ГОСУДАРСТВЕННЫЙ КОМИТЕТ СССР ПО НАРОДНОМУ ОБРАЗОВАНИЮ

МОСКОВСКИЙ ОРДЕНА ОКТЯБРЬСКОЙ РЕВОЛЮЦИИ И ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ ИНСТИТУТ СТАЛИ И СПЛАВОВ

Кафедра инженерной кибернетики

Фараджев И.А.

Одобрено
методическим советом
института

МАТЕМАТИЧЕСКИЕ МЕТОДЫ ДИСКРЕТНОЙ ОПТИМИЗАЦИИ

Курс лекций
для слушателей ФПКП

Москва 1990

АННОТАЦИЯ

Пособие предназначено для студентов специальности 01.02 при изучении курсов " Алгоритмы дискретной математики" и " Исследование операций. Курс состоит из трех частей.

Игорь Александрович Фараджев

МАТЕМАТИЧЕСКИЕ МЕТОДЫ ДИСКРЕТНОЙ ОПТИМИЗАЦИИ
Курс . лекций

Редактор Ю.В.Фролова
Техн.редактор
Рецензенты: д.ф.-м.н.,проф.Р.И.Тышкевич (БГУ)
д.ф.-м.н.,проф. И.В.Романовский (ЛГУ)

Подписано в печать 7.08.90
[illegible] Уч.-изд.л. 6,5 Тираж 140 экз.
Заказ N 1306 цена 24 коп. Тематический план 1990г.
N 51

Московский институт стали и сплавов, Ленинский проспект, 4
Типография ЭОЗ МИСиС, ул.Орджоникидзе, 8/9

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Предлагаемый конспект представляет собой обработанную запись лекций курсов "Алгоритмы дискретной оптимизации" и "Математические методы дискретной оптимизации", читавшихся в 1980-1988 годах для студентов МИСиС специальностей "Кибернетика металлургических процессов" и "Прикладная математика". Этот конспект лекций может также использоваться в качестве учебного пособия по курсу "Алгоритмы дискретной математики" и разделу "Дискретная оптимизация" курса "Исследование операций", предусмотренных новым учебным планом для специальности 01.02 "Прикладная математика".

С точки зрения определений и свойств изучаемых математических объектов курс "Математические методы дискретной оптимизации" базируется на курсе "Введение в конечную математику". Предполагается также, что студенты хорошо ориентируются в применении различных структур данных для манипулирования нечисловой информацией. Кроме того, в курсе используются отдельные элементарные факты из анализа, высшей алгебры, функционального анализа и теории вероятностей. Для успешного овладения курсом, безусловно, нужно обладать некоторым уровнем математической и программистской культуры.

Курс состоит из трех частей.

В первой, вводной, части (раздел 1) приводится формализация задач дискретной оптимизации и описывается проблематика курса.

Вторая часть - теория сложности вычислений (разделы 2,3) посвящена оценке качества алгоритмов по их быстродействию. Здесь автор отказался от традиционного изложения на языке машин Тьюринга, используя более близкую к привычным вычислительным машинам и языкам программирования алгоритмическую систему фон Неймана. Такой подход позволил сделать изложение более доступным при незначительной потере строгости.

В третьей части (разделы 4-8) описаны основные классы алгоритмов решения задач дискретной оптимизации (как эффективные, так и переборные). Совершенно не затронуты два обширных и важных класса задач - задачи математического программирования и задачи теории расписаний. Эти задачи имеют настолько хорошо разработанные теорию и опирающиеся на их специфику методы решения, что их целесообразно выделить в самостоятельные курсы. Изложение алгоритмов проводится на уровне идей в расчете на то, что доведение их до программной реализации будет сделано на практических занятиях. Подробности реализации обсуждаются лишь в той мере, в какой это необходимо для обоснования оценок трудоемкости алгоритмов.

В конце каждого раздела помещены вопросы для самостоятельной работы, которые помогут студентам проверить степень усвоения материала. Вопросы повышенной сложности отмечены звездочкой.

При отборе материала, включенного в лекции, прежде всего преследовалась цель в небольшом по объему курсе познакомить студентов с многообразием идей и методов, используемых при анализе дискретных задач, и подробно изложить некоторые важные технические приемы конструирования алгоритмов их решения. Безусловно, на отборе материала сказались и личные вкусы и пристрастия автора.

На протяжении всего курса автор сознательно жертвовал строгостью доказательств в угоду наглядности и компактности изложения, за что приносятся извинения наиболее дотошным читателям.

Часть использованного в лекциях материала заимствована из журнальных публикаций и ряда прекрасных монографий, перечисленных в списке рекомендованной литературы. Эти книги действительно настойчиво рекомендуются для углубленного изучения предмета.

Автор весьма признателен бывшим студентам МИСиС Ю.Балиной и Т.Фаззулину, предоставившим в его распоряжение свои конспекты, а также всем студентам (их перечень занял бы слишком много места), чья активная работа на лекциях и практических занятиях способствовала совершенствованию курса.

1. ДИСКРЕТНАЯ ОПТИМИЗАЦИЯ

1.1. Проблема выбора

В процессе целенаправленной деятельности человек постоянно сталкивается с *проблемой выбора*: из имеющегося множества способов достижения цели нужно выбрать ровно один. Например, если нужно добраться из одного пункта в городе до другого, можно воспользоваться различными видами и маршрутами общественного транспорта, такси, личной автомашиной, можно, наконец, пойти пешком. Каждый вариант можно охарактеризовать целым спектром качеств: стоимостью и временем перемещения, степенью комфорта и так далее, зависящих, например, от расстояния, времени года и суток, погодных условий. Некоторые варианты могут оказаться неприемлемыми либо с точки зрения ограниченности ресурсов (времени, денег, бензина для автомашины), которыми мы располагаем для достижения цели, либо из-за противоречия самой цели действия (уж очень важным должно быть дело, чтобы идти пешком на значительное расстояние ночью в дождь или гололед). Выбор одного из приемлемых вариантов можно осуществить на основании *предпочтения*, учитывающего как экономность расходования ограниченных ресурсов, так и степень соответствия качеств варианта преследуемой цели (например, низкая степень комфорта при поездке в театр в часы пик общественным транспортом может испортить все удовольствие от спектакля).

Осознание проблемы выбора как математической задачи, основанной на способности предвидеть последствия предпринимаемых действий и оценить близость этих последствий к желаемым, дает большой эффект в жизни каждого человека. Такие общественные установления, как законы, обычаи, и даже поверья, привычки и суеверия, выработаны человеческим обществом из необходимости уберечь его членов от нежелательных (а порою и фатальных) последствий неверного решения проблемы выбора. Неизмеримо большую важность такой подход приобретает при решении проблем выбора, стоящих перед человеческими сообществами (начиная от трудовых коллективов и кончая странами и человечеством в целом), вследствие возрастания значимости целей и ресурсных затрат на их достижение. Игнорирование необходимости строгого математического обоснования предпочтения одного варианта остальным (так называемый волюнтаризм) зачастую приводит к последствиям, прямо противоположным поставленной цели.

Займемся формализацией проблемы выбора. Опишем исследуемый объект с помощью *математической модели*, состоящей из некоторого множества *переменных* и *соотношений*, связывающих значения переменных. Каждая переменная может иметь сложную структуру (например, может быть вектором, матрицей, функцией и тому подобное) и обладает присущей ей областью возможных значений, быть может, зависящей от значений других переменных. В качестве соотношений модели могут фигурировать уравнения различных типов (алгебраические, дифференциальные, интегральные, функциональные), неравенства и даже сколь угодно сложные алгоритмы - правила вычисления значений одних переменных по значениям других.

С точки зрения проблемы выбора множество переменных модели распадается на три группы. Переменные из первой группы описывают состояние внешней по отношению к объекту среды, и их значения являются *исходными данными* (короче, *входом*) задачи выбора.

Значения переменных второй группы - *управляющих* переменных - мы можем задавать по своему усмотрению из области их значений, зависящей, вообще говоря, от входа. Значения этих переменых будем называть *управлениями*, а область их значений - множеством *возможных управлений*. Наконец, значения всех остальных переменных - *внутренних* переменных объекта - должны однозначно вычисляться по входу и управлению с помощью соотношений модели. Не ограничивая общности, можно считать, что в модели есть единственная переменная первого типа с областью значений $X=(x)$. Можно считать также, что модель содержит единственную управляющую переменную с множеством возможных управлений $Y=Y(x)=(y)$. Тот факт, что не все возможные управления *допустимы*, *опишем* с помощью *предиката допустимости* P, определенного на парах (x,y), $x\in X$, $y\in Y(x)$. Область истинности этого предиката называется множеством *допустимых управлений*. Конечно, граница между возможными и допустимыми управлениями вполне условна. Основанием для различения этих понятий (это *мы увидим в дальнейшем) является* то, что множество возможных управлений хотелось бы описывать очень компактно, переложив все трудности описания сложно устроенного множества допустимых управлений на предикат допустимости P.

Наконец, введем на парах (x,y), таких что $x\in X$, $y\in Y(x)$ и $P(x,y)$ истинно, *функционал* $F(x,y)$, описывающий качество функционирования объекта со входом x при выборе управления *y* о точки зрения преследуемой цели. Для решения задачи выбора на такой модели необходимо определить *процедуру предпочтения*, позволяющую выбрать допустимое управление, приводящее к последствиям (как раз и описываемым значением функционала), наиболее соответствующим поставленной цели.

В частном случае, когда значения функционала F представляют собой линейно упорядоченное множество (в дальнейшем мы будем предполагать, что множество значений функционала F содержится в множестве действительных чисел с естественным упорядочением), приходим к задаче *оптимизации*: по заданному входу $x \in X$ найти управление $y \in Y(x)$, которое является допустимым ($P(x,y)$ истинно) и доставляет функционалу F *экстремальное* (максимальное или минимальное) значение среди всех допустимых управлений. Таким образом, задача оптимизации полностью определяется четверкой (X,Y,P,F).

1.2. Примеры задач оптимизации

Пример 1. Пусть нам нужно проехать на автомобиле расстояние S за время T, минимизируя при этом расход топлива. Обозначив через $v(t)$ мгновенную скорость автомобиля, а через $y(t)$ – мгновенный расход топлива, из второго закона Ньютона имеем $m \cdot dv/dt = f(y,v)$, где m – масса автомобиля и f – заданная функция, определяющая приведенное усилие в зависимости от расхода топлива и скорости. Множество возможных управлений в этой задаче – множество всех действительных функций на отрезке $[0,T]$. Множество же всех допустимых управлений определяется граничным условием $\int_0^T v(t)dt = S$. Наконец, минимизации подлежит функционал $F = \int_0^T y(t)dt$. Вход этой задачи состоит из чисел S, T, m и функции f.

Пример 2. Линейное программирование. Пусть имеется n ингридиентов и m свойств, и задана матрица $A = \| a_{ij} \|$, так что a_{ij} –

это значение j-го свойства у i-го ингридиента. Предположим, что значение каждого свойства смеси этих ингридиентов равно средневзвешенному значению свойств каждого из них. Пусть и цена смеси также равна средневзвешенной цене ингредиентов. Требуется найти относительные количества ингридиентов в смеси, так чтобы минимизировать цену смеси при условии, что значение каждого свойства не выйдет за пределы заданных ограничений. В этой задаче входом является совокупность чисел n и m, матрицы A, вектора цен $C=(c_1,\ldots,c_n)$ и вектора ограничений свойств $B=(b_1,\ldots,b_n)$. Управление здесь - это вектор $y=(y_1,\ldots,y_n)$ относительных количеств ингридиентов смеси, область его возможных значений - всевозможные представления числа 1 в виде суммы n неотрицательных слагаемых. Допустимыми являются управления, удовлетворяющие системе линейных неравенств $Ay\leq B$. Функционал, подлежащий минимизации, равен скалярному произведению векторов C и y. Если в этой задаче считать, что $y\in R^n$, включив ограничения $0\leq y_i\leq 1$ и $y_1+\ldots+y_n=1$ в систему линейных неравенств $Ay\leq B$, придем к задаче, носящей название *общей задачи линейного программирования*.

Пример 3. Задача о назначении. Пусть задано множество из n работ и множество из m исполнителей. Соответствие между работами и исполнителями, при котором каждой работе соответствует не более одного исполнителя, а каждому исполнителю - не более одной работы, называется *назначением*. Пусть также задана функция на парах работа-исполнитель, ставящая в соответствие каждой такой паре оценку качества выполнения данной работы данным исполнителем. Требуется выбрать назначение, максимизирующее суммарную оценку качества по всем входящим в него парам работа-исполнитель. В этой задаче вход состоит из чисел n и m и матрицы $A=\|a_{ij}\|$, где a_{ij} - оценка качества выполнения i-й работы j-м исполнителем. Возможные управления $Y=\left\{y\subseteq\{1,\dots,n\}\times\{1,\dots,m\}\right\}$, причем допустимыми являются только те из них, для которых $\left|\left\{i:(i,j)\in y\right\}\right|\leq 1$ при $1\leq i\leq n$ и $\left|\left\{j:(i,j)\in y\right\}\right|\leq 1$ при $1\leq j\leq m$. Максимизировать нужно функционал $F=\sum\limits_{(i,j)\in y} a_{ij}$. В частном случае, когда $a_{ij}\in\{0,1\}$, где $a_{ij}=1$ означает возможность (а $a_{ij}=0$ – невозможность) выполнения i-й работы j-м исполнителем, эта задача называется *булевской* задачей о назначении.

Пример 4. Задача о коммивояжере. Пусть заданы n пунктов и матрица $C=\|c_{ij}\|$, где c_{ij} – стоимость проезда из пункта i в пункт j. Требуется найти замкнутый маршрут, проходящий через каждый пункт в точности по одному разу, и притом с минимальной суммарной стоимостью проезда. Число n и матрица C образуют вход этой задачи. Возможные управления - это n-элементные последовательности $y=\left\{i_1,\dots,i_n\right\}$ пунктов, все элементы которых попарно различны, то есть перестановки множества $\{1,\dots,n\}$. Все возможные управления допустимы, а минимизации подлежит функционал

$$F=c_{i_n,i_1}+\sum_{j=1}^{n-1} c_{i_j,i_{j+1}}.$$

К задаче о коммивояжере, в частности, приводит следующая задача об оптимальной переналадке оборудования. Пусть на одном и том же станке необходимо многократно выполнять совокупность из n операций, причем порядок выполнения операций произволен. Если время выполнения операций не зависит от их порядка, а переход от i-й операции к j-й требует времени t_{ij} для переналадки станка, то решение задачи о коммивояжере с матрицей $T=\|t_{ij}\|$ дает последовательность операций, минимизирующую время выполнения всей их совокупности.

1.3. Задачи дискретной оптимизации

Приведенные в предыдущем пункте задачи оптимизации имеют совершенно разную математическую природу. В первой из них множество возможных управлений бесконечно (имеет мощность гиперконтинуума) и наделено структурой метрического пространства. Множество допустимых управлений может оказаться пустым (невозможно проехать большое расстояние за слишком короткое время), но может также иметь бесконечную мощность. В первом случае задача заведомо не имеет решения. Но она может не иметь решения и в случае наличия допустимых управлений (функционал, даже ограниченный, может не достигать на множестве допустимых управлений своего экстремума). Если же решение существует, оно, за редким исключением, не может быть выражено конечными формулами. Поэтому нахождение оптимального решения подменяется нахождением его приближений в той или иной метрике. Задачами оптимизации этого типа занимаются такие математические дисциплины как вариационное исчисление и теория управления.

Совершенно другая ситуация в задачах, описанных в примерах 3 и 4. Эти задачи характерны тем, что в них множество возможных управлений $Y(x)$ *конечно* для любого входа x. Оптимизационные задачи с таким свойством, называемые задачами ***дискретной оптимизации***, и будут основным объектом исследования в этом курсе лекций. Задачи дискретной оптимизации проще других оптимизационных задач в двух аспектах. Во-первых, если множество допустимых управлений непусто, то оптимальное управление всегда существует. Во-вторых, если есть способ перечислить все элементы множества возможных управлений $Y(x)$, то существует простой и универсальный алгоритм нахождения оптимального решения за конечное время: для каждого управления $y \in Y(x)$ проверим предикат допустимости $P(x,y)$, и если он истинен, вычислим функционал $F(x,y)$. Если область истинности предиката P непуста, найти экстремальное среди конечного числа значений функционала и соответствующее ему управление не составляет труда. Такой алгоритм решения задачи дискретной оптимизации называется алгоритмом *полного перебора*. Для использования алгоритма полного перебора необходимо уметь перечислять элементы множества возможных управлений. Универсальность алгоритма полного перебора основана на том, что как правило удается выбрать множество возможных управлений так, чтобы его элементы можно было легко перечислить. В частности, если мы умеем перечислять элементы конечного множества A, то легко перечислить элементы множества 2^A всех его подмножеств, множества C^k_A его k-элементных подмножеств, множества $A!$ всех его перестановок, множеств R_A и R^k_A его разбиений на произвольное число классов или на k классов (соответствующие методы рассматриваются на практических занятиях). Именно с этим связано различие понятий возможного и допустимого управлений.

Относительно задачи линейного программирования на первый взгляд не видно, что она принадлежит к задачам дискретной оптимизации - ведь множество возможных управлений в ней континуально, и даже множество допустимых управлений может иметь мощность континуума. Чтобы увидеть, что задача линейного программирования может быть сформулирована как задача дискретной оптимизации, обратим внимание на то, что функционал F - линейный функционал в пространстве $\mathbb{R}^n$, а ограничения вырезают в этом пространстве выпуклое замкнутое множество. Согласно известной теореме функционального анализа экстремум функционала в этом случае может достигаться только в граничной точке множества, причем для отыскания этого экстремума достаточно рассмотреть граничные точки минимальной размерности. Граничные точки множества, определяемого системой неравенств $Ay \leq B$, соответствуют обращению некоторых из этих неравенств в равенства. Это обосновывает возможность выбора в качестве возможных управлений конечного множества $2^{\{1,\ldots,m\}}$ всех подмножеств неравенств, которые обращаются в равенства при данном управлении.

1.4. Проблематика курса

Возможности человека существовать в бесконечно разнообразном мире основываются на сравнительно простом устройстве этого мира - большинство задач, возникающих у человека в процессе его деятельности, укладывается в сравнительно небольшое и обозримое множество моделей. Это относится и к задачам оптимизации вообще, и к задачам дискретной оптимизации в особенности (большинство возникающих в разнообразных сферах деятельности задач дискретной оптимизации описывается не слишком большим числом типовых моделей). Этим определяется необходимость разработки методов решения таких типовых задач самих по себе, независимо от предметной области, в которой они возникают.

