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

СОГЛАСОВАНО
Заведующий кафедрой
_____ Котов В.М.
«21» октября_ 2021 г.

СОГЛАСОВАНО
Декан факультета
_____ Недзьведь А.М.
«16» ноября_ 2021 г.

СОГЛАСОВАНО
Председатель
Учебно-методической комиссии факультета
_____ Козловская И.С.
«16» ноября_ 2021 г.

Сборник задач по теории алгоритмов. Организация перебора и приближенные алгоритмы

Электронный учебно-методический комплекс для специальности:
1-31 03 04 «Информатика»

Регистрационный № 2.4.2-12/201

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

Рассмотрено и утверждено на заседании Научно-методического совета БГУ 30.11.2021 г., протокол № 2.

Минск 2021

УДК 510.51(075.8)(076.1)
К 736

Утверждено на заседании Научно-методического совета БГУ
Протокол № 2 от 30.11.2021 г.

Решение о депонировании вынес:
Совет факультета прикладной математики и информатики
Протокол № 3 от 16.11.2021 г.

А в т о р ы :

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

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

Волчкова Галина Петровна, старший преподаватель кафедры дискретной математики и алгоритмики факультета прикладной математики и информатики БГУ.

Рецензенты:

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

Гляков П.В., профессор кафедры информационных технологий в культуре Белорусского государственного университета культуры и искусства, кандидат физ.-мат. наук, профессор.

Котов, В. М. Сборник задач по теории алгоритмов. Организация перебора и приближенные алгоритмы : электронный учебно-методический комплекс для специальности: 1-31 03 04 «Информатика» / В. М. Котов, Е. П. Соболевская, Г. П. Волчкова ; БГУ, Фак. прикладной математики и информатики, Каф. дискретной математики и алгоритмики. – Минск : БГУ, 2021. – 144 с. : ил. – Библиогр.: с. 143–144.

Электронный учебно-методический комплекс (ЭУМК) по учебной дисциплине «Теория алгоритмов» («Модели и алгоритмы задач дискретной оптимизации») предназначен для студентов специальности 1-31 03 04 «Информатика». В ЭУМК содержатся лекционный материал, задания для практических занятий, наборы тестов для проверки правильности программ, задания эвристического типа, список литературы.

ОГЛАВЛЕНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс (ЭУМК) по учебной дисциплине «Теория алгоритмов» («Модели и алгоритмы задач дискретной оптимизации») предназначен для студентов 2 курса специальности 1-31 03 04 Информатика.

Комплекс подготовлен в соответствии с требованиями Положения об учебно-методическом комплексе на уровне высшего образования, утвержденного Постановлением министерства образования Республики Беларусь от 26.07.2011 № 167.

Содержание разделов ЭУМК соответствует образовательным стандартам, структуре и тематике типовых учебных программ по дисциплине «Теория алгоритмов» («Модели и алгоритмы задач дискретной оптимизации»). Главные цели ЭУМК: помощь студентам в организации самостоятельной работы, повышение качества подготовки и усиление практико-ориентированности учебного процесса по дисциплинам.

ЭУМК состоит из следующих разделов.

Теоретический. Включает аннотацию ЭУМК, написанного в соответствии с программой дисциплины. Материал данного комплекса, наряду с конспектом лекций, может быть использован для самостоятельной подготовки студентов к контрольным заданиям и экзамену.

Практический. Содержит набор задач. Данные материалы используются при проведении практических занятий и для самостоятельной работы над курсом.

Раздел контроля знаний представлен наборами тестов для проверки работоспособности программ. Описаны формы диагностики и технология определения оценки по дисциплине с учетом текущей успеваемости.

Вспомогательный раздел включает рекомендуемую литературу.

Учебная дисциплина «Теория алгоритмов» («Модели и алгоритмы задач дискретной оптимизации») знакомит студентов с методами организации перебора NP-трудных задач и методами построения и анализа приближенных алгоритмов. Основой для изучения учебной дисциплины являются учебные дисциплины «Дискретная математика и математическая логика», «Алгоритмы и структуры данных» и «Программирование» («Основы и методологии программирования»). В свою очередь, знания, приобретенные в рамках учебной дисциплины «Теория алгоритмов» («Модели и алгоритмы задач дискретной оптимизации»), являются базой для таких учебных дисциплин, как «Исследование операций», а также ряда дисциплин специализаций.

Изучение учебной дисциплины формирует у студентов знания о методике организации перебора, эффективной реализации перебора с использованием

нижних и верхних оценок, методами построения и анализа приближенных алгоритмов.

Цель преподавания учебной дисциплины – формирование навыков для эффективной реализации алгоритмов решения NP-трудных задач.

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

Основные задачи, решаемые при изучении учебной дисциплины «Теория алгоритмов» («Модели и алгоритмы задач дискретной оптимизации»):

разработка и анализ методов для решения NP-трудных задач и оценка;

построение и исследование различных моделей задач с полной и неполной информацией.

В результате изучения учебной дисциплины, обучающийся должен

знать:

методы построения дерева вариантов и его обхода;

основные методы отсечений при организации перебора вариантов;

способы построения верхних и нижних оценок;

основные подходы при разработке приближенных алгоритмов и методы их оценки;

уметь:

организовывать дерево перебора вариантов и осуществлять отсечения бесперспективных решений;

оценивать качество приближенных алгоритмов;

владеть:

способами организации перебора при решении NP-трудных задач и приемами повышения эффективности перебора;

приемами построения и оценки качества приближенных алгоритмов.

Освоение учебной дисциплины «Теория алгоритмов» должно обеспечить формирование следующих компетенций:

академические компетенции:

АК-1. Уметь применять базовые научно-теоретические знания для решения теоретических и практических задач;

АК-2. Владеть системным и сравнительным анализом;

АК-4. Уметь работать самостоятельно;

АК-5. Быть способным вырабатывать новые идеи (обладать креативностью);

АК-6. Владеть междисциплинарным подходом при решении проблем;

социально-личностные компетенции:

СЛК-3. Обладать способностью к межличностным коммуникациям;

СЛК-5. Быть способным к критике и самокритике (критическое мышление);

СЛК-6. Уметь работать в команде;

профессиональные компетенции:

ПК-6. Принимать оптимальные управленческие решения;

ПК-11. Реализовывать управленческие инновации в профессиональной деятельности;

ПК-14. Использовать достижения науки и передовых технологий в образовательной и научно-исследовательской сферах.

Освоение учебной дисциплины «Модели и алгоритмы задач дискретной оптимизации» должно обеспечить формирование следующих компетенций:

Базовые профессиональные компетенции:

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

Для организации самостоятельной работы студентов и самоподготовки по курсу рекомендуется размещение программы курса, списка необходимой и дополнительной литературы, лекций, заданий, тестов, методических рекомендаций на доступных сетевых ресурсах факультета. Эффективность самоподготовки студентов целесообразно проверять в виде текущего и итогового контроля знаний в форме компьютерного тестирования как по отдельным темам, так и по разделам курса на образовательной платформе iRunner.

Для общей оценки качества усвоения студентами учебного материала рекомендуется использование рейтинговой системы оценивания, когда для каждого задания определен уровень сложности.

В образовательную платформу iRunner интегрирована свободная система управления содержимым (англ. CMS, или content management system) под названием MediaWiki. Этот продукт используется в качестве электронного учебника для размещения материалов лекций и практических занятий, в том числе по теории алгоритмов. https://acm.bsu.by/wiki. Опыт показывает, что создавать материалы в вики-формате удобнее, чем публиковать отдельные doc- или pdf-файлы. Вики-страницы можно легко редактировать, сохраняется история изменений. Страницы связаны между собой ссылками. Фрагменты кода на языках программирования отображаются с подсветкой синтаксиса. Вики-документы хорошо адаптированы для просмотра на мобильных устройствах, а при необходимости легко получить версию для печати.

ЭУМК написан при поддержке гранта Президента Республики Беларусь на 2021 год в области образования, выделенного согласно распоряжению Президента Республики Беларусь от 31 декабря 2020 г. № 260рп «О предоставлении

грантов Президента Республики Беларусь на 2021 год».

1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1. Организация перебора

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

1.1.1. 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 рисунке 1.1, причем вершины изображают подзадачи.

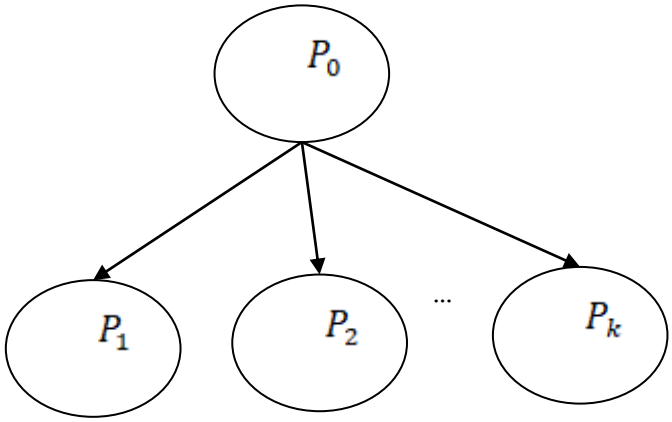


Рисунок 1.1 − Дерево разбиения.

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 рисунке 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

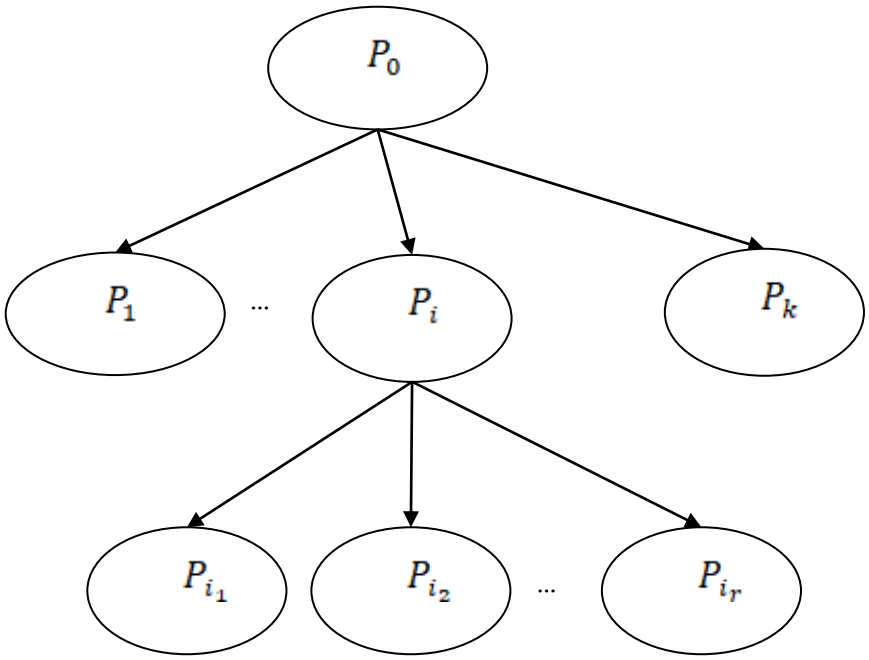


Рисунок 1.2 − Дерево подзадач.

называются висячими вершинами, и на рисунке 1.2 это вершины $P_1, \dots, P_{i_1}, P_{i_2}, \dots, P_{i_r}, \dots, P_k$.

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

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

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

Так как данное соотношение должно быть применено к каждому разбиению, то $\{P_0\} = \cup\{P_j\}$, где P_j – висячая вершина дерева.

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

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

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 $x_i = a$; для P_{i_2} полагаем $x_i = b$; для P_{i_3} полагаем $x_i = c$; и для P_{i_4} полагаем $x_i = d$.

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 (См. рисунок 1.3).

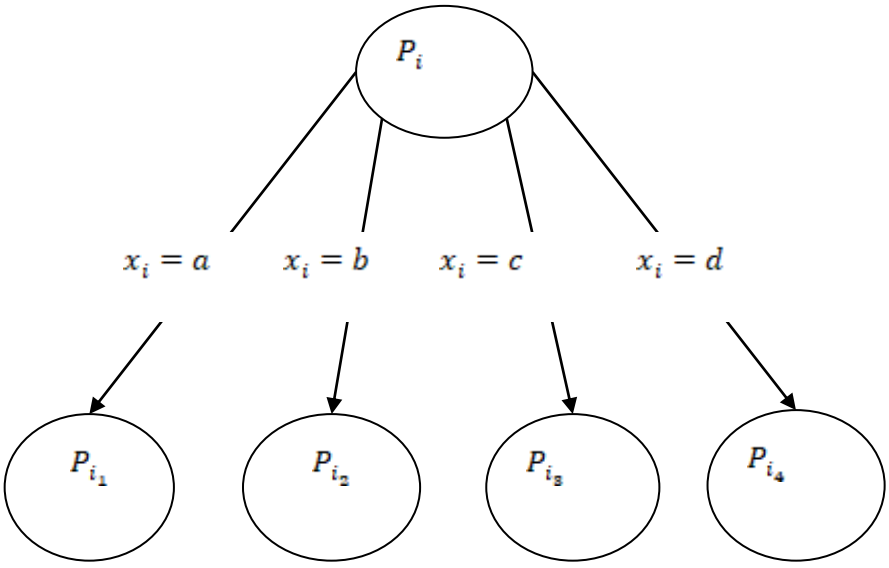


Рисунок 1.3 – Покомпонентное ветвление.

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 $x_i = a$, а для P_{i_2} полагаем $x_i \neq a$, т. е. $x_i = b, c$ или d (См. рисунок 1.4).

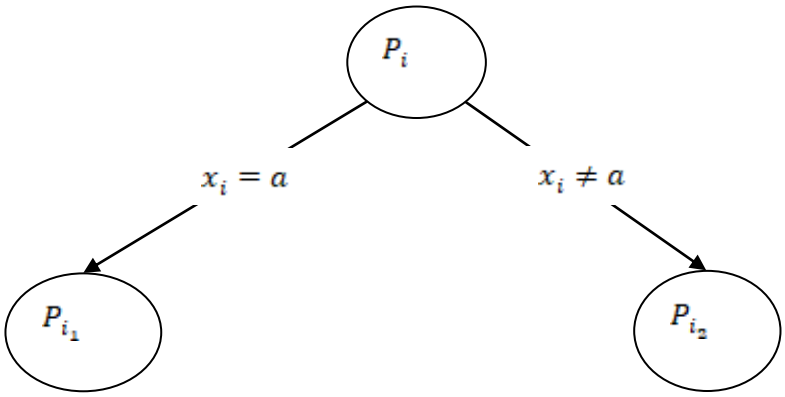


Рисунок 1.4 – Разбиение по признаку.

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

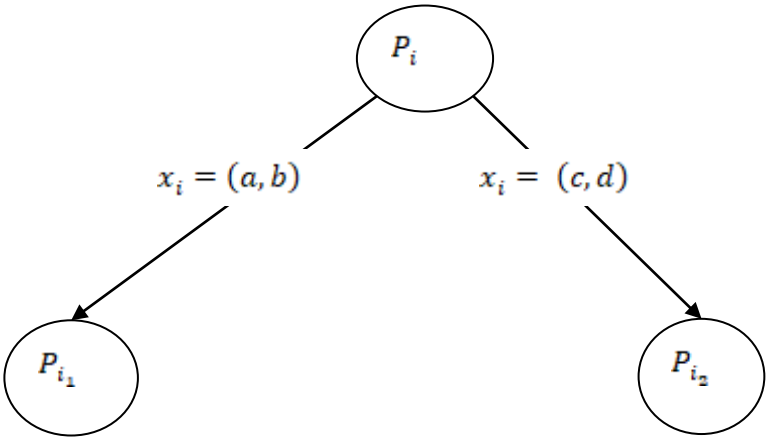


Рисунок 1.5 – Разбиение на две подзадачи.

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 [7]. Но при этом следует отметить, что подзадачи этих подзадач могут и не обладать свойством (1.1). Например, подзадача P_{i_1} оптимальна на подзадаче $x_i = a$ и подзадача P_{i_2}оптимальна на подзадаче $x_i = a$.

1.1.2. 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

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 фронта. Все подзадачи данного уровня должны исследоваться до задач следующего уровня

Вид дерева вариантов при стратегии фронтального ветвления показан на рисунке 1.6.

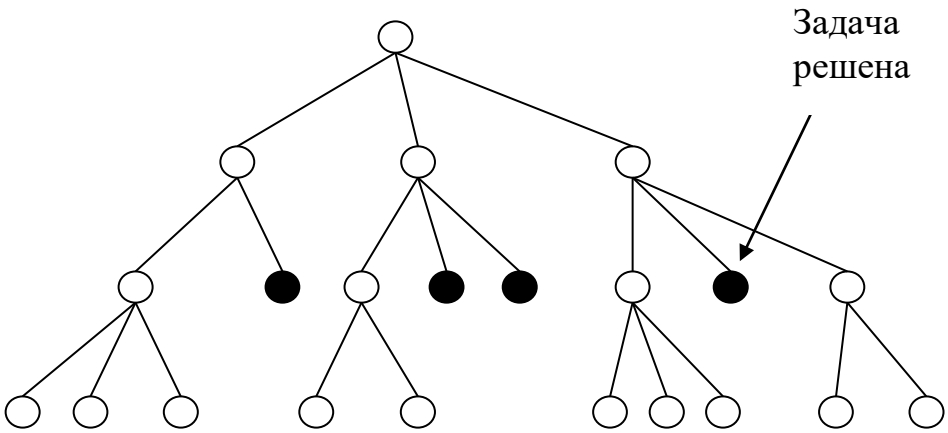


Рисунок 1.6 – Фронтальное ветвление.

Одностороннее ветвление

При такой стратегии 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 одностороннем ветвлении 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 ветвления, когда разрешается первая подзадача, показан на рисунке 1.7, где порядок приоритета исследования получаемых подзадач отражен соответствующей нумерацией.

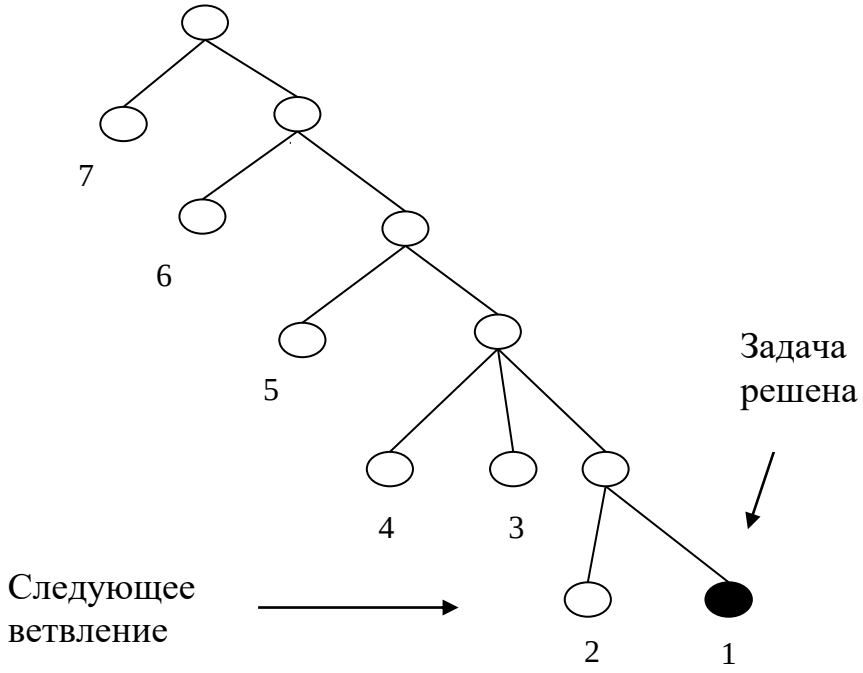


Рис. 1.7 – Одностороннее ветвление.

Такая стратегия часто называется *поиск с возвратом*. Основные ее достоинства – простота реализации и минимизация используемой памяти, так как в отличие от фронтального ветвления нет необходимости хранить формат вершин кандидатов для ветвления. Может показаться, что аналогичным образом ведет себя и *поиск в глубину* при обходе вершин графа [10]. 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.

Как при одностороннем, так и при фронтальном ветвлении выбор очередной вершины для ветвления не был полностью определен. При одностороннем ветвлении, 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. Если мы не указали, в какой именно, то любая из них может рассматриваться как «последняя порожденная». При фронтальном ветвлении, как уже было сказано, все подзадачи данного уровня должны исследоваться до задач следующего уровня, но не был четко определен порядок их исследования на одном уровне. Функция ветвления – это функция, которая позволяет «вычислить», какая из допустимых вершин должна исполь-

зоваться 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 другой стратегии ветвления. Можно использовать функцию ветвления 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. Эта стратегия ветвлени является, вообще говоря, гибридом описанных ранее подходов ветвления, хотя в литературе она часто называется поиском по ширине.

Предположим, что для графа $G = (V, E)$, приведенного на рисунке 1.8, необходимо найти простую цепь, соединяющую вершины 1 и 6, используя алгоритм поиска в глубину [2].

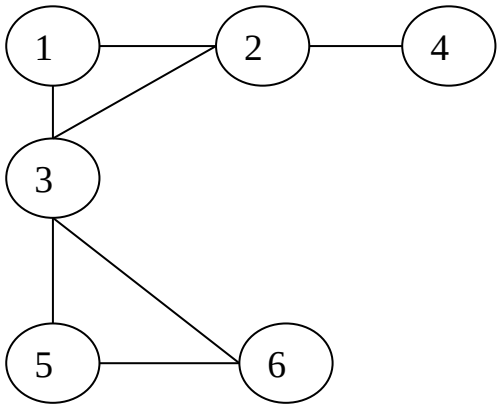


Рисунок 1.8 – Изображение графа.

В результате поиска получаем корневое дерево поиска в глубину, пред-

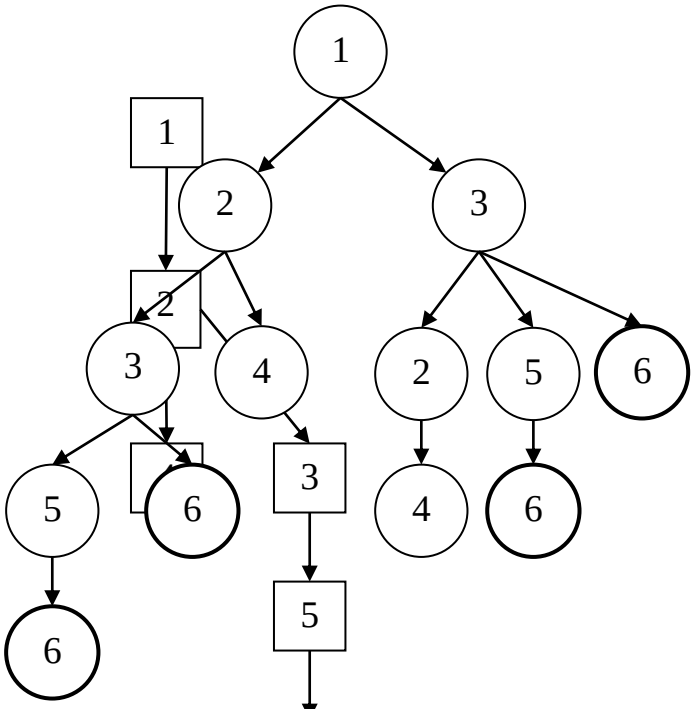


Рисунок. 1.10 – Дерево перебора вариантов.

Рисунок 1.9 – Дерево поиска в глубину.

ставленное на рисунке 1.9.

Несложно увидеть, что в корневом дереве поиска в глубину нет повторяющихся вершин.

Построим теперь для графа, приведенного на рисунке 1.8, все простые цепи, соединяющие вершины 1 и 6, 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 отличны 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 рисунке 1.8, получим следующую древовидную структуру, изображенную на рисунке 1.10.

Сгенерированы четыре попарно различные тривиальные цепи, соединяющие вершины 1 и 6:

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

Несложно увидеть, что в дереве решений на рисунок 1.10 некоторые вершины повторяются.

1.1.3. Сокращение числа необходимых для решения подзадач: отсев вариантов

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 недопустимости* (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 рисунке 1.11, мы имеем три клики:

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

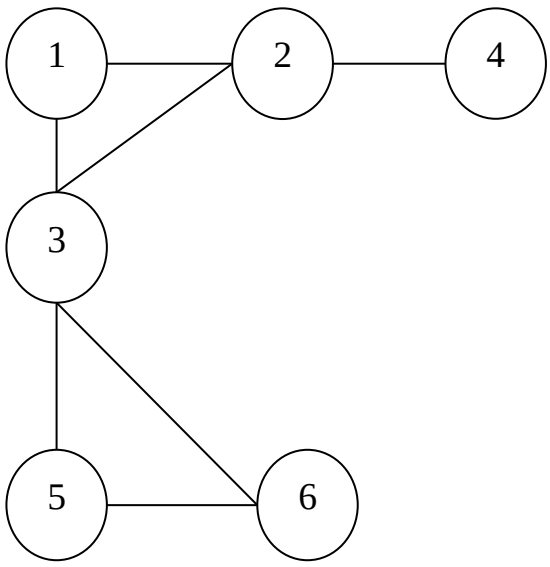


Рисунок 1.11 – Клики графа.

Одним из алгоритмов построения *множества всех клик графа* является поиск с возвратом [12].

Каждая вершина дерева поиска будет соответствовать полному подграфу графа (полный подграф задается множеством его вершин), а каждая дуга дерева поиска – вершине графа.

Корень дерева – пустое множество вершин. Предположим, что некоторой вершине дерева поиска поставлен в соответствие полный подграф графа с множеством вершин С. Пусть вершина *w* графа смежна с каждой из вершин множества С. Тогда вершина $C \cup \{w\}$ будет сыном вершины С, а дуга, идущая от С к $C \cup \{w\}$, будет соответствовать вершине *w*. Если не существует вершин смежных с каждой из вершин множества *C*, то ветвление в соответствующей вершине дерева завершено и данная висячая вершина порождает клику графа.

Дерево решений для графа, изображенного на рисунок 1.11, будет иметь следующий вид (См. рисунок 1.12).

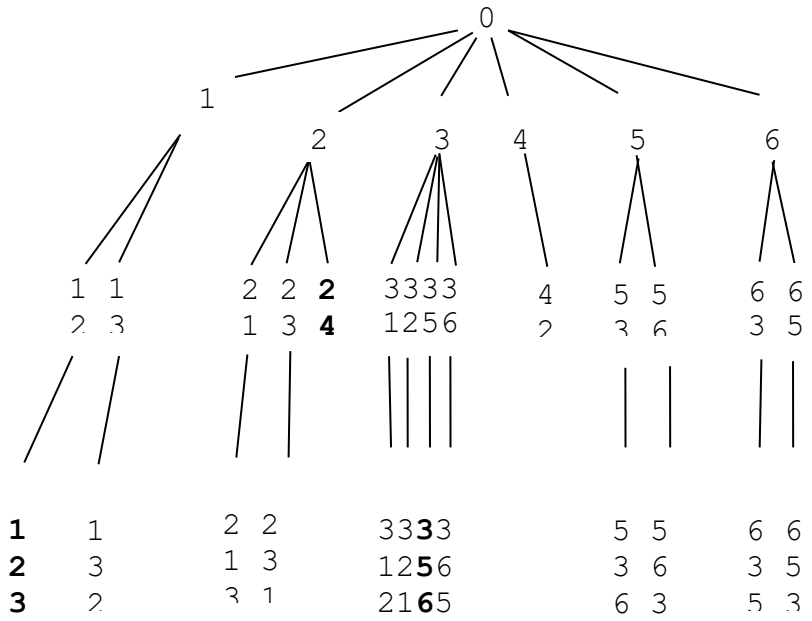


Рисунок 1.12 –Дерево перебора вариантов

На рисунке 1.12 мы видим, что поиск с возвратом привел нас к тому, что мы породили 14 клик, но только три из них различны:

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

Число клик в графе может расти экспоненциально относительно числа вершин. Поэтому важно при построении дерева решений *избегать повторений*. Для этого 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 исследо-

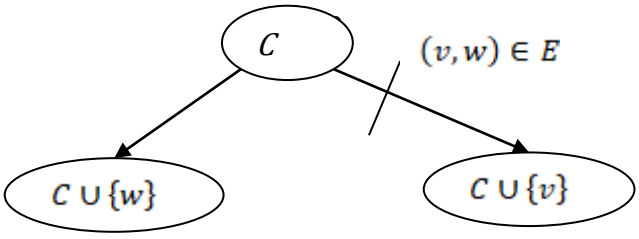


Рисунок 1.13 - Иллюстрация к теореме 1.1.

вать только тех из оставшихся сыновей вершины C, которые не смежны с вершиной w графа (См. рисунок 1.13).

Теорема 1.1 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 неисследованные поддеревья с корнями в вершине

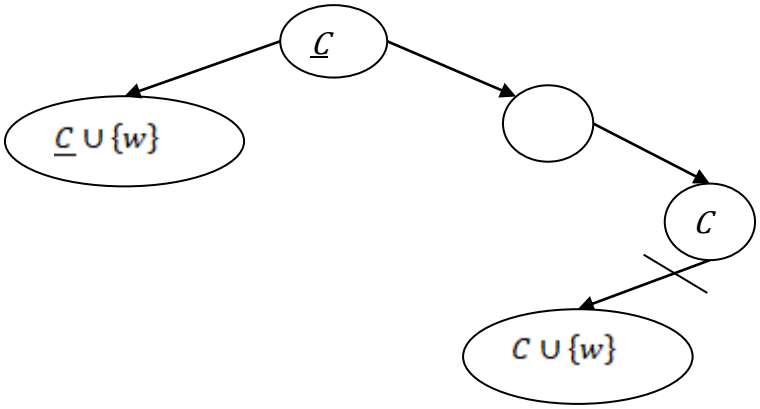


Рисунок 1.14 - Иллюстрация к теореме 1.2.

$C \cup \{w\}$ можно проигнорировать (рисунок 1.14).

