имеется n работ и m работников. Каждая работа характеризуется длительностью ее исполнения каждым работником. Необходимо так рас-

пределить работы среди работников, чтобы максимально загруженный работник был загружен как можно меньше.

Рекомендация: $F^A \leq \frac{5}{3} F^*$.

7. Имеется n программ, m одинаковых процессоров и 1 сервер. Каждая программа характеризуется временем скачивания данных с сервера и временем выполнения ее на процессоре. Необходимо так организовать выполнение программ на процессорах, при котором время завершения последней программы минимально.

Рекомендация: $F^A \leq 2\ F^*$.

8. Имеется n предметов и много контейнеров. Каждый предмет характеризуется массой и объемом. Грузоподъемность и объем контейнера известны. Необходимо упаковать предметы в минимальное число контейнеров.

Рекомендация: $FA \leq 2\ F^*$.

9. Имеется n предметов и много контейнеров. Каждый предмет характеризуется массой и объемом. Грузоподъемность и объем контейнера известны. Необходимо упаковать предметы в минимальное число контейнеров.

Рекомендация: $FA \leq 3\ F^*$.

10. Имеется n городов, каждый из которых является либо потребителем, либо поставщиком продукции. Число $X(к)$ характеризует его спрос (отрицательное) или предложение (положительное). Необходимо определить минимальную грузоподъемность машины и маршрут, которая может объехать все города по разу и удовлетворить потребности (сумма спроса = сумме предложения).

Рекомендация: $FA \leq 2\ F^*$.

11. Имеется n программ, m одинаковых процессоров и 1 сервер. Каждая программа характеризуется временем скачивания данных с сервера и временем выполнения ее на процессоре. Необходимо так организовать выполнение программ на процессорах, при котором время завершения последней программы минимально. Распределение программ по процессорам известно заранее.

Рекомендация: $F^A \leq 2\ F^*$.

12. Коммивояжер хочет объехать все города, побывав в каждом по одному разу и затратив на путешествие наименьшее количество денег. При этом стоимость перелета из одного города в другой есть функция, пропорциональная расстоянию между городами.

Рекомендация: $FA \leq 2\ F^*$.

13. Решить задачу Штейнера с прямоугольной метрикой на плоскости.

Рекомендация: $FA \le \frac{3}{2} F^*$.

14. Необходимо указать такой порядок суммирования векторов в R^m, при котором все частичные суммы попадают в шар минимального радиуса, при условии, что сумма всех векторов = 0.

Рекомендация: $F^A \le 2m\ F^*$.

15. Имеется n работ и 1 работник. Для каждой работы задана длительность ее выполнения рабочим и время, начиная с которого рабочий может выполнять работу. Для каждой работы начисляется штраф, который определяется следующим образом: время, когда рабочий начал выполнять эту работу, минус время, когда рабочий мог начать выполнять данную работу. Определить порядок работ, при котором суммарный штраф будет минимальным.

Рекомендация: $F^A \le \frac{5}{3} F^*$.

16. Дано m работников и n работ. Каждый работник может выполнить любую работу. Каждый работник характеризуется скоростью. Каждая работа характеризуется временем, которое понадобтся работнику с единичной скоростью, чтобы ее выполнить. Реализовать алгоритм, позволяющий таким образом распределить работы, чтобы минимизировать время выполнения работ самым загруженным работником.

Рекомендация: $F^A \le 2\ F^*$.

17. Имеется n работ и m работников. Каждая работа характеризуется длительностью ее исполнения работником единичной производительности. Для каждого работника известна его производительность. Необходимо так распределить работы среди работников, чтобы минимизировать время выполнения последней работы.

Рекомендация: $F^A \le 2\ F^*$.

18. Имеется n работ. Каждая работа характеризуется двумя параметрами: временем подготовки и временем выполнения. Имеется один исполнитель, который выполняет работы. Подготовкой он не занимается. Качество его работы характеризуется штрафом, который для каждой работы равен разности между временем начала работы и временем ее возможного раннего начала. Определить порядок работ, при котором суммарный штраф всех работ минимален.

Рекомендация: $F^A \le n-1\ F^*$.

19. Имеется n деталей и m станков. Каждая деталь характеризуется 3 параметрами: временем доставки, временем обработки и временем доставки на склад. Станок обрабатывает любую деталь сразу, все станки

одинаковы. Определить порядок обработки деталей на станках, когда все детали будут на складе за минимальное время.

Рекомендация: $F^A \le 2m\ F^*$.

20. Имеется N точек на плоскости. Требуется решить задачу коммивояжёра на плоскости с прямоугольной метрикой.

Рекомендация: $F^A \le 2m\ F^*$.

21. В городе имеется несколько маршрутов транспорта. Известны стоимости проезда между станциями. Необходимо поделить город на $к$ зон, установить в каждой зоне стоимость проезда и стоимость проезда между зонами, чтобы максимальное изменение цены проезда между станциями было минимально.

22. Имеется n деталей, m последовательных станков. Для каждой работы известно время выполнения на каждом станке. Необходимо определить порядок обработки деталей, при котором последняя деталь обработается как можно раньше.

23. Имеется n работ и m работников. Каждая работа характеризуется длительностью ее исполнения каждым работником. Необходимо так распределить работы среди работников, чтобы максимально загруженный работник был загружен как можно меньше.

Рекомендация: $F^A \le 4/3 - 1/3 * m\ F^*$.

24. Имеется n деталей, m последовательных станков. Для каждой работы известно время выполнения на каждом станке и частичный порядок предшествования деталей. Необходимо определить порядок обработки деталей, при котором последняя деталь обработается как можно раньше.

Рекомендация: $F^A \le \frac{m+1}{2} \max_i(F_i) \le \frac{m+1}{2} F^*$.

25. Имеется n городов и m машин. Необходимо объехать каждый город ровно 1 раз, используя некоторое число машин, чтобы стоимость проезда была минимальной. Города задаются координатами точек на плоскости.

26. Имеется n деталей и m станков. Для каждой работы известно время, необходимое для обработки этой детали. Необходимо определить порядок обработки деталей, при котором последняя деталь обработается как можно раньше.

Рекомендация: $F^A \le 4/3\ F^*$.

ЛИТЕРАТУРА

1. *Ахо А. В.*, *Хопкрофт Д. Э.*, *Ульман Д. Д.* Структуры данных и алгоритмы: Учеб. пособие/ Пер. с англ. М.: Изд. Дом "Вильямс", 2000. - 384 с.
2. *Кормен Т.*, *Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 1999. - 960 с
3. *Котов В.М., Соболевская Е.П.* Структуры данных и алгоритмы: теория и практика. Учебное пособие (с грифом министерства образования). Мн.: БГУ. 2004.- 252 с.
4. *Ковалев М.Я., Котов В.М., Лепин В.В*. Теория алгоритмов. Часть 2. Приближенные алгоритмы. Курс лекций. – Мн.: БГУ, 2003. - 147 с.
5. *Котов В.М.* Алгоритмы для задач разбиения и упаковки. Научное издание. Мн.: БГУ. 2001. - 97 с.