Наличие универсального алгоритма полного перебора для решения задач дискретной оптимизации означает, что основные проблемы оптимизации (существование оптимального управления и нахождение его точного значения) в этом случае в принципе могут быть решены за конечное число действий. Однако это число действий может быть весьма велико. Например, для задачи о назначении с n работами и n исполнителями число даже допустимых управлений не меньше чем $n!$, так что решение такой задачи при n порядка 20 полным перебором не под силу даже самым мощным вычислительным машинам. Поэтому для задач дискретной оптимизации основной проблемой является конструирование алгоритмов их решения с минимальной затратой вычислительных ресурсов.

В этом курсе мы будем интересоваться только временем, затрачиваемым алгоритмом на получение решения. Понятно, что это время главным образом зависит от *размера* входа (в задаче о назначении - это количество работ и исполнителей, в задаче о коммивояжере - количество пунктов, в задаче линейного программирования - число переменных и ограничений; более точно понятие размера задачи мы введем позже). Оказывается, что для некоторых задач существуют алгоритмы решения, у которых время решения *растет не слишком быстро* с увеличением размера (например, как некоторый полином от размера). Алгоритмы с таким свойством называются *эффективными*, а соответствующие задачи сравнительно просты - они могут быть решены за приемлемое время для практически необходимых размеров. В частности, к таким просто решаемым задачам относятся задача о назначении и задача линейного программирования - для них построены эффективные алгоритмы решения. Для других задач дискретной оптимизации эффективные алгоритмы решения не только неизвестны, но есть основания думать, что их и не существует. Такие задачи трудны для решения - их приходится решать полным или почти полным перебором возможных управлений, так что размер таких задач, поддающихся решению за приемлемое время, значительно меньше реальных потребностей. Такой как раз является задача о коммивояжере.

Оказывается, что большинство типовых задач дискретной оптимизации относятся к числу трудных задач, что можно рассматривать как одно из выражений диалектического единства простоты и сложности окружающего нас мира.

В следующих двух главах будет изложена теория, позволяющая разграничить простые и трудно решаемые задачи. Главы 4-7 мы посвятим конструированию эффективных алгоритмов решения основных типов простых задач дискретной оптимизации. Наконец, в главе 8 будут изложены приемы, позволяющие несколько (а иногда и очень значительно) сократить перебор при решении трудных задач.

Вопросы для самостоятельной работы

1. Пусть мы имеем в пространстве R^n ограничения вида $a_{i1}y_1+\ldots+a_{in}y_n \ \lambda \ b_i$, где $\lambda\in(<,>,=)$. Показать, что любую совокупность таких ограничений можно записать в виде $Ay<B$.

2. Привести пример задачи о назначении с неединственным оптимальным управлением.

3. Пусть в задаче о назначении задана также матрица $C=\|c_{ij}\|$, где c_{ij} - затраты на выполнение i-ой работы j-ым исполнителем. Построить функционал, экстремум которого соответствовал бы максимуму качества при минимуме затрат.

*4. Подсчитать число допустимых управлений в задаче о назначении с n работами и m исполнителями.

*5. Сформулировать задачу о назначении $R=(X,Y,P,F)$ с равным числом работ и исполнителей и задачу о коммивояжере $Q=(X',Y',P',F')$, так чтобы задача о коммивояжере отличалась только сужением множества допустимых управлений, то есть $X'=X$, $Y'=Y$, $F'=F$, $P'=P\&P_1$.

6. Привести пример задачи оптимизации с непустым множест-

вом допустимых управлений и ограниченным функционалом, у которой отсутствует оптимальное управление.

2. ВВЕДЕНИЕ В ТЕОРИЮ СЛОЖНОСТИ

2.1. Алгоритмические системы

Прежде чем говорить о качественных характеристиках алгоритмов решения тех или иных задач, необходимо очертить круг алгоритмов, о которых будет идти речь. Не давая формального определения алгоритма, мы ограничимся интуитивным представлением об алгоритме как программе для некоторой гипотетической машины. Классы таких машин, различающиеся устройством памяти и ограничениями на выполняемые операции, носят название *алгоритмических систем.* В этом разделе мы познакомимся с тремя алгоритмическими системами *последовательного* действия, в каждом такте работы которых выполняется одна операция. В дальнейшем из последовательных систем будут конструироваться *параллельные* алгоритмические системы, могущие выполнять в каждом такте несколько независимых операций.

Простейшей алгоритмической системой является *машина Тьюринга.* Память этой машины представляет собой линейно упорядоченное множество ячеек $S=\{s\}$, пронумерованных целыми числами от $-\infty$ до ∞ (бесконечная лента). Кроме того, имеется выделенная ячейка памяти g, называемая *головкой.* В каждой ячейке памяти (в том числе и в головке) хранится символ некоторого конечного алфавита $A=\{a\}$. В каждый момент времени головка расположена напротив некоторой ячейки ленты. *Состояние* машины Тьюринга в момент времени i полностью определяется отображением x_i : $S \cup \{g\} \to A$ и номером ячейки s_i, напротив которой расположена головка. Функционирование машины задается тремя отображениями α, β и ε, $\alpha: A\times A \to A$, $\beta: A\times A \to A$, $\varepsilon: A\times A \to \{-1,0,1\}$, с помощью которых по состоянию в момент времени i определяется состояние в момент времени $i+1$: $x_{i+1}(s)=x_i(s)$ для $s\neq s_i$; $x_{i+1}(s_i)=\alpha\left(x_i(s_i), x_i(g)\right)$; $x_{i+1}(g)=\beta\left(x_i(s_i), x_i(g)\right)$; $s_{i+1}=s_i+\varepsilon\left(x_i(s_i), x_i(g)\right)$. Таким образом, один шаг работы машины Тьюринга (*операция*) определяется символами, записанными в головке и в ячейке s_i, и состоит в изменении этих символов (в соответствии с отображениями α и β) и перемещении головки (в соответствии с отображением ε). Совокупность отображений α, β и ε является *программой*, по которой работает машина Тьюринга.

Алгоритмическая система *Колмогорова-Успенского* обобщает машину Тьюринга. Памятью этой машины, в отличие от машины Тьюринга, служит множество ячеек, наделенных более общей структурой *дискретной топологии* Ω, ставящей в соответствие каждой ячейке памяти s ее непустую *окрестность* $\Omega(s)$. Для описания функционирования этой машины вводится дополнительно отображение μ, которое по совокупности символов, хранящихся в головке и в ячейке s_i, выделяет из окрестности этой ячейки подмножество *аргументов* операции $X_i\subseteq\Omega(s_i)$ ограниченной мощности $|X_i|<k$. Дальнейшее выполнение операции зависит *только от символов, записанных* в головке и в ячейках из X_i, и заключается в изменении содержимого головки и аргументов операции и перемещении головки в некоторую ячейку из $\Omega(s_i)$. Число k, ограничивающее количество аргументов, обычно называют *адресностью* машины.

Другим крайним частным случаем системы Колмогорова-Успенского, противоположным по свойствам машине Тьюринга, является *машина фон Неймана*, наиболее близкая к традиционным вычислительным машинам и алгоритмическим языкам. К этой алгоритмической системе приходим, полагая, что окрестность каждой ячейки памяти s совпадает со всей памятью S. При этом отпадает необходимость в выделенной ячейке памяти (головке) и функционирование машины фон Неймана можно описать следующим образом. Состояние x_{i+1} в момент времени $i+1$ получается из состояния x_i в момент времени i под действием оператора δ_i, зависящего только от содержимого множества ячеек $X_i \subset S$ ограниченной мощности $|X_i|<k$, причем $x_{i+1}(s)=x_i(s)$ для всех $s \notin X_i$. Таким образом, каждая операция машины фон Неймана определяется содержимым ограниченного множества ячеек, произвольным образом расположенных в памяти, и изменяет состояние только этих ячеек. Последовательность операторов δ_i представляет собой программу этой машины.

Три рассмотренные алгоритмические системы соответствуют трем типам памяти в последовательных вычислительных машинах: машина Тьюринга - памяти последовательного доступа типа магнитной ленты; машина фон Неймана - памяти произвольного доступа с прямой адресацией; машина Колмогорова-Успенского - памяти произвольно-последовательного доступа типа магнитных дисков или виртуальной памяти со страничной организацией.

В этом курсе мы будем рассматривать алгоритмы в системе фон Неймана, не детализируя без необходимости набор имеющихся операций. Следует отметить, что каждая из описанных алгоритмических систем без труда может быть промоцелирована на другой из них, так что большинство фактов, которые мы установим относительно свойств алгоритмов в системе фон Неймана, могут быть перенесены и на другие системы.

2.2. Трудоемкость алгоритмов и сложность задач

Будем говорить, что алгоритм α *решает* задачу R, если, будучи выполнен на машине фон Неймана с начальным состоянием, соответствующим любому входу $x \in X$, он через конечное время останавливается (то есть состояние машины перестает меняться) в состоянии, содержащем решение задачи (отсутствие решения тоже можно считать специальным видом решения). Мы будем рассматривать дискретные задачи (не обязательно задачи дискретной оптимизации), для которых всегда существует алгоритм, находящий решение для *любого* входа x за конечное время. Множество алгоритмов, решающих задачу R, обозначим через $\mathcal{A}(R)$.

Займемся изучением времени $t_R^\alpha(x)$ работы алгоритма $\alpha \in \mathcal{A}(R)$ при решении задачи R со входом x. Хотелось бы научиться *грубо* оценивать это время по небольшому числу параметров, описывающих вход x. Понятно, что время работы алгоритма в очень разной степени зависит от разных компонент входа x. Поэтому естественно выделить небольшое число параметров входа, от которых главным образом зависит время решения, и назвать этот набор параметров *размером* входа r(x). Например, в задаче линейного программирования в качестве размера задачи удобно взять пару (n,m) размеров матрицы ограничений; в задаче о назначении - количество работ и исполнителей; в задаче о коммивояжере - число пунктов. Конечно, размер задачи не позволяет точно вычислить время решения, оно может зависеть и от других компонент входа. Например, в задаче упорядочения последовательности из n чисел, естественно принять за размер задачи число n; однако время работы алгоритма, реализующего метод пузырька, будет зависеть не только от размера n, но и от исходной упорядоченности последовательности. Специфицируя вход при помощи его размера с разной подробностью, мы сможем получать более или менее точные оценки времени работы алгоритма. Существует универсальный способ определения размера входа x посредством его *длины* $|x|$ – количества ячеек памяти, необходимых для его размещения в памяти машины, однако мы будем пользоваться и другими определениями размера входа, при которых длина входа будет по крайней мере ограничиваться сверху некоторой функцией размера. Нас будут интересовать так называемые *массовые* задачи, у которых есть входы сколь угодно большого размера, а мощность множества входов, тем самым, бесконечна.

Пусть для задачи R мы фиксировали понятие размера $r(x)$ как функции от входа $x \in X$. Функция от размера, равная максимальному времени решения алгоритмом α задачи R со входом этого размера, называется *трудоемкостью* алгоритма: $t^{\alpha}_{R}(r) = \max\limits_{r(x)=r} t^{\alpha}_{R}(x)$. Знание трудоемкости алгоритма позволяет, с одной стороны, легко определить верхнюю границу времени решения задачи, а с другой стороны, является важнейшей качественной характеристикой, применяемой для сравнения алгоритмов решения одной и той же задачи.

Понятие трудоемкости алгоритма, позволяя характеризовать алгоритм относительно других алгоритмов решения той же задачи, в то же время не несет информации о его абсолютном качестве, в частности, не позволяет ответить на вопрос о существовании алгоритмов решения этой задачи с меньшей трудоемкостью. Нижняя граница времени решения задачи, как функции от размера входа, устанавливается при помощи *сложности* задачи: $t_R(r) = \min\limits_{\alpha \in \mathcal{A}(R)} t^{\alpha}_{R}(r)$.

Алгоритм, трудоемкость которого совпадает со сложностью решаемой им задачи, называется *оптимальным*.

2.3. Нижние и верхние оценки сложности

Введенное в предыдущем разделе понятие трудоемкости алгоритма не слишком удобно для практического использования по двум причинам. Во-первых, для вычисления трудоемкости алгоритма необходима излишне подробная детализация алгоритмической системы (алфавит, набор операций, способ кодирования входа и результата в памяти и т.д.). Во-вторых, при сравнении двух алгоритмов решения одной и той же задачи может оказаться, что один из них работает быстрее при одних размерах входа, а другой - при других. Оба эти недостатка *можно преодолеть, если вместо* точного значения трудоемкости пользоваться ее скоростью роста при неограниченном увеличении размера входа.

Пусть $f(r)$ и $\varphi(r)$ - положительные функции от набора параметров $r=\{r_1,\dots,r_k\}$. Если $\lim\limits_{r_1\to\infty}\left(\varphi(r)/f(r)\right)<\infty$, то будем говорить, что функция φ *растет не быстрее* функции f, и обозначать этот факт через $\varphi(r)\leqslant O\left(f(r)\right)$. Аналогично, будем пользоваться обозначением $\varphi(r)\geqslant O\left(f(r)\right)$ (функция φ *растет не медленнее* функции f), если $\lim\limits_{r_1\to\infty}\left(\varphi(r)/f(r)\right)>0$. Наконец, в случае $O\left(f(r)\right)\leqslant\varphi(r)\leqslant O\left(f(r)\right)$, говорят, что функции $\varphi(r)$ и $f(r)$ имеют *одинаковую скорость роста*, и используют обозначение $\varphi(r)=O\left(f(r)\right)$. В дальнейшем мы неоднократно будем пользоваться простыми свойствами символа $O()$, считая их известными из курса математического анализа.

Обычно сравнительно легко удается оценить скорость роста трудоемкости алгоритма, даже не вдаваясь в детали его программной реализации. Тем не менее известны случаи, когда скорость роста трудоемкости алгоритма удается получить только в результате весьма нетривиального анализа. Достаточно яркий пример будет приведен в п. 6.7.

Скорость роста трудоемкости любого алгоритма, решающего некоторую задачу, дает верхнюю оценку для скорости роста сложности этой задачи. Получение же нижней оценки сложности, то есть такой функции, что трудоемкость любого алгоритма решения задачи растет не медленнее этой функции, является одной из центральных и наиболее трудных проблем в теории сложности. Только в тех редких случаях, когда удается получить нижнюю и верхнюю оценки с одинаковой скоростью роста, мы узнаем точную скорость роста сложности задачи.

Простейшие *нижние* оценки сложности задачи можно получить из *информационных* соображений. Для многих дискретных задач очевидно (во всяком случае, зачастую это легко доказать), что все компоненты входа x *существенны*, то есть изменение любой из них приводит к изменению результата. Вспоминая, что в одной операции машины фон Неймана участвует фиксированное число ячеек памяти, приходим к выводу, что любой алгоритм решения задачи в этом случае не может работать меньше, чем $O(|x|)$ тактов. Оценивая длину входа через размеры задачи, получим нижнюю оценку ее сложности.

Например, рассмотрим задачу сложения матриц (СМ): необходимо сложить квадратные матрицы $A=\|a_{ij}\|$ и $B=\|b_{ij}\|$ размером $n\times n$ по следующему правилу $A+B=C=\|c_{ij}\|$, где $c_{ij}=a_{ij}+b_{ij}$. Приняв в качестве размера входа число n, видим, что $|x|=O(n^2)$. Поскольку все компоненты входа, очевидно, существенны, получаем $t_{CM}(n)\geq O(n^2)$. Аналогичные рассуждения о длине $|y|$ результата решения задачи показывают, что любой алгоритм должен потратить на решение не менее, чем $O(|y|)$ операций. В качестве примера рассмотрим задачу построения транзитивного замыкания бинарного отношения (ТЗ): для заданного отношения $U, |U|=m$ на множестве $A, |A|=n$ нужно найти минимальное по включению транзитивное отношение $\bar{U}$ на множестве A, содержащее отношение U. Очевидно, что $|\bar{U}|$ может достигать n^2, даже когда $|U|$ существенно меньше. Отсюда получаем $t_{ТЗ}(n,m)\geq O(n^2)$. В следующем пункте мы познакомимся со случаем, когда удается получить менее тривиальную нижнюю оценку сложности.

2.4. Сложность задачи упорядочения

Рассмотрим следующую задачу упорядочения (УП): для заданной последовательности $(a_1,\dots,a_n)$ из n различных элементов линейно упорядоченного множества A найти перестановку π множества $(1,\dots,n)$ такую, чтобы $i<j \Rightarrow a_{\pi(i)}<a_{\pi(j)}$ для всех $1\leq i<n$, $i<j\leq n$. В качестве размера входа примем длину последовательности n.

Пусть нам ничего не известно о структуре множества A и в нашем распоряжении есть только операция сравнения элементов из входной последовательности. Оценим минимальное количество операций сравнения, необходимых для решения задачи упорядочения. Пока мы не провели ни одного сравнения, все перестановки из множества Π_0 всех перестановок множества $(1,\dots,n)$ являются кандидатами в искомую перестановку π и $|\Pi_0|=n!$. Сравним два элемента последовательности a_{i_1} и a_{j_1}. Множество Π_0 разобьется на два класса $\Pi_1^{<}=\{\pi\in\Pi_0 : \pi(a_{i_1})<\pi(a_{j_1})\}$ и $\Pi_1^{>}=\{\pi\in\Pi_0 : \pi(a_{i_1})>$ $>\pi(a_{j_1})\}$. В зависимости от результата сравнения искомая перестановка π содержится либо в классе $\Pi_1^{<}$, либо в классе $\Pi_1^{>}$, который мы и примем в качестве множества Π_1 кандидатов в искомую перестановку. Заметим, что мощность одного из классов $\Pi_1^{<}$ или $\Pi_1^{>}$ не меньше, чем $n!/2$, так что в худшем случае $|\Pi_1|\geq n!/2$. Проведя теперь сравнение элементов a_{i_2} и a_{j_2}, придем к множеству Π_2 кандидатов в искомую перестановку, мощность которого в худшем случае не меньше, чем $n!/4$. Повторяя этот процесс, убеждаемся, что после k сравнений мощность множества Π_k кандидатов в искомую перестановку может достигать величины $n!/2^k$. Поскольку искомая перестановка единственна, получаем отсюда, что $k\geq\log_2 n!$. Использование формулы Стирлинга $\log_2 n!=O(n\log n)$ завершает рассуждения, результатом которых является следующая теорема.

Теорема. $t_{У\Pi}(n)\geq O(n\log n)$.