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

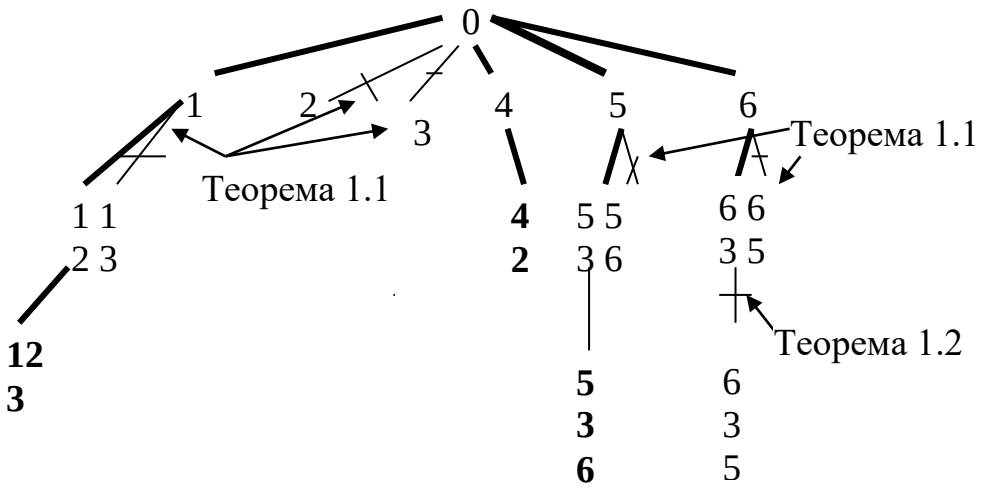


Рисунок 1.15 – Вид дерева после отсечений.м

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 недопустимости

Рассмотрим задачу определения множества всех контуров ориентированного графа (орграфа) $D = (V, E)$. 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 ориентированное как обратное ребро (дуга) [10].

Несложной 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, ориентируя при этом все ребра графа как «древесные» или «обратные». Во время поиска будет построено глубинное дерево (дереву принадлежат все ребра графа, которые были ориентированы как древесные).

Каждое ребро, ориентированное как обратное, при добавлении к глубинному дереву поиска порождает ровно один цикл. Мощность фундаментального множества циклов графа G равна: $|E| - |V| + 1$.

Проиллюстрируем построение фундаментального множества циклов для графа, изображенного на рисунок 1.16. Для заданного графа, начиная, например, с вершины 1, выполним алгоритм поиска в глубину.

На рисунке 1.16 ребра $\{2,1\}$, $\{3,2\}$, $\{4,3\}$, которые были ориентированы как древесные: $(1,2)$, $(2,3)$ и $(3,4)$, выделены жирными линиями. Ребра $\{1,3\}$ и $\{2,4\}$ были ориентированы как обратные: $(3,1)$ и $(4,2)$.

На рисунке 1.17 показано фундаментальное множество циклов графа, приведенного на рисунок 1.16.

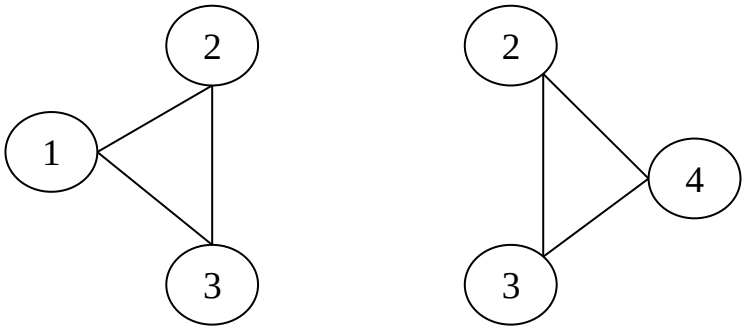


Рисунок 1.17 – Фундаментальное множество циклов.

Интересным является тот факт, что любой цикл в графе может быть представлен в виде линейной комбинации фундаментальных циклов. Здесь под линейной комбинацией циклов понимается такой цикл, в котором присутствуют только ребра, участвующие в линейной комбинации нечетное число раз.

В случае графа, изображенного на рисунке 1.16, цикл (1, 2, 4, 3, 1) является линейной комбинацией циклов (1, 2, 3, 1) и (2, 4, 3, 2).

Более сложной является задача *построения множества всех контуров орграфа* [12]. 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 являет-

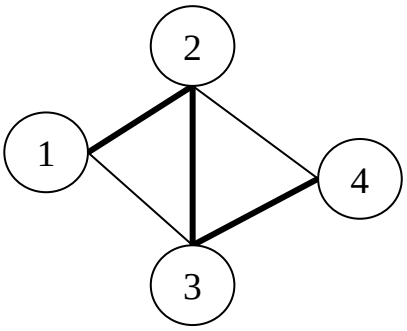


Рисунок. 1.16 –Древесные ребра.

ся задача отсечения повторяющихся контуров, а также отсев неперспективных ветвлений (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. Например, для контура$3 \to 2 \to 6 \to 3$корнем служит вершина 2.

Алгоритм 1.1

Последовательно будем строить контуры с корнями в вершинах $1,2,\dots,n$. Для построения всех контуров с корнем в вершине *s* поступаем следующим образом.

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

2) Поиском в глубину (идем в вершины, которые не заблокированы) строим путь $(s, v_1, v_2, \dots, v_k)$ 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.

3) Контур с корнем в вершине *s* построен, если на некотором шаге алгоритма $v_{k+1} = s$. После построения контура $(s, v_1, v_2, \dots, v_k, s)$ исследуем следующую дугу, выходящую из вершины v_k, и если она ведет в незаблокированную вершину, то продолжаем поиск в глубину из этой вершины. Если же оказывается, что все дуги, выходящие из вершины v_k, исследованы, то возвращаемся в вершину v_{k-1} и 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, не принадлежащие рассматриваемому пути $s \to v_1 \to \cdots \to v_i$ и являющиеся предшественниками вершины v_i (началами дуг, входящих в вершину v_i), также должны быть разблокированы. Таким образом, разблокирование некоторой вершины пути может привести к цепочке разблокирования других ранее заблокированных вершин (*отсечение по недопустимости*).

4) 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 орграфа, изображенного на рисунок 1.18, построить множество всех контуров с корнем в вершине 1.

При поиске контуров с корнем в вершине 1 мы построим путь $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$, после чего все вершины (за исключением вершины 1) станут заблокированными, а контур с корнем в вершине 1 не построен. Осуществляется возврат из вершины 5, но при этом она останется заблокированной. Аналогично произойдет и с вершиной 4 (См. рисунок 1.19).

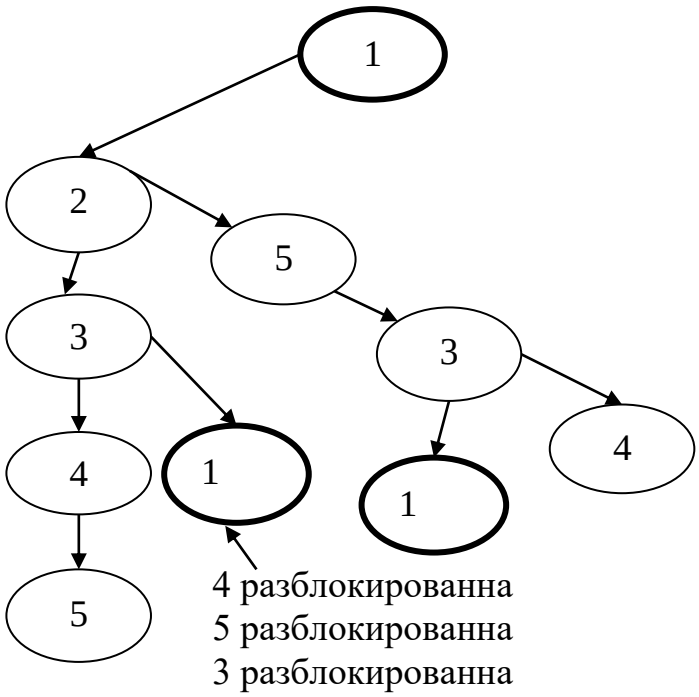


Рисунок 1.19– Принцип разблокировки.

Для пути $1 \rightarrow 2 \rightarrow 3$ существует еще одна не просмотренная дуга $3 \rightarrow 1$, приводящая к контуру. Поэтому после того, как будет осуществлен возврат из вершины 3, она станет разблокированной.

Разблокирование вершины 3 повлечет за собой разблокирование вершины

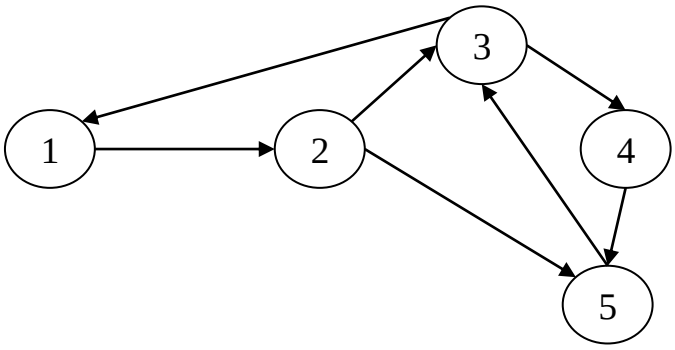


Рисунок 1.18 –Принцип блокировки.

5 (вершина 5 не принадлежит пути $1 \rightarrow 2 \rightarrow 3$ и является предшественником вершины 3 в орграфе).

В свою очередь, разблокирование вершины 5 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 непросмотренной 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 (рисунок 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 непросмотренной 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 (См. рисунок 1.19).

В результате построены два контура с корнем в вершине 1:

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

Пример 1.1. Построить множество контуров, для ориентированного графа, изображенного на рисунке 1.20.

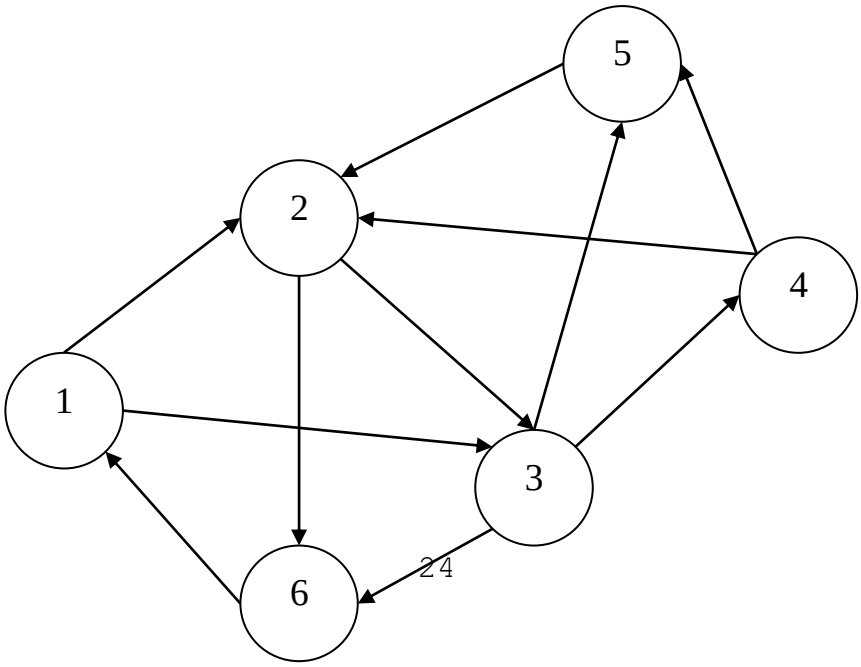


Рисунок 1.20 – Изображение графа.

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

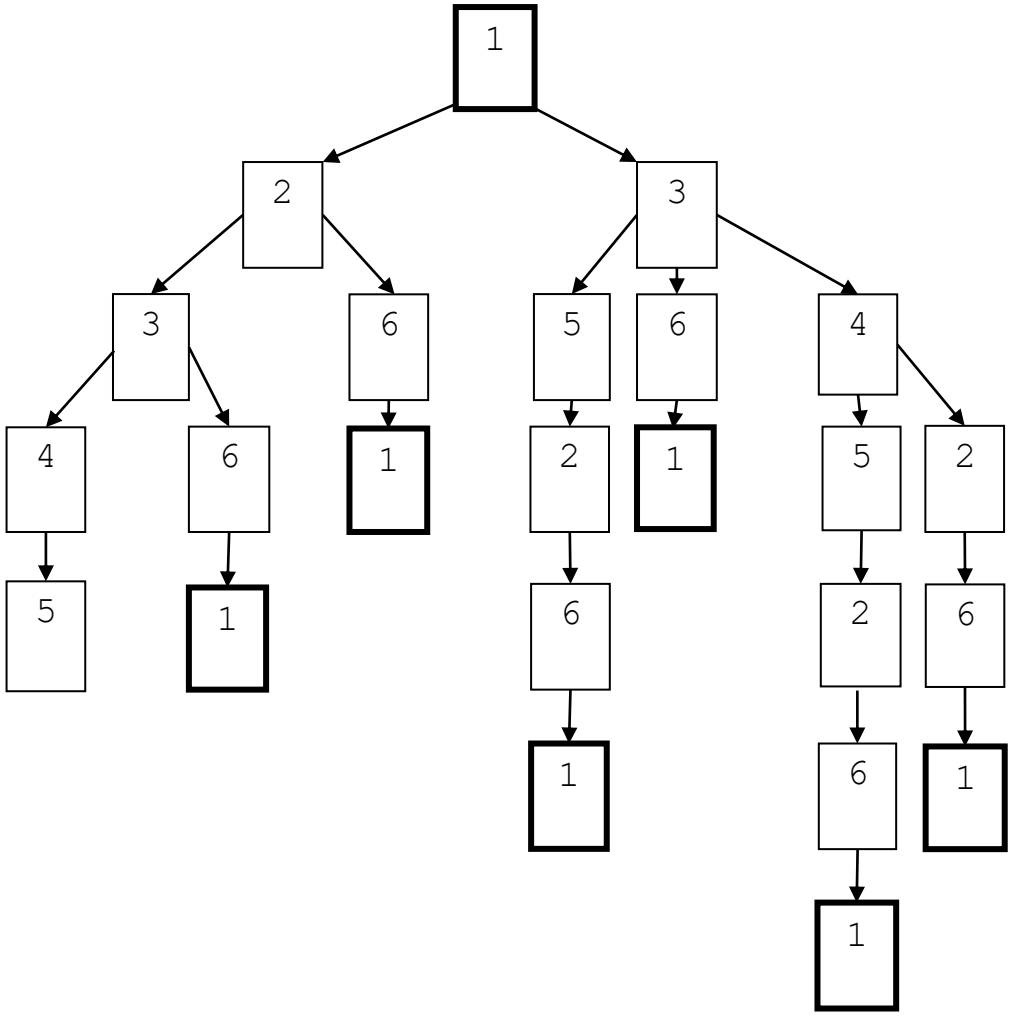


Рисунок 1.21 –Построение контуров с корнем в вершине 1.

Контуры с корнем в вершине 1 построены:

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

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

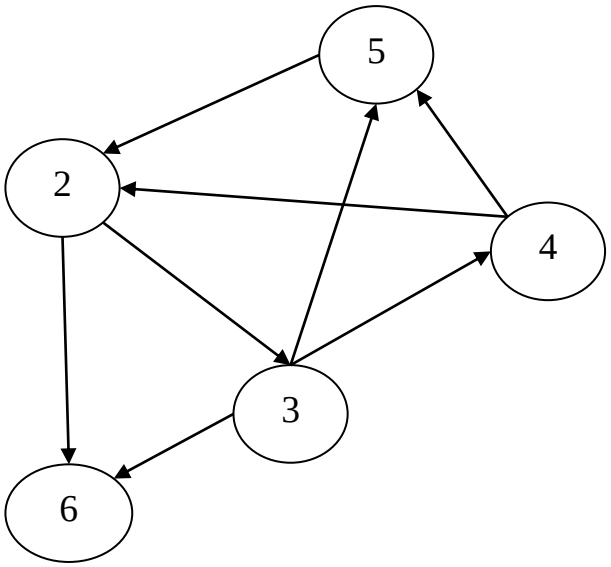


Рисунок 1.22 −Удаление вершины 1.

Для орграфа, приведенного на рисунке 1.22, последовательность действий построения множества контуров с корнем в вершине 2 приведена на рисунке 1.23.

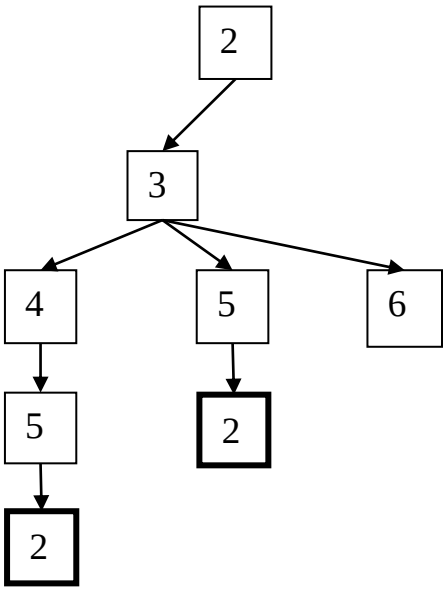


Рисунок 1.23 −Контуры с корнем в вершине 2.

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 рисунке 1.22, вершину 2 вместе с инцидентными ей дугами и для модифицированного орграфа (рисунок 1.24) строим множество контуров с корнем в вершине 3.

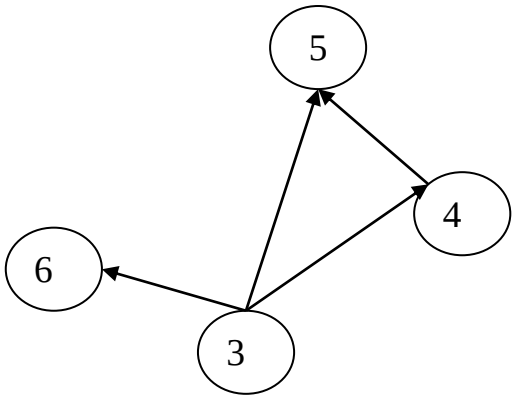


Рисунок 1.24– После удаления вершины 2.

Нетрудно убедиться, что орграф, приведенный на рисунке 1.24, ациклический, поэтому алгоритм построения всех контуров для орграфа, приведенного на рисунке 1. 20, заканчивает свою работу.

Отсев по рекорду (применение оценок)

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

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

Во многих задачах дискретной оптимизации целевая функция удовлетворяет равенству:

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

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

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.

Цель организации поиска – такой обход дерева, который позволяет построить все оптимальные решения, просмотрев минимальное число вершин дерева. Самое простое решение поставленной задачи – это 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 не-

перспективными 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 $F^* \geq 0$).

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

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

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

Для подзадачи определим верхнюю и нижнюю оценки:

$$HO(\bar{x}) \leq \text{опт}(\bar{x}) \leq BO(\bar{x}),$$

где $\text{опт}(\bar{x})$ – значение оптимального решения подзадачи *x* (См. рисунок 1.25).

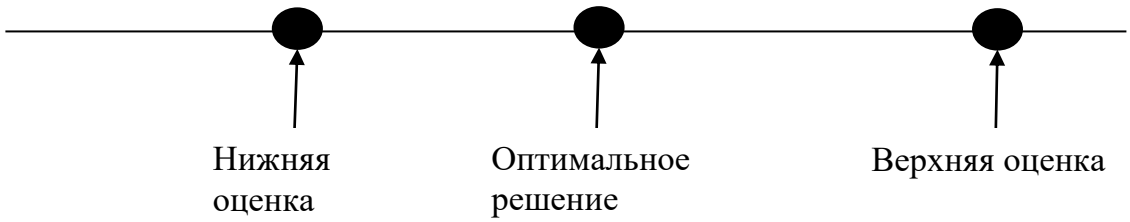


Рисунок 1.25– Оценки.

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 оптимального решения более общей задачи берется в качестве оценки.

Если в задаче минимизации для вершины дерева *x* выполняется неравенство

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

то понятно, что

$$F < F(x) + \text{опт}(\bar{x}),$$

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 $HO(\overline{x}) = 0$. 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

$$F > F(x) + BO(\overline{x}),$$

то

$$F > F(x) + \text{опт}(\bar{x}),$$

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 $BO(\overline{x}) = +\infty$. В этом случае отсечения частичного решения по рекорду не происходят никогда, происходит только пересчет рекорда, следовательно, выполняется полный обход дерева решений.

Следует отметить, что для построения одного оптимального решения можно использовать более сильное отсечение с нестрогим неравенством: $F \leq F(x) + HO(\overline{x})$ для задачи на минимум и $F \geq F(x) + BO(\overline{x})$ для задачи на максимум.

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

Рассмотрим задачу минимизации.

Теорема 1.3. Если нижняя оценка подзадачи совпадает со значением ее оптимального решения, т. е. $HO(\bar{x}) = \text{опт}(\bar{x})$, то это позволяет генерировать для задачи минимизации только оптимальные решения, отсекая в дереве все неперспективные частичные решения.

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

Более того, так как по условию теоремы

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

то нижняя оценка исходной задачи совпадает со значением ее оптимального решения. Поэтому, вычислив для подзадачи $\bar{x} = (x_1, \dots, x_n)$ 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

$$F^* = F(x) + \text{опт}(\bar{x}),$$

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

Аналогично для задачи максимизации справедлива следующая теорема.

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

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

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.

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

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

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:

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

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

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

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

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

и неперспективной в противном случае (отсечение по рекорду).

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

Для эффективного вычисления рекорда F^* и нижних оценок подзадач $\text{НО}(\bar{x})$ сопоставим шахматной доске граф и выполним поиск в ширину [10] (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.

Поскольку 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 $O(N + k \cdot l)$.

Продемонстрируем работу алгоритма на примере.

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

Построим все кратчайшие маршруты коня из *s* в *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 примера представлено на рисунке 1.26. Как видно из рисунка, вершины $x_1,\ x_2$ являлись перспективными, а для вершин $x_3,\ x_4,\ x_7,\ x_8$ было выполнено отсечение по рекорду:

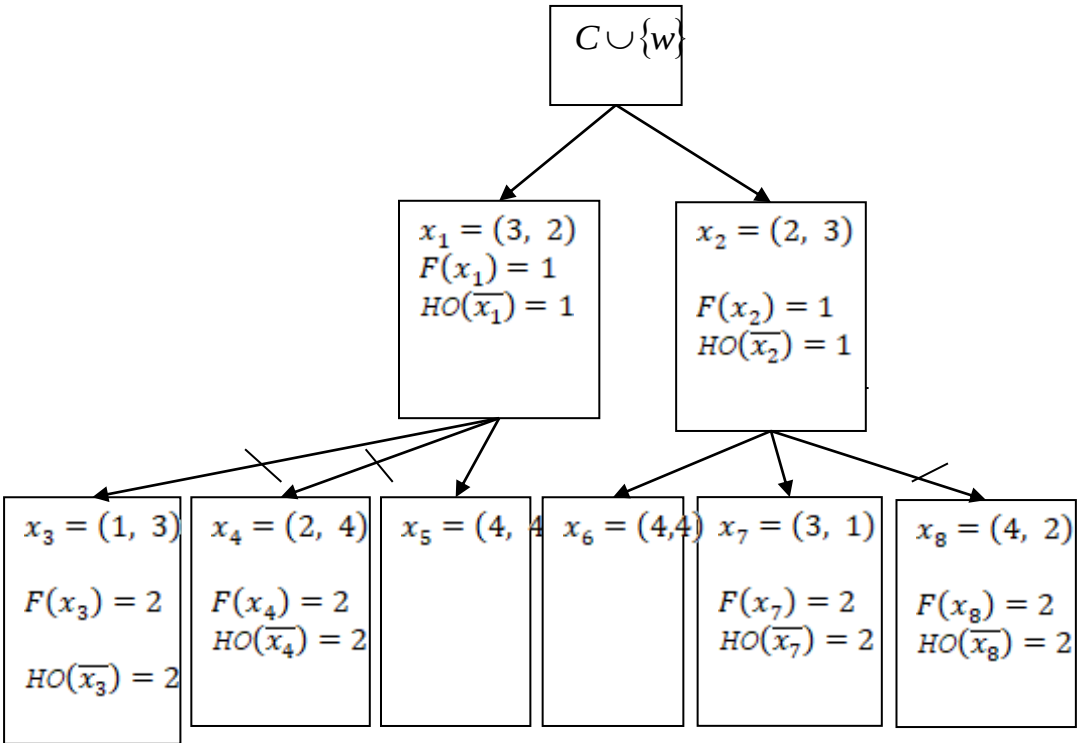


Рисунок 1.26 – Отсечения по рекорду.

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

Вершины x_5 и x_6 порождают оптимальные решения задачи:

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

1.1.4. Способы построения нижних и верхних оценок

Как было отмечено в предыдущих разделах, в задаче минимизации в качестве нижней оценки всегда можно взять число 0, а в задаче максимизации в качестве верхней оценки – бесконечность. Однако для того, чтобы отсечь наибольшее число неперспективных вариантов ветвления, надо стремиться к тому, чтобы оценки были максимально близки к оптимальному решению задачи. Существует несколько способов построения оценок. Один из них состоит в использовании значения решения, построенного алгоритмом, для которого известна гарантированная оценка.

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

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

для задачи на максимум, и

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

для задачи на минимум.

Для задачи на максимум гарантированная оценка $P \leq 1$, полученная алгоритмом A, говорит о том, что алгоритмом A будет построено решение не менее $P \cdot 100\%$ от оптимального (См. рисунок 1.27).

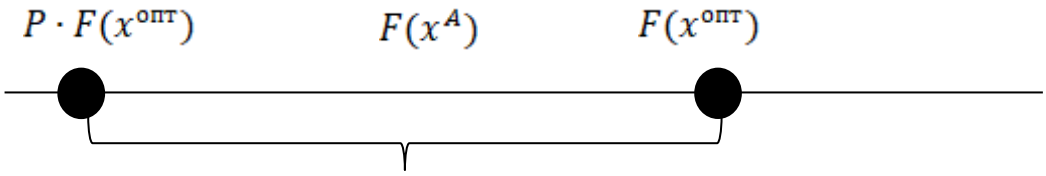
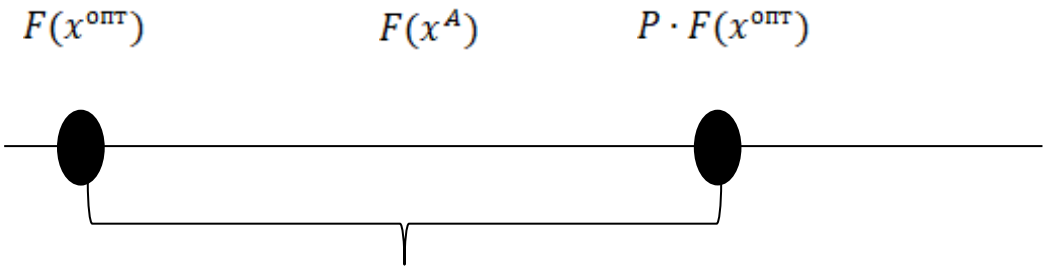


Рисунок 1.27—Гарантированная оценка для задачи на максимум.

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

Для задачи на минимум гарантированная оценка $P \geq 1$, полученная алгоритмом A, говорит о том, что алгоритм A может ошибиться не более, чем в P



раз.

Рисунок 1.28—Гарантированная оценка для задачи на максимум.

Например, если $P = 3$, то алгоритм A может ошибиться не более, чем в 3 раза (полученное решение, не более, чем в 3 раза больше, чем оптимальное) (См. рисунок 1.28).

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

$$F(x^{\text{опт.}}) \leq \frac{F(x^A)}{P}, \qquad P \leq 1,$$

и значение $F(x^A)/P$ можно взять в качестве верхней оценки.

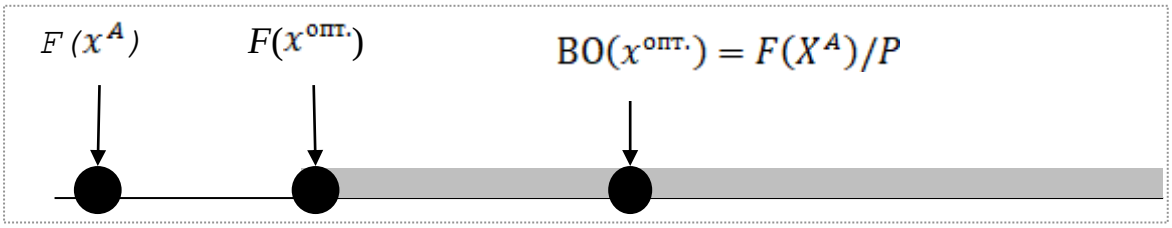


Рисунок 1.29—Верхние оценки для задачи на максимум

На рисунке 1.29 область значений, которые может принимать верхняя оценка $ВО(x^{опт.})$, выделена серой заливкой.

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

$$F(x^{опт.}) \geq \frac{F(x^A)}{P}, \qquad P \geq 1,$$

и значение $F(x^A)/P$ можно взять в качестве нижней оценки. На рисунке 1.30 область значений, которые может принимать нижняя оценка $НО(x^{опт.})$, выделена серой заливкой.

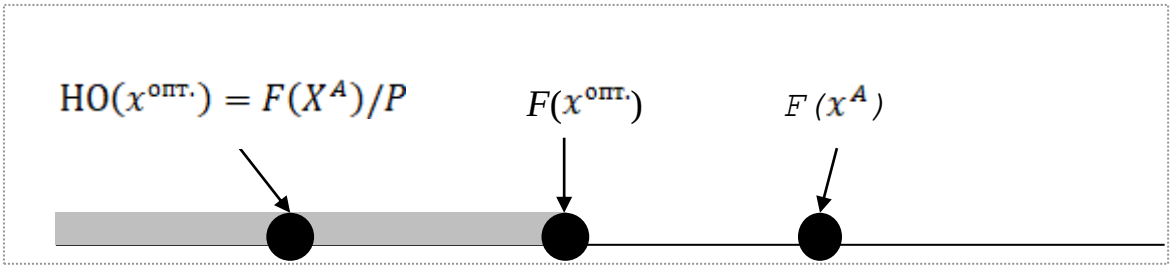


Рисунок. 1.30—*Нижние оценки для задачи на минимум.*

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

Рассмотрим одну из самых известных *NP*-трудных задач комбинаторной оптимизации – симметричную (неориентированную) задачу коммивояжера (англ. *travelling salesman problem)*.

Задан полный взвешенный граф. Необходимо найти кратчайший маршрут, который начинается в некоторой вершине графа, проходит через каждую вершину только один раз и возвращается в начальную вершину. Другими словами, задача состоит в поиске гамильтонова цикла минимального веса [11].

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

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

$$d(x, y) \leq d(x, k) + d(k, y).$$

Задача коммивояжера на плоскости – задача коммивояжера, где каждой вершине графа соответствует точка на плоскости, а вес ребра между вершинами равен расстоянию между соответствующими точками. Существует несколько вариантов задачи в зависимости от того, в какой метрике вычисляется расстояние между двумя точками на плоскости:

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

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

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

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

Можно использовать следующие подходы при построении оценок для задачи коммивояжера.

Предположим, что в задаче коммивояжера мы находимся в состоянии, в котором выбрано *k* ребер и выбранное множество ребер удовлетворяет следующим двум требованиям:

- не образуется циклов длины меньше *n*;
- не образуется вершин степени 3 (т. е. степени вершин 1 или 2).

Другими словами, выбранное множество ребер – набор простых открытых цепей. Множество ребер, которые еще не были выбраны, будем называть *свободными*. Свободное ребро, которое может быть добавлено к текущему состоянию и при этом будут выполняться оба требования, назовем *допустимым*.

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

Действительно, предположим, что выбраны ребра $\{a, b\}$ и $\{c, d\}$, которые пересекаются в точке x (См. рисунок 1.31).

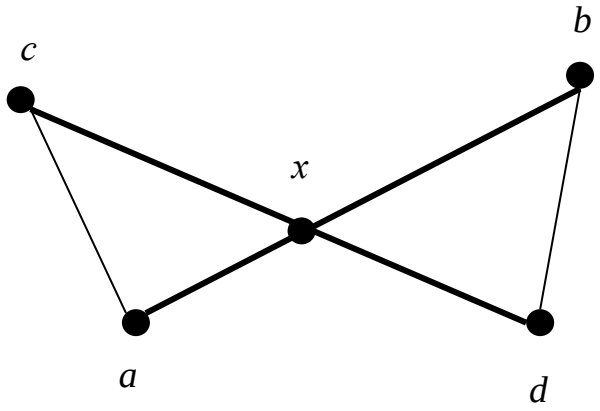


Рисунок 1.31 – Пересечение ребер.

Оценим их суммарную длину, учитывая неравенство треугольника:

$$|ab| + |cd| = |ax| + |xb| + |cx| + |xd| =$$
$$= (|ax| + |cx|) + (|xb| + |xd|) \geq |ca| + |bd|.$$

Существует лучшее решение, в которое входят непересекающиеся ребра $\{c, a\}$ и $\{b, d\}$, а выбор пересекающихся ребер является неперспективным.

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

1) $HO(\overline{x}) = 0$ (т. е. ничем не дополняем текущее состояние).

2) Пусть min_e – длина минимального свободного ребра, тогда учитывая тот факт, что гамильтонов цикл состоит из n ребер, а выбрано пока только k, полагаем

$$HO(\overline{x}) = min_e \cdot (n - k).$$

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

В задаче коммивояжера можно ослабить условие гамильтонова цикла, убирая часть ограничений, что приведет к расширению множества допустимых решений.

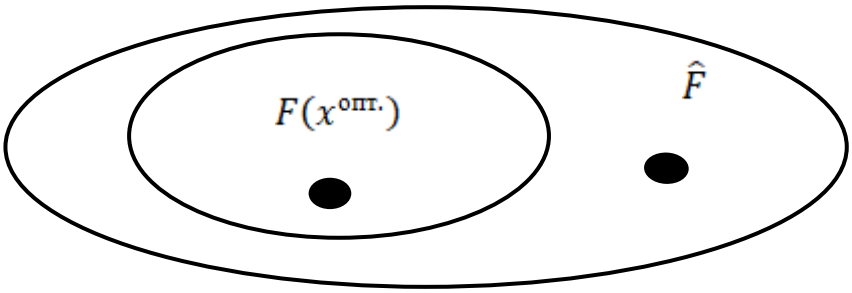


Рисунок 1.32—Ослабление ограниний.

Пусть $\hat{F}$ – решение задачи на расширенной области, тогда, так как $\hat{F} \leq \hat{F}(x^{\text{опт.}})$, можно использовать $\hat{F}$ в качестве нижней оценки $HO(x^{\text{опт.}}) = \hat{F}$ (См. рисунок 1.32).

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

- 1-й способ: уберем ограничение «цикл». Для задачи коммивояжера получаем задачу поиска связного подграфа минимального веса, покрывающего все вершины графа.

Это широко известная задача построения минимального остовного дерева, которую можно решить, алгоритмом Прима или Краскала [10].

- 2-й способ: уберем ограничения «один цикл» и связность. Для задачи коммивояжера получаем задачу построения набора простых циклов минимального суммарного веса, покрывающих все вершины.

Это тоже известная задача построения 2-фактора графа, для которой имеется эффективный точный алгоритм (напомним, что *k*-фактор – регулярный остовный подграф степени *k*, т. е. степени всех вершин остовного подграфа одинаковы и равны *k*).

Пример 1.3. 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 [6].

Рассмотрим в качестве примера шахматное поле, изображенное на рисунке 1.33 (занятые позиции на рисунке имеют серую заливку).

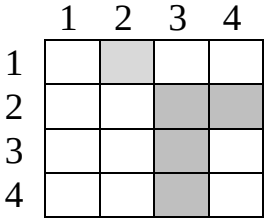


Рисунок 1.33– Пример клеточного поля.

Одна из возможных оптимальных расстановок ладей приведена на рисунке 1.34.

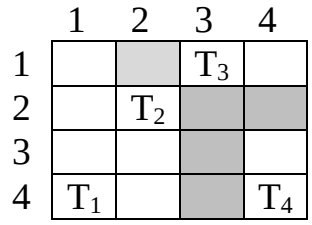


Рисунок 1.34 –Оптимальная расстановка ладей.

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

Рассмотрим возможные отсечения.

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

Построение оценок. Разобьем наше поле на вертикали и горизонтали. Одной вертикали принадлежат соседние клетки одной строки между которыми нет фигур. Аналогично, одной вертикали смежные клетки одного столбца между которыми нет фигур.

Для шахматной доски, приведенной на рисунке 1.33, на рисунке 1.35 показаны семь горизонталей и пять вертикалей:

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

− первая вертикаль состоит из клеток (1, 1), (2, 1), (3, 1) и (4, 1); вторая – (2, 2), (3, 2) и (4, 2); третья – (1, 3); четвертая – (1, 4); пятая – (3, 4), (4, 4).

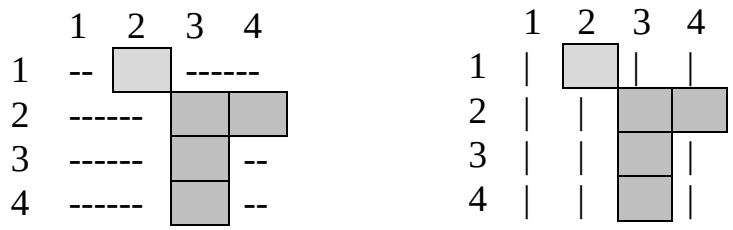


Рисунок 1.35 –Вертикали и горизонтали.

Рассмотрим следующие способы построения нижних оценок.

1. Полагаем нижнюю оценку подзадачи равной 0. Тогда, если на некотором шаге видно, что для частичного решения, соответствующего вершине *x*, количество ладей больше (либо даже не меньше), чем полученный рекорд, то выполняем отсечение вершины *x* по рекорду.

2. Ставим в свободную клетку ладью и считаем число свободных клеток n_0, которые она пробивает (учитываем при подсчете и ту клетку, в которой стоит ладья). Поступаем так для каждой свободной клетки. Пусть n_{max} – максимальное значение среди всех n_0, а s_0 – число всех свободных клеток. Тогда в качестве нижней оценки можно взять величину:

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

Для шахматного поля, приведенного на рисунке 1.33, получаем следующую оценку: $n_{max} = 5, s_0 = 11, \text{НО}(x^{\text{опт.}}) = 2.$

3. Построим двудольный граф. Вершины первой доли – горизонтали шахматной доски, которые не заняты фигурами. Вершины второй доли – вертикали доски, которые не заняты фигурами. Ребро между двумя вершинами двудольного графа проводится в том случае, если соответствующие вертикаль и горизонталь пересекаются (См. рисунок 1.36).

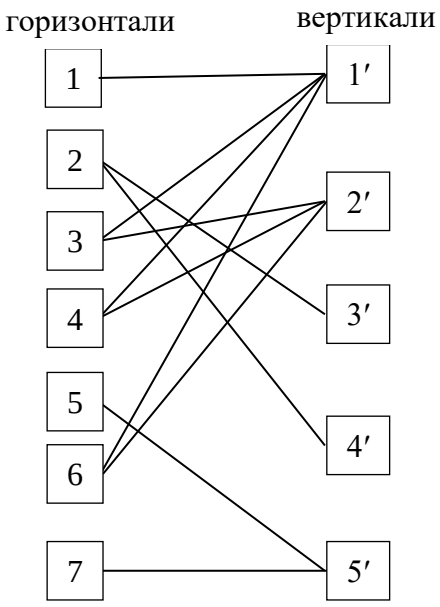


Рисунок 1.36 – Двудольный граф.

Таким образом каждому ребру двудольного графа будет соответствовать позиция шахматной доски (x, y), в которой нет фигур. Если мы ставим ладью в позицию (x, y) (выбираем соответствующее ребро двудольного графа), то этой ладьей будут пробивать все позиции горизонтали x и вертикали y, поэтому существует оптимальное решении, для которого в оптимальной расстановке ладей в каждой горизонтали и вертикали будет стоять не более 1 ладьи.

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

Для двудольного графа, приведенного на рисунке 1.37, мы имеем два максимальных паросочетания:

$$\{\{1,3\}\}$$

и

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

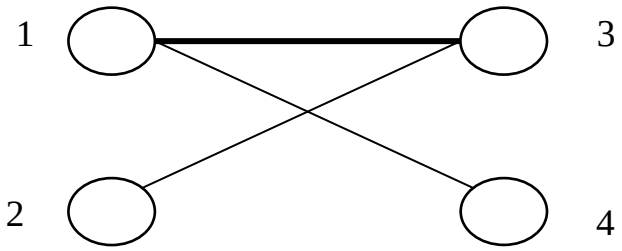


Рисунок 1.37– Паросочетания.

Под наименьшим максимальным паросочетанием будем понимать паросочетание минимальной мощности среди всех максимальных паросочетаний. Для двудольного графа, приведенного на рисунок 1.37, наименьшее максимальное паросочетание: $\{\{1,3\}\}$.

Под наибольшим (по мощности) паросочетанием будем понимать паросочетание с наибольшим числом ребер среди всех паросочетаний графа. Для двудольного графа, приведенного на рисунок 1.37, наибольшее паросочетание: $\{\{1,4\},\{2,3\}\}$. Любое наибольшее паросочетание является максимальным, но обратное верно не всегда.

Точной оценкой для рассматриваемой задачи расстановки минимльного числа ладей так, чтобы все свободные поля пробивались, является мощность наименьшего максимального паросочетания в сгенерированном двудольном графе. Более того, по такому паросочетанию можно восстановить решение: расстановкой являются белые ладьи, соответсвующие ребрам, входящим в наименьшее максимальное паросочетание. Изолированные точки отдельно не рассматриваются, так как они в любом случае обойдутся при обходе в лексикографическом порядке. В этом случае граф будет иметь столько компонент связности, сколько изолированных компонент будет иметь начальная расстановка.

Оценка сначала вычисляется для каждой компоненты связности, после чего неравенства складываются почленно и получается оценка для всего графа.

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

Нетрудно убедиться в том, что множество конечных вершин ребер, которые вошли в максимальное паросочетание *M*, порождают некоторое вершинное покрытие V' графа *G* и мощность этого вершинного покрытия равна $2 \cdot k$.

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

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

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

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

Для двудольного графа существуют алгоритмы, которые позволяют вычислить мощность наименьшего вершинного покрытия за полиномиальное время. Для этого можно воспользоваться теоремой венгерского математика Денеша Кенига (1931 г.), которая говорит о том, что для двудольного графа число вершин наименьшего вершинного покрытия равно числу ребер наибольшего (по мощности) паросочетания. Для нахождения наибольшего паросочетания в двудольном графе $G = (V, E)$ можно использовать, одну из реализаций алгоритма Куна [15], в основе которой лежит алгоритм построения максимального потока в сети (время работы алгоритма – $\text{O}(|V| \cdot |E|)$).

Для двудольного графа приведенного на рисунок 1.36, мощность наибольшего паросочетания равна 4, значит по теореме Кенига мощность нименьшего вершинного покрытия равна также 4 и нижняя оценка для оптимального решения задачи может бытьполучена по формуле:

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

Пример 1.4. На клеточном поле размера $N \times M$, в некоторых позициях которого расставлено *K* черных фигур, необходимо расставить максимальное число белых ладей так, чтобы они не били друг друга.

Решение. Для данной задачи не сложно вычислить за полиномиальное время значение оптимального решения, которое в последующем можно использовать для отсечений по рекорду при построении дерева решений. По аналогии с предыдущей задачей, разобьем шахматную доску на вертикали и горизонтали и построим двудольный граф. Если ладья ставится в некоторую позицию доски (выбрано некоторое ребро двудольного графа), то эта ладья пробивает все позиции соответствующей вертикали и горизонтали, поэтому в оптимальном ре-

шении в свободных клетки этой вертикали и горизонтали не должно стоять других ладей. Следовательно, выбранное множество ребер должно образовывать паросочетание.

Для того, чтобы число ладей было максимальным, паросочетание должно быть наибольшим по мощности.

1.1.5. Минимальные связующие деревья

Рассмотрим проблему построения минимальных связующих деревьев, которая часто называется задачей Штейнера. Покажем, что для решения этой задачи можно использовать динамическое программирование, как общий подход к решению дискретных задач. В этом случае рекомендуется учитывать структурные особенности задачи. При этом динамическое программирование не является полиномиальным алгоритмом.

Исторически имеется несколько формулировок задачи Штейнера.

1. *Задача Штейнера на евклидовой плоскости* формулируется следующим образом: пусть дано множество терминальных вершин на плоскости, требуется найти кратчайшую сеть, соединяющую заданное множество вершин.

Важным условием является то, что для минимизации длины связывающей сети можно добавлять дополнительные точки, называемые *точками Штейнера* (сокр. ТШ).

Кратчайшая сеть, содержащая точки Штейнера, называется *деревом Штейнера* (сокр. ДШ).

Задача поиска деревьев Штейнера играет важную роль в геодезии, проектировании дорожных сетей и т. д. Задача Штейнера является *NP* – полной.

В настоящий момент известно несколько комбинаторных алгоритмов, таких как алгоритмы Мелзака и Кокейна, Дрейфуса и Вагнера дающих оптимальное решение задачи Штейнера за экспоненциальное время. Очевидно, что подобные алгоритмы плохо подходят для практического применения для больших объемов входных данных. В этой связи существует большое количество алгоритмов построения деревьев Штейнера, находящих приближенные решения.

Курант и Робине наряду с формулировкой задачи для произвольного n привели два важных свойства:

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

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

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

Впервые конечная процедура построения МДШ была предложена Мелзаком в 1961 г. Ключевым здесь является подмеченный им факт о том, что если две точки *D* и *B* соединены ребрами с точкой Штейнера *P*, то третье ребро, ин-

цидентное *P*, проходит через вершину *C*, которая образует равносторонний треугольник с вершинами *D* и *B* и лежит в разных с точкой P полуплоскостях, образованных прямой, проходящей через *D* и *B*.

Таким образом, если МДШ с *n* вершинами таково, что две его вершины D и B непосредственно соединены ребрами с ТШ, то заменой пары *D* и *B* на *C* осуществляется редукция к задаче Штейнера с $n-1$ вершинами.

Так как существует два способа выбора точки *C* и

$$\frac{n \cdot (n-1)}{2}$$

способов выбора пары *D, B*, то отсюда с учетом того факта, что число ТШ не превосходит $n-2$, и получается конечная процедура поиска МДШ.

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

Остается заметить, что если МДШ не является ПДШ, то существует его декомпозиция на ПДШ меньшей размерности.

Таким образом, непосредственное использование перечисленных фактов приводит к получению экспоненциального алгоритма построения МДШ. Дальнейшие исследования по построению точных алгоритмов решения задачи проведены Кокинэ.

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

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

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

Пусть дано два множества действительных чисел

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

и

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

Вершинами графа-решетки $G = (V, E)$ являются точки плоскости, первая координата которых принадлежит множеству *X*, а вторая – *Y*.

Упорядоченные наборы первых и вторых координат терминальных вершин образуют множества *X* и *Y*, получается граф-решетка, которая обозначается через $G(A)$.

Ханан доказал, что МДШ в ЗШМП является подграфом $G(A)$. Таким образом, все результаты, относящиеся к задаче Штейнера на графах, применимы и к

рассматриваемой постановке. Однако, как в силу «исторических» причин, так и с точки зрения особенностей, связанных с приложениями, данная задача традиционно выделяется в отдельную задачу.

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

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

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

Алгоритм Дрейфуса-Вагнера основан на использовании двух рекуррентных соотношений.

Пусть $ST(X \cup \{v\}), v \in V \backslash X$, соответствует длине дерева Штейнера на множестве вершин $X \cup \{v\}$, а $ST_v\ (X \cup \{v\})$ соответствует длине дерева Штейнера на множестве вершин $X \cup \{v\}$, при условии, что степень вершины v не меньше 2.

Понятно, что

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

Нетрудно проверить, что

1) $ST_v(X \cup \{v\}) = \min\big(ST(X' \cup \{v\}) + ST(X \backslash X' \cup \{v\})\big)$,

где минимум берется по всем возможным разбиения множества X на подмножества X' и $X \backslash X', v \in V \backslash X$.

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

$\mathrm{Min}\{p(v, w) + ST_w\ (X)\}$, если $w \in V \backslash X\}$.

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

Трудоемкость алгоритма оценивается величиной

$$n \cdot 3^k,$$

где k соответствует мощности дерева Штейнера.

1.2. Эвристики и приближенные алгоритмы

1.2.1. Основные понятия

Задачи, принадлежащие классу *NP*, пока не имеют эффективного решения. 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 (метод ветвей и границ и динамическое программирование) [8,9].

Предположим, что задана задача максимизации:

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

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

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

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

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

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

Если алгоритм A – точный, то получим строгое равенство

$F(x^{\text{опт.}}) = F(x^A)$.

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

$$|F(x^{\text{опт.}}) - F(x^A)|.$$

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

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

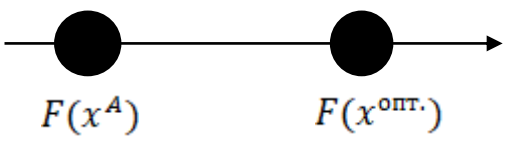
Иногда для оценки качества алгоритма A рассматривают функцию:

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

где inf берется по всем возможным наборам входных данных.

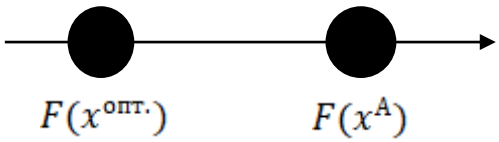
Определение 2.3. Под гарантированной оценкой точности решения, получаемого при помощи алгоритма A, понимаем:

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



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

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



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

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

– либо абсолютная (относительная) погрешность, не зависящая от входных данных;

– либо гарантированная оценка точности.

Для приближенных алгоритмов существует следующая их классификация:

1) *субоптимальный приближенный алгоритм* (такой приближенный алгоритм, для которого величина Δ_A отлична от 0 (для задач максимизации) или от ∞ (для задач минимизации));

2) 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 и величины $1/\varepsilon$);

3) *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 и величины $1/\varepsilon$).

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

Следует отметить, что в силу того, что значение $F(x^{\text{опт.}})$ не известно, как правило, оценивается величина

$$\frac{F(x^A)}{\text{BO}(I)}$$

для задачи максимизации и

$$\frac{F(x^A)}{\text{HO}(I)}$$

для задачи минимизации, где $\text{BO}(I)\big(\text{HO}(I)\big)$ – верхняя (нижняя) оценка оптимального решения для индивидуальной задачи (на определенном наборе входных данных).

Действительно, в этом случае для задачи максимизации

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

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

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

в силу $\text{НО}(I) \leq F(x^{\text{опт}})$ 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, которые наиболее близки к оптимальному решению.

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

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

Тогда

$F(x^{\text{опт.}}) \leq k.$

Предположим, что существует некоторый алгоритм А, для которого

$$F(x^A) \geq k - 1.$$

Тогда А – приближенный алгоритм с абсолютной погрешностью

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

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

1.2.2. Жадные (градиентные) алгоритмы

Алгоритмы данного типа являются наиболее популярной эвристикой для большого количества прикладных задач, и имеют широкое практическое распространение. Поэтому важнейшими вопросами являюся качество (точность), и условия, при которых жадные алгоритмы (англ. g*reedy algorithm*) дают точные решения. В этом разделе будут рассмотрены наиболее популярные алгоритмы градиентного типа для различных модельных задач дискретной оптимизации, будут даны оценки их точности, а также будет рассмотрен вопрос, когда алгоритмы градиентного типа строят точные решения.

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 2.1. 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, инцидентным некоторой вершине.

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

Пример 2.2. Для графа, приведенного на рисунке 1.38, решить задачу

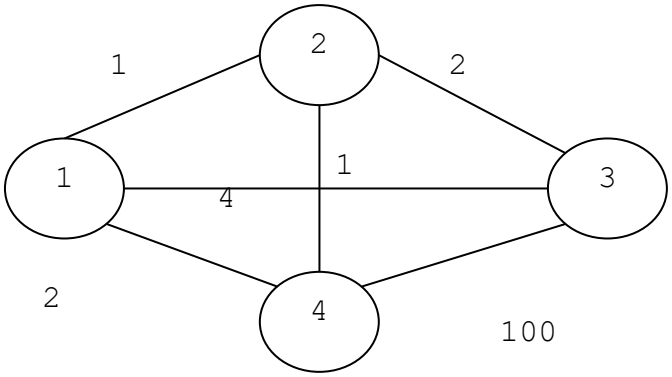


Рисунок 1.38 – Изображение графа.

коммивояжера.

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, так как вершина 2 в этом случае была бы индидентна трем ребрам.

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.

Жадный алгоритм сгенерировал путь $1 \to 2 \to 4 \to 3 \to 1$ стоимости 106 (на

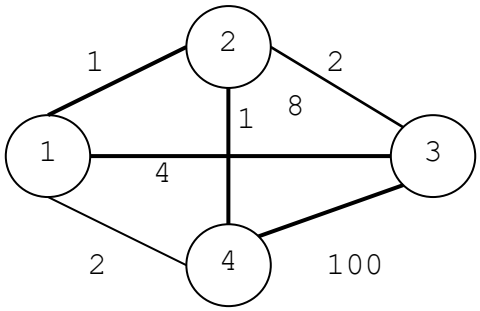


Рисунок 1.39 – Построенный цикл.

рисунке 1.39 выбранные ребра выделены).

Это решение не является оптимальным: есть, по крайней мере, один более короткий путь $1 \to 2 \to 3 \to 4 \to 1$ стоимости 105.

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 2.2. 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 помещается, то отбрасываем его и переходим к рассмотрению следующего объекта списка.

Описанный процесс продолжаем до тех пор, пока рюкзак не заполнится или будет исчерпан упорядоченный список объектов.

Пример 2.3. Упаковать рюкзак объемом $W = 80$ объектами, сведения о которых приведены в таблице 1.1.

Таблица 1.1 – Данные об объектах.

(a_i, c_i)	(20,40)	(20,80)	(20,50)	(15,45)	(30,105)	(35,35)	(20,10)	(10,45)
c_i/a_i	2	4	2.5	3	3.5	1	0.5	4.5
$\frac{c_i}{a_i} \geq \frac{c_{i+1}}{a_{i+1}}$	(10,45) 1	(20,80) 2	(30,105) 3	(15, 45) 4	(20,50) 5	(20, 40) 6	(35,35) 7	(20,10) 8

Так как емкость рюкзака равен 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.

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 2.3. 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 1.2. Предельная сумма объемов предметов множества равна *L*=55.

Необходимо найти такой набор предметов, сумма размеров которых была бы наиболее близка к *L*.

Таблица 1.2 – Объемы предметов.

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 *L*=55, то мы сможем добавить в него в соответствии с нашим алгоритмом три 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 2.4. 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 двухэлементных 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 предметов по убыванию объемов следующая:

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

Предельная сумма объемов предметов множества равна L=55.

Выполним последовательность шагов жадного алгоритма 2.4.

Несложно убедиться, что уже на втором проходе будет построено оптимальное решение задачи (Таблица 1.3).

Таблица 1.3 – Оптимальное решение.

Номер прохода	Размер начального подмножества	Добавляемые элементы	Сумма элементов сгенерированного множества
0	0	27, 22, 1	50
1	{27} {22} {14} {11} {7}	22,1 27, 1 27, 11, 1 27, 14, 1 27, 14, 1 27, 22	50 50 53 53 49 50

	{1}		
2	{27, 22} {27, 14} {27, 11} {27, 7} {27, 1} {22, 14} {22, 11} {22, 7} 22, 1} {14, 11} {14, 7} {14, 1} {11, 7} {11, 1} {7, 1}	1 11, 1 14, 1 14, 1 22 11, 7, 1 14, 7, 1 14, 11, 1 27 27 1 27, 1 27, 11 27, 1 27, 14 27, 14	50 53 53 49 50 55 ! оптимум 55 55 50 53 49 53 46 53 49

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 2.5. 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 c, 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 рисунке 1.40.

Раскраска вершин проводилась в соответствии с приведенным алгоритмом. Вершины графа просматривались в порядке возрастания их номеров. Как видно из рисунка 1.40, минимальное число потребовавшихся красок равно 3.

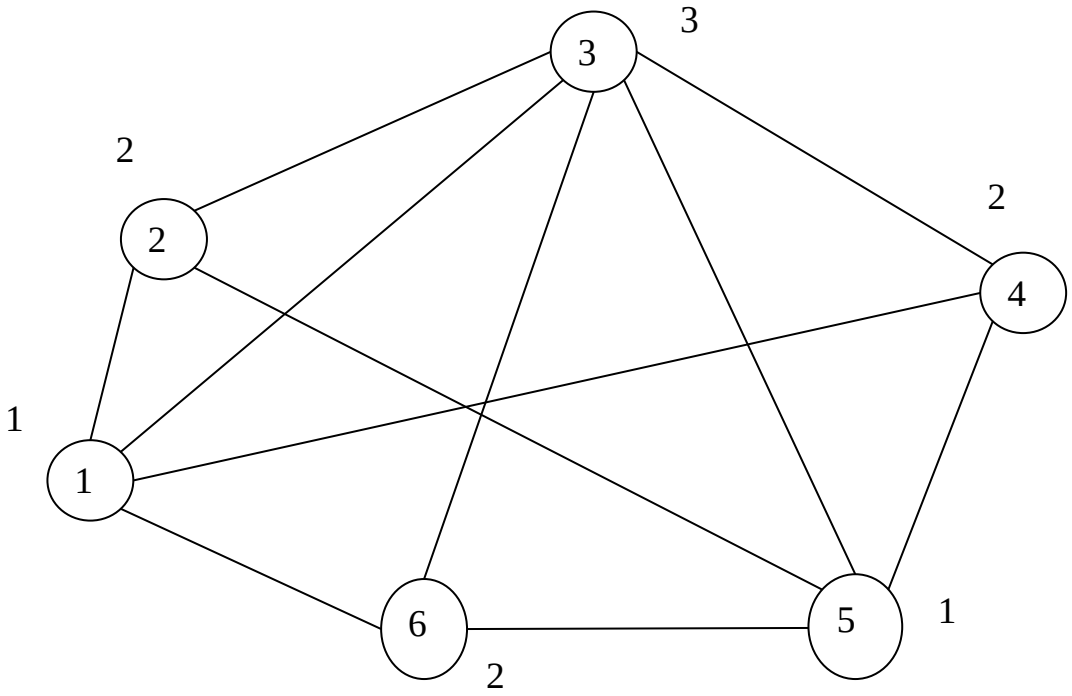


Рисунок 1.40 – Изображение графа.

1.2.3. Локальный поиск

Для любой задачи алгоритм локального поиска начинает свою работу от некоторого исходного решения (найденного каким-то другим алгоритмом или выбранного случайно) и представляет собой итеративный процесс. На каждом шаге локального спуска происходит переход от текущего решения к соседнему решению с лучшим значением целевой функции до тех пор, пока не будет достигнут локальный оптимум. Соседнее решение не обязательно должно быть наилучшим в окрестности, а вот критерий оценки решения не должен меняться по ходу этого итеративного процесса [8].

Таким образом, для любого решения s должно быть задано некоторое множество $N(s)$ соседних решений, которое называется окрестностью s. Грубо говоря, окрестность $N(s)$ состоит из решений незначительно отличающихся от s. Семейство всех окрестностей называется структурой окрестностей. Дадим формальные определения.

Структурой окрестностей для индивидуальной задачи x оптимизационной проблемы V называется отображение N, которое каждому допустимому решению s задачи ставит в соответствие множество решений $N(s)$. Структура

окрестностей может быть достаточно сложной и отношение соседства не всегда симметрично, то есть s может быть соседом s', но s' может не быть соседом s.