Известны алгоритмы решения задачи упорядочения с трудоемкостью $O(n\log n)$ (например, алгоритм разбиения и слияния). В совокупности с доказанной нижней оценкой это означает, что $t_{У\Pi}(n)=O(n\log n)$. Напомним, что такая оценка получена в предположении, что мы умеем только сравнивать элементы последовательности.. Если же нам известна дополнительная информация о множестве A, к которому принадлежат элементы последовательности, возможно, сложность задачи окажется меньше. Например, если имеется операция, вычисляющая количество элементов a множества A, таких что $a_i<a<a_j$ (в случае, когда A - отрезок натурального ряда, количество таких элементов равно a_j-a_i-1), то решить задачу упорядочения можно с трудоемкостью меньшей, чем $O(n\log n)$ (алгоритм ящичной сортировки).

2.5. Сводимость и эквивалентность задач

В предыдущих разделах мы продемонстрировали схему анализа задачи с целью установления нижних и верхних оценок ее сложности. Этот анализ, в частности, включает в себя разработку хорошего (с малой трудоемкостью) алгоритма ее решения. Очевиден порочный круг - понятие сложности задачи введено с целью оценки близости к оптимуму алгоритма ее решения, а для вычисления сложности необходимо сконструировать алгоритм, по возможности близкий к оптимальному. Разорвать этот порочный круг позволяет другой подход к получению оценок сложности, связанный с возможностью использовать при решении одной задачи алгоритмы решения другой задачи (*сведение* одной задачи к другой).

Пусть R и Q - дискретные задачи с одним и тем же множеством значений размера входа. Для упрощения рассуждений мы будем считать, что размер входа $r(x)$ задается единственным целочисленным параметром n; обобщение на случай, когда размер входа является набором параметров, не вносит ничего принципиально нового.

Рассмотрим множество $\Psi=\left\{\varphi(n)\right\}$ положительных функций от целого аргумента n, обладающих следующими свойствами:

1- все функции $\varphi(n)\in\Psi$ возрастают при достаточно больших n;

2- Ψ содержит тождественную функцию $\varphi(n)=n$;

3- для любых $\varphi_1(n), \varphi_2(n)\in\Psi$ существует $\varphi(n)\in\Psi$, такая что $\left(\varphi_2(n)\right)<\varphi(n)$.

Пусть с помощью любого алгоритма $\alpha\in\mathcal{A}(R)$ решения задачи R мы можем построить алгоритм $\beta\in\mathcal{A}(Q)$ (использующий α как подпрограмму) решения задачи Q с трудоемкостью $t^{\beta}_{Q}(n)<t^{\alpha}_{R}\bigl(\varphi(n)\bigr)$, где $\varphi\in\Psi$. В этом случае будем говорить, что задача Q *Ψ-сводится* к задаче R: $Q\xrightarrow[\Psi]{}R$. Выполнение условий 2 и 3 для множества функций Ψ гарантирует рефлексивность и транзитивность отношения Ψ-сводимости.

В качестве множества Ψ чаще всего рассматривают либо множество L всех линейных функций $an+b$ при $a>0$, либо множество P всех полиномов от n с положительным старшим коэффициентом. В этих двух случаях имеет место *линейная* сводимость (символ L мы будем опускать и писать $Q\rightarrow R$), либо *полиномиальная* сводимость $Q\xrightarrow[P]{}R$.

Пусть, например, мы умеем преобразовать вход x задачи Q во вход x' задачи R, так чтобы по решению y' задачи R найти решение y задачи Q, причем как трудоемкости прямого и обратного преобразований, так и размер r(x'), ограничены полиномами от размера r(x). Легко видеть, что в этом случае $Q\xrightarrow[P]{}R$.

Если установлена Ψ-сводимость задачи Q к задаче R, то по верхней оценке сложности задачи R можно вычислить верхнюю оценку сложности задачи Q, а по нижней оценке сложности задачи Q – нижнюю оценку сложности задачи R. Действительно, из определения сводимости, используя в качестве α оптимальный алгоритм решения задачи R, немедленно получаем, что из $Q\xrightarrow[\Psi]{}R$ и $t_R(n)\leqslant O\bigl(f(n)\bigr)$ следует $t_Q(n)\leqslant O\bigl(f(\varphi(n))\bigr)$. Наоборот, если $Q\xrightarrow[\Psi]{}R$ и $t_Q(n)\geqslant O\bigl(f(n)\bigr)$, то $t_R(n)\geqslant O\bigl(f(\varphi^{-1}(n))\bigr)$. В случае линейной сводимости задачи Q к задаче R это означает простой перенос верхней оценки с R на Q, а нижней оценки – с Q на R.

Пусть $Q\xrightarrow[\Psi]{}R$ и $R\xrightarrow[\Psi]{}Q$. В этом случае имеет место *взаимная Ψ-сводимость* или *Ψ-эквивалентность* задач R и Q.

Сводимость и эквивалентность задач является мощным инструментом как анализа сложности задач, так и конструирования хороших алгоритмов их решения, в чем мы неоднократно убедимся на протяжении этого курса. В частности, если Q полиномиально сводится к R и для решения задачи R известен эффективный (с полиномиальной трудоемкостью) алгоритм ее решения, то мы можем построить эффективный алгоритм и для решения задачи Q.

2.6. Задачи существования

Наряду с задачами дискретной оптимизации рассмотрим класс более простых дискретных задач - дискретные задачи *существования*. В то время, как задача оптимизации определяется четверкой (X,Y,P,F), задача существования задается только тройкой (X,Y,P) и заключается в нахождении по входу $x \in X$ любого возможного управления $y \in Y(x)$, доставляющего истинное значение предикату $P(x,y)$. Задачи существования можно рассматривать как частный случай задач оптимизации: любая задача существования $Q=(X,Y,P)$ линейно сводится к задаче оптимизации $R=(X,Y,P,F)$ с постоянным функционалом $F(x,y)=\mathrm{const}$. С другой стороны, широкий класс задач оптимизации полиномиально сводится к некоторому классу задач существования.

Для задачи дискретной оптимизации $R=(X,Y,P,F)$ будем рассматривать *соответствующую* ей задачу существования $Q=(X',Y,P')$, (где $X'=\{x'\}=\left\{x \cup (f^-,f^+) : x \in X\right\}$, $P'(x',y)=P(x,y) \,\&\, \left(f^- \le F(x,y) \le f^+\right)$), заключающуюся в нахождении допустимого управления задачи оптимизации, доставляющего функционалу значение из заданного отрезка $[f^-,f^+]$.

Рассмотрим одно важное свойство функционала F. Обозначим через $\Delta(x)$ длину отрезка, к которому принадлежат значения функционала при данном входе x: $\Delta(x)=\max\limits_{y\in Y(x)} F(x,y) - \min\limits_{y\in Y(x)} F(x,y)$, и через $\delta(x)$ минимум разности двух значений функционала: $\delta(x)=\min\limits_{y,y'\in Y(x)} |F(x,y)-F(x,y')|$, где минимум берется по парам управлений, доставляющих функционалу не совпадающие значения. Функция $\chi_F(r)=\max\limits_{r(x)=r} \log_2\left(\Delta(x)/\delta(x)\right)$ называется *дискретностью* функционала F.

Теорема. Если дискретность $\chi_F(r)$ функционала F задачи дискретной оптимизации R ограничена полиномом от размера входа, то задача R полиномиально сводится к соответствующей задаче существования.

Доказательство. Для определенности будем считать, что в задаче R ищется максимум функционала F. Рассмотрим следующий алгоритм решения задачи R, использующий многократное решение соответствующей задачи существования. Прежде всего, решим задачу существования в границах $[f_0^-, f_0^+]$ возможных значений функционала. Если решения задачи существования нет, то отсутствует и решение задачи оптимизации. Если решение существует, положим $f_1^-=(f_0^-+f_0^+)/2$, $f_1^+=f_0^+$ и снова решим задачу существования в границах $[f_1^-, f_1^+]$. В случае существования решения максимум функционала лежит в этих границах, в противном случае – в границах $[f_0^-, f_1^-]$. В любом случае получаем отрезок длиной $\Delta(x)/2$, в котором лежит максимум функционала. Будем точно так же действовать дальше, решая на каждом шаге задачу существования в границах, определяющих верхнюю половину найденного на предыдущем шаге отрезка, к которому принадлежит максимум функционала. Легко видеть, что через k шагов мы будем иметь отрезок $[f_k^-, f_k^+]$ длиной $\Delta(x)/2^k$. Как только длина этого отрезка станет меньше $\delta(x)$, что произойдет при $k>\chi_F(r)$, последнее из найденных решений задачи существования и будет решением задачи оптимизации. Из предположения о полиномиальной ограниченности дискретности функционала следует, что для нахождения решения задачи оптимизации нам достаточно решить задачу существования ограниченное полиномом от размера входа число раз. Это и означает полиномиальное сведение задачи оптимизации к соответствующей задаче существования.

Дискретность функционала многих типовых задач дискретной оптимизации полиномиально ограничена, что позволяет вместо анализа сложности этих задач исследовать сложность соответствующих задач существования.

Следует отметить, что с точки зрения сведения задачи оптимизации к задаче существования достаточно рассматривать соответствующую задачу существования с односторонним ограничением $F(x,y)>f$ для задачи на максимум функционала или $F(x,y)<f$ для задачи на минимум.

Вопросы для самостоятельной работы

1. Дать формальное определение машин Тьюринга и фон Неймана как частных случаев алгоритмической системы Колмогорова-Успенского. Какова адресность этих машин?

2. Доказать, что если длина входа задачи ограничена функцией от размера, то множество входов фиксированного размера конечно.

*3. Доказать существование оптимального алгоритма решения дискретной задачи.

4. Показать, что $\varphi(x)<\dot{O}\bigl(f(x)\bigr) \leftrightarrow f(x)>0\bigl(\varphi(x)\bigr)$.

5. Доказать, что отношение "расти не быстрее" есть отно-

шение толерантности на множестве функций с одними и теми же аргументами.

6. Доказать, что отношение "иметь одинаковую скорость роста" есть отношение эквивалентности на множестве функций с одними и теми же аргументами.

7. Доказать, что отношение ∀-сводимости есть отношение толерантности на множестве задач с одинаковой областью значений размера входа.

8. Доказать, что отношение взаимной ∀-сводимости есть отношение эквивалентности на множестве задач с одинаковой областью значений размера входа.

*9. Построить множество функций, удовлетворяющее условиям 1-3 п. 2.5 и отличное от множеств линейных функций и полиномов.

10. Сформулировать задачу существования, соответствующую задаче коммивояжера.

11. Показать полиномиальную ограниченность дискретности функционала в булевской задаче о назначении.

3. СЛОЖНОСТНАЯ КЛАССИФИКАЦИЯ ЗАДАЧ

3.1. Классы $\mathcal{P}$ и $N\mathcal{P}$

Множество дискретных задач, сложность которых ограничена сверху полиномом от размера входа, образует класс $\mathcal{P}$ *полиномиально разрешимых* задач. Безусловно, к классу $\mathcal{P}$ принадлежат задачи дискретной оптимизации, у которых мощность множества возможных управлений ограничена полиномом от размера, а предикат и функционал могут быть вычислены за полиномиальное время (для таких задач полиномиальную трудоемкость имеет алгоритм полного перебора). В то же время эффективные алгоритмы решения могут существовать и для тех задач, у которых мощность множества возможных управлений (и мощность множества допустимых управлений, и даже мощность множества оптимальных управлений) растет при увеличении размера входа экопоненциально. Примером может служить булевская задача о назначении; эффективный алгоритм ее решения будет рассмотрен в п. 6.8.

Для определения более широкого, чем класс $\mathcal{P}$, класса задач нам понадобится ввести новую алгоритмическую систему. Пусть в нашем распоряжении есть бесконечно много последовательных машин фон Неймана. Соединим их в древовидную иерархическую структуру, так что корневой машине (машине 0-го уровня) подчинено несколько машин 1-го уровня, каждой машине 1-го уровня подчинено несколько машин 2-го уровня и так далее. В систему команд каждой последовательной машины введем операции передачи содержимого ограниченного числа ячеек памяти в память подчиненной машины и приема содержимого ячеек памяти подчиненной машины. Такая алгоритмическая система называется *параллельной* машиной фон Неймана. Входящие в ее состав последовательные машины могут работать одновременно и независимо, каждая по своей программе. Функционирует такой коллектив машин следующим образом. Корневая машина может либо решить задачу сама, либо может разбить задачу на ряд подзадач, поручив их решение подчиненным машинам, и завершить решение исходной задачи с использованием результатов решения подзадач. Каждая из подчиненных машин действует таким же образом, то есть либо решает сама порученную ей подзадачу, либо разбивает ее на более мелкие подзадачи и поручает их решение подчиненным ей машинам. Таким образом, в каждом такте работы такой машины может выполняться одновременно много операций. Те последовательные машины в этом коллективе, которые не используют подчиненных машин, будем называть *терминальными*.

Дискретные задачи, которые могут быть решены на параллельной машине фон Неймана с трудоемкостью, ограниченной некоторым полиномом от размера входа, образуют класс $N\mathcal{P}$. Конечно, для принадлежности задачи классу $N\mathcal{P}$ необходимо и достаточно, чтобы были ограничены полиномами от размера входа следующие три величины: время работы каждой последовательной машины, количество подчиненных у каждой машины и уровень каждой терминальной машины. Очевидно, что каждая полиномиально разрешимая задача принадлежит классу $N\mathcal{P}$, то есть $\mathcal{P}\subseteq N\mathcal{P}$. Приведем достаточное условие принадлежности задачи дискретной оптимизации этому классу.

Утверждение. Пусть в задаче дискретной оптимизации $R=(X,Y,P,F)$ для любого входа x любое допустимое управление $y\in Y(x)$ может быть закодировано словом в некотором алфавите, причем и длина слова, и мощность алфавита ограничены полиномами от размера входа; пусть также предикат P(x,y) и функционал F(x,y) могут быть вычислены с полиномиальной трудоемкостью. Тогда $R \in N\mathcal{P}$.

Доказательство. Разобьем множество Y=Y(x) на подмножества Y_{α_1}, Y_{α_2}, ..., Y_{α_i}, ... , каждое из которых состоит из слов , начинающихся с одного и того же символа α_i. Каждое из этих подмножеств, например Y_{α_1}, разобьем на подмножества Y_{α_1,α_1}, Y_{α_1,α_2}, ..., каждое из которых состоит из слов, начинающихся с двух одинаковых символов $\alpha_i \alpha_j$, и так далее до тех пор, пока все элементы разбиения не станут одноэлементными подмножествами. Полученная система T(Y) подмножеств частично упорядочена относительно включения в виде дерева с корнем Y, причем и ветвление (количество элементов разбиения на каждом уровне), и высота дерева (максимальный уровень одноэлементных подмножеств) ограничены полиномом от размера входа. Теперь можно использовать следующий алгоритм решения задачи R на параллельной машине: последовательной машине предлагается в качестве подзадачи найти экстремум функционала на подмножестве допустимых управлений, содержащихся в некотором подмножестве из системы T(Y). Если это подмножество не одноэлементно, то машина порождает подзадачи в соответствии с его разбиением, в противном случае она сама решает подзадачу, вычисляя функционал и предикат.

В случае, когда Y - множество всех подмножеств, или множество всех перестановок, или множество всех разбиений множества A, мощность которого ограничена полиномом от размера задачи, возможные управления легко представляются словами полиномиально ограниченной длины в алфавите A, так что к классу $N\mathcal{P}$ принадлежит большое количество задач дискретной оптимизации, для которых не известны эффективные алгоритмы решения (например, задача о коммивояжере; много примеров будет приведено в п. 3.5).

3.2. $N\mathcal{P}$-полные задачи

Класс $\mathcal{P}\subseteq N\mathcal{P}$ полиномиально разрешимых задач замкнут относительно полиномиальной сводимости на последовательных машинах фон Неймана. Это означает, что любые две задачи из класса $\mathcal{P}$ полиномиально сводятся друг к другу. Конечно, задачи из класса $\mathcal{P}$ можно дальше классифицировать по степени полинома, ограничивающего их сложность (линейные, квадратичные и так далее). Сейчас нам достаточно констатировать, что к классу $\mathcal{P}$ принадлежат самые простые задачи из $N\mathcal{P}$ – для них существуют эффективные алгоритмы решения на последовательных машинах. Перейдем к определению класса самых сложных в $N\mathcal{P}$ задач, также обладающего замкнутостью относительно полиномиальной сводимости.

Задача R из класса $N\mathcal{P}$ называется *$N\mathcal{P}$-полной* (обозначается $R\in\overline{N\mathcal{P}}$), если к ней полиномиально сводится любая задача из $N\mathcal{P}$: $R\in\overline{N\mathcal{P}} \leftrightarrow R\in N\mathcal{P} \,\&\, \left(\forall Q\in N\mathcal{P} : Q \underset{P}{\to} R\right)$.

Как доказывать принадлежность задачи к классу $\overline{N\mathcal{P}}$? Один, весьма трудоемкий, путь состоит в непосредственном использовании определения класса $N\mathcal{P}$-полных задач и функционирования параллельной машины фон Неймана. Этим способом мы докажем в п. 3.4 *$N\mathcal{P}$-полноту одной* задачи существования. Другой способ основан на использовании простой леммы.

Лемма. Пусть $R\in\overline{N\mathcal{P}}$, $Q\in N\mathcal{P}$ и $R \underset{P}{\to} Q$. Тогда $Q\in\overline{N\mathcal{P}}$.

Доказательство немедленно следует из транзитивности *P*-сводимости.

Применение этой леммы позволит нам легко показать

NP-полноту разнообразных задач существования и оптимизации (п. 3.5). Обращаем внимание на то, что этот подход применим, только когда нам уже известна принадлежность классу $\overline{NP}$ хотя бы одной задачи.

3.3. Сужение алгоритмической системы

Чтобы показать существование NP-полных задач, нам понадобится более точное описание класса NP и функционирования параллельной машины фон Неймана. Сузим класс NP, рассматривая только задачи существования. На самом деле, рассуждения того же типа, которые мы будем проводить для задач существования, проходят и для задач оптимизации, но в этом случае они становятся еще более громоздкими. Некторое основание для такого сужения дает теорема п. 2.6.

Примем следующую схему функционирования параллельной машины фон Неймана при решении задачи существования. Каждая нетерминальная машина занимается только генерацией и распределением подзадач своим подчиненным машинам, причем передает информацию каждой из них однократно. Каждая терминальная машина заканчивает работу приходом в одно из двух заключительных состояний, соответствующих наличию или отсутствию решения поставленной ей подзадачи. Решение исходной задачи существования существует, когда хотя бы одна из терминальных машин нашла решение своей подзадачи.

Детализировать функционирование параллельной машины фон Неймана при решении задачи существования из класса NP позволяет следующее утверждение.