Для данного решения s поиск более хорошего решения в окрестности $N(s)$ может быть осуществлен двумя способами:

− выбор наилучшего решения во всей окрестности;

− выбор первого лучшего, когда окрестность каким-либо образом перебирается, и этот перебор сразу же прекращается, как только будет найдено более хорошее решение.

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

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

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

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

Если же это свойство не выполняется, то стараются получить хотя бы свойство вполне связности, когда из любой вершины существует путь в вершину с минимальным значением целевой функции. Если же и этого свойства нет, то мы теряем уверенность в достижении глобального оптимума локальными методами и должны либо ограничиться локальными оптимумами, либо переопределить функцию окрестности.

Переходу от одного текущего решения к другому в графе соседства соответствует дуга. Поэтому локальному поиску для задачи x соответствует путь в графе соседства, который заканчивается в локально-оптимальной вершине s_r свойством, что для любой вершины s, такой, что $(s_r, s) \in U$ значение целевой функции не лучше, чем в вершине s_r.

В худшем случае локальный поиск может исследовать экспоненциальное число решений прежде, чем достигнет локального оптимума. По этой причине, часто в эвристике локального поиска используется правило остановки.

Алгоритмы локального поиска основаны на следующей общей схеме.

Схема локального поиска

`Input`: Индивидуальная задача x;

Output: Решение s;

begin

s : = исходное допустимое решение s_0;

(* N – структура окрестностей *)

repeat

Выбрать еще не рассмотренное решение $s' \in N(s)$;

Если $F(s')$ лучше $F(s)$ то $s := s'$;

until все решения из $N(s)$ не будут рассмотрены;

return s

end.

Отметим, что для данной индивидуальной задачи поведение этого алгоритма зависит от следующих факторов:

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

2. Начального решения s_0. Его можно находить с помощью алгоритма (например, конструктивной эвристикой), который выдает хорошее допустимое решение, или с помощью процедуры случайной генерации.

3. Стратегии выбора новых решений. Например, все решения из $N(s)$ просматриваются, выбирается лучшее из них, и сравнивается с s. Это означает, что если s нелокально-оптимально, то осуществляется переход к наилучшему соседу. Или, наоборот, осуществляется переход к первому лучшему решению, найденному в окрестности.

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

Ниже приводится несколько примеров таких задач. Данные примеры указывают на важность локального поиска при построении оптимизационных алгоритмов и достаточно общий характер этого подхода.

1. Линейное программирование. Геометрически алгоритм симплексметода можно интерпретировать как движение по вершинам многогранника допустимой области. Вершина не является оптимальной, если и только если существует смежная с ней вершина с меньшим значением целевой функции. Алгебраически, предполагая невырожденность задачи, базисное допустимое решение не

является оптимальным, если и только если оно может быть улучшено локальным изменением базиса, т. е. заменой одной базисной переменной на небазисную. Получающаяся таким образом окрестность является точной и имеет полиномиальную мощность.

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

3. Наибольшее паросочетание. Паросочетание не является наибольшим, если и только если существует увеличивающий путь. Два паросочетания называют соседними, если их симметрическая разность образует путь. Определенная таким образом окрестность является точной и имеет полиномиальную мощность.

Аналогичные утверждения справедливы для взвешенных паросочетаний, совершенных паросочетаний минимального веса, задач о максимальном потоке и потоке минимальной стоимости.

На каждом шаге локального спуска структура окрестностей N задает множество возможных направлений движения. Очень часто это множество состоит из нескольких элементов и имеется определенная свобода в выборе следующего решения. Правило выбора может оказать существенное влияние на трудоемкость алгоритма и результат его работы. Например, в задаче о максимальном потоке алгоритм Форда-Фалкерсона (который тоже можно рассматривать как вариант локального спуска) имеет полиномиальную временную сложность при выборе наименьшего по количеству дуг пути для увеличения потока и экспоненциальную временную сложность без гарантии получить глобальный оптимум при произвольном выборе пути. Таким образом, при разработке алгоритмов локального поиска важно не только правильно определить окрестность, но и верно задать правило выбора направления спуска. Интуитивно кажется, что в окрестности надо брать элемент с наименьшим значением целевой функции. Однако, как мы увидим ниже, разумным оказывается не только такой выбор, но и движение в «абсурдном» направлении, когда несколько шагов с ухудшением могут привести (и часто приводят) к лучшему локальному оптимуму.

При выборе окрестности хочется иметь множество $N(s)$ как можно меньшей мощности, чтобы сократить трудоемкость одного шага. С другой стороны, более широкая окрестность, вообще говоря, приводит к лучшему локальному оптимуму. Поэтому при создании алгоритмов каждый раз приходится искать оптимальный баланс между этими противоречивыми факторами. Ясных принципов разрешения этого противоречия на сегодняшний день не известно, и для каждой задачи этот вопрос решается индивидуально.

Сложность локального поиска

Анализ вычислительной сложности локального поиска в последние годы интенсивно ведется в двух направлениях: эмпирическом и теоретическом. Как ни странно, но эти направления дают существенно разные оценки возможностям локального поиска.

Эмпирические результаты. Для многих *NP*-трудных задач локальный поиск позволяет находить решения, близкие по целевой функции к глобальному оптимуму. Трудоемкость алгоритмов часто оказывается полиномиальной, причем степень полинома достаточно мала.

Так для задачи о разбиении множества вершин графа на две равные части разработаны алгоритмы локального поиска со средней трудоемкостью $O(n \cdot \log n)$, n – число вершин, которые дают всего несколько процентов погрешности.

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

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

Напомним, что задача коммивояжера состоит в нахождении минимального по длине гамильтонова цикла в полном взвешенном ориентированном (или неориентированном) графе с n вершинами. Для удобства ниже будет рассматриваться только неориентированные графы с симметричной целочисленной неотрицательной матрицей расстояний $w(i,j),\ 1 < i,\ j < n$.

Рассмотрим 2-оптимальную (*сокр*. 2-опт.) эвристику для задачи о коммивояжере. Этот метод основан на следующей 2-обменной окрестности: для любого цикла C 2-обменная окрестность $N(C)$ это множество всех циклов C', которые могут быть получены из C после удаления двух ребер $\{x,y\}$ и $\{v,z\}$, и добавления двух новых ребер $\{x,v\}$ и $\{z,y\}$. Процесс продолжается до тех пор, пока не происходит улучшение целевой функции.

Алгоритм 2-опт. можно модифицировать в алгоритм локального поиска с большими окрестностями. Например, 3-опт. эвристика основывается на 3-обменных окрестностях. Для тура C его 3-обменная окрестность $N(C)$ это мно-

жество всех циклов t', которые могут быть получены из C после замены не более чем трех ребер.

Алгоритм 3-опт. эвристика имеет более хорошее приближение, но хуже по трудоемкости.

Однако можно построить примеры, для которых эти эвристики находят решения со стоимостью, далекой от оптимальной. Как уже говорилось раньше, результат, полученный на выходе эвристики, сильно зависит от выбора начального цикла C.

Следующая теорема показывает, что существуют такие начальные циклы, при использовании которых эвристика дает решение хуже оптимального в произвольное число раз.

Теорема 2.1. Для любых

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

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

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

1.2.4. Приближенные алгоритмы с гарантированной оценкой

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

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

Предположим, что $F^* > 0$. Пусть приближенный алгоритм H находит элемент множества X со значением F^H целевого функционала. Введем в рассмотрение величину $\Delta_H = F^H/F^*$. 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 $P//\sum w_i \cdot C_i$. 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 [13,14,16].

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

Перенумеруем требования в порядке SWPT так, что

$$p_1/w_1 \le \dots \le p_n/w_n.$$

Организуем m подмножеств $N_1, N_2, \dots, N_m$, полагая вначале

$$N_1 = N_2 = \dots = N_m = \emptyset.$$

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

Временная сложность алгоритма *A* равна $O(n \log n)$, так как значения $P_1, P_2, \dots, P_m$ можно хранить в виде древовидной структуры данных *куча* или сбалансированного поискового 2−3-дерева [11, 4].

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

Очевидно, что тогда

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

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

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

.

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

Лемма 2.1. Пусть $b_1, b_2, \dots, b_n$ − действительные числа. Тогда

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

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

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

Теорема 2.2. Для любого расписания справедливо

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

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 $p_1/w_1 = \dots = p_q/w_q > p_{q+1}/w_{q+1}$.

Положим $\gamma = \frac{w_{q+1}\cdot p_1}{(p_{q+1}\cdot w_1)}$ и рассмотрим вспомогательную задачу, в которой веса обозначены $\overline{w}_j$ и для первых q требований $\overline{w}_j = \gamma\cdot w_j$, $j = 1,\dots,q$. 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

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

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

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

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

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

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 $\left(\overline{X}^q + \delta(\overline{X})\right)\left(\overline{Y} + \gamma\delta(\overline{Y})\right) > \left(\overline{X}^q + \gamma\delta(\overline{X})\right)\left(\overline{Y}^q + \delta(\overline{Y})\right)$.

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

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

Сейчас нетрудно получить оценку сверху величины Δ_A:

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

поскольку $F_n^* \leq F^*$.

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 как R_l, называется разница между d и суммарной длительностью 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 $p_j \le 1$, $j = 1, \dots, n$.

Докажем ряд вспомогательных утверждений для алгоритмов B_1 и B_2.

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

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

Далее будем рассматривать расписания, построенные одним из алгоритмов B_1 либо B_2.

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

Не ограничивая общности, предположим, что $p_1 \ge \dots \ge p_j$.

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

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

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

Возможны следующие варианты:

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

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

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

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

При вариантах 1) и 2) имеем соответственно

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

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

При варианте 3) имеем

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

При варианте 4) имеем

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

Для прибора l обозначим через $\psi(h)$ максимальный резерв времени для приборов $h = 1, \dots, l-1$, 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 $R_h < {}^1/_2$, $h = 1, \dots, l-1$ и длительность обслуживания первого назначенного на прибор l требования больше, чем Q_l. Обозначим эту длительность через τ. Если $\tau > \frac{1}{2}$, то лемма доказана. Если $\tau \le \frac{1}{2}$, то $R_l \ge {}^1/_2$, и длительность второго назначаемого на прибор l требования больше, чем Q_l. Обозначим эту длительность через τ'. Если $\tau + \tau' \le \frac{1}{2}$, то лемма 2.5 доказана. Если $\tau + \tau' > \frac{1}{2}$, 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 $\tau > Q_l \ge \frac{1}{2}$, что и требуется.

Лемма 2.6. Пусть на прибор l назначены требования $1, \dots, j$ *и* $Q_l < {}^1/_2$. Если $\sum_{i=1}^{j} p_i \ge 1 - Q_l$, то $\sum_{i=1}^{j}\psi(p_i) \ge 1$.

Доказательство. Не ограничивая общности, предположим, что $p_1 \ge \dots \ge p_j$. Если $p_1 > {}^1/_2$, то $\psi(p_1) = 1$ и справедливость утверждения леммы доказана.

Пусть $p_1 \le {}^1/_2$. Поскольку $\sum_{i=1}^{j} p_i \ge 1 - Q_l > {}^1/_2$, имеем $l \ge 2$. Из леммы 2.5 и $p_1 \le {}^1/_2$ следует, что $p_2 > Q_l$.

Возможны варианты:

1) $Q_l \in \left(0, {}^1/_6\right]$,

2) $Q_l \in \left({}^1/_6, {}^1/_3\right]$,

3) $Q_l \in \left({}^1/_3, {}^1/_2\right]$.

В условиях первого варианта

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

Пусть в условиях второго варианта $j = 2$. Поскольку $p_1 + p_2 > 1/2$ имеем либо $p_2 \geq 1/3$, либо $p_1 \geq 1/3$ и $Q_l < p_2 < 1/3$. В первом случае $\psi(p_1)+\psi(p_2)=\frac{6}{5}p_1+\frac{9}{5}p_2=\frac{6}{5}(p_1+p_2)+\frac{3}{5}p_2>\frac{6}{5}(1-Q_l)+\frac{3}{5}Q_l=\frac{6}{5}-\frac{3}{5}Q_l \geq 1$, поскольку $Q_l \leq \frac{1}{3}$.

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 $1,\dots,j$ и $Q_l < \frac{1}{2}$. Если $\sum_{i=1}^{j}\psi(p_i) = 1-\alpha$ *и* $\alpha > 0$, то $p_i \leq 1/2$, $i = 1,\cdots,j$, и $\sum_{i=1}^{j} p_i \leq 1 - Q_l - \frac{5}{9}\alpha$ при $j \geq 2$.

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

Пусть $l \geq 2$. Поскольку $p_1 \leq 1/2$, то в силу леммы 2.6 имеем $p_1 \geq p_2 > Q_l$, $\sum_{i=1}^{j} p_i = 1 - Q_l - \gamma$, где $\gamma > 0$.

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

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

Для любых x и y $0 \leq x < y \leq 1/2$ выполняется соотноешине $\psi(y) \leq \psi(x) + \frac{9}{5}(y-x)$. Поэтому $\psi(\delta_1) + \psi(\delta_2) \leq \psi(p_1) + \psi(p_2) + \frac{9}{5}\cdot\gamma$. Отсюда получаем $\sum_{i=1}^{j}\psi(p_i) + \frac{9}{5}\cdot\gamma \geq 1$.

Следовательно, $\frac{9}{5}\cdot\gamma \geq \alpha$ и $\gamma \geq \frac{5}{9}\cdot\alpha$. Лемма 2.7 доказана.

Теорема 2.3. Справедливы неравенства

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

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

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

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

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

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

Для любого $i \in N_{l'_h}$, $h = 1, \dots, k$, выполняется $p_i \leq {}^1/_2$, поскольку в противном случае $\psi\left(N_{l'_h}\right) \geq 1$.

Отсюда следует, что $Q_{l'_h} < {}^1/_2$, $h = 1, \dots, k$.

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

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

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

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

Таким образом, $\sum_{h=1}^{k-1} \alpha_h \leq \frac{9}{5} \sum_{h=1}^{k-1} \left(Q_{l'_{h+1}} - Q_{l'_h}\right) = \frac{9}{5}\left(Q_{l'_k} - Q_{l'_1}\right) < \frac{9}{10}$, поскольку $Q_{l'_h} < {}^1/_2$, $h = 1, \dots, k$. Отсюда, а также из $\alpha_k < 1$ получаем $\sum_{h=1}^{k} \alpha_h < 2$. Как нетрудно убедиться, $\psi(N_l) \geq 1$, 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 $m_1 \leq \sum_{l \in \Theta} \psi(N_l) + k \leq \sum_{l \in \Theta} \psi(N_l) + \sum_{h=1}^{k} \psi\left(N_{l'_h}\right) + \sum_{h=1}^{k} \alpha_h < \Psi + 2 \leq \frac{17}{10} m^* + 2$. Аналогично, $m_2 < \frac{17}{10} m^* + 2$. Теорема 2.3 доказана.

Из Теоремы 2.3 следует, что $\Delta_H \leq \frac{17}{10} + \frac{2}{m^*}$ для алгоритма $H \in \{B_1, B_2\}$. Поскольку $m^* \geq \lceil P \rceil$, где $P = \sum_{j=1}^{n} p_j$, то $\Delta_H \leq \frac{17}{10} + \frac{2}{\lceil P \rceil}$.

Напомним, что здесь предполагается $d = 1$ и $p_j \leq 1$, $j = 1, \dots, n$.

Если d и $p_j \leq 1$, $j = 1, \dots, n$ произвольные числа, то

$$\Delta_H \leq \frac{17}{10} + {}^2\Big/\left[{}^P/_d\right].$$

Перейдем к рассмотрению алгоритмов B_3 и B_4.

Теорема 2.4. Справедливы неравенства

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

Доказательство. Схема доказательства теоремы 2.4 аналогична схеме доказательства теоремы 2.3. Оно сводится к построению такой весовой функции, аналогичной $\psi(x)$, что суммарный вес требований, назначенных на один и тот же прибор, не превосходит некоторой константы $C \ge 1$ 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 $p_j \le 1$, $j = 1, \dots, n$ − произвольные числа, из теоремы 2.3 следует, что $\Delta_H \le \frac{11}{9} + \frac{4}{\lceil {}^{P}/_{d} \rceil}$, $H \in \{B_3, B_4\}$.

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. Так как суммарная длительность уже распределенных работ равна $p_1 + \cdots + p_k$, то существует процессор, загрузка которого не превышает величину

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

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:

$$LB \ge \frac{p_1 + \cdots + p_n}{m},$$

причем $LB \ge Z_k$ для любого k. Поэтому, учитывая предыдущие соотношения, на каждом шаге новая загрузка процессора, на который назначена очередная работа, не превышает величины

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

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 $p_1, \dots, p_n$ строит решение, в котором загрузка максимального процессора не превосходит величины $LB + (m-1) \cdot LB/m = (2 - 1/m) \cdot LB$.

Задача коммивояжера

В общем случае для задачи коммивояжера построение полиномиального алгоритма с любой гарантированной оценкой точности будет означать построение точного полиномиального алгоритма.

Поэтому рассматривают версии задач, когда возможно построение приближенного алгоритма с гарантированной оценкой.

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

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

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

Понятно, что в любом метрическом пространстве это соотношение выполняется. Например, когда вершинам соответствуют точки на плоскости, а расстояния между вершинами равны расстоянию между соответствующими точками в какой-то метрике, например Евклидовой. Эта версия будет рассмотрена в разделе 2.5.

1.2.5. Методики построения приближенных алгоритмов с гарантированной оценкой точности

Выше мы ограничивались анализом алгоритмов градиентного типа.

В данном разделе будут приведены некоторые техники, которые позволяют строить и оценивать приближенные алгоритмы с гарантированной оценкой.

Метод релаксации является одним из самых основных методов разработки приближенных алгоритмов.

Идея состоит в том, чтобы построить решение данной проблемы P, используя оптимальное или неоптимальное решение другой задачи P'.

При таком подходе существуют две возможность.

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

2.Этот подход в некотором смысле противоположен предыдущему и подразумевает расширение пространства возможных решений за счет включения ранее недопустимых решений. В этом случае необходимо решить более общую задачу P''. Алгоритм аппроксимации для P решает задачу P'' (оптимально или неоптимально), а затем преобразует в такое решение в то, которое допустимо для *P*.

Тесно связан с методом релаксации подход, обеспечивающий преобразование релаксации к допустимому решению исходной задачи. Идея преобразования состоит в том, чтобы преобразовать недопустимое решение релаксационной задачи к допустимому решению исходной задачи. Существуют различные методы преобразования.

Алгоритмы аппроксимации, основанные на методологии линейного программирования, подпадают под эту категорию.

Проиллюстрируем данный метод на двух классических задачах: построение минимального дерева Штейнера, задача коммивояжера.

Дерево Штейнера и задачи коммивояжера (TSP) являются классическими задачами комбинаторной оптимизации.

Алгоритмы, которые мы обсуждаем для TSP, являются одними из наиболее известных алгоритмов аппроксимации для любой задачи.

Задача о дереве Штейнера является классической задачей комбинаторной оптимизации. Давайте определим задачу о дереве Штайнера над полным метрическим графом с ребрами $G = (V, E, w)$, где V − множество n вершин, E – множество $m = (n^2 - n)/2$ ребер и w: $E \to R^+$ весовая функция для ребер. Поскольку граф метрический, набор весов удовлетворяет неравенству треугольника, т. е. для каждой пары вершин $i,\ j,\ w(i,j)$ меньше или равно сумме веса ребер на любом пути от вершины *i* до вершины *j*.

Задача о дереве Штейнера состоит из метрического графа $G = (V, E)$ и подмножества вершин $T \subseteq V$. Проблема состоит в том, чтобы найти дерево, включающее все вершины в T плюс некоторые другие вершины в графе, чтобы сумма веса ребер в дереве была наименее возможной.Проблема дерева Штейнера в *NP*-сложной задаче.При таком подходе можно решать аппроксимационную задачу построения минимального остовного дерева на множестве вершин T в модифицированном графе, в котором веса ребер между соответствующими вершинами соответствуют длинам кратчайших путей между вершинами в исходном графе. В этом случае используется сведение исходной задачи Штейнера к более частной задаче построения минимального остовного дерева. Данный подход используется при построении аппроксимиционных схем (PTAS_FPTAS).

Продемонстрируем это на примере задачи о рюкзаке, которая формулируется следующим образом

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

$$\sum a_i \cdot x_i \leq B,$$

$x_i = 0 \cup 1.$

Вместо решения исходной задачи, для решения которой можно использовать динамическое программирование с оценкой трудоемкости $O(n \cdot \min(B, c \cdot x^{\text{опт}}))$. Решается огрубленная задача, в которой новые коэффициенты целевой функции c_i заменяются на

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

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

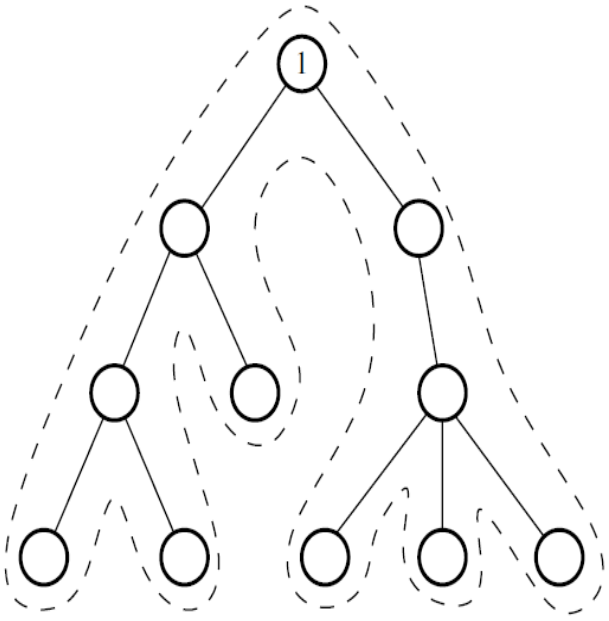
После этого решается огрубленная задача с помощью динамического программирования, но при этом трудоемкость становится $O\left(n \cdot \min\left(B, \frac{c \cdot x^{\text{опт.}}}{k}\right)\right)$.

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

Лучшие алгоритмы для метрической задачи коммивояжера тоже в качестве алгоритма аппроксимации используют минимальное остовное дерево, но в этом случае решается более общая задача, так как ослабляются ограничения.

Широко известен алгоритм аппроксимации «связующего дерева с двойным минимальным весом» (DMWST). Формальный подход заключается в построении Эйлерова цикла в мультиграфе (графе с несколькими ребрами между вершинами), состоящем из двух копий ребер оставного дерева. Эйлеров цикл – это путь, который начинается и заканчивается в одной и той же вершине и посещает каждое ребро мультиграфа один раз.

Оптимальное решение задачи коммивояжера имеет вес, превышающий вес минимального остовного дерева, из чего следует, что вес полученного тура, не более чем в два раза превышает значение оптимального решения.



Алгоритм аппроксимации, который мы называем DMWST, строит остовное дерево минимального веса, копирует все ребра в дереве, а затем генерирует Эйлеров цикл с весом, равным удвоенному весу связующего дерева минимального веса.

Допустимое решение (гамильтонов цикл) получится из эйлерова цикла путем вычеркивания повторяющихся вершин, кроме первой и последней.

В силу условия неравенства треугольника такая трансформация не приводит к увеличению длины получаемого допустимого решения относительно длины Эйлерова цикла.

Сердюков и Христофидес модифицировали вышеуказанный подход таким образом, чтобы созданные туры имели общий вес в пределах в 1,5 раза больше веса оптимального тура. Сначала обратим внимание, что существует множество различных способов преобразования связующего дерева в мультиграф Эйлера. Идея состоит в том, чтобы преобразовать связующее дерево в мультиграф Эйлера с помощью добавление наименьшего количества ребер с наименьшим возможным общим весом. Таким набором может служить совершенное паросочетание минимального веса, которое покрывает вершины нечетной степени в минимальном остовном дереве. В этом случае суммарный вес ребер мультиграфе, построенном по алгоритму Сердюкова-Христофидеса, не более чем в 1,5 раза больше веса оптимального тура.

Время работы алгоритма Христофидеса $O(n^3)$, и в нем преобладает время, необходимое для построения совершенного паросочетания с минимальным весом.

Этот подход аналогичен подходу, применяемому Эдмондсом и Джонсоном для задачи китайского почтальона. Задача китайского почтальона состоит в том, чтобы построить цикл с минимальным весом, возможно, с повторяющимися ребрами, который содержит каждое ребро в графе по крайней мере один раз [10].

В настоящее время лучший алгоритм для решения этой проблемы занимает $O(n^3)$ времени, и он использует кратчайшие пути и алгоритм построения совершенного паросочетания минимального веса.

1.2.6. Оптимальность градиентного алгоритма

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

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

Элементы семейства J называются *независимыми*.

Пример 2.7. Пусть задано конечное множество

$$E = \{1,2,3,4\}$$

и семейства подмножеств этого множества:

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

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

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

Возникает вопрос: как задавать систему подмножеств? Можно предложить выписывать все независимые подмножества из E, используя подходящее представление. Однако такой способ представления может оказаться очень неэффективным, так как семейство подмножеств J может содержать до $2^{|E|}$ подмножеств. Поэтому все рассматриваемые системы подмножеств будем представлять с помощью алгоритма, который по данному подмножеству I множества E решает, верно ли, что $I \in J$ (т. е. является независимым).

Наша цель заключается в поиске алгоритмов решения комбинаторных задач оптимизации для некоторых систем подмножеств. Рассмотрим сейчас общую схему «жадного» алгоритма.

«Жадный" алгоритм»

1. $I = \emptyset$.

2. Пока $E \neq \{\emptyset\}$ выполнять следующую последовательность шагов:

а)пусть $e \in E$ и имеет наибольший вес $w(e)$;

б)удалить e из E, т. е. $E = E \backslash e$;

в)если $I \cup e \in J$, то $I = I \cup e$.

Некоторые комбинаторные задачи для систем подмножеств могут быть решены корректно (точно) «жадным» алгоритмом, а некоторые нет (получается некоторое приближенное решение).

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

Приведем сейчас примеры различных задач, сформулируем их как комбинаторные задачи оптимизации для системы подмножеств, затем применим к ним «жадный» алгоритм и посмотрим, решает ли он корректно рассматриваемую задачу.

Пример 2.8. Пусть E – множество ребер графа $G = (V, E)$ каждому ребру графа приписан вес $w(e)$ *и* X – семейство паросочетаний графа. Семейство X замкнуто относительно включения, так как если из некоторого паросочетания удалить ребро, то оно также останется паросочетанием.

Таким образом, $S = (E, X)$ – система подмножеств.

Комбинаторная задача оптимизации для системы S заключается в нахождении паросочетания графа максимального веса.

Рассмотрим граф, изображенный на рисунке 1.41, для которого нужно найти паросочетание максимального веса:

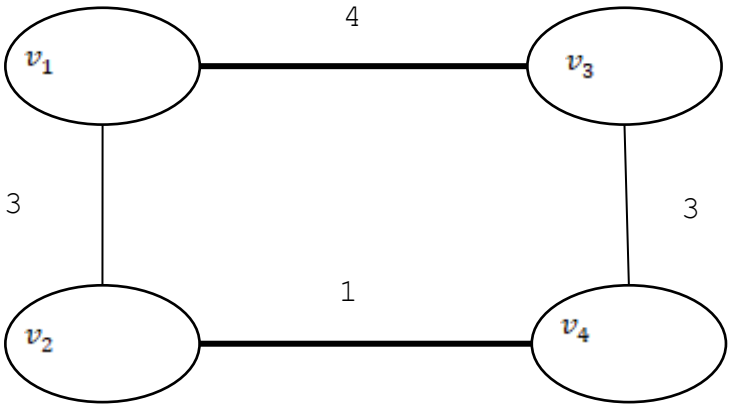


Рисунок 1.41 –Изображение графа.

Применим «жадный» алгоритм. Он построит следующее решение $I = \{(v_1, v_3), (v_2, v_4)\}$ которое имеет вес $w(I) = 5$ (на рисунке 1.41 выбранные ребра выделены).

Однако построенное «жадным» алгоритмом решение не является оптимальным, так как существует множество $I' = \{(v_1, v_2), (v_3, v_4)\}$ которое имеет вес $w(I') = 6$. Так как $w(I) < w(I')$, то «жадный» алгоритм не решает корректно задачу о максимальном взвешенном паросочетании, а следовательно, рассмотренная система S не является матроидом.

Пример 2.9. Задан орграф $G = (V, E)$, для каждой дуги $e \in E$ определен вес $w(e) \geq 0$. Требуется найти в множестве E такое подмножество B наибольшего веса, что никакие две дуги из B не имеют общего конца. Сформулируем данную задачу как комбинаторную задачу оптимизации для системы подмножеств.

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

Рассмотрим ориентированный граф, приведенный на рисунке 1.42.

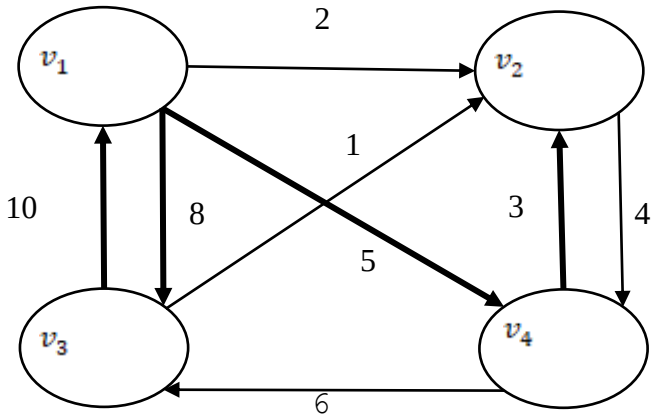


Рисунок 1.42 – Матроид разбиения.