Утверждение. Пусть задача существования R принадлежит классу $N\mathcal{P}$. Тогда она может быть решена за полиномиальное время на параллельной машине фон Неймана со следующими ограничениями:

1- каждая последовательная машина работает в двоичном алфавите {0,1};

2- каждая машина в параллельной системе имеет не более двух подчиненных;

3- все машины одного уровня работают по одной и той же программе;

4- все терминальные машины находятся на одном и том же уровне.

Доказательство. 1. Символы алфавита любой машины фон Неймана можно закодировать в двоичном алфавите, потратив на хранение символа фиксированное число ячеек двоичной машины. Программу произвольной машины фон Неймана можно проинтерпретировать на двоичной машине, заменив каждую операцию исходной машины последовательностью операций двоичной машины фиксированной длины. Таким образом, при переходе к двоичной машине время работы увеличится в константу раз.

2. Пусть в исходной системе некоторая машина i-го уровня использует $k>2$ подчиненных машин $(i+1)$-го уровня, которым поручает решение подзадач $R_1,\ldots,R_k$. Оставим на уровне $i+1$ две машины, которым поручим решать подзадачи R_1 и $\{R_2,\ldots,R_k\}$. Второй из этих машин подчиним две машины уровня $i+2$, решающие подзадачи R_2 и $\{R_3,\ldots,R_k\}$. Действуя так же и дальше, мы заменим $(i+1)$-ый уровень исходной системы $k-1$ уровнем двоичного дерева. Так как количество подчиненных машин было ограничено полиномом от размера задачи, то общее число уровней увеличится не более чем в этот полином раз, и следовательно останется полиномиально ограниченным.

3. Каждая машина i-го уровня однозначно идентифицируется траекторией, ведущей к ней от корневой машины. Если у каждой машины не более двух подчиненных, траекторию можно задать словом длины i в алфавите {влево, вправо}. Будем передавать каждой машине траекторию вместе со входом ее подзадачи. Объединим все программы машин i-го уровня и предпошлем этой общей программе разбирательство по траектории, какую именно ее часть необходимо исполнить. Это удлинит время работы каждой машины на величину, ограниченную полиномом от размера задачи.

4. Пусть корневая машина вычисляет по своему входу и передает всем подчиненым машинам максимальный уровень терминальных машин. Передавая каждой машине траекторию, мы, в частности, даем ей возможность узнать свой собственный уровень. Пусть теперь каждая машина, обнаружив, что она терминальная и ее уровень не равен максимальному, не решает свою подзадачу, а передает ее единственной подчиненной ее машине и так далее до достижения максимального уровня.

В следующем разделе мы предъявим задачу существования из класса $N\mathcal{P}$, к которой полиномиально сводится любая задача существования, решаемая за полиномиальное время на ограниченной приведенными выше условиями параллельной машине фон Неймана.

3.4. $N\mathcal{P}$-полнота задачи о выполнимости булевой функции

Пусть $\{z_1, \dots, z_n\}$ - множество булевых переменных, принимающих значения из множества {0,1}, и пусть $F(z_1, \dots, z_n)$ - булева функция от этих переменных, записанная в базисе $(\&, \vee, \bar{\ })$. Задача существования, заключающаяся в нахождении значений переменных, доставляющих функции F истинное значение, называется задачей о *выполнимости булевой функции* (ВБФ). В качестве размера задачи удобно принять число m вхождений переменных в запись функции F (длину формулы, выражающей F); будем считать, что $m \geq n$.

Теорема Кука. Задача о выполнимости булевой функции $N\mathcal{P}$-полна.

Доказательство. Задача ВБФ принадлежит классу $N\mathcal{P}$ в силу утверждения из п. 3.1: возможные управления - значения переменных - могут быть закодированы словами длины $n \leq m$ в двоичном алфавите и функция F требует для вычисления $O(m)$ действий.

Пусть $R=(X,Y,P)$ - произвольная задача существования из класса $N\mathcal{P}$, то есть существует алгоритм α, решающий эту задачу на параллельной машине, затрачивая на решение для входа x размером r время, ограниченное полиномом от r. Это значит, что число используемых уровней не превышает $p(r)$, а время работы каждой последовательной машины ограничено величиной $q(r)$, где p и q - полиномы. В силу ограниченной адресности число ячеек памяти, используемых при этом каждой машиной, также ограничено полиномом $s(r)$. Займемся построением булевой функции, описывающей функционирование параллельной машины, выполняющей алгоритм α для входа x.

Введем булевы переменные z^j_{it}, которые будем интерпретировать как состояние i-ой ячейки памяти машины j-го уровня в t-ом такте от начала ее работы $\left(1 \leq i \leq s(r),\ 0 \leq j \leq p(r),\ 0 \leq t \leq q(r)\right)$.

Обозначим через Φ_0 функцию, описывающую соответствие начального состояния корневой машины входу x. Пусть компоненты двоичной кодировки входа $x=(x_1,\ldots,x_d)$ должны быть расположены в ячейках с номерами $i_1, \dots, i_d$. Тогда $\Phi_0 = \bigwedge_{u=1}^{d} \left(z^0_{i_u,0} = x_u\right)$.

Для построения функции f^0_t, описывающей изменение состояния корневой машины при выполнении t-го такта работы, нужно вспомнить, что в одном такте работы машины фон Неймана изменяется содержимое конечного числа ячеек с номерами $I_t = \{i_1, \dots, i_k\}$, причем это изменение определяется оператором δ_t, зависящим только от состояния ячеек из I_t. Учитывая это, имеем: $f^0_t = \bigwedge_{u=1}^{k} \left(z^0_{i_u,t+1} = \delta_t(z^0_{i_1,t}, \dots, z^0_{i_k,t})\right) \& \bigwedge_{i \notin I_t} (z^0_{i,t+1} = z^0_{i,t})$. Таким образом, функция $F_0 = \bigwedge_{t=1}^{q(r)} f^0_t$ полностью описывает работу корневой машины.

Передача корневой машиной подзадачи левой из подчиненных машин 1-го уровня заключается в переписи содержимого ячеек корневой машины с номерами $i_1, \dots, i_d$ в ячейки подчиненной машины с номерами $i'_1, \dots, i'_d$, и описывается функцией $\Phi^Л_1 = \bigwedge_{u=1}^{d} \left(z^1_{i'_u,0} = z^0_{i_u,q(r)}\right)$. Аналогично записывается функция $\Phi^П_1$, описывающая передачу подзадачи правой подчиненной машине.

Точно так же конструируются функции F_j, описывающие функционирование машины j-го уровня, и функции $\Phi^Л_j$, $\Phi^П_j$, описывающие передачу подзадач машинами $(j-1)$-го уровня своим подчиненным при $j \leqslant p(r)$.

Наконец, можно считать, что факт нахождения терминальной машиной решения фиксируется состоянием 1 некоторой ячейки с номером i, и описывается функцией $\Psi = \left(z^{p(r)}_{i,q(r)} = 1\right)$.

Рассмотрим булеву функцию $F = \Phi_0 \& F_0 \& (\Phi^Л_1 \vee \Phi^П_1) \& F_1 \& \dots \& (\Phi^Л_{p(r)} \vee \Phi^П_{p(r)}) \& F_{p(r)} \& \Psi$. Нетрудно убедиться, что длина каждого конъюнкта этой функции ограничена полиномом от размера задачи. Поскольку тем же свойством обладает и число входящих в F конъюнктов, то и длина формулы F полиномиально ограничена. Сейчас мы докажем, что функция F выполнима тогда и только тогда, когда исходная задача существования R со входом x имеет решение.

Пусть задача существования R со входом x имеет решение. Тогда алгоритм α находит это решение, приводя в соответствующее состояние некоторую терминальную машину. Присвоим каждой из переменных z^j_{it} значение, равное содержимому i-ой ячейки памяти в t-ом такте работы машины j-го уровня, лежащей на траектории от корня к терминальной машине, нашедшей решение. Из построения функции F следует, что все ее конъюнкты будут истинны при таких значениях переменных, и следовательно, функция F выполнима.

Пусть теперь функция F выполнима. Это значит, что существует такой набор значений переменных z^j_{it}, при котором истинны все конъюнкты F, и в частности, все дизъюнкции вида $\Phi^Л_j \vee \Phi^П_j$. Но для истинности этого выражения необходимо, чтобы был истинным хотя бы один из дизъюнктов - левый или правый. Составим слово в алфавите {влево,вправо}, j-ый символ которого указывает, какой из дизъюнктов выражения $\Phi^Л_j \vee \Phi^П_j$ истинен. Это слово определяет траекторию в параллельной машине фон Неймана от корневой машины к некоторой терминальной машине. Будем интерпретировать значения переменной z^j_{it} как содержимое i-ой ячейки памяти в t-ом такте работы машины j-го уровня, лежащей на этой траектории. Убеждаемся, что выполнение алгоритма α со входом x приводит терминальную машину, лежащую на конце траектории, в состояние, соответствующее нахождению ею решения. Следовательно, задача R со входом x имеет решение.

Таким образом, существует следующий алгоритм решения задачи существования R на последовательной машине фон Неймана. По входу X размера r и по алгоритму α решения задачи R на параллельной машине с полиномиальной относительно размера задачи трудоемкостью построим булеву функцию F, длина которой также ограничена полиномом от r, и решим задачу о выполнимости функции F. В случае существования решения этой задачи, решение задачи R также существует и его легко восстановить за полиномиальное время. Это и означает, что $R \underset{P}{\to}$ ВБФ. Поскольку это справедливо для любой задачи существования R из класса NP, задача ВБФ NP-полна.

На самом деле построенную нами булеву функцию F можно преобразовать в конъюктивную нормальную форму $F = \bigwedge_{u=1}^{s} \bigvee_{v=1}^{k_u} y_{uv}$, где y_{uv} – переменные или их отрицания, так что длина формы, равная $\sum_{u=1}^{s} k_u$, будет ограничена полиномом от длины записи исходной функции (этот факт мы примем без доказательства). Таким образом, имеет место более сильная теорема.

Теорема. Задача о выполнимости конъюктивной нормальной формы (КНФВ) NP-полна.

3.5. Другие примеры NP-полных задач

В этом разделе мы докажем NP-полноту нескольких задач существования (соответствующих типовым задачам оптимизации), используя лемму из п. 3.2. Принадлежность этих задач классу NP легко следует из утверждения в п. 3.1 и доказательство этого факта мы будем опускать.

Задача о *клике в графе.* Пусть $\Gamma=(V,E)$ - неориентированный граф с $|V|=n$ вершинами. Подмножество вершин $W\subseteq V$ называется *кликой* размером $|W|$, если любые две вершины из W соединены ребром в Γ. Задача о клике в графе (КГ) заключается в определении *по заданному графу Γ и числу $k<n$ содержит ли граф Γ клику размером* не меньше k. Размером входа в этой задаче будем считать число вершин n.

Теорема. КНФВ $\underset{P}{\to}$ КГ, и задача о клике в графе NP-полна.

Доказательство. Пусть задана конъюнктивная нормальная форма $F=\bigwedge_{i=1}^{s}\bigvee_{j=1}^{k_i} y_{ij}$ от булевых переменных $z_1,\dots,z_t$, где каждое y_{ij} - либо переменная, либо ее отрицание. Построим неориентированный граф $\Gamma=(V,E)$, взяв в качестве вершин $V=\{y_{ij}\}$ все вхождения переменных в форму F, и соединив два вхождения y_{ij} и $y_{i'j'}$ ребром, если эти вхождения принадлежат разным конъюнктам формы F и не являются отрицаниями друг друга: $E=\{(y_{ij},y_{i'j'}):(i\neq i')\&\&(\overline{y}_{ij}\neq y_{i'j'})\}$. Нетрудно понять, что в графе Γ нет клик размером больше s. Докажем, что в графе Γ есть клика размером s тогда и только тогда, когда форма F выполнима.

Пусть форма F выполнима. Тогда в каждом конъюнкте найдется хотя бы один истинный дизъюнкт. Пусть это вхождения $y_{1,j_1},\dots,y_{s,j_s}$. Но эти вхождения образуют клику в графе Γ, так как они принадлежат разным конъюнктам формы и не могут быть отрицаниями друг друга в силу одновременной истинности.

Пусть в графе Γ есть клика размером s. Тогда составляющие ее вхождения принадлежат различным конъюнктам формы F (по одному для каждого конъюнкта) и не являются отрицаниями друг друга. Это значит, что можно так присвоить значения переменным, чтобы все вхождения клики были истинными, а при этом и вся форма примет истинное значение.

Задача о *вершинной базе графа.* Подмножество W вершин неориентированного графа $\Gamma=(V,E)$, $|V|=n$, называется *вершинной базой* размером $|W|$, если по крайней мере один из концов каждого ребра в Γ принадлежит W. Задача о вершинной базе графа (ВБГ) состоит в определении, имеет ли данный граф Γ вершинную базу размера, не превышающего данного числа $k<n$.

Теорема. КГ $\xrightarrow[P]{}$ ВБГ, и задача о вершинной базе графа NP-полна.

Доказательство. По заданному графу $\Gamma=(V,E)$, для которого нужно решить задачу о клике, построим дополнительный граф $\bar{\Gamma}=(V,\bar{E})$, где $\bar{E}=\{(a,b),\ a,b\in V:(a\neq b)\&(a,b)\notin E\}$. Нетрудно видеть, что если $W\subseteq V$ – клика в графе Γ, то $V\setminus W$ является вершинной базой в графе $\bar{\Gamma}$ и наоборот. Действительно, вершины клики W не соединены ни одним ребром в графе $\bar{\Gamma}$, следовательно, хотя бы один конец любого ребра графа $\bar{\Gamma}$ принадлежит $V\setminus W$. Обратно, если $V\setminus W$ – вершинная база графа $\bar{\Gamma}$, то в $\bar{\Gamma}$ нет ребер, соединяющих вершины из W, следовательно, W – клика в графе Γ.

Задача *о вершинном разрезе циклов графа.* Ориентированный граф называется *ациклическим,* если из его дуг нельзя составить никакого цикла. Подмножество вершин $W\subseteq V$ ориентированного графа $\Gamma=(V,D)$ называется *вершинным разрезом циклов* размера $|W|$, если граф $\Gamma'=(V\setminus W,D')$, полученный из Γ удалением всех вершин из W и всех инцидентных им дуг, является ациклическим графом. Задача о вершинном разрезе циклов (ВРЦ) состоит в определении, имеется ли в заданном ориентированном графе $\Gamma=(V,D)$, $|V|=n$, вершинный разрез циклов размера, не превышающего заданного числа $k<n$.

Теорема. ВБГ $\xrightarrow[P]{}$ ВРЦ, и задача о вершинном разрезе циклов

NP-полна.

Доказательство. Пусть мы хотим найти вершинную базу размера k в неориентированном графе $\Gamma=(V,E)$. Построим ориентированный граф $\Gamma_1=(V,D)$, заменив каждое ребро графа Γ парой противоположно направленных дуг: $D=\{(a,b),(b,a):(a,b)\in E\}$. Утверждается, что $W\subseteq V$ – вершинная база в графе Γ тогда и только тогда, когда W – вершинный разрез циклов в графе Γ_1. Действительно, если W – вершинная база графа Γ, то в графе Γ_1', полученном из Γ_1 удалением вершин W и всех дуг, инцидентных вершинам из W, вовсе нет дуг. Обратно, если W – вершинный разрез циклов в графе Γ_1, то в графе Γ_1' нет ни одной дуги, так как вместе с любой дугой (a,b) в нем была бы и дуга (b,a) и они образовывали бы цикл.

Задача о *дуговом разрезе циклов графа*. Подмножество $M\subseteq D$ дуг ориентированного графа $\Gamma=(V,D)$ называется *дуговым разрезом циклов* размера $|M|$, если граф $\Gamma'=(V,D\setminus M)$, полученный из Γ удалением всех дуг, принадлежащих M, является ациклическим. Задача о дуговом разрезе циклов (ДРЦ) состоит в определении наличия в данном графе с n вершинами и m дугами дугового разреза циклов размера, не превышающего заданного числа $k<m$.

Теорема. ВБГ $\underset{P}{\to}$ ДРЦ, и задача о дуговом разрезе циклов NP-полна.

Доказательство. По неориентированному графу $\Gamma=(V,E)$, задачу о вершинной базе которого нам нужно решить, построим ориентированный граф $\Gamma_1=(V_1,D)$ с $2n$ вершинами и $n+2m$ дугами: $V_1=V\times\{0,1\}$, $D=D_1\cup D_2$, где $D_1=\{((a,0),(a,1))\}$ (прямые дуги), $D_2=\{((a,1),(b,0)),((b,1),(a,0)):(a,b)\in E\}$ (косые дуги). Очевидно, что любой путь в графе Γ_1 состоит из чередующихся прямых и косых дуг – всякая прямая дуга входит в вершину вида $(a,1)$, из которой исходят только косые дуги, и наоборот, всякая косая дуга входит в вершину вида $(a,0)$, из которой исходит только прямая дуга. Отсюда следует, что в любом дуговом разрезе циклов графа Γ_1, содержащем косую дугу $\big((a,1),(b,0)\big)$, можно заменить эту дугу на прямую дугу $\big((a,0),(a,1)\big)$. Таким образом, если в графе Γ_1 есть дуговой разрез циклов размера k, то в нем есть разрез не большего размера, состоящий только из прямых дуг. Докажем, что в графе Γ есть вершинная база размера k тогда и только тогда, когда в графе Γ' есть дуговой разрез циклов размера k.

Пусть W - вершинная база графа Γ. Тогда $M=\big\{\big((a,0),(a,1)\big):a\in W\big\}$ - дуговой разрез циклов в графе Γ_1. Действительно, пусть в графе $\Gamma_1'=(V_1,D\setminus M)$ есть хотя бы один цикл. Этот цикл должен содержать какую либо косую дугу $\big((a,1),(b,0)\big)$. Но тогда в этом цикле содержатся прямые дуги $\big((a,0),(a,1)\big)$ и $\big((b,0),(b,1)\big)$, а это невозможно, так как из $(a,b)\in E$ следует, что по крайней мере одна из этих дуг принадлежит M.

Пусть теперь M - дуговой разрез циклов в графе Γ_1, состоящий только из прямых дуг. Тогда $W=\big\{a\in V:\big((a,0),(a,1)\big)\in M\big\}$ - вершинная база в графе Γ. Предположим противное - пусть есть ребро $(a,b)\in E$, такое что $a\notin W$ и $b\notin W$. Но тогда дуги $\big((a,0),(a,1)\big)$, $\big((a,1),(b,0)\big)$, $\big((b,0),(b,1)\big)$ и $\big((b,1),(a,0)\big)$ образуют цикл в графе $\Gamma_1'=(V_1,D\setminus M)$.

Задача об *изоморфном вложении графов* (ИЗОВ). Для двух данных неориентированных графов $\Gamma=(V,E)$ и $\Gamma'=(V',E')$ при $|V|\geq|V'|$ *требуется* определить, существует ли взаимно однозначное соответствие $\varkappa$ множества V' и некоторого подмножества множества V, такое что $\big(\varkappa(a),\varkappa(b)\big)\in E$ для любого ребра $(a,b)\in E'$. Если такое соответствие существует, то говорят, что граф Γ' *изоморфно* *вложен* в граф Г.

Теорема. КГ $\xrightarrow[P]{}$ ИЗОВ, и задача об изоморфном вложении графов NP-полна.

Доказательство. Пусть мы хотим определить, есть ли в графе $\Gamma=(V,E)$ клика размера k. Положим $K=(1,\ldots,k)$ и определим граф $\Gamma'=(K,E')$, где $E'=((i,j),\ i,j\in K,\ i\neq j)$ (полный граф с k вершинами). Легко видеть, что в графе Г существует клика размера k тогда и только тогда, когда в граф Г изоморфно вложен граф Г'. Действительно, если в графе Г есть клика $\{a_1,\ldots,a_k\}$, то отображение π, определенное по правилу $\pi(i)=a_i$, является изоморфным вложением графа Г' в граф Г. Обратно, если отображение π осуществляет изоморфное вложение графа Г' в граф Г, то вершины $\{\pi(1),\ldots,\pi(k)\}$ образуют клику в графе Г.

Задача *квадратичного выбора* (КВ). Даны две квадратные матрицы $A=\|a_{ij}\|$ и $B=\|b_{ij}\|$ размером $n\times n$ с неотрицательными элементами. Рассматривается функционал F, определенный на перестановках π множества $(1,\ldots,n)$: $F(\pi)=\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ij}\cdot b_{\pi(i),\pi(j)}$. Спрашивается, существует ли перестановка π n-элементного множества, для которой $F(\pi)$ не меньше заданного числа C.

Теорема. КГ $\xrightarrow[P]{}$ КВ, и задача квадратичного выбора NP-полна.

Доказательство. Пусть $\Gamma=(V,E)$ - неориентированный граф с множеством вершин $V=\{v_1,\ldots,v_n\}$ и нужно определить, существует ли в нем клика размера k. Возьмем в качестве матрицы A матрицу смежности графа Г, то есть положим $a_{ij}=1$, если $(v_i,v_j)\in E$, и $a_{ij}=0$ в противном случае. Матрицу B определим как матрицу смежности полного k-вершинного графа, к которому добавлено $n-k$ изолированных вершин: $b_{ij}=1$, если $i,j\leq k$ и $i\neq j$; в остальных случаях

$b_{ij}=0$. В матрице B $k(k-1)/2$ единиц, поэтому значение функционала в задаче квадратичного выбора с матрицами A и B не превышает $k(k-1)/2$. Легко показать, что значение этого функционала равно $k(k-1)/2$ тогда и только тогда, когда в графе Г есть клика размера k.

Задача *целочисленного линейного программирования* (ЦЛП). В пространстве R^n задан выпуклый многогранник, определяемый системой линейных неравенств $AX\leq B$, где X – n-мерный вектор координат, B – m-мерный вектор, A – матрица размером $m\times n$. Требуется определить, есть ли в этом многограннике точка с целочисленными координатами.

Теорема. КГ $\underset{P}{\to}$ ЦЛП, и задача целочисленного линейного программирования NP-полна.

Доказательство. Задачу определения, есть ли в графе $\Gamma=(V;E)$, $V=\{v_1,\dots,v_n\}$ клика размера k, легко сформулировать в терминах линейных неравенств. Поставим в соответствие вершине v_i переменную x_i с ограничениями $0\leq x_i\leq 1$, причем значение $x_i=1$, будем интерпретировать, как принадлежность v_i искомой клике. Невозможность вхождения в клику двух вершин, не соединенных ребром, описывается неравенством $x_i+x_j\leq 1$ для любой пары различных вершин $(v_i,v_j)\notin E$. Наконец, ограничение на размер искомой клики можно задать неравенством $x_1+\dots+x_n\geq k$. Таким образом, для получения решения задачи о клике в графе с n вершинами и m ребрами достаточно решить задачу целочисленного линейного программирования с n переменными и $n(n-1)/2+2n-m+1$ ограничениями.

Сформулируем еще несколько NP-полных задач без доказательства их принадлежности классу $\overline{NP}$.

Задача о *гамильтоновом цикле в графе.* Требуется определить, существует ли в заданном ориентированном графе гамильтонов цикл – цикл, проходящий через каждую вершину графа в точности по одному разу.

Задача о *раскраске графа.* Раскраской неориентированного графа $\Gamma=(V,E)$ называется отображение $\varphi: V \to C$ вершин графа в конечное множество *цветов* C. Раскраска φ – *правильная,* если смежные вершины раскрашены в разные цвета: $\varphi(a)\neq\varphi(b)$ для любого ребра $(a,b)\in E$. Для заданного графа нужно определить, существует ли его правильная раскраска не более, чем в k цветов.

Задача о *покрытии множества.* Задана система подмножеств $S=\{A_1,\ldots,A_n\}$ конечного множества A. Спрашивается, существует ли подмножество $I\subseteq\{1,\ldots,n\}$ n-элементного множества, такое что $|I|\leq k$ и $\bigcup\limits_{i\in I} A_i = A$.

Задача о *рюкзаке.* Заданы n пар положительных чисел $(w_1,c_1),\ldots,(w_n,c_n)$. Нужно определить, существует ли подмножество $I\subseteq\{1,\ldots,n\}$ n-элементного множества, такое что $\sum\limits_{i\in I} w_i \leq W$ и $\sum\limits_{i\in I} c_i \geq C$. Если числа w_i интерпретировать как веса предметов, а c_i как их стоимости, то задача о рюкзаке заключается в выборе предметов с суммарным весом не больше заданного W и суммарной стоимостью не меньше заданной C.

3.6. Центральная проблема теории сложности

В предыдущих разделах этой главы мы ввёли два полиномиально замкнутых класса дискретных задач: класс $\mathcal{P}$ самых простых задач в $N\mathcal{P}$ (любая задача из $\mathcal{P}$ полиномиально сводится к любой задаче из $N\mathcal{P}$) и класс $\overline{N\mathcal{P}}$ самых сложных задач в $N\mathcal{P}$ (любая задача из $N\mathcal{P}$ полиномиально сводится к любой задаче из $\overline{N\mathcal{P}}$). Однако мы нигде не утверждали, что классы $\mathcal{P}$ и $\overline{N\mathcal{P}}$ не пересекаются. Здесь, в принципе, могут быть две ситуации. Если $\mathcal{P}\cap\overline{N\mathcal{P}}\neq\emptyset$, то есть существует полиномиально разрешимая $N\mathcal{P}$-полная задача, то из определения класса $\overline{N\mathcal{P}}$ следует, что $\overline{N\mathcal{P}}\subset\mathcal{P}$ и $N\mathcal{P}=\mathcal{P}$. В противном случае класс $N\mathcal{P}$ действительно шире класса $\mathcal{P}$.

Какая же из этих двух альтернатив имеет место? Для ответа на этот вопрос достаточно либо построить алгоритм решения какой либо $N\mathcal{P}$-полной задачи с полиномиальной трудоемкостью, либо для какой либо задачи из класса $N\mathcal{P}$ получить неполиномиальную нижнюю оценку сложности. Несмотря на массированные усилия мощных научных коллективов во всех странах мира, ответ на этот вопрос до сих пор не получен, и он остается центральной проблемой теории сложности. Более того, есть основания думать, что аксиоматики теории множеств, на основе которой построена теория сложности, вообще недостаточно для решения этой проблемы (возможно, ситуация здесь такая же, как в знаменитой континуум гипотезе).

С точки зрения решения практических дискретных задач это заставляет действовать, считая, что для $N\mathcal{P}$-полных задач не существует алгоритмов с существенно меньшей трудоемкостью, чем алгоритм полного перебора. Получив дискретную задачу неизвестной сложности, нужно либо построить алгоритм ее решения с полиномиальной трудоемкостью (например, путем полиномиального сведения ее к какой либо известной задаче из класса $\mathcal{P}$), либо доказать ее $N\mathcal{P}$-полноту, полиномиально сведя к ней какую либо из известных $N\mathcal{P}$-полных задач. Основные приемы конструирования эффективных алгоритмов и некоторые типы полиномиально разрешимых задач будут описаны в разделах 4-7. Если же удалось доказать принадлежность задачи классу $\overline{N\mathcal{P}}$, не следует пытаться построить эффективный алгоритм ее решения (его, скорее всего, не существует), а нужно использовать алгоритм полного перебора с разумными сокращениями, описанными в разделе 8. Альтернативой здесь может быть тщательный анализ математической модели с целью выяснения дополнительных ограничений на множество входов. Часто оказывается, что задача $R=(X,Y,P)$ $N\mathcal{P}$-полна, но задача $R'=(X',Y,P)$ для $X'\subset X$ полиномиально разрешима. Например, хотя задача о клике в графе принадлежит к классу $\overline{N\mathcal{P}}$, нахождение клики заданного размера в графе ограниченной валентности можно выполнить за время, полиномиальное относительно числа вершин графа.

Иногда для задачи не удается ни построить эффективный алгоритм решения, ни доказать $N\mathcal{P}$-полноту. Такие задачи представляют особый интерес с точки зрения теории сложности и оказываются в центре внимания исследователей, работающих в этой области. Долгое время в такой ситуации была общая задача линейного программирования, эффективный алгоритм для ее решения построен лишь недавно. Сейчас такой является задача об *изоморфизме графов* - частный случай задачи об изоморфном вложении графов с равным числом вершин.

Вопросы для самостоятельной работы

1. Доказать, что любая задача из класса $\mathcal{P}$ полиномиально сводится к любой задаче из класса $N\mathcal{P}$.

2. Подмножество W вершин графа $\Gamma=(V,E)$ называется *независимым*, если $(a,b)\notin E$ для любых $a,b\in W$. Доказать $N\mathcal{P}$-полноту задачи нахождения в заданном графе независимого множества заданного размера.

*3. Доказать $N\mathcal{P}$-полноту задачи о коммивояжере с целыми ограниченными расстояниями, полиномиально сведя к ней задачу о гамильтоновом цикле в графе.

4. Доказать $\mathcal{P}\cap\overline{N\mathcal{P}}\neq\varnothing \to \mathcal{P}=N\mathcal{P}=\overline{N\mathcal{P}}$.

5. Построить алгоритм с полиномиальной трудоемкостью для нахождения клики заданного размера в графе ограниченной валентности.

*6. Пусть $S=\{A_1,\ldots,A_n\}$ - система подмножеств m-элементного множества A. Показать $N\mathcal{P}$-полноту задачи об *упаковке подмножеств*, заключающуюся в нахождении заданного числа $k<n$ непересекающихся подмножеств из S, полиномиально сведя к ней задачу о клике в графе.

4. АЛГОРИТМЫ УМНОЖЕНИЯ БУЛЕВСКИХ МАТРИЦ

4.1. Пути в графах и алгебра булевских матриц

Пусть $\Gamma_1=(V,D_1)$ и $\Gamma_2=(V,D_2)$ - ориентированные графы с одним и тем же множеством вершин V. Определим граф $\Gamma=(V,D)=\Gamma_1\circ\Gamma_2$, называемый *композицией* графов Γ_1 и Γ_2, следующим образом: $(v,v')\in D \Leftrightarrow \exists w\in V:(v,w)\in D_1 \,\&\, (w,v')\in D_2$. Рассмотрим квадратную булевскую матрицу $A=\|a_{ij}\|$, $a_{ij}\in\{0,1\}$, размером $n\times n$ как матрицу смежности $A(\Gamma)$ ориентированного графа Γ с n вершинами. Определим на булевских матрицах одинакового размера операцию умножения, порожденную операцией композиции графов, матрицами смежности которых *они являются*: $A(\Gamma_1\circ\Gamma_2)=A(\Gamma_1)\cdot A(\Gamma_2)$. Если $A=\|a_{ij}\|$, $B=\|b_{ij}\|$ и $C=A\cdot B=\|c_{ij}\|$, то справедлива формула $c_{ij}=\bigvee_{k=1}^{n} a_{ik}\&b_{kj}$, легко выводимая из определения композиции графов.

Аналогично, операция *объединения* графов $\Gamma_1=(V,D_1)$ и $\Gamma_2=(V,D_2)$: $\Gamma=\Gamma_1\cup\Gamma_2=(V,D_1\cup D_2)$ порождает операцию сложения булевских матриц: $A(\Gamma_1\cup\Gamma_2)=A(\Gamma_1)+A(\Gamma_2)$, для которой справедлива формула $C=A+B \Leftrightarrow c_{ij}=a_{ij}\vee b_{ij}$.

Таким образом, имеем две изоморфные алгебры: алгебру графов с операциями объединения и композиции, и алгебру булевских матриц с операциями сложения и умножения. Это означает, что любой факт в алгебре графов может быть выражен в терминах булевских матриц, что и объясняет большую роль введенных операций над булевскими матрицами в конструировании алгоритмов для ориентированных графов.

В силу ассоциативности умножения булевских матриц (проверьте!) можно определить *степень* A^k булевской матрицы A как произведение k ее экземпляров. Степень булевской матрицы легко интерпретируется в алгебре графов. Если $A=A(\Gamma)$ - матрица смежности графа $\Gamma=(V,D)$, то $A^k=A(\Gamma^k)$ - матрица смежности графа $\Gamma^k=(V,D^k)$, определяемого следующим образом: $(v,v')\in D^k \leftrightarrow$ (в графе Γ существует путь длины k из v в v').

Будем говорить, что вершина y в графе Γ *достижима* из вершины x, если в графе Γ существует путь из x и y. Очевидно, что отношение достижимости рефлексивно и транзитивно, то есть является отношением толерантности. Можно показать, что отношение достижимости в графе Γ - минимальное по включению отношение толерантности, содержащее отношение смежности графа Γ. Для графа $\Gamma=(V,D)$ определим граф $\bar{\Gamma}=(V,\bar{D})$: $(v,v')\in\bar{D} \leftrightarrow$ (вершина v' достижима в графе Γ из вершины v), называемый *транзитивным замыканием* графа Γ. Учитывая, что если вершина y достижима из вершины x в графе Γ с n вершинами, то в этом графе существует путь из x в y длиной $0<k<n-1$, получим следующую интерпретацию графа $\bar{\Gamma}$ в алгебре матриц: $\bar{A}(\Gamma)=A(\bar{\Gamma})=\sum_{k=0}^{n-1} A(\Gamma)^k$ ($A^0=E$ -единичная матрица).

Приведенные выше формулы дают алгоритмы решения задач сложения булевских матриц (СБМ), умножения булевских матриц (УБМ) и построения транзитивного замыкания графа (ТЗ) с трудоемкостью $O(n^2)$, $O(n^3)$ и $O(n^4)$, соответственно. Из информационных соображений ясно, что сложность этих задач не может быть меньше, чем $O(n^2)$. Таким образом, имеем $t_{СБМ}(n)=O(n^2)$, $O(n^2)<t_{УБМ}(n)<O(n^3)$ и $O(n^2)<t_{ТЗ}(n)<O(n^4)$. В следующих разделах этой главы мы пока-

жем линейную эквивалентность задач умножения булевских матриц и построения транзитивного замыкания, и опишем два алгоритма умножения булевских матриц с трудоемкостью меньшей, чем $O(n^3)$.

4.2. Эквивалентность задач умножения булевских матриц и построения транзитивного замыкания графа

Утверждение. ТЗ → УБМ.

Доказательство. Пусть нам нужно найти транзитивное замыкание графа $\Gamma=(V,D)$, $|V|=n$. Разобьем множество вершин V на два подмножества V_1 и V_2: $|V_1|=|V_2|=n/2$. Тогда множество дуг D разобьется на четыре подмножества $D_{ij}=\{(a,b)\in D: a\in V_i \,\&\, b\in V_j\}$, $i,j\in\{1,2\}$. Пронумеровав вершины графа так, чтобы вершины из V_1 получили меньшие номера, чем вершины из V_2, представим матрицу смежности графа Γ в виде $A=A(\Gamma)=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$, где блоки A_{ij}, $i,j\in\{1,2\}$, размером $n/2\times n/2$ соответствуют подмножествам дуг D_{ij}. Матрицу смежности транзитивного замыкания будем искать в таком же виде: $B=\overline{A}=A(\overline{\Gamma})=\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$, где матрицы B_{ij}, $i,j\in\{1,2\}$, описывают достижимость вершин подмножества V_j из вершин подмножества V_i. По такому описанию матриц A и B в терминах графа нетрудно получить выражения для блоков матрицы B в алгебре булевских матриц, порожденной блоками матрицы A.

Любой путь из вершины $v\in V_1$ в вершину $v'\in V_1$ можно составить из путей двух типов. Каждый путь первого типа состоит из единственной дуги $d\in D_{11}$, и все такие пути описываются матрицей A_{11}. Пути второго типа имеют вид $d\,L\,d'$, где $d\in D_{12}$, $d'\in D_{21}$, а путь L целиком состоит из дуг, принадлежащих D_{22}. Пути такого типа описываются матрицей $A_{12}\cdot\overline{A_{22}}\cdot A_{21}$. Таким образом $B_{11}=\overline{S}$, где $S=A_{11}+A_{12}\cdot\overline{A_{22}}\cdot A_{21}$.

Любой путь L из вершины $v\in V_1$ в вершину $v'\in V_2$ можно представить в виде $L_1 d L_2$, где L_1 – путь из вершины v в вершину w – последнюю вершину в пути L, принадлежащую V_1; $d\in D_{12}$; L_2 – путь, целиком состоящий из дуг, принадлежащих D_{22}. Отсюда следует, что $B_{12}=B_{11}\cdot A_{12}\cdot\bar{A}_{22}$.

Аналогично выводятся формулы $B_{21}=\bar{A}_{22}\cdot A_{21}\cdot B_{11}$ и $B_{22}=\bar{A}_{22}+$ $+\bar{A}_{22}\cdot A_{21}\cdot B_{12}$.