Применим «жадный» алгоритм, который строит следующее решение (на рисунок 1.42 выбранные дуги выделены):

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

Анализируя орграф, который приведен выше на рисунке, можно заметить, что при выборе дуги, входящей в вершину v_1, не имеет смысла выбирать никакую другую дугу, кроме дуги, имеющей наибольший вес. Это связано с тем, что этот выбор локален в том смысле, что он никак не влияет на допустимость выборов в других вершинах. То же самое справедливо для всех остальных вершин. Поэтому в данной задаче "жадный" алгоритм будет давать оптимальное решение.

Так как данную комбинаторную задачу «жадный» алгоритм решает корректно, то рассмотренная в примере система подмножеств $S = (E, X)$ является матроидом, который называют *матроидом разбиения*.

Пример 2.10. Задан граф $G = (V, E)$, для каждого ребра $e \in E$ определен вес $w(e) \geq 0$. Рассмотрим в графе множества ребер, которые являются объединениями вершинно непересекающихся путей. Семейство этих множеств ребер обозначим через J (множество ребер X является независимым, т. е. принадлежит J, если оно является объединением вершинно непересекающихся путей). Комбинаторная задача оптимизации для некоторой системы подмножеств E, J заключается в нахождении подмножества ребер вершинно непересекающихся путей и имеющего наибольший вес.

Рассмотрим следующий пример. На рисунке 1.43, *а* изображен исходный граф. Применим «жадный» алгоритм и получим следующее решение (См. рисунок 1.43, *б*): $I = \{\{v_1, v_4\}, \{v_4, v_2\}, \{v_2, v_3\}\}$, $w(I) = 18$.

Данное решение не является оптимальным, так как оптимальным является следующее независимое множество (См. рисунок 1.43, *в*):

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

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

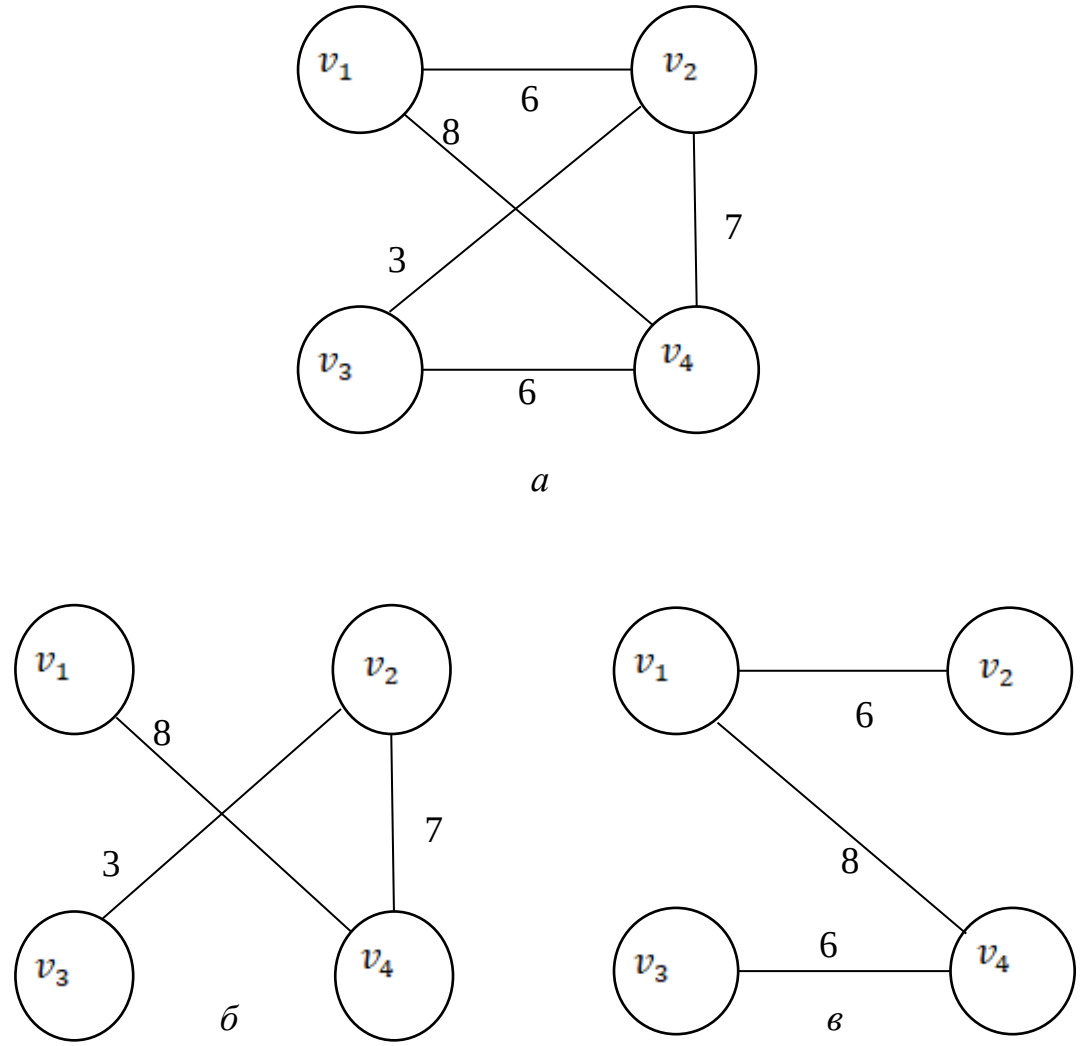


Рисунок 1.43 – Возможные решения.

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

Оказывается, что «жадный» алгоритм корректно решает комбинаторную задачу оптимизации, связанную с этой системой подмножеств. Этот факт будет получен позже, как простое следствие доказанной далее теоремы и элементарных факторов из линейной алгебры. Рассмотренную в данном примере систему подмножеств называют *матричным матроидом*.

Задача о минимальном остовном дереве также решалась «жадным» алгоритмом. Однако эта задача не может быть сформулирована в терминах системы подмножеств, так как если обозначить через J семейство остовных деревьев не-

которого графа, то это семейство не будет являться замкнутым относительно включения (любое подмножество ребер остовного дерева уже не будет являться остовным деревом исходного графа).

Однако существует другая задача, которая, по существу, может рассматриваться как задача о минимальном остовном дереве, но в то же время может быть сформулирована как комбинаторная задача оптимизации для системы подмножеств. Рассмотрим эту задачу.

Пример 2.12. Задан граф $G = (V, E)$. Каждому ребру графа поставлен в соответствие некоторый вес $w(e) \geq 0$.

Требуется найти лес – ациклический подграф графа G, имеющий максимальный общий вес.

Предположим, что граф является связным. Так как веса положительны, то любой оптимальный вес можно сделать максимальным по включению, т. е. остовным деревом графа G.

Кроме того, все остовные деревья графа G имеют $|V| - 1$ ребер. Поэтому можно построить функцию расстояний для E, полагая

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

где

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

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

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

Отсюда сразу следует, что минимальное остовное дерево в графе G с расстояниями d совпадает с лесом максимального веса в графе G с весами w.

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

Таким образом, задачи о минимальном остовном дереве и задачу о лесе максимального веса можно рассматривать, по существу, как одну и ту же задачу.

Следующая теорема является аналогом основного свойства минимального остовного дерева.

Теорема 2.5. Пусть $\{(U_1, T_1), (U_2, T_2), \ldots, (U_k, T_k)\}$ − остовный лес на множестве V, и пусть $\{v, u\}$ – ребро, исходящее из U_1 и имеющее наибольший вес. Тогда среди всех лесов, содержащих $T = \bigcup_{i=1}^{k} T_i$ существует оптимальный лес, содержащий также $\{v, u\}$.

Таким образом, задачу о лесе максимального веса можно решить, используя любой из двух алгоритмов для минимального остовного дерева [10].

Теорема 2.6. «Жадный» алгоритм корректно решает задачу о лесе максимального веса.

Пример 2.13. Рассмотрим задачу построения минимального остовного леса для графа, изображенного на рисунке 1.44. Каждому ребру на рисунке поставлен в соответствие вес w.

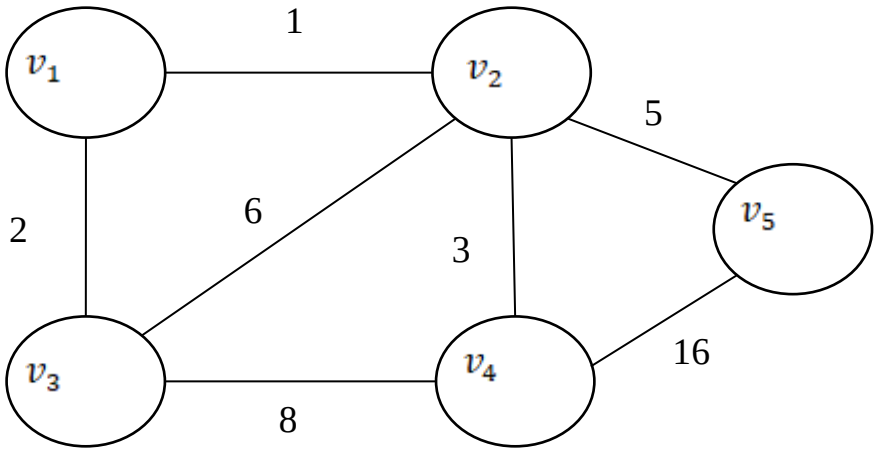


Рисунок 1.44 – Функция весов w.

Данная задача эквивалентна задаче о минимальном остовном дереве в графе, который приведен на рисунке 1.45. На рисунке каждому ребру графа поставлено в соответствие расстояние d по формулам:

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

где

$$W = \max_{e \in E} w(e) = 15.$$

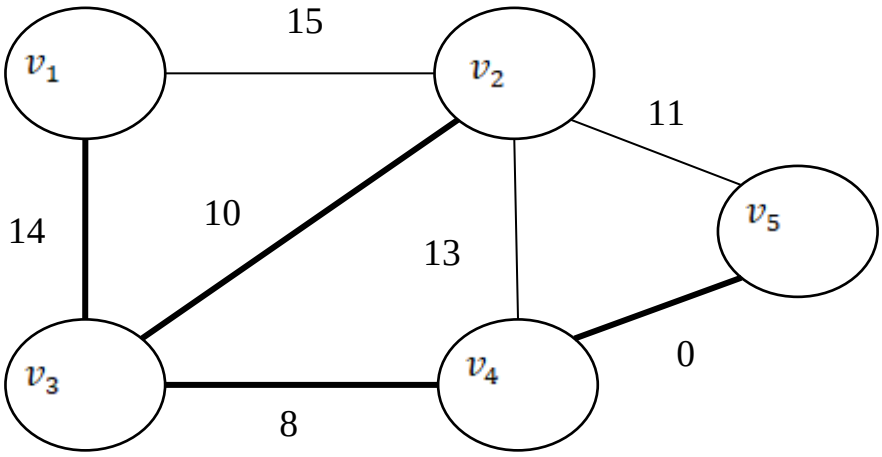


Рисунок 1.45 – Функция весов d.

На рисунке 1.45 минимальное остовное дерево графа с матрицей весов d построено, а ребра, вошедшие в дерево, выделены.

Решая эти задачи «жадным» алгоритмом, получим лес максимального веса, который изображен на рисунке 1.46.

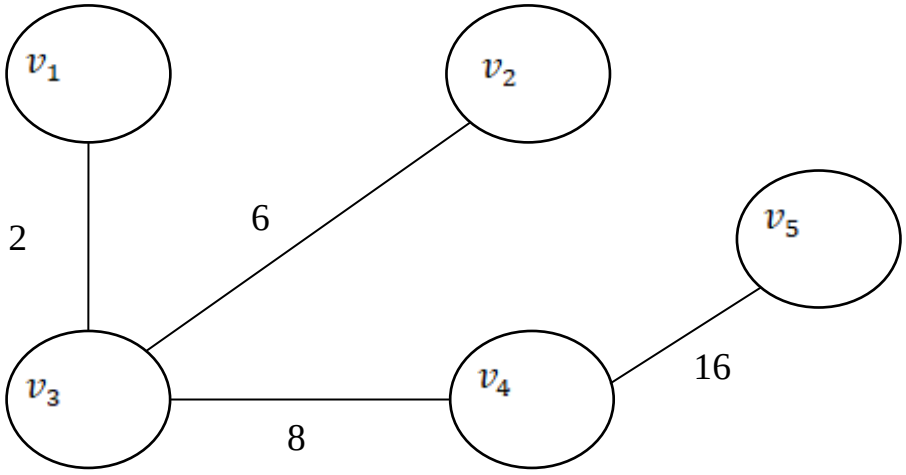


Рисунок 1.46 – Лес максимального веса.

Рассмотренная в примере 2.13 система подмножеств является матроидом, который называют *графическим*.

Докажем сейчас основную теорему об эквивалентных определениях матроида.

Теорема 2.7. Пусть $M = (E, J)$ – система подмножеств. Тогда следующие утверждения эквивалентны.

1. M – матроид.
2. Если $I_p, I_{p+1} \in J$ (независимые подмножества), где
$$|I_p| = p,\ |I_{p+1}| = p + 1,$$
то существует элемент $e \in I_{p+1} - I_p$ такой, что $I_p \cup e \in J$.
3. Если $A \subseteq E$ и I, I'– максимальные по включению подмножества множества A, то $|I| = |I'|$ (т. е. максимальные по включению независимые подмножества множества A являются также максимальными и по количеству элементов).

Доказательство. $1 \Rightarrow 2$.

Предположим, что M – матроид, но утверждение (2) не выполняется, т. е. существуют два независимых подмножества I_p, I_{p+1}, где $|I_p| = p, |I_{p+1}| = p + 1$, но ни для какого элемента $e \in I_{p+1} - I_p$ множество $I_p \cup e \notin J$ (т. е. не является независимым).

Рассмотрим функцию весов w на E, которую определим по следующему правилу:

$$w(e) = \begin{cases} p+2, e \in I_p, \\ p+1, e \in I_{p+1} - I_p, \\ 0, \quad e \notin I_p \cup I_{p+1}. \end{cases}$$

Оценим вес множеств I_p и I_{p+1}:

$$w(I_{p+1}) \geq (p+1) \cdot (p+1) = p^2 + 2p + 1 > p(p+2) = w(I_p).$$

Таким образом, получаем, что

$$w(I_{p+1}) > w(I_p). \tag{2.1}$$

Из неравенства (2.1) следует, что множество I_p не оптимально.

Применим «жадный» алгоритм, так как M – матроид, то он должен построить оптимальное множество. Для данной функции весов он начнет с элементов множества I_p, так как у них самые большие веса, и все их выберет (так как множество I_p независимо).

После чего он попытается добавлять элементы из множества $I_{p+1} - I_p$, но по предположению таких элементов e не существует $\forall e \in I_{p+1} - I_p,\ I_p \cup e \notin J$. Затем останутся лишь элементы нулевого веса, которые не изменят веса оптимального множества, построенного «жадным» алгоритмом.

Таким образом, «жадный» алгоритм построит в качестве оптимального множества множество I', которое имеет такой же вес, что и множество I_p.

Но в силу (2.1) выполняется неравенство $w(I') = w(I_p) < w(I_{p+1})$, что противоречит оптимальности множества I', построенного «жадным» алгоритмом. Приходим к противоречию, следовательно, предположение о том, что ни для какого элемента $e \in I_{p+1} - I_p$ множество $I_p \cup e \notin J$ не верно. Таким образом, утверждение (2) теоремы верно.

Доказательство $2 \Longrightarrow 3$.

Предположим, что утверждение (2) выполняется. Пусть I, I' – два независимых максимальных по включению подмножества множества $A \subseteq E$ Предположим, что утверждение (3) теоремы не выполняется, т. е.

$$|I| < |I'|.$$

Тогда отбросим из большего множества I' количество элементов, равное числу $(|I'| - |I| - 1)$. Получим такое множество $I'' \subseteq I'$ с количеством элементов

$$|I''| = |I'| - (|I'| - |I| - 1) = |I| + 1.$$

Заметим, что множество I'' является независимым подмножеством множества A (любое подмножество независимого множества также является независимым).

Таким образом, имеем два независимых подмножества I, I'' мощности которых отличаются на 1, и так как выполняется утверждение (2), то существует элемент $e \in I'' - I (e \in A)$ такой, что $I \cup e \in J$.

Заметим, что $I \cup e \in A$ и является независимым подмножеством множества A, но по условию множество I являлось максимальным по включению независимым подмножеством множества A. Приходим к противоречию. Следовательно, сделанное предположение о том, что мощность множества I меньше мощности множества I' не верно (аналогично приходим к противоречию при предположении, что мощность множества I' меньше мощности множества I). Таким образом, $|I| = |I'|$, и утверждение (3) теоремы верно.

Доказательство $3 \Longrightarrow 1$.

Предположим, что утверждение (3) верно, и покажем, что «жадный» алгоритм корректно решает любую комбинаторную задачу на M, тогда система M будет являться матроидом.

Проведем доказательство от противного.

Пусть существует такая функция весов $w(e), \forall e \in E$ что «жадный» алгоритм строит независимое множество $I \in J$, которое максимально по включению

$$I = \{e_1, e_2, \dots, e_i\}.$$

В то время как существует другое независимое максимальное по включению множество

$$\bar{J} = \{e'_1, e'_2, \dots, e'_j\}.$$

такое, что

$$w(\bar{J}) > w(I) \tag{2.2}$$

Т. е множество, построенное «жадным» алгоритмом, не является оптимальным.

Предположим, что элементы множеств I и $\bar{J}$ упорядочены так, что

$$w(e_1) \geq w(e_2) \geq \cdots \geq w(e_i),$$
$$w(e'_1) \geq w(e'_2) \geq \cdots \geq w(e'_j).$$

Учитывая, что $I, \bar{J} \in J$ и являются максимальными по включению, то в силу выполнения утверждения (3) теоремы (при $E = A$) выполняется, что $|I| = |\bar{J}|$ поэтому $i = j$, и мы можем считать, что

$$I = \{e_1, e_2, \dots, e_i\},$$
$$\bar{J} = \{e'_1, e'_2, \dots, e'_j\}.$$

Покажем, что для всех $m = 1,2, \dots, i$ верно неравенство

$$w(e_m) \geq w(e'_m). \tag{2.3}$$

Тогда будет верно, что $w(I) \geq w(\bar{J})$, и мы придем к противоречию с неравенством (2.2).

Доказательство проведем по индукции. Пусть $m = 1$.

Неравенство (2.3) верно, так как "жадный" алгоритм выбирает всегда элемент с самым большим весом $w(e_1) \geq w(e_1')$.

Предположим, что неравенство выполняется для $m \leq i-1$ т. е.

$w(e_s) \geq w(e_s'), \forall s = 1,2,\dots,i-1.$

Нам надо показать, что неравенство верно для $m = i$, т. е. что $w(e_i) \geq w(e_i')$. Пусть это не так, т. е.

$$w(e_i) < w(e_i'). \qquad (2.4)$$

Рассмотрим множество

$$A = \{e \in E : w(e) \geq w(e_i')\}.$$

Множество $\{e_1, e_2, \dots, e_{i-1}\}$ – максимальное по включению независимое подмножество множества A, так как если бы существовало множество $\{e_1, e_2, \dots, e_{i-1}, e\} \in J$ и в силу неравенства (2.4) должно было бы выполняться неравенство $w(e) \geq w(e_i') > w(e_i)$, то «жадный» алгоритм взял бы на своем очередном шаге при построении оптимального множества не элемент e_i, а элемент e. Однако существует независимое множество $\{e_1', e_2', \dots, e_m'\} \in A$ и имеющее мощность на 1 большее, чем множество $\{e_1, e_2, \dots, e_{i-1}\}$. Приходим к противоречию с утверждением (3) теоремы. Следовательно, неравенство (2.4) не выполняется, и индукция по *m* доказана.

Таким образом, $w(I) \geq w(\bar{J})$, что противоречит предположению (2.2). Следовательно, предположение о существовании множества $\bar{J}$ не верно, поэтому «жадный» алгоритм корректно решает любую задачу на M, а следовательно, M – матроид. ■

В примере 2.7 задача о максимальном взвешенном паросочетании не являлась матроидом, так как для нее не выполняется утверждение 3 теоремы: максимальные по включению независимые подмножества множества A являются также максимальными и по количеству элементов. Максимальное по включению паросочетание не всегда будет максимальным по количеству элементов.

Определение 2.8. Пусть $M = (E, J)$ – матроид и $A \subseteq E$. *Рангом* $r(A)$ множества A в M называется мощность максимальных по включению независимых подмножеств множества A. Максимальные по включению независимые подмножества множества E называются *базисами*.

В примере 2.13 рассмотренная система являлась матричным матроидом, так как для нее выполняется утверждение 3 теоремы. Это следует из хорошо известного факта линейной алгебры: все максимальные линейно независимые подмножества множества векторов E имеют одинаковую мощность. Эта мощность в алгебре называется рангом подматрицы A, определяемой множеством столбцов E.

Матричные матроиды образуют более широкий класс, чем графические матроиды и матроиды разбиения. Можно показать, что любой графический матроид или матроид разбиения представим матричным матроидом, если подо-

брать соответствующую матрицу A над некоторым полем K. Однако это верно не для всех матроидов, поэтому теория матроидов является существенным обобщением линейной алгебры.

Определение 2.9. Подмножество D множества E, не входящее в J, называется зависимым. Минимальное по включению зависимое подмножество C в E называется циклом.

Теорема 2.8. Пусть $I \in J$ и $e \in E$. Тогда либо $I + e \in J$, либо $I + e$ содержит единственный цикл.

Если мы рассматриваем графический матроид, то циклы матроида – это теоретико-графовые циклы графа G. Ранг подмножества E' из E равен $|V| - c(E')$ где $c(E')$ – число связных компонент графа $G' = (V, E')$.

1.3. Версии задач с неполной информацией (online и semi online)

В последнее время большой интерес вызывают версии задачи с неполной информацией, так называемые on-line и semi on-line. Рассмотрим различные версии для классической задачи теории расписаний. Имеется m идентичных процессоров, когда требуется распределить работы таким образом, чтобы время завершения последней выполненной работы было минимально.

В on-line версии нет никакой информации о количестве работ, их длительности выполнения на процессоре. Работы поступают одна за другой, и прежде, чем поступит информация о длительности следующей работы или об отсутствии работ, текущая работа должна быть назначена на конкретный процессор, причем это назначение окончательное, т.е. не может быть изменено в дальнейшем.

Для оценки качества работы алгоритмов для версий задач с неполной информацией обычно используется сравнительный анализ. Для этого сравниваются значения решений, получаемых алгоритмом для on-line версии, с оптимальными значениями соответствующей off-line версии (когда все данные заранее известны), и устанавливается гарантированная оценка алгоритма.

Следует отметить, что при анализе алгоритмов для задач с неполной информацией особую роль играют так называемые нижние границы (НГ) для гарантированных оценок on-line и semi on-line алгоритмов. Смысл НГ для версий задач с неполной информацией состоит в том, что невозможно построить алгоритма, гарантированная оценка которого лучше, чем НГ.

1.3.1. Online алгоритмы для задач теории расписания

Одним из первых результатов в этой области является алгоритм LS (в минимально загруженный), для которого было доказано, что его гарантированная оценка равна 2-1/m, где m – количество процессоров.

Известно также, что в случае m=2 это алгоритм является асимптотически оптимальным, так как НГ точности алгоритмов для этой задачи равна 3/2.

Утверждение. Для онлайн версии задачи не существует алгоритма, гарантированная оценка которого меньше, чем 3/2.

Доказательство. Предположим, что такой алгоритм существует. Подадим на вход последовательно две работы длительности 1. Если алгоритм назначит их на один процессор, то на следующем шаге дадим информацию, что работы закончились. Значение оптимального решения равно 1, а алгоритм построил решение со значением 2. Гарантированная оценка равна 2, поэтому такое назначение невозможно.

Если же алгоритм назначит две поступившие работы на разные процессоры, то на вход подадим последнюю работу с длительностью 2.

Значение оптимального решения равно 2, а построенного равно 3. Поэтому гарантированная оценка не лучше 3/2. Противоречие. ■

Semi on-line версия задач предполагает наличие дополнительной информации общего характера. Для рассматриваемой задачи известно несколько semi on-line версий:

• известно, что работы поступают в отсортированном порядке;

• известно оптимальное значение решения;

• известна общая сумма длительностей выполнения;

• наличие буфера.

Для построения и анализа алгоритма используется несколько подходов.

Динамический пересчет нижней оценки оптимального решения

Был LPT (в минимально загруженный). Гарантированная оценка равна 4/3-1/(3m).

Алгоритм 4.1

• Вначале назначаем первые m работ на разные процессоры;

• Вычислить LB=max{a_1, a_m+a_{m+1}}

• Назначать очередную работу на максимально загруженный процессор, но с условием, что сумма не превосходила 5LB/4

Алгоритм 4.1 имеет гарантированную оценку, равную 5/4

Построение инвариантов

Основная идея подхода состоит в том следующем:

− на множестве объектов вводится определенная классификация;

− определяются инвариантные группы, определяемые параметрами, связанными с гарантированной оценкой алгоритма;

− определяются множества правил распределения объектов по группам, обеспечивающим поддержание инварианта.

Фиксированное количество групп, согласованность классификации и структура инвариантов позволяют, с одной стороны обеспечить эффективность

алгоритмов (как по времени, так и по памяти), а с другой упрощают доказательство гарантированной точности алгоритма, так как она уже заложена изначально в инвариантах.

В классической задаче минимизации времени завершения проекта на многопроцессорной системе рассматривается система из m машин с соответствующими им скоростями $s_1, s_2, \dots, s_m$ и N работ, с временами обработки $p_1, p_2, \dots, p_N$. Требуется распределить все работы таким образом, чтобы время их обработки было минимальным. В on-line версии данной задачи, все работы поступают последовательно, и каждую работу необходимо назначить на выполнение одной из машин сразу после поступления и, не имея никакой информации о последующих работах. Мы исследуем случай on-line версии задачи, когда $m - k$ процессоров имеют единичную скорость, а у k процессоров, где k – некоторая константа, скорости удовлетворяют соотношениям $1 \leq sm - k + 1 = \cdots = sm = s \leq 2$.

Предлагается параметрический алгоритм, который разбивает работы на классы, процессоры разбивает на группы, и вводит такой приоритет назначения очередной работы на процессор, при котором гарантируется, что всегда существует процессор, на который можно назначит очередную работу при заданной гарантированной точности решения задачи. Отличительной особенностью предлагаемого алгоритма является тот факт, что предлагаемая схема позволяет построить решение с гарантированной оценкой $2 + \alpha$ для любого $\alpha > 0$ при достаточно больших m.

Обозначим через J_j длительность J -го задания в списке σ, и будем считать, что задание J_j прибывает на шаге j согласно σ. Обозначим через p_j время обработки задания J_j. Если работа с временем p_j назначается на машину со скоростью s_i, то $\frac{p_j}{s_i}$ – время необходимое для выполнения данной работы.

Качество онлайн-алгоритма Alg определяется как наименьшее число c, такое, что для каждого списка заданий σ мы имеем $F(\text{Alg}, \sigma) \leq c\ \text{Opt}(\sigma)$, где $F(\text{Alg}, \sigma)$ обозначает время обработки расписания, которое получается как результат применения алгоритма Alg к списку σ, и Opt(σ) обозначает время обработки некоторого оптимального расписания работ σ, вычисляемое при рассмотрении σ как набора заданий. В этом случае говорят, что гарантированная оценка алгоритма Alg равна c.

Рассмотрим параметрическую схему для онлайн версии задачи теории расписаний и униформ-процессорами. Прежде чем представить основные результаты, введем некоторые обозначения: m – общее количество машин; k – число машин со скоростью $1 < s \leq 2$, где k – константа.

Далее опишем стратегию (алгоритм), которая позволит нам назначать, для любого индекса $j = 1, \ldots,$ Length(s), работу J_j с временем обработки p_j, которая прибывает на шаге j согласно списку заказа s, на некоторые машины $M_i, i = 1, \ldots, m.$ Мы должны построить расписание таким образом, что J_j будут назначены сразу после завершения последнего задания, которое было получено. По сути, нет приоритета по отношению заданий, и задания, назначенные на одной машине, будут последовательно выполняться. Поэтому, каждый раз, когда мы имеем дело с текущим заданием J_j входного списка σ, мы обозначим через:

- $L_{i,j}$ – текущая загрузка машины i до назначения работы J_j;
- $L^*_{i,j}$ – текущая загрузка машины i после назначения работы J_j;
- V_j – теоретическое оптимальное решение для офф-лайн версии задачи для множества работ $J(j) = \{J_1, \ldots, J_j\}$.

Легко проверить, что, если мы обозначим через $q_1, \ldots, q_j$ времена обработки заданий из $J(j)$, отсортированных по невозрастанию, т. е. $q_1 \geq q_2 \geq \cdots \geq q_j$, то будет выполнено следующее утверждение.

Лемма 3.1. Справедливы следующие неравенства:

$$V_j \geq (q_1 + q_2 + \cdots + q_j)(m - k + s \cdot k),$$

$$V_j \geq q_{\frac{1}{s}},\ V_j \geq \min\left\{\frac{(q_k + q_{k+1})}{s},\ q_{k+1}\right\}.$$

Доказательство очевидно. Важно заметить, что последнее неравенство $V_j \geq min\left\{\frac{(q_k+q_{k+1})}{s},\ q_{k+1}\right\}$ вытекает из предположения $1 \leq s \leq 2$. Следовательно, для каждого шага j имеем

$$LB_j = \mathrm{Sup}\left\{\frac{(q_1+q_2+\cdots+q_j)}{(m-k+s\cdot k)},\ q_{\frac{1}{s}},\ \min\left\{\frac{(q_k+q_{k+1})}{s},\ q_{k+1}\right\}\right\}. \quad (3.1)$$

Ясно, что величина LB_j является нижней оценкой для оптимального значения офф-лайн версии задачи, соответствующей шагу j, причем $LB_{j-1} \leq LB_j$ (т. е. LB_j является монотонной).