Для вычисления по полученным формулам матрицы B размером $n\times n$ требуется построить транзитивные замыкания матриц A_{22} и S размером $n/2\times n/2$ и, p и q раз (значения p и q не играют роли) произвести умножение и сложение булевских матриц размером $n/2\times n/2$. Таким образом, $t_{ТЗ}(n)\leq 2t_{ТЗ}(n/2)+pt_{УБМ}(n/2)+qt_{СБМ}(n/2)$. Так как $t_{ТЗ}(n)\geq O(n^2)$, то предположив, что $t_{ТЗ}(n)=O(n^{2+\varepsilon})$, $\varepsilon>0$, получим $t_{ТЗ}(n/2)\leq t_{ТЗ}(n)/4$. Аналогично, из предположения $t_{УБМ}(n)=O(n^{2+\varepsilon})$, $\varepsilon>0$, следует $t_{УБМ}(n/2)\leq t_{УБМ}(n)/4$. Подстановка этих неравенств дает $t_{ТЗ}(n)\leq(p/2)t_{УБМ}(n)+O(n^2)$. Учитывая, что $t_{УБМ}(n)\geq O(n^2)$, получаем окончательно $t_{ТЗ}(n)\leq O\big(t_{УБМ}(n)\big)$, что и означает линейную сводимость задачи ТЗ к задаче УБМ.

Тем самым, естественное ощущение, что задача ТЗ сложнее задачи УБМ, оказывается неверным. В частности, из $t_{УБМ}(n)\leq O(n^3)$ следует $t_{ТЗ}(n)\leq O(n^3)$.

Использованный нами при доказательстве метод сведения задачи к той же задаче меньшего размера широко применяется как при анализе сложности дискретных задач, так и при конструировании алгоритмов их решения. При этом часто предполагается, что размер задачи обладает определенными свойствами делимости. Так, в приведенном выше доказательстве неявно считается, что число вершин графа n является степенью двойки. Однако это не ограничивает общности, так как мы можем увеличить n до ближайшей степени двойки, добавив к графу необходимое число изолированных вершин (из которых не исходит и в которые не входит ни одна дуга). Размер задачи увеличится при этом не более чем вдвое, и это не повлияет на оценку трудоемкости алгоритма. Аналогичные рассуждения проходят и в других случаях, когда требуются те или иные свойства делимости размера задачи; в дальнейшем мы будем эти рассуждения опускать.

Утверждение. УБМ → ТЗ.

Доказательство. Пусть нам нужно найти произведение $A \cdot B$ квадратных булевских матриц размером $n \times n$. Построим матрицу M размером $3n \times 3n$, состоящую из 3×3 блоков размером $n \times n$:

$$M = \begin{pmatrix} 0 & A & 0 \\ 0 & 0 & B \\ 0 & 0 & 0 \end{pmatrix}.$$ Легко проверить, что $$M^2 = \begin{pmatrix} 0 & 0 & A \cdot B \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$ и $M^k = 0$ при $k > 2$.

Таким образом, $$\bar{M} = \begin{pmatrix} E & A & A \cdot B \\ 0 & E & B \\ 0 & 0 & E \end{pmatrix}$$ и правый верхний блок транзитивного замыкания равен искомому произведению. Так как построение матрицы M по матрицам A и B и выделение произведения $A \cdot B$ из матрицы $\bar{M}$ может быть выполнено за $O(n^2)$ действий, $t_{УБМ}(n) \leqslant t_{ТЗ}(3n) + O(n^2)$. Так как $t_{ТЗ}(n) \leqslant O(n^3)$, то предположение $t_{ТЗ}(n) = O(n^{3-\varepsilon})$, $\varepsilon > 0$, приводит к $t_{ТЗ}(3n) \leqslant 27 t_{ТЗ}(n)$. Поскольку $t_{ТЗ}(n) \geqslant O(n^2)$, получаем $t_{УБМ}(n) \leqslant O\left(t_{ТЗ}(n)\right)$, что и означает линейную сводимость задачи УБМ к задаче ТЗ.

Поскольку мы показали, что УБМ → ТЗ и ТЗ → УБМ, нами доказана следующая теорема.

Теорема. Задачи умножения булевских матриц и построения транзитивного замыкания линейно эквивалентны.

Утверждение теоремы, в частности, означает, что транзитивное замыкание можно построить с той же трудоемкостью, с какой мы умеем умножать булевские матрицы. Алгоритм решения задачи ТЗ с трудоемкостью $O(n^3)$, но более простой, чем можно получить с помощью описанного сведения, будет изложен в разделе 5. Два эффективных алгоритма умножения булевских матриц, к описанию которых мы переходим, позволяют сконструировать алгоритмы решения задачи ТЗ с трудоемкостью меньшей, чем $O(n^3)$.

4.3. Алгоритм четырех русских

Введенные в п. 4.1 операции сложения и умножения квадратных булевских матриц естественным образом распространяются на прямоугольные булевские матрицы, в том числе на вектор-строки (матрицы размером $1\times n$), вектор-столбцы (матрицы размером $n\times 1$) и скаляры (матрицы размером 1×1).

Для нахождения произведения $A\cdot B$ булевских матриц размером $n\times n$ представим матрицу A в виде $A=(A_1,\dots,A_k)$, а матрицу B в виде $B=\begin{pmatrix} B_1 \\ \dots \\ B_k \end{pmatrix}$, где A_i – матрицы размером $(n/k)\times n$, а B_i – матрицы размером $n\times(n/k)$. Искомое произведение при этом определяется формулой $A\cdot B=\sum_{i=1}^{k} A_i\cdot B_i$.

Займемся теперь вычислением произведения $R=S\cdot T$ прямоугольных булевских матриц $S=\|s_{ij}\|$ размером $n\times m$ и T размером $m\times n$, $m<n$. Если через R_i и T_i обозначить i-ые строки матриц R и T, соответственно, то можно записать $R_i=\sum_{j=1}^{n} s_{ij}\cdot T_j$. Обозначив через $U_i=\{1\leq j\leq m : s_{ij}=1\}$ множество столбцов матрицы S, в которых i-ая строка содержит единицы, перепишем предыдущее равенство в виде

$R_i=\sum_{j\in U_i} T_j$, из которой видно, что каждая строка произведения $R=S\cdot T$ является суммой некоторых строк матрицы T. Излагаемый алгоритм получения произведения $R=S\cdot T$ состоит из двух этапов: вначале строится совокупность всевозможных сумм строк матрицы T, а затем из полученной совокупности по матрице S выбираются необходимые элементы в качестве строк матрицы R.

На множестве $X=2^{\{1,\dots,m\}}$ всех подмножеств m-элементного множества введем функцию $\nu: X \to \{0,1,\dots,2^m-1\}$, ставящую в соответствие подмножеству $x=\{x_1,\dots,x_p\}$ его *номер* $\nu(x)=\sum_{i=1}^{p} 2^{x_i-1}$. Нетрудно видеть, что соответствие ν взаимно однозначное, то есть по номеру $q=\nu(x)$ однозначно вычисляется подмножество $x=\nu^{-1}(q)$. Рассмотрим матрицу Z размером $2^m\times n$, строки которой пронумерованы от 0 до 2^m-1, так что i-ая строка матрицы Z есть сумма строк матрицы T по подмножеству $\nu^{-1}(i)$: $Z_i=\sum_{j\in\nu^{-1}(i)} T_j$. Нулевая строка матрицы Z соответствует пустому подмножеству и целиком состоит из нулей. Введем на множестве $\{1,\dots,2^m-1\}$ функции α и β: $\alpha(i)=\min\nu^{-1}(i)$ – минимальный элемент из $\nu^{-1}(i)$, и $\beta(i)=i-2^{\alpha(i)-1}$ – номер подмножества $\nu^{-1}(i)\setminus\{\alpha(i)\}$. Легко проверить, что $\nu^{-1}(i)=\nu^{-1}(\beta(i))\cup\{\alpha(i)\}$, и выражение для Z_i при $i>1$ можно переписать в виде $Z_i=Z_{\beta(i)}+T_{\alpha(i)}$. Так как $\beta(i)<i$, а на вычисление $\alpha(i)$ и $\beta(i)$ достаточно $O(m)$ действий, по этой формуле можно вычислять строки матрицы Z в порядке возрастания номеров, затрачивая на каждую строку $O(n)$ действий, а на вычисление всей матрицы – $O(n2^m)$ действий. После построения матрицы Z матрица R вычисляется по формуле $R_i=Z_{\nu(U_i)}$ за $O(n^2)$ действий. Итого, трудоемкость описанного алгоритма умножения прямоугольных булевских матриц равна $O(n^2+n2^m)$.

Вернемся теперь к исходной задаче получения произведения квадратных матриц размером $n\times n$. Для ее решения нам нужно k раз найти произведение прямоугольных матриц A_i размером $n\times m$ и B_i размером $m\times n$, где $m=n/k$. При $k=n/\log n$, то есть при $m=\log n$, произведение $A_i\cdot B_i$ изложенным выше алгоритмом можно найти за $O(n^2)$ действий, а все $n/\log n$ произведений, необходимых для вычисления $A\cdot B$, за $O(n^3/\log n)$ действий. Такую же трудоемкость имеют $n/\log n$ сложений матриц $A_i\cdot B_i$. Таким образом, трудоемкость умножения булевских матриц размером $n\times n$ алгоритмом четырех русских равна $O(n^3/\log n)$. Отметим, что при тщательной программной реализации этот алгоритм работает быстрее простейшего алгоритма с трудоемкостью $O(n^3)$ уже при n порядка 50.

4.4. Алгоритм Штрассена

Наряду с введенным в п. 4.1 произведением $C=A\cdot B=\|c_{ij}\|$ булевских матриц $A=\|a_{ij}\|$ и $B=\|b_{ij}\|$ размером $n\times n$, рассмотрим их обычное произведение над кольцом целых чисел $D=AB=\|d_{ij}\|$, определяемое формулой $d_{ij}=\sum\limits_{k=1}^{n} a_{ik}b_{kj}$. Сравнивая формулы для c_{ij} и d_{ij}, легко установить, что $c_{ij}=0 \Leftrightarrow d_{ij}=0$. Поскольку вычисление матрицы C по матрице D выполняется за $O(n^2)$ действий, а $t_{УБМ}(n)\geq O(n^2)$, имеет место линейная сводимость задачи УБМ к задаче умножения матриц (УМ) над кольцом целых чисел.

В связи с этим займемся задачей УМ. Представим каждую из матриц A, B и $D=AB$ в виде блочной матрицы с 2×2 блоками размером $n/2\times n/2$: $\begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix}=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$. Матрицы D_{ij}, $i,j\in\{1,2\}$ определяются формулами

$$D_{11}=A_{11}B_{11}+A_{12}B_{21};\quad D_{12}=A_{11}B_{12}+A_{12}B_{22};$$

$$D_{21}=A_{21}B_{11}+A_{22}B_{21};\quad D_{22}=A_{21}B_{12}+A_{22}B_{22},$$

для вычисления по которым требуется выполнить 8 умножений и 4 сложения матриц размером $n/2\times n/2$.

Штрассен изобрел остроумный способ вычисления D_{ij}, $i,j\in(1,2)$ за 7 умножений и 18 сложений/вычитаний. Для этого сначала вычисляются вспомогательные матрицы

$$M_1=(A_{12}-A_{22})(B_{21}+B_{22});\quad M_2=(A_{11}+A_{22})(B_{11}+B_{22});$$

$$M_3=(A_{11}-A_{21})(B_{11}+B_{12});\quad M_4=(A_{11}+A_{12})B_{22};$$

$$M_5=A_{11}(B_{12}-B_{22});\quad M_6=A_{22}(B_{21}-B_{11});\quad M_7=(A_{21}+A_{22})B_{11}.$$

После этого вычисляются матрицы D_{ij}, $i,j\in(1,2)$:

$$D_{11}=M_1+M_2-M_4+M_6;\quad D_{12}=M_4+M_5;$$

$$D_{21}=M_6+M_7;\quad D_{22}=M_2-M_3+M_5-M_7.$$

Оценим трудоемкость алгоритма Штрассена, имея в виду, что вычисления произведений матриц размером $n/2\times n/2$ будут производится тем же методом с доведением рекурсии до матриц размером 1×1, умножение которых требует константы действий. Имеем $t_{УМ}(n)<7t_{УМ}(n/2)+18t_{СМ}(n/2)<7^2t_{УМ}(n/4)+7\cdot 18t_{СМ}(n/4)+18t_{СМ}(n/2)<$

$$<\ldots<7^k t_{УМ}(n/2^k)+18\sum_{i=0}^{k-1}7^i t_{СМ}(n/2^{i+1})<7^{\log_2 n}+(7/4)^{\log_2 n}O(n^2)=$$

$=O(n^{\log_2 7})\sim O(n^{2.81})$.

Таким образом, имеем алгоритм умножения булевских матриц с трудоемкостью $O(n^{\log_2 7})<O(n^3/\log n)$. Следует отметить, что хотя алгоритм Штрассена асимптотически эффективнее алгоритма четырех русских, из-за накладных расходов рекурсивной программной реализации он работает быстрее алгоритма четырех русских только начиная с n порядка нескольких тысяч.

Вопросы для самостоятельной работы

1. Доказать ассоциативность умножения булевских матриц.

*2. Доказать, что минимальное по включению отношение толерантности, содержащее отношение смежности графа Г, единственно и совпадает с отношением достижимости в этом графе.

3. Вывести формулы $B_{21}=\bar{A}_{22}\cdot A_{21}\cdot B_{11}$ и $B_{22}=\bar{A}_{22}+\bar{A}_{22}\cdot A_{21}\cdot B_{12}$ в доказательстве сведения ТЗ → УБМ.

4. Разработать алгоритмы вычисления $v(U_i)$, $\alpha(i)$ и $\beta(i)$, необходимые для реализации алгоритма четырех русских, с трудоемкостью $O(m)$.

*5. Доказать, что если научиться умножать блочные $k\times k$ матрицы с размером блоков $n/k\times n/k$ с использованием $s<k^3$ умножений блоков, то можно построить алгоритм умножения матриц штрассеновского типа с трудоемкостью $O(n^{\log_k s})<O(n^3)$. За сколько умножений блоков нужно умножать блочные матрицы размером 3×3, чтобы получить алгоритм с меньшей трудоемкостью, чем алгоритм Штрассена?

5. АЛГОРИТМЫ, ОСНОВАННЫЕ НА ОБХОДАХ ГРАФОВ

5.1. Обходы графа

Пусть $\Gamma=(V,D)$ – ориентированный граф. Последовательность дуг $d_1, d_2, \ldots, d_s$ графа Γ называется его *обходом*, если каждая дуга $d\in D$ встречается в этой последовательности не более одного раза. Если каждая дуга $d\in D$ встречается в обходе ровно один раз, обход называется *полным*. Накладывая на последовательность дуг различные дополнительные ограничения, будем получать частные случаи обходов – обходы различных типов.

Опишем несколько типов обходов, лежащих в основе эффективных алгоритмов решения многих задач для ориентированных графов.

Обход исчерпыванием. Обход этого типа удовлетворяет следующим условиям:

1- дуга $(a,v)\in D$ может встретиться в обходе только после всех дуг $(v',a)\in D$;

2- все дуги $(a,v)\in D$, исходящие из одной и той же вершины a, расположены в обходе подряд.

Обход исчерпыванием удобно использовать для решения задач на ациклических графах (п. 5.2).

Фронтальный обход с корнем x. Этот тип обхода описывается правилами:

1- дуги $(x,v)\in D$, исходящие из корня обхода, расположены в обходе раньше всех остальных дуг;

2- дуга $(a,v)\in D$ при $a\neq x$ может встретиться в обходе только после какой-либо дуги $(v',a)\in D$;

3- все дуги $(a,v)\in D$, исходящие из одной и той же вершины a, расположены в обходе подряд.

На основе фронтального обхода строятся алгоритмы анализа метрических свойств графа (п. 5.3). Фронтальный обход графа часто называют *поиском в ширину*.

Радиальный обход с корнем x удовлетворяет следующим условиям:

1- первой дугой обхода является какая-либо дуга $(x,v)\in D$, исходящая из корня обхода;

2- дуга $(a,v)\in D$ при $a\neq x$ может встретиться в обходе только после какой-либо дуги $(v',a)\in D$;

3- если в обходе есть дуга $(a,b)\in D$, являющаяся первой в обходе дугой, заходящей в вершину b, то следующая дуга, исходящая из вершины a, расположена в обходе после всех дуг, исходящих из вершины b.

На радиальном обходе основаны алгоритмы анализа циклической структуры графа (п. 5.4). Радиальный обход графа часто называют *поиском в глубину*.

При оценке трудоемкости алгоритмов на графах с n вершинами и m дугами мы всегда будем предполагать, что $O(n)<m<O(n^2)$.

5.2. Алгоритмы для ациклических графов

Ориентированный граф $\Gamma=(V,D)$ называется *ациклическим*, если из его дуг нельзя образовать ни одного цикла. Для вершины $v\in V$ обозначим через $s^+(v)$ количество дуг, заходящих в вершину v (*полустепень захода*), а через $s^-(v)$ - количество дуг, исходящих из вершины v (*полустепень исхода*).

Важное свойство ациклического графа устанавливает следующая простая лемма.

Лемма. В ациклическом графе $\Gamma=(V,D)$ найдется вершина v, такая что $s^+(v)=0$.

Доказательство. Предположим противное: пусть в ациклическом графе $\Gamma=(V,D)$ полустепень захода $s^+(v)>0$ для всех $v\in V$. Возьмем произвольную вершину $v_1\in V$. Так как по предположению $s^+(v_1)>0$, то существует дуга $(v_2,v_1)\in D$. Аналогично из $s^+(v_2)>0$ следует существование дуги $(v_3,v_2)\in D$. Продолжая этот процесс, получим бесконечную последовательность $v_1,v_2,\ldots$ вершин графа, такую что $(v_{i+1},v_i)\in D$. Но в силу конечности $|V|$, в этой последовательности какая-либо вершина встретится дважды. При этом дуги, соединяющие вершины последовательности, расположенные между вхождениями одной и той же вершины, образуют цикл, что противоречит предположению об ацикличности графа Γ.

Основанием для использования обхода исчерпыванием для решения некоторых задач на ациклических графах является следующая теорема.

Теорема (о полном обходе исчерпыванием). Полный обход исчерпыванием графа $\Gamma=(V,D)$ существует тогда и только тогда, когда этот граф - ациклический.

Доказательство. Пусть из дуг графа Γ можно составить цикл $(v_0,v_1),(v_1,v_2),\ldots,(v_s,v_0)$. Тогда никакая дуга из этого цикла не может входить в обход исчерпыванием, так как она может быть расположена в обходе только после предшествующей дуги цикла, та, в свою очередь, только после предшествующей ей дуги цикла, и так далее. В конце концов получаем, что каждая дуга цикла может быть расположена в обходе исчерпыванием только после себя самой.

Наоборот, если граф Γ - ациклический, можно предложить алгоритм построения полного обхода исчерпыванием. В соответствие с леммой, в графе Г найдется вершина $v \in V$, для которой $s^+(v)=0$. Удалим из графа вершину v и все дуги, исходящие из этой вершины. Получившийся граф Г', очевидно, ациклический и мы можем повторять описанную процедуру редукции до исчерпания вершин и дуг графа. Нетрудно проверить, что последовательность удаления дуг из графа является его полным обходом исчерпыванием, что и завершает доказательство теоремы.