Опишем теперь процесс назначения. Предположим, что некоторое положительное число α, а дается вместе с тремя целыми числами R, m1 and m2, для которых справедливы соотношения:

$$(1 + \alpha)s \cdot k + \left(1 + \frac{\alpha}{2}\right) m_1 \geq s \cdot k + m_1 + m_2, \quad (3.2)$$

$$k + m_1 + m_2 = m, \quad (3.3)$$

$$m_2 = R \cdot k, \quad (3.4)$$

$$R \geq \log 1 + \frac{\alpha}{2}\left(\frac{\left(1+\frac{\alpha}{2}\right)}{(2+\alpha-s)}\right). \quad (3.5)$$

Легко заметить, что если мы зафиксируем *k*, *s* и α, а также потребуем, чтобы *R* и m_1 принимали минимальные возможные значения, то *R*, *m*, m_1 и m_2 полностью определяются значениями *k*, *s* и α. Это наблюдение указывает путь, как m машин может быть разбито на наборы машин, составляющие три класса:

– машины со скоростью s называются Fast;

– выбираем m_1 машин из $m - k$ машин со скоростью 1 и называем их нормальными (Normal);

– остальные m_2 машин со скоростью 1 называются зарезервированными (Reserved), их количество равно $m_2 = R \cdot k$. Зарезервированные машины делятся на *R* групп $G_0, \dots, G_{R-1}$, где каждая из групп содержит ровно *k* машин.

Аналогичным образом будем говорить, что работа J_j, поступающая на шаге *j* является:

– Small, если время выполнения p_j не превосходит величины $\left(1 + \frac{\alpha}{2}\right) LB_j$;

– Large, в противном случае.

Наконец, мы будем говорить, что работа J_j подходит для машины M_i, $i = 1, \dots, m$, если $L_{i,j} + \frac{p_j}{s_i} \leq (2 + \alpha) LB_j$.

Лемма 3.2. Если работа J_j из класса Large не подходит для машины *i* из класса Fast, то справедливо $L_{i,j} > (1 + \alpha) LB_j$.

Доказательство. Следует непосредственно из того факта, что $\frac{p_j}{s_i} = \frac{p_j}{s} \leq LB_j$.

Приведенные выше факты позволяют описать онлайн алгоритм, который будет работать корректно на любом входном наборе для задачи теории расписаний с униформ-процессорами, базируясь на значениях m, *k*, *s* в соответствии с соотношениями (3.2) – (3.5).

Основная идея здесь заключается в том, что на каждом шаге *j*, мы будем иметь возможность назначать работы J_j на некоторые машины *i* таким образом, что будет выполняться следующее неравенство $L^*_{i,j} \leq (2 + \alpha) LB_j$. При этом это неравенство очевидно в случае, когда J_j – это работы из класса Small. Таким образом, если J_j – это большая работа (из класса Large), то всегда существует машина, для которой эта работа подходит. Для этого специальным образом подберем параметры, при которых гарантируется, что если работа J_j не подходит для любых машины из классов Fast и Normal, то она, по крайней мере, подойдет для некоторых машин в группе из класса Reserved.

Алгоритм Assign

Инициализация: полагаем $n = 0$; (*n* соответствует индексу активной машины из класса Reserved в группе G_0; машины в каждой группе G_r пронумерованы от 0 to $k - 1$); полагаем $j = 0$; $LB_j = 0$;

Прочитать список σ;

Пока список σ не пуст, повторять следующие шаги: $j = j + 1$;

Взять очередную работу J_j и выполнить шаг *j* следующим образом:

1.Пересчитать значение LB_j в соответствии с формулой (1.1).

2.Если работа J_j подходит для некоторой машины *i* из классов Fast или Normal, то назначить работу *j* на эту машину *i*. Иначе:

если $n < k$, то назначить работу J_j на машину с номером *n* в группе G_0; Пусть i_0 будет номер машины из класса Normal с минимальной текущей суммарной загрузкой. Провести обмен машин n и i_0 между классами Normal и G_0 таким образом, что машина i_0 попадает в группу G_0 с номером n, а машина из группы попадает в класс Normal. Положить $n = n + 1$;

если n = k, то перенумеровать группы $G_0, \dots, G_{R-1}$ таким образом, что группа с номером r, $1 \le r \le R - 1$ получает новый номер $r - 1$, а группа с номером 0 получает номер $R - 1$. Положить $n = 1$.

Предложенный алгоритм работает на входных данных $(M_1, M_2, \dots, M_m;\ s_1, s_2, \dots, s_m)$, для которых выполняется: $s_i = s \in [1,2]$ для $i = 1, \dots, k;\ s_i = 1$ для $i = k+1, \dots, m;\ m$ можно представить в виде $m = k + m_1 + m_2 = k + m_1 + Rk$, где числа m_1, m_2 и *R* удовлетворяют соотношениям (3.2) – (3.5).

Сейчас докажем, что если k, α фиксированы, а величина m достаточно велика, то гарантированная оценка предлагаемого алгоритма не превосходит величины $(2 + \alpha)$. Более точно, мы докажем, что если последовательность работ σ является входными данными для алгоритма Assign, то значение решения F(Assign, σ) в построенном расписании не превосходит $(2 + \alpha)LB(\sigma)$, где $LB(\sigma)$ соответствует нижней границе оптимального значения Opt(σ).

Лемма 3.3. На любом шаге *j* во время выполнения алгоритма Assign существует либо машина *i* из класса Fast, такая, что $L_{i,j} \le (1 + \alpha)LB_j$ или машина *i* из класса Normal, такая, что $L_{i,j} \le \left(1 + \frac{\alpha}{2}\right) LB_j$.

Доказательство. Предположим противное. Пусть существует шаг *j*, такой, что для любой машины i из класса Fast $L_{i,j} > (1 + \alpha)LB_j$, и для любой машины *i* из класса Normal $L_{i,j} > \left(1 + \frac{\alpha}{2}\right) LB_j$. Это значит, что

$$\left(p_1 + p_2 + \cdots + p_j\right) = s\left(\Sigma i \in \text{Fast} L_{i,j}\right) + \left(\Sigma i \in \text{Normal} \cup \text{Reserved} L_{i,j}\right)$$
$$\gg ks(1 + \alpha)LB_j + m_1\left(1 + \frac{\alpha}{2}\right) LB_j.$$

Из леммы 3.1 следует, что $\left(p_1 + p_2 + \cdots + p_j\right) \le (sk + m_1 + m_2)LB_j$, так как условие (3.2) гарантирует, что

$$ks(1 + \alpha)LB_j + m_1\left(1 + \frac{\alpha}{2}\right) LB_j = sk + m_1 + m_2.$$

Получили противоречие. ■

Лемма 3.4. Если текущая работа J_j из класса Small, то существует машина в одном из классов Fast или Normal, для которой эта работа подходит.

Доказательство. Применим лемму 3.2 и рассмотрим машину i как в утверждении леммы 3.3. Если i из класса Fast, то $L_{i,j} \leq (1+\alpha)LB_j$. Учитывая тот факт, что $\frac{p_j}{s_i} = \frac{p_j}{s} \leq LB_j$ получаем $L_{i,j} + \frac{p_j}{s_i} \leq (2+\alpha)LB_j$ как результат. Если i из класса Normal, то $L_{i,j} \leq \left(1+\frac{\alpha}{2}\right)LB_j$ и $L_{i,j} + \frac{p_j}{s_i} = L_{i,j} + p_j \leq (2+\alpha)LB_j$.

■

Лемма 3.5. Для $q = 1, \ldots, Q$, выполняется $L_{i(q),j(q)} \leq (2+\alpha-s)LB_{j(q)}$.

Теорема 3.1. Пусть задана величина α, и пусть для m, k, s выполняются соотношения (3.2)−(3.5). Тогда для любой входной последовательности σ, алгоритм Assign строит расписание, для которого выполнено $F(\text{Assign}, \sigma) \leq (2+\alpha)\text{Opt}(\sigma)$.

Теорема 3.2. Пусть даны скорости s, $1 < s \leq 2$, машин и количество k машин со скоростью s. Тогда для любого $\alpha > 0$ существует m_0, такое, что для каждого $m \geq m_0$, алгоритм Assign можно применить таким образом, что $F(\text{Assign}, \sigma) \leq (2+\alpha)\text{Opt}(\sigma)$.

2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

2.1. Задачи для самостоятельного решения по теме 1.1

Задача 1. Минимальное число коней

Имеется клеточное поле размера $n \times m$, в некоторых позициях которого расставлены черные фигуры (m – число столбцов, а n – число строк). Необходимо расставить минимальное число белых коней, чтобы пробивались все свободные позиции.

Формат входных данных

Первая строка задает размеры поля: целые числа m и n ($1 \le m, n \le 100$), вторая строка содержит число k занятых клеток ($0 \le k \le m \cdot n - 2$).

В последующих k строках находятся координаты занятых клеток – пары целых чисел x_i и y_i ($1 \le x_i \le m$, $1 \le y_i \le n$).

Формат выходных данных

В первой строке должно находиться число p расстановок.

Следующая строка должна быть пустой.

Далее выведите p досок размера $m \times n$, разделенных пустой строкой (где 0 – свободная клетка, 1 – конь, 2 – препятствие).

входной файл	*выходной файл*
5 5 5 2 2 2 4 3 3 4 2 4 4	2 00100 02120 10201 02120 00100 00100 02020 11211 02020 00100

Задача 2. Минимальное число ладей

Имеется клеточное поле размера $n \times m$, в некоторых позициях которого расставлены k черных фигур. Необходимо расставить минимальное число белых ладей, чтобы пробивались все свободные позиции.

Формат входных данных

В первой строке находятся целые числа n, m и k ($1 \le n, m \le 7$, $0 \le k \le n \cdot m$). В каждой из последующих k строк находится по одному целому числу x

($1 \le x \le n \cdot m$) – координата черной фигуры (нумерация координат в лексикографическом порядке).

Гарантируется, что все k координат фигур различны.

Формат выходных данных

В первой строке выведите число p всех расстановок.

В каждой из последующих p строк выведите по одной расстановке ладей. Каждая расстановка должна быть отсортирована в лексикографическом порядке.

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

Задача 3. Минимальное число ферзей

Имеется клеточное поле размера $n \times m$, в некоторых позициях которого расставлены k черных фигур. Необходимо расставить минимальное число белых ферзей, чтобы пробивались все свободные позиции.

Формат входных данных

В первой строке находятся целые числа n, m и k ($1 \le n, m \le 7$, $1 \le k \le n \cdot m$).

В каждой из последующих k строк находится целое число x – координата черной фигуры (нумерация координат в лексикографическом порядке, $1 \le x \le n \cdot m$).

Формат выходных данных

Выведите расстановки, отсортированные в лексикографическом порядке, по одной в строке.

В последней строке – число всех расстановок.

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

Задача 4. Минимальное число слонов

Имеется клеточное поле размера $n \times m$, в некоторых позициях которого расставлены k черных фигур. Необходимо расставить минимальное число белых слонов, чтобы пробивались все свободные позиции.

Формат входных данных

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

Формат выходных данных

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

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

Задача 5. Расстановка коней

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

Формат входных данных

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

Формат выходных данных

В первой строке выведите число m решений. В каждой из следующих m строк выведите решение: разделенные пробелом координаты клеток, куда следует расставить коней.

Координаты должны быть упорядочены по возрастанию (приоритет координаты x (строка) больше, чем приоритет координаты y (столбец)). Строка завершается пробелом.

входной файл	*выходной файл*
1 3 3 1 1 1	8 0 0 0 1 0 2 0 0 0 1 1 0 0 0 1 0 2 0 0 1 0 2 1 2 0 2 1 2 2 2 1 0 2 0 2 1

	1 2 2 1 2 2 2 0 2 1 2 2

Задача 6. Расстановка ферзей

Необходимо расставить минимальное число ферзей, чтобы они атаковали каждую свободную клетки доски не менее k раз. Клетка считается свободной, если она изначально пустая. В задаче следует учитывать, что доска может содержать препятствия, которые в свою очередь могут сужать область пробитых клеток у ферзей. При этом другие ферзи такими препятствиями друг для друга не являются.

Обратите внимание, что клетка, занятая ферзем, также должна быть атакована не менее k раз, включая атаку стоящего на ней ферзя.

Формат входных данных

Первая строка содержит число k.

Вторая строка содержит ширину w, затем высоту h доски.

Третья строка содержит число n занятых клеток доски (препятствий).

Следующие n строк содержат координаты занятых точек (строка, столбец).

Гарантируется, что

$1 \le k$, $1 \le w$, $1 \le h$, $0 \le n$, $n + k \le w \cdot h$, $w \cdot h \le 100$, $w + h \le 25$.

Формат выходных данных

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

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

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

Задача 7. Максимальное число ладей

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

Формат входных данных

В первой строке указаны через пробел n, m и k, где n и m – размеры поля, k – число черных фигур.

В каждой из последующих k строк находятся 2 числа i и j, разделенных пробелом, где (i, j) – координаты черной фигуры (считается, что левая верхняя клетка имеет координаты (1, 1)).

Формат выходных данных

В выводе одной расстановки выводятся построчно координаты всех белых ладей в формате (*j*, пробел, *i*).

Потом вывести комбинацию символов <--->.

В конце вывести число полученных расстановок.

входной файл	*выходной файл*
3 3 4 2 2 2 3 3 2 3 3	3 1 1 3 <---> 3 1 1 2 <---> 2 1 1 3 <---> 2 1 1 2 <---> 4

Задача 8. Максимальное число ферзей

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

Формат входных данных

В первой строке находится целое число n ($1 \le n \le 10$).

Во второй строке – целое число m ($1 \le m \le 10$). В третьей строке – число b черных фигур.

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

Формат выходных данных

В первой строке выведите максимальное число ферзей. Во второй строке – число p решений.

В каждой из последующих p строк выведите последовательность чисел, разделенных пробелами, где каждая пара чисел – координаты ферзя.

Последовательности вывести в лексикографическом порядке.

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

Задача 9. Максимальное число слонов

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

Формат входных данных

В первой строке находится целое число m, во второй строке – целое число n ($1 \le m, n \le 10$). В каждой из последующих m строк находится по n чисел 0 или 1, разделенных пробелом: 0 означает, что клетка пуста, 1 – что там стоит черная фигура.

Формат выходных данных

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

входной файл	*выходной файл*
4 6 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0	5 12 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 1 2 2 2 3 2 5 3 5 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 2 2 3 2 5 3 0 3 5 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 2 2 5 3 0 3 2 3 5 0 0 0 1 0 2 0 3 0 5 1 5 2 0 2 2 3 0 3 2 3 4 3 5 0 0 0 2 0 3 0 5 1 0 1 5 2 0 2 2 2 3 2 5 3 0 3 5

Задача 10. Кратчайшие маршруты коня

Имеется шахматное поле размера $n \times m$, некоторые позиции которого могут быть заняты фигурами. Необходимо найти все кратчайшие маршруты коня между двумя заданными позициями.

Формат входных данных

В первой строке находится число n столбцов ($1 \le n \le 10^6$). Во второй строке находится число m строк ($1 \le m \le 10^6$, $m \cdot n \le 2 \cdot 10^6$). В третьей строке находится число k фигур ($0 \le k$, $n \cdot k \le 200$, $m \cdot k \le 200$). В каждой из последующих k строк находится по два числа: номера столбца и строки (начиная с 0), где находится фигура. В последних двух строках указаны начальная и конечная позиция коня (столбец, строка, начиная с 0).

Формат выходных данных

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

Поскольку порядок вывода путей, вообще говоря, не однозначен, то необходимо определить порядок вывода. Примем клетку $(0, 0)$ за левый верхний угол, а $(n-1, m-1)$ – за правый нижний. Тогда рассмотрим первый ход. Пути

нужно выводить в порядке направления движения в порядке «вправо-вправо-вниз», «вправо-вниз-вниз», «влево-вниз-вниз», «влево-влево-вниз», «влево-влево-вверх», «влево-вверх-вверх», «вправо-вверх-вверх», «вправо-вправо-вверх». Если же есть более одного решения, у которых первый ход совпадает, то аналогичным образом они сортируются по второму ходу, далее по третьему и т. д.

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

Задача 11. Кратчайшие маршруты ладьи

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

Необходимо найти все кратчайшие маршруты ладьи между двумя заданными позициями.

Формат входных данных

В первой строке находится число n столбцов. Во второй строке находится число m строк. В третьей строке находится число k фигур. В каждой из последующих k строк находится по два числа: номера столбца и строки (начиная с 0), где находится фигура. В последних двух строках указаны начальная и конечная позиция ладьи (столбец, строка, начиная с 0).

Формат выходных данных

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

Поскольку порядок вывода маршрутов, вообще говоря, не однозначен, то необходимо определить порядок вывода. Примем клетку $(0, 0)$ за левый верхний угол, а $(m - 1, n - 1)$ – за правый нижний. Тогда рассмотрим первый ход.

Маршруты нужно выводить в порядке направления движения (главный параметр сортировки) в порядке «вправо», «вниз», «влево», «вверх» и длине хода (вторичный параметр сортировки). Если же есть более одного решения, у которых первый ход совпадает, то аналогичным образом они сортируются по второму ходу, далее по третьему и т. д.

входной файл	*выходной файл*
8 6 8 1 1 1 4 3 0 3 5 4 0 4 5 6 1 6 4 0 0 7 5	3 0 0 0 2 7 2 7 5 3 0 0 0 3 7 3 7 5

Задача 12. Кратчайшие маршруты ферзя

Имеется шахматное поле размера $n \times m$, некоторые позиции которого могут быть заняты фигурами. Необходимо найти все кратчайшие маршруты ферзя между двумя заданными позициями. Под кратчайшим маршрутом понимается маршрут содержащий минимально возможное число ходов ферзя.

Формат входных данных

В первой строке находится число n (число столбцов).

Во второй строке – число m (число строк). В третьей строке находится число k (число черных фигур).

В каждой из последующих k строк находится по два числа: номер столбца и номер строки, где находится черная фигура (начиная с 0). В последних двух строках указаны начальная и конечная позиция ферзя (столбец, строка, начиная с 0).

Гарантируется, что $1 \le n, m \le 1024$, $n \cdot m \le 60\,000$.

Формат выходных данных

Если решение существует, то последовательно выведите все возможные пути в следующем виде.

В первой строке – число s ходов ферзя. В следующих $s + 1$ строках – клетки пути (столбец, строка, начиная с 0).

В конце файла – символ перевода строки. Если решений нет, то выведите *no_solutions* (без символа перевода строки).

Поскольку порядок вывода путей, вообще говоря, неоднозначен, то необходимо определить порядок вывода. Примем клетку $(0, 0)$ за левый верхний угол, а $(m - 1, n - 1)$ – за правый нижний. Тогда рассмотрим первый ход.

Пути надо выводить в порядке направления движения (главный параметр сортировки) в порядке «вправо», «вправо-вниз», «вниз», «влево-вниз», «влево», «влево-вверх», «вверх», «вправо-вверх» и длине хода (вторичный параметр сортировки). Если же число решений, у которых первый ход совпадает, более одного, то аналогичным образом они сортируются по второму ходу, далее по третьему

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

Задача 13. Эйлеровы циклы

Необходимо найти все эйлеровы циклы в неориентированном графе.

Формат входных данных

Первая строка содержит число n вершин и число m ребер графа ($1 \le n \le 100$, $0 \le m \le n \cdot (n-1)/2$).

Следующие m строк содержат по одному ребру, заданному как два номера концевых вершин, разделенных пробелом.

Формат выходных данных

В первой строке выведите число k эйлеровых циклов в заданном графе.

В каждой из следующих k строк выведите по одному эйлерову циклу в виде подпоследовательности, содержащей все вершины, разделенные пробелами.

Для каждого цикла выберите только одну соответствующую последовательность вершин – лексикографически минимальную.

Циклы следует выводить в лексикографическом порядке.

Если $k > 10\,000$, выведите только первые 10 000 циклов.

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

Задача 14. Разрезы доски

Найдите все возможные различные разрезы доски размером $n \times n$ ($n = 2 \cdot k$) на две одинаковые по форме связные фигуры.

Доской будем называть множество точек координатной плоскости *OXY*, для которых $\max \{|x|, |y|\} \le k$.

Границей доски будем называть множество точек координатной плоскости *OXY*, для которых $\max \{|x|, |y|\} = k$.

Разрезом доски будем называть ломаную, удовлетворяющую следующим аксиомам:

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

Фигуру будем называть *связной*, если для любых двух внутренних точек этой фигуры существует кривая, соединяющая эти точки и состоящая только из внутренних точек этой фигуры.

Две фигуры будем называть одинаковыми по форме, если при помощи преобразований параллельного переноса, поворота и симметрии можно получить совпадающие фигуры.

Формат входных данных

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

Формат выходных данных

Выведите все различные разрезы доски по одному в строке.

Если ломаная $L_1 L_2 \ldots L_n$ – разрез доски (причем считается, что все звенья ломаной имеют длину 1), то формат выходной строки:

$x_1\ y_1\ d_2 d_3 \ldots d_n$ (между x_1 и y_1, y_1 и d_2 должно стоять по одному пробелу), где

x_1, y_1 – целочисленные координаты начальной точки L_1 разреза;

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

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

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

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

где (x_i, y_i) – координаты точки L_i.

входной файл	*выходной файл*
1	-1 0 RR 0 -1 UU

Задача 15. Магические квадраты

Необходимо составить из костяшек одного набора домино все магические квадраты размера $n \times n$. Костяшки можно класть только горизонтально, костяшка занимает две клетки по горизонтали и одну по вертикали.

Каждая костяшка из набора в каждом отдельно взятом квадрате используется не более одного раза.

Магическим квадратом называется квадратная числовая таблица, у которой суммы элементов во всех строках, всех столбцах и на двух главных диагоналях совпадают.

Формат входных данных

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

Формат выходных данных

Выведите полученные квадраты следующим образом.

Каждый магический квадрат должен занимать $n+1$ строк: первая строка должна содержать восемь символов - (дефисоминус, ASCII-код 45) и служит для разделения квадратов, а последующие четыре строки – соответствующие строки самого квадрата (в каждой из них четыре числа, обозначающих числа на костяшках домино).

Пустой клетке костяшки соответствует число 0.

Соседние числа разделяются одним пробелом.

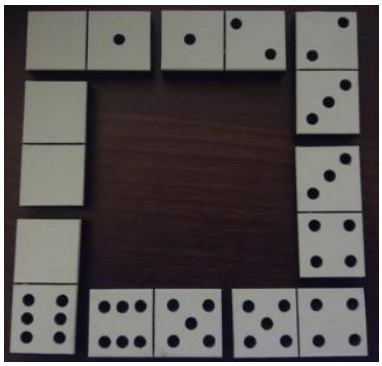
Обратите внимание, что после последнего квадрата строка -------- не выводится, но она выводится перед первым квадратом.

Последняя строка файла имеет вид *Count=x*, где *x* – число найденных магических квадратов.

Порядок вывода магических квадратов не имеет значения.

входной файл	*выходной файл*
4	-------- 0 0 6 0 1 4 0 1 1 2 0 3 4 0 0 2 -------- и так далее

Задача 16. Цепочки домино



Необходимо составить из стандартного набора костяшек домино все возможные замкнутые цепочки. Костяшки соприкасаются гранями сторон с одинаковыми числами. Под замкнутой цепочкой подразумевается прямоугольная рамка толщины 1. При этом разные прямоугольные рамки, полученные «сворачиванием» одной и той же линейной последовательности костей домино, считаются одинаковыми цепочками.

Формат входных данных

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

Формат выходных данных

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

входной файл	*выходной файл*
2	wrong count
4	2 2.0 3 3.0 4 4.0 5 5.0 6 6.0 3 3.0 4 4.0 и так далее 210

Задача 17. Таблица из костей домино

Даны две таблицы с числами. Необходимо покрыть их с помощью набора костяшек домино. Костяшки в наборе повторяться не могут.

Формат входных данных

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

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

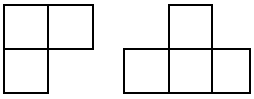
Формат выходных данных

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

входной файл	*выходной файл*
2 2 2 3 2 5 2 2 6 5 0 1	YES 4 1 2 1 2 3 4 3 4 1 2 1 2 3 3 4 4 1 1 2 2 3 4 3 4 1 1 2 2 3 3 4 4

Задача 18. Замощение фигурами

Имеется клеточное поле размера $n \times m$, в некоторых позициях которого расставлены черные фигуры. Необходимо замостить его фигурами вида:

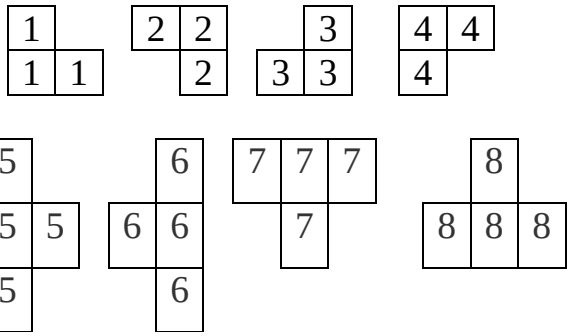


Формат входных данных

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

Формат выходных данных

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



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

входной файл	*выходной файл*
5 5 00000 00000 00001 00000 00000	0 12222 11232 22330 12122 11112 12222 11232 22330 12443 11433 и так далее

Задача 19. Раскраска графа

Задан неориентированный граф без петель и кратных ребер.

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

Формат входных данных

В первой строке указано число *n* вершин графа.

Если граф состоит только из вершин, то больше в файле ничего нет. Иначе пронумеруем вершины графа произвольным образом.

Следующие строки входного файла содержат по 2 числа *i* и *j* через пробел, если вершины *i* и *j* ($i \neq j$) соединены ребром.

Формат выходных данных

Каждому цвету поставьте в соответствие произвольный номер, начиная с 1 и т. д. по порядку. Число цветов не больше *n*.

В первой строке выведите минимальное число цветов, в которые данный граф раскрашивается, при выполнении требований, указанных в условии.

Во второй строке – номера цветов через пробел в порядке следования вершин, т. е. сначала номер цвета 1-й вершины, затем номер цвета 2-й вершины и т. д.

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

Задача 20. Расстановка фишек

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.

Формат входных данных

В первой строке находятся целые числа n и m ($n, m \geq 1$).

Последующие n строк содержат таблицу размера $n \times m$ с элементами 0 и –1, где 0 означает пустое место, –1 означает место, занятое черной фигурой.

Формат выходных данных

Каждый вариант расстановки фишек записывается в виде таблицы размера $n \times m$ с элементами 0, 1 и −1, где 0 означает пустое место, −1 означает место, занятое черной фигурой, 1 – место, на котором стоит фишка.

Таблицы разделяйте пустыми строками.

входной файл	*выходной файл*
4 4 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 -1 0 1 1 0 0 1 0 1 1 0

Задача 21. Детали – станки

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.

Формат входных данных

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

Формат выходных данных

В первой строке выведите минимальное время окончания обработки всех деталей. Далее последовательно опишите одно из возможных назначений деталей на станки. Оно имеет следующий формат: m строк, каждая из которых описывает назначение деталей на соответствующий станок (1-я строка – 1-й станок, 2-я – 2-й и т. д.). Значения, которые должна содержать данная строка, должны быть времена обработки, соответствующие деталям, которые назначены на данный станок. В случае отсутствия деталей на каком-либо станке, строка, соответствующая назначению деталей на текущий станок, должна содержать единственное число −1.

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

Задача 22. Склад

Финская компания имеет большой прямоугольный склад. На складе есть рабочий и менеджер. Стороны склада в порядке обхода называются соответственно расположению на схеме: левая, верхняя, правая и нижняя. Площадь склада разбита на квадраты с равной площадью, составляющие строки и столбцы. Строки пронумерованы сверху вниз целыми числами $1, 2, \ldots$, а столбцы пронумерованы слева направо целыми числами $1, 2, \ldots$ Склад имеет контейнеры, которые используются для хранения отдельных устройств. Контейнеры имеют попарно различные идентификационные номера. Каждый контейнер занимает ровно один квадрат. Склад настолько большой, что число прибывающих контейнеров меньше, чем число строк и число столбцов. Контейнеры не удаляются со склада, но иногда прибывают новые контейнеры. Вход в склад в левом верхнем углу склада. Рабочий разместил контейнеры вокруг левого верхнего угла таким образом, чтобы он мог найти их по их идентификационным номерам. Он использует следующий метод. Предположим, что идентификационный номер следующего контейнера, который нужно вставить, есть k (контейнер k для краткости). Рабочий идет вдоль первой строки и ищет первый контейнер, с идентификационным номером больше, чем k. Если он не находит такого контейнера, то он ставит его непосредственно за самым правым контейнером в строке. Если такой контейнер t найден, то контейнер t заменяется контейнером k, а контейнер t вставляется в следующую строку, используя тот же самый метод. Если рабочий достигает строки, в которой нет ни одного контейнера, он ставит контейнер в позицию самого левого квадрата этой строки.

Предположим, что контейнеры 3, 4, 9, 2, 5, 1 прибыли на склад в этом порядке. Тогда размещение контейнеров на складе будет таким:

1	4	5
2	9	
3		

К рабочему подходит менеджер, и между ними происходит следующий диалог:

– Контейнер 5 прибыл перед контейнером 4?

– Нет, это невозможно.

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

– Вообще говоря, нет. Например, контейнеры, находящиеся на складе, могли прибыть в порядке 3, 2, 1, 4, 9, 5, или в порядке 3, 2, 1, 9, 4, 5, или в одном из 14 других порядков.

Поскольку менеджер не хочет выглядеть намного глупее рабочего, он уходит. Вы должны помочь менеджеру и написать программу, которая по данному размещению контейнеров вычисляет все возможные порядки, в которых они могли прибывать.