В доказательстве теоремы, по существу, описан алгоритм, позволяющий проверить, является ли граф Г ациклическим. Для этого нужно применять к графу Г процедуру редукции либо до исчерпания его элементов (тогда граф Г - ациклический, так как построен его полный обход исчерпыванием), либо до тех пор, когда в редуцированном графе Г' полустепени исхода всех вершин окажутся положительными (в этом случае, в соответствие с леммой, в графе Г', а следовательно, и в графе Г есть циклы). При подходящем выборе структур данных описанный алгоритм имеет на графе с m дугами трудоемкость $O(m)$.

Пронумеруем вершины ациклического графа в порядке вхождения исходящих из них дуг в полный обход исчерпыванием. Легко видеть, что такая нумерация, называемая *топологической*, имеет замечательное свойство: у каждой дуги графа номер ее конца больше номера начала.

Рассмотрим задачу дискретной оптимизации, известную под названием задачи *сетевого планирования*. Имеется множество $A=\{a_1, a_2, \ldots, a_n\}$ работ, подлежащих выполнению, причем для каждой работы $a \in A$ задана ее длительность $\tau(a)>0$. Кроме того, на множестве работ определено бинарное отношение $R \subset A^2$: $(a,b) \in R$ означает, что выполнение работы b может быть начато только после завершения работы a (работа a *предшествует* работе b). Требуется найти минимально возможные сроки начала и окончания всех работ (*календарный план*). В качестве модели этой задачи удобно выбрать взвешенный ориентированный граф $\Gamma=(V,D)$, вершины которого соответствуют моментам начала и окончания работ $V=A\times\{н,к\}$. Вершины (a,н) и (a,к) соединим дугой с длиной $\tau(a)$ для всех $a\in A$, а вершины (a,к) и (b,н) соединим дугой с нулевой длиной для всех $(a,b)\in R$. Таким образом, $|V|=2|A|$, $|D|=|A|+|R|$. Построенный граф носит название *сетевого графика*. Нетрудно убедиться, что максимумы длин путей, заканчивающихся в вершинах графа, дают искомый календарный план, и что весь комплекс работ может быть выполнен за конечное время только тогда, когда граф Γ – ациклический. Самый длинный путь в сетевом графике называется *критическим путем*; его длина равна минимальному времени выполнения комплекса работ.

Для произвольного ациклического графа $\Gamma=(V,D)$ с весами дуг $w(d)>0$ максимумы $\rho(v)$ длин путей, оканчивающихся в вершине v, можно вычислить для всех $v\in V$ в процессе обхода этого графа исчерпыванием. Очевидно, что длины этих путей удовлетворяют соотношениям: $\rho(v)=0$ при $s^+(v)=0$; $\rho(v)=\max\limits_{(v',v)\in D}\big(\rho(v')+w(v',v)\big)$ при $s^+(v)>0$. Используя эти соотношения, можно вычислить все $\rho(v)$ в порядке топологической нумерации вершин графа. Если в процессе вычисления $\rho(v)$ для каждой вершины с $s^+(v)>0$ запоминать вершину v', на которой достигается максимум $\rho(v')+w(v',v)$, то по окончании вычислений можно "раскрутить" самый длинный путь, оканчивающийся в любой вершине, и в частности, критический путь, оканчивающийся в вершине v, для которой $\rho(v)$ максимально. Нетрудно убедиться, что все перечисленные задачи решаются на графе с m дугами с трудоемкостью $O(m)$.

5.3. Алгоритмы анализа метрических свойств графа

В ориентированном графе $\Gamma=(V,D)$ (быть может, с дугами, взвешенными функцией $w:D \to R^+$) *расстоянием* $\rho(x,y)$ между вершинами x и y называется минимальная длина пути, начинающегося в x и заканчивающегося в y (длина *кратчайшего* пути между x и y). Напомним, что в невзвешенном графе длина пути равна количеству дуг в этом пути, а во взвешенном графе - сумме весов этих дуг. Если вершина y не достижима из вершины x, положим $\rho(x,y)=\infty$. Легко проверить, что определенное таким образом расстояние удовлетворяет аксиомам метрики (вообще говоря, несимметрической):

1- $\forall x,y\in V$ $\rho(x,y)\geq 0$ (аксиома неотрицательности);

2- $\rho(x,y)=0 \iff x=y$ (аксиома рефлексивности);

3- $\forall x,y,z\in V$ $\rho(x,y)\leq\rho(x,z)+\rho(z,y)$ (аксиома треугольника).

Пусть в графе $\Gamma=(V,D)$, $|V|=n$, вершины пронумерованы произвольным образом: $V=\{v_1,v_2,\ldots,v_n\}$. Квадратная (размером $n\times n$) матрица $R(\Gamma)=\|\rho(v_i,v_j)\|$ называется *матрицей расстояний* графа Γ, а ее максимальный элемент - *диаметром* $d(\Gamma)$ этого графа.

Вычисление любой строки матрицы расстояний, то есть расстояний от произвольной вершины x графа $\Gamma=(V,D)$ до всех остальных его вершин, можно выполнить с помощью фронтального обхода с корнем x.

Пусть S - некоторая последовательность дуг, удовлетворяющая требованиям фронтального обхода с корнем x. Обозначим через $V^+(S)$ множество вершин графа, состоящее из корня обхода и концов дуг, принадлежащих S: $V^+(S)=(x)\cup\{v\in V:\ \exists(v',v)\in S\}$ (*достиг-*

нутые обходом S вершины); а через $V^-(S)$ – множество вершин, из которых исходят дуги обхода S: $V^-(S)=\{v\in V: \exists(v,v')\in S\}$ (вершины, *пройденные* обходом S). Множество вершин $F(S)=V^+(S)\setminus V^-(S)$ называется *фронтом* обхода S. Любой путь L из вершины x в некоторую вершину $v\notin V^+(S)$ обязательно проходит через какую-либо вершину из F(S). Действительно, не все дуги пути L принадлежат S, ведь в противном случае вершина v принадлежала бы $V^+(S)$. Пусть (a,b) – первая из дуг пути L, не принадлежащая S. Но тогда $a\in F(S)$. Нетрудно убедиться, что дуги, исходящие из какой-либо вершины фронта обхода, и только они, могут быть добавлены к обходу, так что получившаяся последовательность снова будет фронтальным обходом.

Любой фронтальный обход с корнем x для графа Г можно построить, исходя из пустого обхода $S=\emptyset$ с фронтом $F(\emptyset)=(x)$, многократно повторяя процедуру выбора из фронта произвольной вершины и добавления к обходу всех исходящих из нее дуг. Естественно, эту процедуру можно продолжать до тех пор, пока фронт обхода не станет пустым. Такой обход будем называть *тупиковым*. Очевидно, что если S – тупиковый фронтальный обход с корнем x для графа Г, то $V^+(S)=V^-(S)$ и эти множества состоят из тех и только тех вершин $v\in V$, для которых $\rho(x,v)<\infty$. Устанавливая различные правила выбора вершины из фронта обхода, получим разные модификации описанного алгоритма построения фронтального обхода.

Для нахождения расстояний от некоторой вершины до всех остальных вершин невзвешенного графа Г можно использовать модификацию фронтального обхода, известную под названием *алгоритма расширяющихся окрестностей*. В этом алгоритме вершины фронта упорядочиваются в порядке их появления в фронте и на каждом шаге алгоритма выбирается первая из вершин фронта в этом упорядочении. Определим на вершинах из $V^+(S)$ функцию r, положив $r(x)=0$ и $r(v)=r(v')+1$ для $v \in V^+(S) \setminus \{x\}$, где (v',v) - первая дуга в S, заходящая в v.

Лемма. Пусть S - фронтальный обход, построенный алгоритмом расширяющихся окрестностей, и пусть $F(S)=\{v_1, v_2, \dots, v_s\}$ - фронт этого обхода, вершины которого упорядочены в порядке их появления в фронте. Тогда $r(v_1) \le r(v_2) \le \dots \le r(v_s) \le r(v_1)+1$.

Доказательство проведем по индукции. Для $S=\emptyset$ утверждение леммы выполнено. Пусть оно выполнено для некоторого фронтального обхода S. Произведем шаг алгоритма расширяющихся окрестностей, добавив к обходу S все дуги, исходящие из вершины v_1. Фронт получившегося обхода S' будет при этом содержать вновь достигнутые вершины $v_{s+1}, \dots, v_t$ с $r(v_i)=r(v_1)+1$ для $s+1 \le i \le t$. Первой вершиной фронта $F(S')$ будет вершина v_2. Но из предположений индукции $r(v_s) \le r(v_1)+1$ и $r(v_1) \le r(v_2)$ следует $r(v_s) \le r(v_i) \le r(v_2)+1$ для $s+1 \le i \le t$, что и завершает доказательство.

Утверждение. Если S - фронтальный обход с корнем x для графа Γ, построенный алгоритмом расширяющихся окрестностей, то $r(v)=\rho(x,v)$ для всех $v \in V^+(S)$.

Доказательство также проведем по индукции. Для $S=\emptyset$ утверждение справедливо. Пусть оно справедливо для обхода S, построенного алгоритмом расширяющихся окрестностей, и пусть $F(S)=\{v_1, \dots, v_s\}$ - упорядоченная последовательность вершин фронта. Выполним шаг алгоритма расширяющихся окрестностей, построив обход S'. Для любой вновь достигнутой вершины $v \in V^+(S') \setminus V^+(S)$ получим $r(v)=r(v_1)+1$. Очевидно, что $\rho(x,v) \le r(v)$ по аксиоме треугольника. С другой стороны, так как $v \notin V^+(S)$, то любой путь из x в v проходит через некоторую вершину $v_i \in F(S)$.

Но в силу леммы и предположения индукции $\rho(x,v_1)\geq\rho(x,v_1)$, из чего легко следует $\rho(x,v)\geq r(v)$. Таким образом, $\rho(x,v)=r(v)$ и утверждение доказано.

Выполняя алгоритм расширяющихся окрестностей с корнем x вплоть до получения тупикового обхода, найдем расстояния от x до всех достижимых из x вершин. Использование подходящих структур данных (в частности, для хранения фронта обхода удобно использовать *очередь*) позволяет реализовать алгоритм расширяющихся окрестностей для графа с m дугами с трудоемкостью $O(m)$. Тем самым для невзвешенного графа с n вершинами можно вычислить матрицу расстояний и диаметр путем n-кратного применения этого алгоритма с корнем в каждой из вершин графа с трудоемкостью $O(nm)$.

Для взвешенных графов лемма не выполняется и алгоритм расширяющихся окрестностей не позволяет определить расстояния от корня обхода до всех остальных вершин графа. Решить эту задачу позволяет другая модификация фронтального обхода, носящая название *алгоритма Дейкстры.* Для фронтального обхода S функцию r на достигнутых вершинах определим следующим образом: $r(x)=0$, $r(v)=\min\limits_{(v',v)\in S}\left(r(v')+w(v',v)\right)$ для $v\in V^+(S)\setminus(x)$. Легко видеть, что введенная таким образом функция r определяет расстояния от вершины x до всех вершин графа $\Gamma_S=\left(V^+(S),S\right)$. Шаг алгоритма Дейкстры заключается в выборе вершины $v\in F(S)$, для которой $r(v)\leq r(v')$ при всех $v'\in F(S)$ (*ведущая* вершина фронта), и добавлении к обходу S всех исходящих из нее дуг.

Утверждение. Пусть фронтальный обход S с корнем x построен алгоритмом Дейкстры на графе $\Gamma=(V,D)$. Тогда $r(v)=\rho(x,v)$ для всех $v\in V^-(S)$.

Доказательство проведем по индукции. Утверждение верно для пустого обхода. Пусть оно верно для обхода S', полученного на предыдущем шаге алгоритма Дейкстры. Для доказательства справедливости утверждения для обхода S достаточно установить, что $r(v)=\rho(x,v)$ для ведущей вершины v фронта $F(S)$. Предположим, что это не так, то есть кратчайший в графе Г путь L из x в v имеет длину $r(L)<r(v)$. Тогда не все дуги из L принадлежат обходу S. Пусть (a,b) - первая из таких дуг. Очевидно, что $a \in F(S)$ и $r(a)=\rho(x,a)$. В силу положительности веса каждой дуги получаем $r(L)>r(a)$, откуда следует $r(a)<r(v)$, что противоречит выбору ведущей вершины фронта по минимальному значению r.

Таким образом, построив алгоритмом Дейкстры тупиковый фронтальный обход с корнем x для взвешенного графа Г, мы определим расстояния от вершины x до всех достижимых из нее вершин графа. Оценим трудоемкость алгоритма Дейкстры для взвешенного графа с n вершинами и m дугами. Если для хранения фронта использовать структуру очереди, на выбор ведущей вершины фронта необходимо $O(n)$ действий. Учитывая, что число шагов алгоритма не превышает $O(n)$, и на остальные операции тратится $O(m)$ действий, получим реализацию алгоритма Дейкстры с трудоемкостью $O(n^2)$. Использование для хранения фронта структуры данных типа сбалансированного двоичного дерева позволяет снизить трудоемкость алгоритма до $O(m+n\log n)$. Тем самым построение матрицы расстояний и вычисление диаметра графа можно произвести с трудоемкостью $O(n^3)$ или $O(nm+n^2\log n)$.

Интересно отметить, что до сих пор неизвестен алгоритм нахождения в графе одного расстояния между заданной парой вершин, более быстрый, чем описанные алгоритмы нахождения целой строки матрицы расстояний.

Приведем примеры еще нескольких задач для ориентированных графов (в частности, взвешенных), которые можно решить приведенными модификациями фронтального обхода.

Если в процессе выполнения фронтального обхода с корнем x для каждой пройденной вершины $v \neq x$ запоминать вершину v', такую что $r(v)=r(v')+1$ (или $r(v)=r(v')+w(v',v)$ в случае взвешенного графа), то по окончании вычислений можно "раскрутить" кратчайшие пути из вершины x во все достижимые из нее вершины. Совокупность этих кратчайших путей образует в графе *кратчайшее связывающее дерево* с корнем x.

В графе $\Gamma=(V,D)$ кратчайших путей между вершинами x и y может быть несколько. Совокупность всех вершин и дуг этих кратчайших путей образует *подграф кратчайших путей* из x в y – $\Gamma_{xy}=(V_{xy},D_{xy})$, где $V_{xy}\subseteq V$, $D_{xy}\subseteq D$. С помощью фронтального обхода с корнем x для графа Γ и фронтального обхода с корнем y для графа $\Gamma^{-1}=(V,D^{-1})$, где $D^{-1}=\{(b,a):(a,b)\in D\}$, вычислим $\rho(x,v)$ и $\rho(v,y)$ для всех $v\in V$. Теперь Γ_{xy} можно построить, используя легко проверяемые соотношения:

$v\in V_{xy} \Leftrightarrow \rho(x,v)+\rho(v,y)=\rho(x,y)$;

$(a,b)\in D_{xy} \Leftrightarrow \rho(x,y)-\rho(x,a)-\rho(b,y)=1$ (или $=w(a,b)$ в случае взвешенного графа).

5.4. Алгоритмы анализа циклической структуры графа

Вершины x и y ориентированного графа $\Gamma=(V,D)$ называются *взаимно достижимыми*, если существует и путь $L_{xy}\subset D$ из x в y, и путь $L_{yx}\subset D$ из y в x. Другими словами, вершины x и y взаимно достижимы, если x=y или в графе Γ существует цикл, проходящий как через x, так и через y. Рефлексивность и симметричность отношения взаимной достижимости очевидна. Покажем его транзитивность. Пусть (x,y) и (y,z) – пары взаимно достижимых вершин, то есть в графе Г существуют пути L_{xy}, L_{yx}, L_{yz} и L_{zy}. Но тогда $L_{xy}L_{yz}$ – путь из x в z, а $L_{zy}L_{yx}$ – путь из z в x. Тем самым вершины x и z взаимно достижимы. Таким образом, отношение взаимной достижимости на вершинах графа является отношением эквивалентности. Но отношение эквивалентности порождает разбиение своего носителя на классы попарно эквивалентных элементов. Классы разбиения вершин графа, порожденного отношением взаимной достижимости, называются *компонентами сильной связности* (КСС) графа. КСС называется *тривиальной*, если она состоит из единственной вершины x и $(x,x)\notin D$.

Задача нахождения компонент сильной связности ориентированного графа является одной из важнейших задач анализа графов и имеет многочисленные приложения. Одна группа таких приложений основана на следующей модели.

Пусть $X=\{x_1, x_2, \dots, x_n\}$ – множество переменных и пусть задана система уравнений вида $x_i=f_i(X_i)$, где $X_i\subseteq X$, $1<i<n$. Из i-го уравнения можно найти значение x_i, если известны значения всех $x_j\in X_i$. Построим ориентированный граф $\Gamma=(X,D)$, вершинами которого являются переменные x_i, и $(x_j,x_i)\in D \Leftrightarrow x_j\in X_i$. Наличие в графе Г дуги (x_j,x_i) интерпретируется как невозможность вычислить переменную x_i раньше переменной x_j. Ясно, что если x_i и x_j взаимно достижимы в графе Г, то их значения нельзя вычислить независимо, а только путем решения некоторой системы уравнений, содержащей и уравнение $x_i=f_i(X_i)$, и уравнение $x_j=f_j(X_j)$. КСС построенного графа как раз и определяют те минимальные по включению подсистемы исходной системы уравнений, которые необходимо и достаточно решить для вычисления значений всех переменных.

Прежде чем перейти к решению задачи нахождения КСС графа с использованием радиального обхода, изучим некоторые свойства такого обхода и опишем алгоритм его построения.

Пусть S – радиальный обход с корнем x для графа $\Gamma=(V,D)$. Обозначим через $V^+(S)$ подмножество вершин графа, содержащее корень обхода и все вершины, в которые заходят дуги обхода S: $V^+(S)=\{x\}\cup\left\{v\in V:\ \exists(v',v)\in S\right\}$ (вершины, *достигнутые* обходом S). Последовательность $R(S)=\left\{v_1,v_2,\dots,v_s\right\}$ вершин из $V^+(S)$ назовем *радиусом* обхода S, если она максимальна по включению среди последовательностей, удовлетворяющих условиям:

1- $\exists(v_s,v)\in D\setminus S$, то есть не все дуги, исходящие из последней вершины последовательности, содержатся в обходе;

2- $(v_i,v_{i+1})\in S$ при $1\leq i\leq s-1$, причем (v_i,v_{i+1}) – первая из содержащихся в обходе S дуг, заходящая в v_{i+1}.