Формат входных данных

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

Формат выходных данных

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

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

Задача 23. Упаковка семи деталей

Петя и Вася очень любят решать головоломки. Однажды, воскресным утром они собрались вместе и долго думали, какую же головоломку им поре-

шать. Они уже разгадали не одну сотню японский кроссвордов, судоку и других подобных задач. И тут Петя придумал новую задачу. Он достал с полки старую бумагу для вырезания, 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 семь прямоугольников в прямоугольник минимальной площади. Они договорились о размерах этих семи прямоугольников. Помогите Пете решить эту непосильную задачу. Также Петя понял, что, возможно, 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 воскресный вечер. Для заданных размеров прямоугольников необходимо определить минимальную площадь прямоугольника, в который можно уложить эти прямоугольники.

Формат входных данных

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

Формат выходных данных

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

входной файл	*выходной файл*
5 3 8 2 3 4 1 6 5 3 8 4 3 5	114

Задача 24. Минимальные остовные деревья

Необходимо построить все минимальные остовные деревья в графе.

Формат входных данных

В первой строке находятся два целых числа n и m, разделенные пробелом, где n – число вершин, а m – число ребер графа. В последующих m строках находится по 3 целых числа i, j и d, где d – вес ребра, соединяющего вершины i и j.

Формат выходных данных

В первой строке выведите число k минимальных остовных деревьев данного графа. В последующих k строках выведите по $n-1$ целых чисел, разделен-

ных пробелом, каждое из которых соответствует номеру ребра. Номера ребер определяются порядком их ввода.

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

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

Задача 25. Распределенная задача с сервером

Есть сервер и два процессора. Есть n_1 задач, которые могут выполняться только на первом процессоре и n_2 задач, которые могут выполняться только на втором процессоре. Перед выполнением каждой задачи, она сначала должна загрузиться с сервера на необходимый процессор, что занимает положительное время $s(k, i)$ для задачи с номером i исполняемой только на процессоре k. При этом процессе сервер и процессор k могут заниматься только загрузкой этой задачи.

Сразу после загрузки процессор начинает исполнение только что загруженной задачи, что занимает у него положительное время $p(k, i)$. После окончания выполнения загруженной задачи процессор может загружать следующую предназначенную для него задачу. То есть в каждый момент времени сервер может заниматься загрузкой не более одной задачи и каждый процессор выполняет не более чем одну задачу в каждый момент времени. Процессор не может одновременно загружать одну задачу и исполнять другую.

Необходимо найти такое расписание, при котором выполнялись бы все задачи и время окончания выполнения последней задачи было бы как можно меньшим.

Формат входных данных

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

Формат выходных данных

В первой строке требуется написать время окончания выполнения последней задачи.

В последующих $n_1 + n_2$ строках – вывести порядок, в котором задачи загружаются с сервера. Задача задается двумя числами: номером процессора и номером задачи.

Нумеруются задачи с 1, в порядке, заданном входным файлом.

входной файл	*выходной файл*
1 2 2 5 4 1 5 5	15 2 2 1 1 2 1

Задача 26. Нераспределенная задача с сервером

Есть один сервер и два процессора. Есть *n* задач, каждая из которых сначала должна быть загружена с сервера на любой процессор, а сразу после этого исполняться на этом процессоре. При этом, сервер может заниматься не более одной загрузкой в каждый момент времени и каждый процессор может заниматься исполнением не более одной задачи в каждый момент времени.

Процессор не может одновременно исполнять одну задачу и загружать другую. Для задачи *i* известно время s_i загрузки с сервера на процессор ($s_i > 0$) и время p_i исполнения на процессоре ($p_i > 0$), $i = 1, \ldots, n$. Необходимо найти такое расписание, при котором выполнялись бы все задачи и время окончания выполнения последней задачи было бы наименьшим.

Формат входных данных

В первой строке записано число *n*.

Далее идет *n* строк, в каждой по паре чисел: s_i и p_i.

Формат выходных данных

В первой строке требуется написать время окончания выполнения последней задачи. В последующих $n_1 + n_2$ строках – вывести порядок, в котором задачи загружаются с сервера.

Задача задается двумя числами: номером процессора и номером задачи. Нумеруются задачи с 1, в порядке, заданном входным файлом.

входной файл	*выходной файл*
4 12 3 8 7 3 12 1 14	33 3 1 4 2

Задача 27. Open-shop+

Имеется n работ и m приборов. Требуется составить такой маршрут обслуживания каждой работы на каждом приборе, чтобы время обслуживания последней работы было минимально.

В любой момент времени каждая работа должна обслуживаться не более чем одним прибором и каждый прибор должен обслуживать не более одной работы. При этом никакой прибор не может простаивать, если некоторая работа готова к выполнению на нем.

Формат входных данных

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

Формат выходных данных

Выведите минимальное время выполнения всех n работ на m приборах.

входной файл	*выходной файл*
3 4 4 12 4 6 8 6 2 10 2 4 18 3	27

Задача 28. Мультиразрез

Задан взвешенный неориентированный граф и зафиксировано некоторое подмножество T его вершин. Необходимо удалить из графа множество ребер с минимальным суммарным весом, чтобы все вершины множества T оказались в разных компонентах связности.

Формат входных данных

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

Формат выходных данных

В первой строке выведите минимальный возможный вес удаляемого множества. Во второй строке – число r ребер, входящих в это множество. В следующих r строках выведите номера ребер, входящих в это множество, по одному в строке. Ребра нумеруются, начиная с 1, в порядке, в котором они были заданы во входном файле. Если существует несколько решений с минимальным весом удаляемого множества, выведите любое из них.

входной файл	*выходной файл*

6 7 1 2 3 2 3 1 2 6 3 5 6 1 3 5 5 3 4 2 4 5 3 3 1 3 5	9 4 2 4 5 6

Задача 29. Job-shop

Есть n станков и 2 работы. Каждая работа состоит из s_i этапов.

Этап характеризуется парой (k, t) чисел, где k – номер станка, а t – продолжительность этапа. Для каждой работы порядок этапов строго задан. Любой этап можно приостановить в любой момент и позже продолжить с того же момента. В каждый момент времени любая работа может выполняться только на одном станке и любой станок может выполнять только одну работу. Необходимо составить такое расписание, чтобы все работы были выполнены за минимальное время.

Формат входных данных

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

Формат выходных данных

Выведите минимальное время выполнения всех n работ на m приборах.

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

Задача 30. Пятнашки

Имеется клеточное поле размера $n \times m$, в клетках которого произвольным образом расставлены $(n \cdot m - 1)$ костяшек, пронумерованных целыми числами $1, 2, \ldots, n \cdot m - 1$. Одна клетка пустая. На пустую клетку можно передвигать соседние по горизонтали или вертикали костяшки. Необходимо добиться такого расположения, чтобы костяшки располагались по возрастанию (слева направо по строкам, сверху вниз по столбцам), а правая нижняя угловая клетка стала пустой. Определить кратчайшее решение. Гарантируется, что решение существует.

Формат входных данных

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

Формат выходных данных

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

входной файл	*выходной файл*
3 3 1 0 3 4 2 6 7 5 8	3 2 5 8

Задача 31. n работ, 1 исполнитель

Есть n работ и 1 работник. Для каждой работы известно время поступления работы и время выполнения работы работником. Штраф за работу есть разность между временем начала работы и временем поступления. Необходимо минимизировать суммарный штраф.

Формат входных данных

В первой строке выведите минимальный возможный штраф. В следующих строках выведите все последовательности выполнения работ, на которых достигается минимальный штраф. Последовательности работ должны быть отсортированы в лексикографическом порядке. Номер работы отсчитывается от единицы.

Формат выходных данных

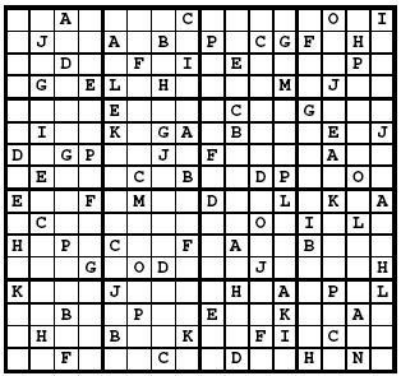
Выведите минимальное время выполнения всех n работ на m приборах.

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

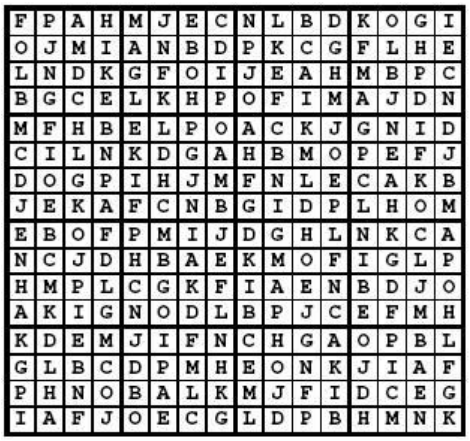
3 1	

Задача 32. Судоку

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



Игра заключается в том, чтобы заполнить все пустые ячейки буквами от A до P так, чтобы каждая буква встретилась только один раз в каждой строке, каждом столбце и каждом квадрате. Первоначальная расстановка букв такова, что удовлетворяет ограничениям, указанным выше, и гарантирует существование решения (решение головоломки на рисунке ниже).



Напишите программу, которая решает Судоку.

Формат входных данных

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

Формат выходных данных

Выведите все решения головоломки. Каждое решение состоит из 16 строк по 16 символов в каждой, которое представляет одно из корректных решений головоломки. Решения можно выводить в любом порядке. Различные решения головоломки нужно разделять пустой строкой.

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

входной файл	*выходной файл*
--A----C-----O-I -J--A-B-P-CGF-H- --D--F-I-E----P- -G-EL-H----M-J-- ----E----C--G--- -I--K-GA-B---E-J D-GP--J-F----A-- -E---C-B--DP--O- E--F-M--D--L-K-A -C--------O-I-L- H-P-C--F-A--B--- ---G-OD---J----H K---J----H-A-P-L --B--P--E--K--A- -H--B--K--FI-C-- --F---C--D--H-N-	FPAHMJECNLBDKOGI OJMIANBDPKCGFLHE LNDKGFOIJEAHMBPC BGCELKHPOFIMAJDN MFHBELPOACKJGNID CILNKDGAHBMOPEFJ DOGPIHJMFNLECAKB JEKAFCNBGIDPLHOM EBOFPMIJDGHLNKCA NCJDHBAEKMOFIGLP HMPLCGKFIAENBDJO AKIGNODLBPJCEFMH KDEMJIFNCHGAOPBL GLBCDPMHEONKJIAF PHNOBALKMJFIDCEG IAFJOECGLDPBHMNK -1

Задача 33. Поиск Гамильтонова цикла

Имеется полный граф с n вершинами. Каждая вершина задается своими целочисленными координатами на плоскости. Расстояние $d(i, j)$ между вершинами i и j задается по следующим образом: $d(i, j) = |x_i - x_j| + |y_i - y_j|$. Нужно найти гамильтонов цикл минимального веса в этом графе.

Формат входных данных

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

Формат выходных данных

В первой строке выведите вес найденного гамильтонова цикла.

Во второй строке – $n + 1$ чисел, номера вершин в порядке обхода в цикле.

входной файл	*выходной файл*
5 3 7	22 1 5 4 2 3 1

7 2 6 3 8 8 4 7	

Задача 34. Job-shop: *m* работ

Есть n станков и m работ. Каждая работа состоит из s_i этапов. Этап характеризуется парой (k, t) чисел, где k – номер станка, а t – продолжительность этапа. Для каждой работы порядок этапов строго задан. Любой этап можно приостановить в любой момент и позже продолжить с того же момента. В каждый момент времени любая работа может выполняться только на одном станке и любой станок может выполнять только одну работу.

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

Нужно найти гамильтонов цикл минимального веса в этом графе.

Формат входных данных

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

Формат выходных данных

Выведите минимальное время выполнения.

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

Задача 35. Задача коммивояжера на плоскости

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

Формат входных данных

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

Формат выходных данных

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

входной файл	*выходной файл*
4 0 0 1 0 1 1 0 1	4.000000

Задача 36. Изоморфизм графов

Даны два неориентированных графа без петель. Требуется проверить, являются ли графы изоморфными, а также найти пример изоморфизма и число изоморфизмов.

Формат входных данных

В первой строке находятся два числа n_1 и m_1– число вершин и ребер первого графа соответственно ($1 \le n_1 \le 16$).

В каждой из следующих m_1 строк расположена пара (u, v) чисел от 0 до $n_1 - 1$, означающая, что вершины u и v соединены ребром.

Далее абсолютно аналогичным образом задается и второй граф.

Формат выходных данных

Выведите в единственной строке NO, если графы не являются изоморфными.

В противном случае выведите в первой строке YES.

Во второй строке – номера вершин второго графа в порядке их соответствия вершинам первого графа, то есть попарно различные значения $\varphi(0), \varphi(1), \ldots, \varphi(n_1 - 1)$, для которых верно, что вершины u и v первого графа смежны тогда и только тогда, когда смежны вершины $\varphi(u)$ и $\varphi(v)$ второго графа.

Если таких функций φ несколько, выведите любую.

В третьей строке выведите число изоморфизмов первого графа на второй, то есть число таких функций φ.

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

Задача 37. Сбалансированные скульптуры

Полимино – это фигура из одинаковых квадратных плиток, соединенных через общие стороны. Полимино может содержать дыры.)

Скульптурой порядка n назовем полимино со следующими свойствами:

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

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

сбалансированные скульптуры заданного порядка, которые можно получить из заданной скульптуры добавлением блоков. Повороты и сдвиги не допускаются.

Формат входных данных

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

Формат выходных данных

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

входной файл	*выходной файл*
3 1 0 0 0 1	2 # # # # ### _#_

Задача 38. Испытание шаха

Известна легенда, что в древней Лимонии любой претендент на должность визиря при шахе должен был выдержать следующее испытание. Ему дается доска размером $M \times M$ и некоторое количество шахматных фигур: ферзей, ладей, слонов, коней и королей. Претендент должен расставить их на доске таким образом, чтобы ни одна из фигур не била другие фигуры, и все фигуры были выставлены на доске. Если претендент выдерживал испытание, он назначался визирем, а если не выдерживал, то не назначался. Напишите программу, которая будет решать эту головоломку.

Формат входных данных

Первое число во входном файле задает размер доски M $(2 \le M \le 12)$. Следующие пять целых неотрицательных чисел *K, Q, R, B, N* задают соответственно количество королей, ферзей, ладей, слонов и коней, которые требуется расставить. Общее количество фигур не превосходит M^2. Фигуры подобраны так, что искомая расстановка существует.

Формат выходных данных

Вывести доску с расставленными фигурами в виде *M* строк по *M* символов в каждой. Пустые поля обозначаются символом "." (точка), поля с королями – "*K*", ферзями – "*Q*", ладьями – "*R*", слонами – "*B*", конями – "*N*".

входной файл	*выходной файл*
4 1 0 0 1 2	.... B.NN .... K...

Задача 39. Задача категории Б

Квадратная таблица $n \times n$ $(n > 4, n \neq 6)$ называется красивой, если в каждой клетке матрицы записано 0 или 1, при том, что сумма в любой строке, в любом столбце диагонали не превосходит 1, а сумма чисел во всей таблице равна *n*. Найдите любую красивую квадратную таблицу размера $n \times n$.

Формат входных данных

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

Формат выходных данных

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

входной файл	*выходной файл*
5	00010 10000 00100 00001 01000

2.2. Задачи для самостоятельного решения по темам 1.2-1.3

Задача 1. Станки − детали с тремя параметрами

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

Формат входных данных

В первой строке через пробел идут число *n* деталей и число *m* станков ($1 \leq n, m \leq 300\,000$). В каждой из последующих *n* строк записаны через пробел время доставки, время обработки и время доставки на склад очередной детали (неотрицательные целые числа, не превосходящие 10^{13}).

Формат выходных данных

В первой строке выведите минимальное время, полученное алгоритмом. В каждой последующей строке в порядке назначения на станки выведите номер

детали и номер станка, на который она назначена. Полученное время не должно превосходить оптимальное более чем вдвое.

входной файл	*выходной файл*
7 3 2 4 1 0 4 2 1 6 0 1 2 3 0 0 4 3 4 1 2 5 0	11 5 1 2 1 3 2 1 3 4 1 6 1 7 3

Задача 2. Станки – детали с двумя параметрами

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

Формат входных данных

В первой строке находятся натуральные числа n и m ($1 \le n, m \le 300\,000$). В каждой из последующих n строк находятся два числа, разделенных пробелом, – время доставки детали и время обработки очередной детали (неотрицательные целые числа, не превосходящие 10^{13}).

Формат выходных данных

В первой строке выведите минимальное время, полученное алгоритмом. В каждой последующей строке в порядке назначения на станки выведите номер детали и номер станка, на который она назначена. Полученное время не должно превосходить удвоенное оптимальное.

входной файл	*выходной файл*
7 3 10 6 1 15 10 10 5 15 14 8 6 7 4 12	37 5 1 3 2 4 3 1 2 7 3 2 1 6 2

Задача 3. Станки двух типов

Имеется n деталей и $2m$ станков двух типов. Деталь обрабатывается в две 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 число. Необходимо

определить порядок обработки деталей на станках, когда все детали будут обработаны за минимальное время.

Формат входных данных

В первой строке находится число n деталей ($1 \le n \le 300\,000$). Во второй строке – число m станков одного типа ($1 \le m \le 300\,000$). Последующие n строк содержат по два неотрицательных вещественных числа, записанных через пробел, – время обработки каждой детали на станке первого и второго типа.

Формат выходных данных

В первой строке выведите минимальное время, за которое детали можно обработать, с точностью до 5 разрядов после запятой. В каждой последующей строке в порядке назначения на станки выведите номер детали и номер станка, на который она назначена. Станки первого типа имеют номера от 1 до m, второго типа – от $m + 1$ до $2m$. Полученное время не должно быть больше удвоенного оптимального.

входной файл	*выходной файл*
5 3 2 3 1 4 4 2 5 1 2 2	8.00000 1 1 2 2 5 3 2 5 3 2 1 4 5 6 4 3 3 4 4 4

Задача 4. Наименее загруженный работник

Имеется n работ и m работников. Каждая работа характеризуется временем выполнения.

Требуется распределить работы между работниками, чтобы минимально загруженный работник выполнял максимальный объем работы.

Формат входных данных

В первой строке находятся целые числа n и m ($1 \le n, m \le 300\,000$). В каждой из последующих n строк находится время выполнения работы (неотрицательное целое число, не превосходящее 10^{13}).

Формат выходных данных

В первой строке выведите общее время всех работ, которые выполнит минимально загруженный работник.

В последующих n строках для каждой работы выведите номер работника, который ее выполнит.

Время работы минимально загруженного работника не должно быть меньше оптимального более чем в полтора раза.

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

Задача 5. Максимальное число работников

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 *x* единиц времени, при этом число занятых работников было как можно больше.

Формат входных данных

В первой строке находится положительное целое число *x*, а затем в каждой строке находится время выполнения очередной работы (неотрицательное целое число, не превосходящее 10^{13}).

Число работ не превосходит 300 000. Сумма времен выполнения всех работ не меньше *x*.

Формат выходных данных

В первой строке выведите число *k* занятых работников. Далее для каждой работы в отдельной строке выведите номер работника (от 1 до *k*), который будет выполнять соответствующую работу.

Число занятых работников должно быть либо не меньше двух третьих от оптимального, либо меньше оптимального на 1.

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

Задача 6. Наиболее загруженный работник

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

Формат входных данных

В первой строке задается число m работников ($1 \le m \le 300\,000$). В следующей строке задаются последовательно продолжительности работ (неотрицательные целые числа, не превосходящие 10^{13}). Число работ не превосходит 300 000.

Формат выходных данных

В первой строке выведите полученное время работы наиболее загруженного работника, превосходящее минимальное возможное не более чем в 4 / 3 раз. Во второй строке выведите для каждой работы номер ее исполнителя.

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

Задача 7. Сервер и процессоры

Имеется n программ, m одинаковых процессоров и 1 сервер.

Каждая программа характеризуется временем скачивания данных с сервера и временем выполнения ее на процессоре. Необходимо организовать выполнение программ на процессорах, чтобы время завершения последней программы было минимальным.

С сервера не могут одновременно скачиваться данные более чем для одной программы. Скачивание данных для программы не может быть приостановлено. Процессор в любой момент времени может осуществлять только одну операцию (скачивание или выполнение) ровно с одной программой. Процессор не может выполнять программу, пока данные для нее не получены. Выполнение программы также не может быть приостановлено.

Формат входных данных

В первой строке находится число m процессоров ($1 \le m \le 300\,000$). Во второй строке – число n программ ($1 \le n \le 300\,000$). В каждой из последующих n строк находятся время загрузки программы и время ее выполнения (неотрицательные целые числа, не превосходящие 10^{13}).

Формат выходных данных

В первой строке выведите время выполнения всех программ, не более чем в 2 раза превосходящее минимально возможное. В каждой из последующих n строк выведите номер очередной выполняемой программы и номер процессора, на котором будет выполняться эта программа.

входной файл	*выходной файл*
3 6 5 3 4 2 4 3	29 5 1 6 2 1 3 2 1

2 1 8 5 5 4	3 2 4 3

Задача 8. Грузовики и камни

Имеется n камней и m машин в очереди. Камни характеризуются массой, машины – грузоподъемностью. Необходимо определить порядок загрузки, при котором минимизируется число используемых машин.

Известно, что любой камень помещается в любую машину, а также что если искомое размещение существует, то общее число машин не менее чем вдвое превосходит минимально возможное число машин.

Формат входных данных

В первой строке находится число m грузовиков ($1 \le m \le 300\,000$). В следующих m строках записаны грузоподъемности грузовиков – целые числа от 1 до 10^{13}. В следующей строке находится число n камней ($1 \le n,\ n + m \le 500\,000$). Далее в n строках записаны массы камней – целые числа от 1 до 10^{13}.

Формат выходных данных

Если решение существует, то выведите $2m + 1$ строк. В первой – число m, а далее для каждой машины в первой строке должна находиться грузоподъемность грузовика, а во второй – последовательно через пробел массы камней, положенных в грузовик (пустая строка, если в грузовик ничего не положено). Число используемых машин не должно превышать минимально возможное более чем в два раза. Если решений нет, то выведите `no solution`.

входной файл	*выходной файл*
6 15 14 17 18 16 20 10 5 6 8 3 5 3 9 4 5 3	6 20 9 8 3 18 6 5 5 17 5 4 3 3 16 15 14

Задача 9. Упаковка предметов в контейнеры

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

Формат входных данных

В первой строке находится число *n* предметов ($1 \le n \le 300\,000$). Далее в *n* строках записано по два целых числа: масса и объем каждого предмета. Грузоподъемность и объем каждого контейнера одинаковы и равны 100, а каждый предмет помещается в контейнер.

Формат выходных данных

Выведите число *k* контейнеров, не превосходящее удвоенное минимально возможное. В каждой из последующих *k* строк выведите характеристики предметов, попавших в один контейнер.

входной файл	*выходной файл*
8 25 28 1 2 56 10 89 90 12 55 51 41 51 10 49 1	6 89, 90; 56, 10; 51, 41; 51, 10; 12, 55; 25, 28; 1, 2; 49, 1;

Задача 10.

Имеется *n* городов, каждый из которых является либо потребителем, либо поставщиком продукции. Для города *k* число x_k характеризует спрос (отрицательное) или предложение (положительное). Необходимо определить минимальную грузоподъемность машины и маршрут, чтобы объехать все города по разу и удовлетворить потребности (сумма спроса равна сумме предложения).

Формат входных данных

В первой строке находится целое число *n* ($1 \le n \le 300\,000$). Во второй строке – последовательность (x_k) целых чисел, разделенных пробелами ($|x_k| \le 10^{13}$).

Формат выходных данных

В первой строке выведите необходимую грузоподъемность машины, не превосходящую удвоенную минимально возможную. Во второй строке – последовательность городов, разделенных пробелами, в том порядке, в котором машина будет их объезжать.

входной файл	*выходной файл*
15 -4 16 -3 -5 -8 -9 -3 -8 1 7 4 -5	16 16 -9 8 -8 7 -8 -5 5 -5 4 -4 4 -3

8 4 5	1 -3

Задача 11. Задачи с предписанным распределением по процессорам

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. С сервера не могут одновременно скачиваться данные более чем для одной программы. Скачивание данных для программы не может быть приостановлено. Процессор в любой момент времени может осуществлять только одну операцию (скачивание или выполнение) ровно с одной программой. Процессор не может выполнять программу, пока данные для нее не получены. Выполнение программы также не может быть приостановлено.

Формат входных данных

В первой строке находится число n программ ($1 \le n \le 300\,000$). Во второй строке – число m процессоров ($1 \le m \le 300\,000$). В каждой из последующих n строк находятся тройки (x_i, y_i, z_i) целых чисел: время скачивания данных с сервера, время работы на процессоре и номер процессора для i-й программы ($0 \le x_i, y_i \le 10^{13}$, $1 \le z_i \le m$).

Формат выходных данных

В первой строке выведите полученное время завершения выполнения программ, не превосходящее удвоенное оптимальное. Во второй строке выведите номера программ в порядке скачивания данных с сервера.

входной файл	*выходной файл*
4 3 3 10 1 4 5 3 7 6 2 5 5 1	24 1 2 3 4

Задача 12.

Решите задачу коммивояжера на плоскости с евклидовой метрикой, если положения городов задаются их координатами.

Формат входных данных

В первой строке находится целое число n городов ($3 \le n \le 3000$).

В каждой из последующих n строк находятся пары (x, y) действительных чисел – координаты городов (целые числа, по модулю не превосходящие 10^9).

Формат выходных данных

В первой строке выведите длину маршрута с точностью до трех знаков после запятой. Эта длина не должна превосходить оптимальную более чем в два раза.

Во второй строке выведите вершины полученного маршрута в порядке посещения.

входной файл	*выходной файл*
4 0 0 1 1 0 1 1 0	4.000 3 1 4 2

Задача 13. Сумма векторов

Необходимо указать такой порядок суммирования векторов в R^m, при котором все частичные суммы попадают в шар минимального радиуса в евклидовой метрике. Известно, что сумма всех векторов равна $\vec{0}$.

Формат входных данных

В первой строке находится число *n* векторов ($1 \le n \le 3000$).

Во второй строке – размерность *m* пространства ($1 \le m \le 10$).

Далее идут *n* строк, содержащих по *m* целых чисел, разделенных пробелом, – координаты каждого вектора (каждое число не превосходит 10^9 по абсолютной величине).

Формат выходных данных

Выведите *n* строк, содержащих номера векторов в порядке суммирования. Радиус шара, определяемый предложенным порядком суммирования, не должен превосходить минимально возможный более чем в $2 \cdot m$ раз.

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

Задача 14. Детали с суммарным временем обработки

Имеется *m* станков и детали, о которых известна только информация о суммарном времени *s* их обработки. Каждая деталь характеризуется временем обработки. Станок обрабатывает любую деталь сразу, все станки одинаковы. При этом детали поступают по очереди, причем нет никакой информации о следующей детали и их числе. Поступившая деталь должна быть сразу назначена на один из станков, и это назначение не может быть изменено позже.

Необходимо определить порядок обработки деталей на станках, когда все детали будут обработаны за минимальное время.

Формат входных данных

В первой строке находится целое число m станков ($1 \le m \le 300\,000$). Во второй строке – целое число s, суммарное время обработки ($0 \le s \le 10^{18}$). В третьей строке находятся положительные целые числа, разделенные пробелом,– информация о времени обработки очередной детали. Число деталей не превосходит 300 000.

Формат выходных данных

В первой строке выведите полученное время обработки деталей, не превосходящее оптимальное более чем в $2m / (m + 1)$ раз. Во второй строке выведите для каждой детали номер станка, на котором она обрабатывается.

входной файл	*выходной файл*
4 154 12 10 5 7 16 11 15 17 18 9 14 3 10 7	39 1 2 2 3 3 1 3 4 4 1 2 4 2 1

Задача 15. Работники со списком работ

Дано m работников и n работ. Каждый работник может выполнить любую работу. Каждый работник характеризуется скоростью исполнения. Каждая работа характеризуется временем, которое понадобится работнику с единичной скоростью, чтобы ее выполнить. Необходимо реализовать алгоритм, позволяющий таким образом распределить работы, чтобы минимизировать время выполнения работ самым загруженным (по времени) работником, при условии, что все работники выполняют почти одинаковое число работ (для некоторого k каждый работник – либо k, либо k - 1 работ).

Формат входных данных

В первой строке находятся разделенные пробелом число n работ и число m работников ($1 \le n, m \le 3000$).

Далее следует n строк действительных чисел, каждое из которых характеризует время выполнения работы работником единичной скорости.

Затем следует m строк действительных чисел, характеризующих скорость выполнения работы работником.

Формат выходных данных

В первой строке выведите время выполнения работ самым загруженным работником.

Далее для каждой работы в отдельной строке выведите номер ее исполнителя (целое число от 1 до m). Полученное время работы не должно превосходить удвоенное оптимальное.

входной файл	*выходной файл*
4 2 1.0 3.0 3.0 5.0 1.0 1.0	6.0 1 2 2 1

Задача 16. Назначение работ рабочим с различной производительностью

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.

Формат входных данных

В первой строке находится число n работ ($1 \le n \le 3000$).

Во второй строке – n натуральных чисел, характеризующих время выполнения каждой из работ работником единичной производительности.

В третьей строке – число m работников ($1 \le m \le 3000$).

В четвертой строке – m натуральных чисел, характеризующих производительности этих работников.

Формат выходных данных

В первой строке выведите время выполнения всех работ с пятью знаками после запятой. Во второй строке выведите для каждой работы номер ее исполнителя (число от 1 до m). Полученное время выполнения всех работ не должно превосходить удвоенное оптимальное.

входной файл	*выходной файл*
9 2 2 2 2 2 2 2 2 4 16 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2.00000 2 3 4 5 6 7 8 9 1

Задача 17. n работ, 1 исполнитель

Есть n работ и один работник. Для каждой работы есть время работы и время подготовки. Штраф за работу есть разность между временем начала работы и временем возможного начала работы.

Необходимо минимизировать суммарный штраф.

Формат входных данных

Первая строка содержит число n работ ($1 \le n \le 300\,000$). Следующие n строк содержат по два целых числа от 0 до 10^8: время задержки и время выполнения.

Формат выходных данных

В первой строке выведите число n работ и полученный штраф через пробел. Во второй строке выведите номера работ в порядке выполнения. Штраф не должен превосходить минимально возможный более чем в n - 1 раз.

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

Задача 18. Обработка в две стадии

Имеется n деталей, m станков первого типа и один станок второго типа. Деталь обрабатывается в две стадии: сначала на станке первого типа, затем на станке второго типа. Каждая деталь характеризуется двумя параметрами: временем обработки на станке первого типа и временем обработки на станке второго типа.

Станок обрабатывает любую деталь сразу. Необходимо определить порядок обработки деталей на станках, при котором все детали будут обработаны за минимальное время.

Формат входных данных

В первой строке находится число m станков первого типа ($1 \le m \le 300\,000$). Во второй строке – число n деталей ($1 \le n \le 300\,000$). В каждой из последующих n строк находится по два целых числа a и b, разделенных пробелом, где a – время обработки i-й детали на станке первого типа, b – на станке второго типа ($0 \le a, b \le 10^{13}$).

Формат выходных данных

В первой строке выведите полученное время обработки всех деталей. В каждой из следующих m строк выведите для соответствующего станка первого типа число обработанных им деталей и их номера в порядке обработки. В последней строке выведите номера деталей в порядке обработки на втором станке. Найденное время работы не должно превышать удвоенное оптимальное.

входной файл	*выходной файл*
3 6 20 3 16 8	58 2 1 2 2 3 4 2 5 6

19 1 16 5 15 20 15 6	5 1 2 4 6 3

Задача 19. Задача коммивояжера с прямоугольной метрикой

Решите задачу коммивояжера на плоскости, если положение городов задается координатами, а расстояние считается в прямоугольной метрике.

Формат входных данных

В первой строке находится число n городов ($1 \le n \le 10\ 000$). В каждой из последующих n строк находится по два числа целых x_i и y_i, разделенных пробелом, – координаты i-го города ($|x_i|, |y_i| \le 10^9$).

Формат выходных данных

В первой строке выведите $n + 1$ целых чисел – города в порядке их обхода. Номер первого города в списке должен совпадать с номером последнего. Во второй строке выведите длину маршрута. Длина полученного маршрута не должна превосходить длину оптимального более чем вдвое.

входной файл	*выходной файл*
13 4 4 -4 -4 -4 4 4 -4 3 3 -3 -3 -3 3 3 -3 2 2 -2 -2 -2 2 2 -2 1 1	1 5 9 13 3 7 11 2 6 10 12 8 4 1 46

Задача 20. Разделение на зоны

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

Под изменением цены понимается абсолютная величина разности новой и старой стоимости.

Формат входных данных

В первой строке содержатся два целых числа k и n – число зон и число станций соответственно ($1 \le k \le n \le 15$, $k \le 5$).

Далее следует n строк по n чисел, разделенных пробелом. На месте (i, j) стоит стоимость проезда от станции номер i до станции номер j (если 0, то маршрута между этими станциями нет). Стоимости – целые числа от 0 до 10^9.

Формат выходных данных

В первых k строках выведите распределение станций по зонам: в i-й строке отсортированные по возрастанию номера станций, принадлежащих i-й зоне, разделенные пробелом, а также стоимость проезда в зоне.

Строки должны быть отсортированы по возрастанию по номеру первой станции. Станции нумеруются с нуля.

Далее выведите k строк, i-я из которых должна содержать стоимость проезда между i-й зоной и всеми остальными, или 0, если между зонами нет проезда.

Максимальное полученное изменение цены проезда не должно превышать удвоенное оптимальное.

входной файл	*выходной файл*
3 5 0 4 0 0 1 4 0 2 5 0 0 2 0 3 0 0 5 3 0 1 1 0 0 1 0	0 1 3 4.5 2 0.0 4 0.0 0.0 2.5 1.0 2.5 0.0 0.0 1.0 0.0 0.0

Задача 21. Последовательные станки

Имеется n деталей и m станков. Для каждой детали известно время выполнения на каждом станке и последовательность прохождения станков каждой деталью.

Необходимо определить порядок обработки деталей, при котором последняя деталь обработается как можно раньше. В последовательности прохождения станков деталью ни один станок не встречается более одного раза. Запрещено останавливать станок до окончания обработки очередной детали.

Формат входных данных

В первой строке содержатся два целых числа n и m ($1 \le n \le 3000$, $1 \le m \le 300$). Далее следуют n блоков. При этом i-й блок начинается со строки, содержащей число k_i этапов в процессе обработки i-й детали.

Далее следуют k_i строк, задающих последовательность прохождения станков i-й деталью, каждая строка содержит два положительных целых числа – номер станка и время обработки. Время обработки не превосходит 10^{12}.

Формат выходных данных

В первой строке выведите время окончания обработки последней детали. Далее выведите m строк, по n чисел в каждой.

В *i*-й строке *j*-е число соответствует времени, когда *i*-й станок начнет обработку *j*-й детали или равно 0 в случае, если *j*-ю деталь не надо обрабатывать на *i*-м станке. Затраченное время не должно превосходить оптимальное более, чем в $(n + 1) / 2$ раз.

входной файл	*выходной файл*
5 5 4 1 10 2 10 3 10 4 10 4 2 10 3 10 4 10 5 10 4 3 10 4 10 5 10 1 10 4 4 10 5 10 1 10 2 10 4 5 10 1 10 2 10 3 10	40 0 0 30 20 10 10 0 0 30 20 20 10 0 0 30 30 20 10 0 0 0 30 20 10 0

Задача 22. Частичный порядок обработки деталей

Имеется n деталей (пронумерованы от 1 до n) и m последовательных станков. Для каждой детали известно время обработки на каждом станке и частичный порядок предшествования деталей.

Необходимо определить порядок обработки деталей, при котором последняя деталь обработается как можно раньше.

Формат входных данных

В первой строке находятся целые числа n и m ($2 \le n, m \le 100$). Далее в n строках, по m элементов в каждой, идут целые числа x ($1 \le x \le 10^{13}$).

Если число x находится в i-й строке на j-м месте, то время обработки i-й детали на j-м станке равно x.

В следующей строке идет число k деталей, от которых зависит хотя бы одна другая деталь ($0 \le k \le 100$).

Далее в k строках записаны числа: первое число в строке – номер a детали, далее идут номера деталей, зависящих от нее по времени (должны выполнится позже a). Гарантируется существование решения задачи при заданных входных условиях.

Формат выходных данных

Выведите в первой строке минимальное время обработки последней детали. Далее в m строках выведите перестановки чисел от 1 до n – в i-й строке порядок обработки деталей на i-м станке (если вариантов несколько, то вывести любой). Полученное время обработки деталей не должно превосходить оптимальное более чем в $(m+1)/2$ раз.

входной файл	*выходной файл*
4 2 2 2 2 2 2 2 1 2 1 3 2	9 4 3 2 1 4 3 2 1

Задача 23. Минимальная стоимость проезда

Имеется n городов и m машин. Необходимо объехать каждый город ровно один раз, при этом минимизировать стоимость проезда. Города задаются координатами точек на плоскости. Изначально машины могут располагаться в любых городах. Каждой машине необходимо вернуться в тот же город, из которого она выехала.

Формат входных данных

В первой строке находятся два числа n и m – число городов и число машин соответственно ($1 \le m \le n \le 10\,000$). Далее идут n строк, содержащих по паре целых чисел, не превосходящих 10^9 по абсолютной величине,– координаты городов.

Формат выходных данных

В первой строке выведите полученную сумму длин маршрутов. Далее в m строках выведите число посещенных очередной машиной городов и номера этих городов в порядке посещения. Сумма длин маршрутов не должна превосходить удвоенную минимально возможную.

входной файл	*выходной файл*
6 1 171 40 172 58 209 95 228 71 226 47 198 41	180.701 6 1 2 3 4 5 6 1

Задача 24. Станки – детали

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, при котором время окончания обработки последней обрабатываемой детали было бы минимальным.

Формат входных данных

В первой строке содержатся два целых числа n и m, где n – число деталей, а m – число станков ($1 \le n \le 100\,000$, $1 \le m \le 50\,000$). Во второй строке содержатся n целых чисел от 1 до 100 000 – время обработки каждой детали.

Формат выходных данных

В первой строке выведите время окончания обработки всех деталей. Во второй строке для каждой детали выведите номер станка, на котором она будет обработана. Полученное время работы не должно превосходить $4/3$ от оптимального.

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

В приведенном примере детали с номерами 1 и 2 будут обрабатываться на первом станке, а детали 3 и 4 – на втором. Или, наоборот. Тогда первый станок завершит выполнение через $t_1 = 5$ единиц времени с начала обработки, а второй соответственно через $t_2 = 6$. Таким образом, минимальное время завершения обработки $T = \max\{ t_1, t_2 \} = 6$.

Задача 25. Распределенная задача с сервером

Есть сервер и два процессора. Есть n_1 задач, которые могут выполняться только на первом процессоре, и n_2 задач, которые могут выполняться только на втором процессоре. Перед выполнением каждой задачи, она сначала должна загрузиться с сервера на необходимый процессор, что занимает время $s_{k,i}$ ($s_{k\,i} > 0$), для задачи с номером i, исполняемой только на процессоре k. При этом процессе сервер и процессор k могут заниматься только загрузкой этой задачи. Сразу после загрузки процессор начинает исполнение только что загруженной задачи, что занимает у него время $p_{k,i}$ ($p_{k,i} > 0$). После окончания выполнения загруженной задачи процессор может загружать следующую предназначенную для него задачу. Т. е. в каждый момент времени сервер может заниматься загрузкой не более одной задачи и каждый процессор выполняет не более чем одну задачу в каждый момент времени. Сервер не может загрузить две задачи подряд, которые должны исполняться на одном процессоре. Процессор не может одновременно загружать одну задачу и исполнять другую. Необходимо найти такое расписание, при котором выполнялись бы все задачи и время окончания выполнения последней задачи было бы как можно меньшим.

Формат входных данных

В первой строке записаны числа n_1 и n_2 ($1 \le n_1, n_2 \le 100\,000$). Далее идет n_1 пар ($p_{1,i}, s_{1,i}$) чисел. Далее идет n_2 пар ($p_{2,i}, s_{2,i}$) чисел. Все числа положительные целые, не превосходящие 10^{13}.

Формат выходных данных

В первой строке выведите время окончания выполнения последней задачи.

В последующих $n_1 + n_2$ строках выведите порядок, в котором задачи загружаются с сервера. Задача задается двумя числами – номер процессора и номер задачи. Задачи нумеруются с 1, в порядке появления во входном файле. Полученное расписание не должно превосходить по общему времени оптимальное более чем в полтора раза.

входной файл	*выходной файл*
1 2 2 5 4 1 5 5	15 2 2 1 1 2 1

В примере одна задача для 1-го процессора: время выполнения 2, время загрузки 5.

Две задачи для 2-го процессора.

Задачи можно выполнить за 15 единиц времени.

Первой загружается задача 2 для второго процессора, потом задача 1 для первого процессора, потом задача 1 для второго процессора.

Задача 26. Нераспределенная задача с сервером

Есть один сервер и два процессора. Есть n задач, каждая из которых сначала должна быть загружена с сервера на любой процессор, а сразу после этого исполняться на этом процессоре.

При этом сервер может заниматься не более чем одной загрузкой в каждый момент времени и каждый процессор может заниматься исполнением не более чем одной задачи в каждый момент времени.

Процессор не может одновременно исполнять одну задачу и загружать другую.

Для задачи i известно время s_i загрузки с сервера на процессор и время p_i исполнения на процессоре.

Необходимо найти такое расписание, при котором выполнились бы все задачи и время окончания выполнения последней задачи было бы минимальным возможным.

Формат входных данных

В первой строке находится число n, далее следуют n строк по два положительных целых числа s_i и p_i в каждой ($1 \le n \le 300\,000$, $1 \le s_i, p_i \le 10^{13}$).

Формат выходных данных

Выведите в первой строке время завершения обработки всех задач, а во второй строке – порядок загрузки задач с сервера (перестановку чисел от 1 до n).

Полученное время не должно превосходить оптимальное более чем в 3 / 2 раза.

входной файл	*выходной файл*
4 12 3 8 7 3 12 1 14	33 3 1 4 2

Задача 27. Мультиразрез

Дан взвешенный неориентированный граф и зафиксировано некоторое подмножество T его вершин. Необходимо удалить из графа множество ребер с минимальным суммарным весом так, чтобы все вершины множества T оказались в разных компонентах связности.

Формат входных данных

Первая строка содержит число n вершин и число m ребер графа ($1 \le n \le 150$). Следующие m строк содержат тройки (u, v, w) чисел, свидетельствующие о наличии ребра между вершинами u и v с весом w ($1 \le u, v \le n$, $0 \le w \le 1\,000\,000$), В $m+2$-й строке находится число k вершин, входящих во множество T ($2 \le k \le n$). В следующей строке через пробел записаны k попарно различных чисел от 1 до n – номера вершин, входящих во множество T.

Формат выходных данных

В первой строке выведите вес удаляемого множества. Во второй строке – число r ребер, входящих в это множество. В следующих r строках выведите номера ребер, входящих в это множество, по одному в строке. Ребра нумеруются, начиная с 1, в том же порядке, в котором они были заданы на входе. Вес удаляемого множества должен отличаться от оптимального не более чем в $2 - 2 / |T|$ раз.

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

Задача 28. Flow-shop permutation

Имеется N работ и m приборов. Каждая работа обслуживается приборами $1, 2, \ldots, m$ в этом порядке. Обслуживание требования следующим по порядку прибором может быть начато не ранее завершения его обслуживания предыдущим прибором. Требуется составить такое расписание обслуживания всех работ на всех приборах, чтобы время обслуживания последней работы было минимально. В любой момент времени каждая работа должна обслуживаться не более чем одним прибором и каждый прибор должен обслуживать не более одной работы. При этом никакой прибор не может простаивать, если некоторая работа готова к выполнению на нем.

Формат входных данных

В первой строке записаны два числа n и m ($1 \le n, m \le 1000$). Далее следует n строк по m чисел в каждой: время выполнения i-й работы на j-м приборе – неотрицательное целое число, не превосходящее 10^9.

Формат выходных данных

В первой строке выведите время выполнения всех n работ на m приборах.

В каждой из следующих m строк выведите по n чисел − номера работ в порядке выполнения на соответствующем станке. Полученное время выполнения не должно превышать оптимальное, умноженное на $\lceil m / 2 \rceil$.

входной файл	*выходной файл*
2 3 1 4 3 5 3 7	16 1 2 1 2 1 2

Задача 29. Open-shop

Имеется n работ и m приборов. Требуется составить такой маршрут обслуживания каждой работы на каждом приборе, чтобы время обслуживания последней работы было минимально.

В любой момент времени каждая работа должна обслуживаться не более чем одним прибором и каждый прибор должен обслуживать не более одной работы.

При этом никакой прибор не может простаивать, если некоторая работа готова к выполнению на нем.

Формат входных данных

В первой строке записаны числа n и m ($1 \le n, m \le 1000$).

Далее следуют n строк по m чисел в каждой.

В i-й строке j-е число задает время выполнения i-й работы на j-м приборе (каждое число неотрицательное целое, не превосходящее 10^9).

Формат выходных данных

В первой строке выведите время выполнения всех n работ на m приборах.

Далее в каждой из $n \cdot m$ строк выведите по два числа: номер работы и номер станка.

Каждая такая пара чисел соответствует событию начала выполнения соответствующей работы на соответствующем станке. Все события должны следовать в хронологическом порядке. Полученное время выполнения не должно превышать удвоенное оптимальное.

входной файл	*выходной файл*
3 4 4 12 4 6 8 6 2 10 2 4 18 3	27 1 4 2 2 3 3 1 2 2 1 2 4 1 3 3 1 3 2 1 1 2 3 3 4

Задача 30. Поиск Гамильтонова цикла

Имеется полный граф на n вершинах. Каждая вершина задается своими целочисленными координатами на плоскости. Вес ребра $\{ u, v \}$ задается по следующей формуле:

$$d(u, v) = |x_u - x_v| + |y_u - y_v|.$$

Вам нужно найти гамильтонов цикл минимального веса в этом графе.

Формат входных данных

Первая строка содержит одно целое число n ($3 \le n \le 10\,000$). Далее следуют n строк с координатами вершин: i-я строка задает координаты i-й вершины, разделенные пробелом. Координаты – целые числа, не превосходящие по модулю 100 000.

Формат выходных данных

Первая строка должна содержать вес найденного гамильтонова цикла, вторая строка – $(n+1)$ чисел, номера вершин в порядке обхода в цикле. Вес найденного цикла не должен превосходить удвоенный вес оптимального.

входной файл	*выходной файл*
5 3 7 7 2 6 3 8 8 4 7	22 1 5 4 2 3 1

Задача 31. Пожарные станции

В стране расположены n городов, связанных двусторонними дорогами. Из каждого города можно добраться в каждый.

Необходимо разместить в городах k пожарных станций таким образом, чтобы максимальное из расстояний от города до пожарной станции было минимально.

Формат входных данных

В первой строке расположены три числа: число n городов, число m дорог и число k пожарных станций ($1 \le k \le n \le 1000$, $k \le 200$). В последующих m строках расположено по три натуральных числа i, j и d, где d – длина дороги из города i в город j ($1 \le d \le 10^9$). Каждая пара городов соединена не более чем одной дорогой.

Формат выходных данных

В первой строке выведите полученное максимальное расстояние от города до пожарной станции.

Во второй строке выведите k номеров городов, в которых размещаются станции. Полученное расстояние не должно превосходить удвоенное оптимальное.

входной файл	*выходной файл*
6 7 3 1 2 10 1 3 5 2 4 3 3 4 1 4 5 19 4 6 15	3 1 4 5

5 6 2	

Задача 32. Наименьшее максимальное паросочетание

Паросочетанием называется независимое множество ребер графа, то есть множество ребер, в котором любые два ребра не смежны (не имеют общих вершин).

Максимальным паросочетанием называется паросочетание, которое не содержится ни в каком другом паросочетании.

Наименьшим максимальным паросочетанием называется максимальное паросочетание наименьшей мощности. Необходимо найти в заданном графе наименьшее максимальное паросочетание.

Формат входных данных

Первая строка содержит число n вершин и число m ребер ($1 \le n \le 500\,000$, $0 \le m \le 500\,000$).

Каждая из последующих m строк содержит пару номеров концов очередного ребра (числа от 1 до n). Гарантируется, что граф не содержит петель и кратных ребер.

Формат выходных данных

В первой строке необходимо вывести мощность максимального паросочетания, не более чем в 2 раза превосходящую минимальную возможную.

Во второй строке – номера (начиная с 1) входящих в него ребер.

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

Задача 33. Всемирная выставка

Организаторы Всемирной выставки 2584 года (объявленного годом Фибоначчи) решили, что давно пора уже превзойти успех Эйфеля с его экспонатом 1889 года. Все-таки наука и техника сделали огромные шаги за долгое время. В результате многочисленных обсуждений они пришли к тому, что за основу будет взята идея другого экспоната Всемирной выставки – давно забытый Атомиум (1958 год), который можно увидеть на фотографии. Основное отличие, конечно, не в том, что число комнат будет увеличено с 9 до n штук, а в том, что в некоторых комнатах будут установлены голографические проекторы. Каждый такой проектор будет создавать объемное изображение

не в комнате, где и без него будет много интересного, а в запутанных переходах. Более точно, один проектор, установленный в комнате, может работать во всех переходах, ведущих в эту комнату, по всей длине перехода (каким запутанным он бы ни был!), и только в них. И, конечно же, нужно обеспечить каждый коридор хотя бы одним таким проектором. Но вот беда, денег, как всегда, на все не хватает, поэтому нужно минимизировать затраты на проекторы. Или хотя бы потратить не более чем вдвое больше по сравнению с минимально возможной суммой, раз уж точное решение этой задачи до сих пор не могут эффективно находить нейронные сети и квантовые компьютеры.

Формат входных данных

В первой строке содержатся число n комнат и число m переходов ($1 \le n \le 200\,000$, $0 \le m \le 500\,000$).

В каждой из следующих m строк содержится пара номеров комнат (числа от 1 до n), соединенных очередным переходом. Гарантируется, что каждый переход соединяет две разных комнаты и никакие две комнаты не соединены более чем одним переходом.

Формат выходных данных

В первой строке выведите число k используемых Вами проекторов. Во второй строке выведите k разделенных пробелом номеров комнат с проекторами.

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

Задача 34. Два станка

Есть два одинаковых станка и n деталей. Для каждой детали известно время ее обработки на станке. Обработка детали выполняется только на одном станке и не может быть приостановлена. Необходимо распределить детали между станками, чтобы время завершения обработки последней детали было минимально возможным.

Формат входных данных

В первой идет число n деталей ($1 \le n \le 300\,000$). Во второй строке записано n чисел, разделенных пробелом, – время обработки каждой детали (неотрицательные целые числа, не превосходящие 10^{13}).

Формат выходных данных

В первой строке выведите время, полученное алгоритмом. Во второй строке выведите для каждой детали номер станка, на который она назначена. Полученное время обработки не должно превосходить минимально возможное более чем на 4 %.

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

Задача 35. Искусство

Василий создает интернет-ресурс, на котором определяются рейтинги самых разных произведений искусства, будь то картины, книги, музыкальные композиции, фотографии или фильмы. Рейтинг каждого произведения определяется в результате голосования. При этом в голосовании пользователь не выставляет какие-либо субъективные оценки, а просто сравнивает одно с другим и выбирает лучшее. Каждое сравнение одного произведения с другим будем называть *голосом*. И задача, которая стоит перед Василием состоит в том, чтобы ранжировать произведения искусства с учетом результатов голосования. Хорошо, когда можно просто расставить все произведения по порядку, чтобы каждый пользователь говорил, что произведение с меньшим рангом лучше, чем произведение с большим рангом. Но иногда это принципиально невозможно. Поэтому Василий ищет порядок, учитывающий в описанном смысле максимальное число голосов.

Формат входных данных

В первой строке записано число n произведений искусства и число m голосов ($1 \le n, m \le 300\,000$). Все произведения искусства занумерованы числами от 1 до n. В каждой из следующих m строк записана тройка чисел x, y и z ($1 \le x, y \le n$, $x \ne y$, $1 \le z \le 2$), соответствующих одному голосу, где x и y – номера сравниваемых произведений, а z равно 1, если произведение x понравилось пользователю больше, чем y, или 2 в противном случае.

Формат выходных данных

В первой строке выведите число голосов, которые будут учтены в предложенном порядке.

Во второй строке выведите перестановку чисел от 1 до n – порядок, при котором не меньше половины от максимально возможного числа голосов отдает предпочтение произведению, идущему в этом порядке раньше.

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

В примере максимальное число голосов, которые можно учесть, равно двум. Предложенный ответ учитывает только один – третий.

3. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

3.1. Набор тестов

При обучении основам алгоритмизации студенты решают задачи. Набор разнообразных задач, которые предлагаются для решения, формируется преподавателями. Задачи разбиты на шесть тем: деревья поиска, рекуррентные соотношения, структуры данных, алгоритмы на графах, организация перебора вариантов, приближённые алгоритмы. Задачи требуют творческого подхода и развивают креативное мышление.

Задачи хранятся в системе. Каждая задача состоит из двух важнейших частей: из текста условия и набора тестов.

Условия задач в системе по традиции имеют единую структуру и стиль оформления. В условие обычно входят следующие элементы: легенда (сказка) или формальная постановка; описание формата входных данных; описание формата выходных данных; примеры. Студенты могут читать условия всех задач, находящихся в системе.

Тесты к задаче. Под тестом для задачи понимается набор входных и выходных данных. Тесты могут выглядеть следующим образом:

Тест №1: вход 1, выход 1
Тест №2: вход 2, выход 2
Тест №3: вход 4, выход 5
…
Тест №16: вход 10000000, выход 640540120

Тесты составляются преподавателями. Студентам содержимое тестов недоступно (кроме примеров из условия).

3.2. Средства диагностики

Решением является программа, написанная на некотором языке программирования. Наличие заготовленных тестов позволяет автоматически проверить правильность решения. Для этого нужно запускать программу, передавая ей входные данные, и сравнивать тот ответ, который выдаёт программа, с правильным ответом, который хранится в базе. Эту работу и делает iRunner. Образовательная платформа при этом предоставляет дружественный интерфейс преподавателю курса для назначения студентам задач (индивидуальных, общих, дополнительных, штрафных), для отслеживания прогресса, для расчёта итоговых оценок по курсу (журнал, ведомость) [1,3].

ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

3.2.Рекомендуемая литература

Основная

1. Опыт использования образовательной платформы Insight Runner на факультете прикладной математики и информатики Белорусского государственного университета. Роль университетского образования и науки в современном обществе : материалы междунар. науч. конф., Минск, 26–27 февр. 2019 г. / редкол.: А. Д. Король (пред.) [и др.]. – Минск : БГУ, 2019. – С. 263 – 267.

2. Сборник задач по теории алгоритмов : учеб.-метод. пособие / В. М. Котов [и др.] – Минск : БГУ, 2017. – 183 с.

3. Соболь, С. А. Методика преподавания дисциплин по теории алгоритмов с использованием образовательной платформы iRUNNER / С. А. Соболь, В. М. Котов, Е. П. Соболевская // Электронный науч.-методич. журнал «Педагогика информатики». – 2020 – № 2. http://pcs.bsu.by/2020_2/6ru.pdf

4. Сборник задач по теории алгоритмов. Структуры данных : учеб.-метод. пособие / С. А. Соболь [и др.] – Минск : БГУ, 2020. – 159 с.

Дополнительная

5. Алгоритмы: построение и анализ / Т. Кормен [и др.] – М. : Вильямс, 2005. – 1296 с.

6. Волчкова, Г. П. Сборник задач по теории алгоритмов. Организация перебора вариантов и приближенные алгоритмы : для студентов спец. 1- 31 03 04 «Информатика» / Г. П. Волчкова, В. М. Котов, Е. П. Соболевская. – Минск : БГУ, 2008. – 59 с.

7. Ковалев, М. М. Матроиды в дискретной оптимизации / М. М. Ковалев – М. : Едиториал УССР, 2003. – 224 с.

8. Ковалев, М.Я. Теория алгоритмов. Ч. 2. Приближенные алгоритмы : курс лекций / М. Я. Ковалев, В. М. Котов, В. В. Лепин – Минск : БГУ, 2003. – 147 с.

9. Котов, В.М. Алгоритмы для задач разбиения и упаковки / В. М. Котов – Минск : БГУ, 2001. – 97 с.

10. Котов, В. М. Алгоритмы и структуры данных : учеб. пособие / В. М. Котов, Е. П. Соболевская, А. А. Толстиков. – Минск : БГУ, 2011. – 267 с. – (Классическое университетское издание).

11. Пападимитриу, Х.Комбинаторная оптимизация. Алгоритмы и сложность / Х. Пападимитриу, К. Стайглиц – М. : Мир, 1985. – 512 с.

12. Рейнтгольд, Э. Комбинаторные алгоритмы. Теория и практика / Э. Рейнтгольд., Ю. Нивергельт, Н. Део – М. : Мир, 1980. – 466 с.

13. Танаев, В.С. Теория расписаний. Многостадийные системы / В. С. Танаев, Ю. Н. Сотсков, В. А. Струсевич – М. : Наука, 1989. – 912 с.

14. Танаев, В.С.Теория расписаний. Групповые технологии / В. С. Танаев, М. Я. Ковалев, Я. М. Шафранский – Минск : Ин-т техн. кибернетики НАН Беларуси, 1998. – 289 с.

15. Теория алгоритмов : учеб. пособие / П. А. Иржавский [и др.] – Минск : БГУ, 2013. – 159 с.

16. Kovalyov, M.Y. Approximation scheduling algorithms: a survey / M. Y. Kovalyov [et al.] // Optimization. – 1989. –№. 6. – P. 859–878.

3.3.Электронные ресурсы

1. Образовательный портал БГУ [Электронный ресурс]. – Режим доступа: https://edufpmi.bsu.by/course/view.php?id=96. – Дата доступа: 02.09.2021.

2. Образовательная платформа Insight Runner [Электронный ресурс]. – Режим доступа: https://acm.bsu.by. – Дата доступа: 02.09.2021.

3. Ковалев, М.Я. Теория алгоритмов. Часть 2. Приближенные алгоритмы / М.Я.Ковалев, В.М.Котов, В.В.Лепин. - Мн.: БГУ, 2002. - 157 с. [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/27561. – Дата доступа: 02.12.2021.

4. Котов, В. М. Алгоритмы и структуры данных : учеб. пособие / В. М. Котов, Е. П. Соболевская, А. А. Толстиков. – Минск : БГУ, 2011. – 267 с. – (Классическое университетское издание). [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/8522. – Дата доступа: 02.12.2021.

5. Теория алгоритмов : учеб. пособие / П. А. Иржавский [и др.] – Минск : БГУ, 2013. – 159 с. [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/91612. – Дата доступа: 02.12.2021.

6. Сборник задач по теории алгоритмов : учеб.-метод. пособие / В. М. Котов [и др.] – Минск : БГУ, 2017. – 183 с. [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/181529. – Дата доступа: 02.12.2021.

7. Опыт использования образовательной платформы Insight Runner на факультете прикладной математики и информатики Белорусского государственного университета. Роль университетского образования и науки в современном обществе : материалы междунар. науч. конф., Минск, 26–27 февр. 2019 г. / редкол.: А. Д. Король (пред.) [и др.]. – Минск : БГУ, 2019. – С. 263 – 267. Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/231914. – Дата доступа: 02.12.2021.

8. Сборник задач по теории алгоритмов. Структуры данных : учеб.-метод. пособие / С. А. Соболь [и др.] – Минск : БГУ, 2020. – 159 с. Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/255033. – Дата доступа: 02.12.2021