Положим $V^-(S)=V^+(S)\setminus R(S)$ (вершины, *пройденные* обходом S). Из максимальности $R(S)$ по включению следует, что $V^-(S)$ содержит только те вершины, для которых все исходящие дуги содержатся в обходе S (но не обязательно все такие вершины). Если $R(S)=\emptyset$ (то есть для всех вершин, в которые заходят дуги обхода S, все исходящие дуги также содержатся в обходе), то обход S называется *тупиковым*. Для нетупикового обхода первая вершина радиуса всегда совпадает с корнем обхода.

Лемма. Радиус $R(S)$ обхода S определяется однозначно.

Доказательство. Предположим существование двух различных последовательностей $R(S)=\left\{v_1,\dots,v_s\right\}$ и $R'(S)=\left\{v'_1,\dots,v'_t\right\}$, удовлетворяющих определению радиуса. Пусть $s\leq t$. Тогда найдется $1\leq i\leq s-1$, такое что $v_j=v'_j$ при $1\leq j\leq i$, но $v_{i+1}\neq v'_{i+1}$. Рассмотрим дуги (v_i,v_{i+1}) и (v_i,v'_{i+1}). Пусть для определенности дуга (v_i,v_{i+1}) расположена в обходе раньше дуги (v_i,v'_{i+1}). Но тогда, в соответствие с правилом 3 построения радиального обхода (п. 5.1), между этими дугами в S должны быть расположены все дуги, исходящие из вершин, достижимых из v_{i+1}, в том числе и все дуги, исходящие из вершины v_u. Но это противоречит условию 1 в определении радиуса, что и доказывает лемму.

Из определений радиального обхода и его радиуса следует, что добавление к обходу S любой дуги (v,v'), исходящей из последней вершины v радиуса R(S) (и только такой дуги), приводит к радиальному обходу S'. Радиус R(S') обхода S' получается при этом из R(S) одним из следующих способов:

1- если $v'\notin V^+(S)$ и $s^-(v')>0$ (то есть в графе есть дуги, исходящие из v'), то R(S') получается из R(S) добавлением вершины v';

2- если $v'\in V^+(S)$ и (v,v') - не единственная исходящая из вершины v дуга, не содержащаяся в обходе S, то R(S')=R(S);

3- в остальных случаях, то есть если $v'\notin V^+(S)$ и $s^-(v')=0$ или $v'\in V^+(S)$ и все дуги, исходящие из вершины v, содержатся в обходе S , R(S') получается из R(S) удалением последних вершин, для которых все исходящие дуги содержатся в S'.

Будем действовать таким образом, начиная с пустого обхода $S=\emptyset$ с радиусом $R(S)=(x)$, последовательно добавляя к обходу дугу, исходящую из последней вершины радиуса, вплоть до получения тупикового обхода S с радиусом $R(S)=\emptyset$. Поскольку в описанном процессе на обработку каждой дуги обхода S и каждой вершины из $V^+(S)$ при подходящем выборе структур данных (в частности, для хранения радиуса обхода удобно использовать стек) тратится константа действий, то рассмотренный алгоритм построения тупикового радиального обхода для графа с m дугами имеет трудоемкость O(m).

Использование радиального обхода для выделения КСС графа основано на том обстоятельстве, что при добавлении к обходу S дуги (v,v') с $v'\in R(S)$ обнаруживается цикл в графе $\Gamma_{S'}=(V,S')$, проходящий через все вершины $v',\dots,v$, расположенные в $R(S)$ подряд от v' до v; следовательно, все эти вершины принадлежат одной и той же КСС графов $\Gamma_{S'}$ и Γ. Если обозначить через $K(S)$ разбиение множества вершин V на КСС в графе $\Gamma_S=(V,S)$ (для пустого обхода S $K(S)$ - разбиение на одноэлементные подмножества), то $K(S')$ получается из $K(S)$ объединением всех компонент, к которым принадлежат вершины $v',\dots,v$. Построив тупиковый радиальный обход S с корнем x для графа $\Gamma=(V,D)$, получим разбиение на КСС всех достижимых из x вершин графа. Если не все вершины графа Γ достижимы из x, повторим процесс для подграфа графа Γ, порожденного не пройденными обходом S вершинами, выбрав в качестве корня обхода любую из них. В процессе применения описанного алгоритма к графу с n вершинами объединение классов разбиений производится не более n-1 раз (при каждом объединении количество классов уменьшается минимум на единицу), а одно объединение классов можно выполнить за $O(n)$ действий, что дает трудоемкость алгоритма $O(n^2)$.

Можно построить более экономный алгоритм разбиения вершин графа на КСС, не требующий хранения разбиений $K(S)$. Определим на достигнутых радиальным обходом S вершинах графа $\Gamma=(V,D)$ две функции: $\nu(v)$, равную порядковому номеру появления вершины v в $V^+(S)$, и $\mu(v)$, равную минимальному из номеров $\nu(v')$ вершин, принадлежащих в графе $\Gamma_S=(V,S)$ той же КСС, что и вершина v (вершина v' при этом называется *ведущей* вершиной компоненты). При первом появлении вершины v в $V^+(S)$ положим $\mu(v)=\nu(v)$. Добавляя к обходу S дугу (v,v') с $v'\in R(S)$, пересчитаем $\mu(v)$:

$\mu(v)=\min\{\mu(v),\mu(v')\}$. Если при удалении вершины v из $R(S)$ окажется, что $\mu(v)=\nu(v)$, то v – ведущая вершина КСС и все вершины этой компоненты пройдены обходом S. Если же окажется $\mu(v)<\nu(v)$, то предшествующая вершине v в $R(S)$ вершина v' принадлежит той же КСС, что и вершина v, и необходимо положить $\mu(v')=\min\{\mu(v'),\mu(v)\}$. Выделение КСС таким алгоритмом в графе с m дугами требует всего $O(m)$ действий.

Приведем еще пример задачи анализа циклической структуры графа, которая может быть решена радиальным обходом. Назовем вершину графа *циклической вершиной*, если в графе существует проходящий через нее цикл. Аналогично, дуга, содержащаяся в каком-либо цикле графа, называется *циклической дугой*. Возможность выделения циклических вершин и/или циклических дуг графа в процессе построения его радиального обхода обосновывает следующее очевидное утверждение.

Утверждение. Вершина графа – циклическая тогда и только тогда, когда она принадлежит нетривиальной КСС. Дуга графа является циклической дугой тогда и только тогда, когда ее концы принадлежат одной и той же КСС.

Вопросы для самостоятельной работы

1. Доказать, что в ациклическом графе существует вершина v с $s^-(v)=0$.

*2. Доказать, что если S и S' – тупиковые фронтальные обходы с одинаковым корнем для одного и того же графа, то множества входящих в них дуг совпадают.

3. Разработать алгоритм построения транзитивного замыкания графа $\Gamma=(V,D)$, $|V|=n$, $|D|=m$, с использованием фронтального

обхода с трудоемкостью $O(nm)$.

4. Построить пример, показывающий невозможность использования алгоритма расширяющихся окрестностей для вычисления расстояний во взвешенном графе.

5. Доказать, что поддерево $T=(V,D')$ графа $\Gamma=(V,D)$, $D'\subseteq D$, все расстояния $\rho(x,v)$ в котором (при фиксированном x) совпадают с соответствующими расстояниями в графе Γ, является его кратчайшим связывающим деревом с корнем x.

*6. Предложить модификацию обхода исчерпыванием, позволяющего вычислить расстояния от вершины x до всех остальных вершин взвешенного ациклического графа Γ с m дугами за $O(m)$ действий.

7. Разработать алгоритм нахождения кратчайшего пути из вершины x в вершину y в подграфе кратчайших путей Γ_{xy} с трудоемкостью $O(r)$, где $r=\rho(x,y)$.

8. Можно ли использовать радиальный обход для построения транзитивного замыкания графа? Какова трудоемкость такого алгоритма?

*9. *Фактор-графом* по сильной связности графа $\Gamma=(V,D)$ называется граф $\Gamma_\Phi=(K,D')$, вершины которого - компоненты сильной связности графа Γ, и $D'=\left\{(k,k'): \left(\exists v,v'\in V: v\in k, v'\in k', (v,v')\in D\right)\right\}$. Доказать, что фактор-граф любого графа Γ - ациклический.

10. Пусть Γ и Γ_Φ - граф и его фактор-граф по сильной связности, и пусть $\overline{\Gamma}=(V,\overline{D})$ и $\overline{\Gamma}_\Phi=(K,\overline{D}')$ - транзитивные замыкания графов Γ и Γ_Φ. Доказать, что $(v,v')\in\overline{D} \leftrightarrow k(v)=k(v') \vee \vee \left(k(v),k(v')\right)\in\overline{D}'$, где $k(v)$ - компонента сильной связности графа Γ, содержащая вершину v.

11. Доказать, что полный (фронтальный или радиальный) обход с корнем x для графа Γ существует тогда и только тогда, когда из x достижима любая его вершина.

6. ПОТОКОВЫЕ АЛГОРИТМЫ

6.1. Потоки в сети

Сетью $G=(V,D,s,t)$ называется ориентированный граф с множеством вершин V, $|V|=n$, множеством дуг D, $|D|=m$ и двумя выделенными вершинами: *источником* s *и стоком* t, называемыми *полюсами* сети. Остальные вершины $V\setminus\{s,t\}$ сети называются *внутренними*.

Для произвольной функции $f:D\to\mathbb{R}$ определим *дивергенцию* $\mathrm{div}_f(v)$ этой функции в вершине $v\in V$:

$$\mathrm{div}_f(v)=\sum_{(v,v')\in D} f(v,v')-\sum_{(v',v)\in D} f(v',v).$$

Функция f, определенная на дугах сети G, такая что $\mathrm{div}_f(v)=0$ для любой внутренней вершины $v\in V\setminus\{s,t\}$, $\mathrm{div}_f(s)>0$, $\mathrm{div}_f(t)<0$, называется *потоком* в этой сети.

Просуммируем дивергенции произвольной функции f по всем вершинам сети: $\sum\limits_{v\in V}\mathrm{div}_f(v)=\sum\limits_{v\in V}\Big(\sum\limits_{(v,v')\in D} f(v,v')-\sum\limits_{(v',v)\in D} f(v',v)\Big)$.

Каждая дуга $d\in D$ появляется в правой части этого выражения дважды, причем с противоположными знаками. Следовательно, $\sum\limits_{v\in V}\mathrm{div}_f(v)=0$. Таким образом, если f - поток, то дивергенции f в полюсах сети равны по величине и противоположны по знаку. Величина $M(f)=\mathrm{div}_f(s)=-\mathrm{div}_f(t)$ называется *мощностью* потока f в сети G.

Легко видеть, что если f и g - потоки в сети G, то для любых $\lambda,\mu\in\mathbb{R}$ $\lambda f+\mu g$ - также поток в G, имеющий мощность $M(\lambda f+\mu g)=\lambda M(f)+\mu M(g)$, то есть потоки в сети G образуют *линейное пространство*, причем мощность потока - *линейный функционал* на этом пространстве.

Функция f, равная 0 на всех дугах сети, называется *нулевым* потоком 0.

Введем на множестве функций, определенных на дугах сети G, частичный порядок: $f>g \leftrightarrow \left(\forall d\in D\ f(d)\geq g(d)\right) \& \left(\exists d\in D\ f(d)>g(d)\right)$.

Поток $f>0$ называется *положительным.* Приведем примеры положительных потоков.

Пусть L - простой путь из источника s в сток t. Задавшись числом $\rho>0$, определим функцию φ_L на дугах сети: $\varphi_L(d)=\rho$, если $d\in L$; $\varphi_L(d)=0$, если $d\notin L$. Функция φ_L - положительный поток, называемый *потоком вдоль простого пути* L. Его мощность $M(\varphi_L)=\rho$.

Пусть C - простой цикл в G. Функция φ_C, определенн' ' формулой $\varphi_C(d)=\rho$, если $d\in C$; $\varphi_C(d)=0$, если $d\notin C$, при $\rho>0$ является положительным потоком в сети G и называется *циркуляцией вдоль простого цикла* C. Мощность циркуляции $M(\varphi_C)=0$.

Потоки φ_L и φ_C называются *элементарными* потоками в сети G.

Теорема. Пусть f - положительный поток в сети G. Тогда f может быть представлен в виде суммы не более $m=|E|$ элементарных потоков.

Доказательство. Из положительности потока f следует существование дуги (a_0, a_1), такой что $f(a_0, a_1)>0$. Если $a_1\neq t$, то из $\operatorname{div}_f(a_1)\geq 0$ следует существование дуги (a_1, a_2) с $f(a_1, a_2)>0$. Аналогично, если $a_0\neq s$, то существует дуга (a_{-1}, a_0), для которой $f(a_{-1}, a_0)>0$. Повторяя эти рассуждения для вершин a_2 и a_{-1}, в конце концов получим либо простой путь L из s в t, либо простой цикл C. В обоих случаях для любой дуги d, входящей в этот путь S (S=L или S=C), $f(d)>0$. Положим $\rho=\min\limits_{d\in S} f(d)$ и определим элементарный поток φ_S: $\varphi_S(d)=\rho$, если $d\in S$; $\varphi_S(d)=0$, если $d\notin S$. Рассмотрим поток $f'=f-\varphi_S$. Очевидно, что $f'\geq0$. Если $f'=0$, разложение получено. В противном случае повторим описанную процедуру для положительного потока f'. Так как существует по крайней мере одна дуга d, такая что $f(d)>0$ и $f'(d)=0$ (это та дуга из S, для которой $f(d)=\rho$), то до получения нулевого потока эту процедуру придется повторить не более m раз.

6.2. Допустимые потоки в сети с ограничениями

Пусть теперь на дугах сети G задана функция $c : D \to \mathbb{R}^+$, ставящая в соответствие дуге d ее *пропускную способность* $c(d)>0$.

Поток f, удовлетворяющий условию $0\leq f(d)\leq c(d)$ для любой дуги $d\in D$, называется *допустимым* потоком в сети с ограничениями. Допустимый поток максимальной мощности называется *максимальным* потоком. Задача нахождения максимального потока в сети с ограничениями в описанной постановке не является задачей дискретной оптимизации - ведь множество потоков (и даже допустимых потоков) бесконечно и непрерывно. В такой ситуации само существование лтимального решения не очевидно и требует изучения.

Теорема. В сети с ограничениями существует максимальный поток.

Доказательство. Произвольную функцию f на дугах сети можно рассматривать как элемент пространства $\mathbb{R}^m$. Потоки в сети, удовлетворяющие $n-2$ линейным условиям $\mathrm{div}_f(d)=0$ для $v\notin(s,t)$, образуют в $\mathbb{R}^m$ $(m-n+2)$-мерное линейное подпространство, являющееся замкнутым множеством. Неравенства $\mathrm{div}_f(s)\geq0$ и $\mathrm{div}_f(t)\leq0$ выделяют в этом подпространстве замкнуты.. многогранник. Условия допустимости $0\leq f(d)\leq c(d)$ для $d\in D$ выделяют в $\mathbb{R}^m$ прямоугольный параллелепипед - замкнутое и ограниченное множество. Таким образом, множество допустимых потоков также является замкнутым и ограниченным. Теперь достаточно вспомнить, что мощность потока - линейный функционал, и воспользоваться теоремой из функционального анализа о том, что линейный функционал на замкнутом ограниченном множестве достигает своих экстремальных значений.

6.3. Остаточная сеть и потоки в ней

Пусть f - поток в сети G. Обозначим через $\bar{d}$ дугу, обратную к дуге d, и определим *симметризованный поток* $\tilde{f}$: $\tilde{f}(d)=f(d)-f(\bar{d})$. При этом мы полагаем $f(\bar{d})=0$, если $\bar{d}\notin D$. В терминах симметризованного потока выражение для дивергенции имеет простой вид: $\operatorname{div}_f(v)=\sum\limits_{(v,v')\in D}\tilde{f}(v,v')$.

Для сети G и допустимого потока f в ней построим *остаточную сеть* G_f с пропускными способностями c_f с помощью следующей процедуры:

1- добавим в сеть G все дуги $\bar{d}$, такие что $d\in D$, $\bar{d}\notin D$, и положим для них $c(\bar{d})=f(\bar{d})=0$;

2- для всех дуг (в том числе и для добавленных) вычислим новые пропускные способности $c_f(d)=c(d)-\tilde{f}(d)$;

3- удалим все дуги, для которых $c_f(d)=0$.

Из допустимости потока f в сети G следует, что $c_f(d)>0$ для всех дуг сети G_f.

Займемся изучением связей между допустимыми потоками в сети G и в остаточной сети G_f с пропускными способностями c_f.

Пусть f -допустимый поток в сети G, а g - допустимый поток в сети G_f. Определим операцию $\oplus$ сложения потоков:

$(f\oplus g)(d)=\max\{0,\tilde{f}(d)+\tilde{g}(d)\}$.

Утверждение. $h=f\oplus g$ – допустимый поток в сети G и его мощность $M(h)=M(f)+M(g)$.

Доказательство. Из определения операции $\oplus$ следует $\tilde{h}=\tilde{f}+\tilde{g}$. Тогда $\mathrm{div}_h(v)=\mathrm{div}_f(v)+\mathrm{div}_g(v)$, следовательно, $\mathrm{div}_h(v)=0$ для $v\notin\{s,t\}$, $\mathrm{div}_h(s)\geq 0$, $\mathrm{div}_h(t)\leq 0$, то есть h – поток в сети G. Выполнение ограничения $h(d)\geq 0$ очевидно. Покажем, что $h(d)\leq c(d)$: $h(d)=\tilde{f}(d)+\tilde{g}(d)=\left(c(d)-c_f(d)\right)+\left(g(d)-g(\bar{d})\right)=c(d)-\left(c_f(d)-g(d)\right)-g(\bar{d})\leq$ $\leq c(d)$. Таким образом, h – допустимый поток в сети G. Его мощность $M(h)=\mathrm{div}_h(s)=\mathrm{div}_f(s)+\mathrm{div}_g(s)=M(f)+M(g)$.

Для допустимых в сети G потоков f и f' с $M(f)\leq M(f')$ определим операцию $\ominus$ вычитания потоков:

$(f'\ominus f)(d)=\max\{0,\tilde{f}'(d)-\tilde{f}(d)\}$.

Утверждение. $h=f'\ominus f$ – допустимый поток в сети G_f и его мощность $M(h)=M(f')-M(f)$.

Доказательство аналогично доказательству предыдущего утверждения.

Теорема. Если f – допустимый поток в сети G, а g – допустимый поток в сети G_f, то $f\oplus g$ – максимальный поток в сети G тогда и только тогда, когда g – максимальный поток в сети G_f.

Доказательство. Пусть поток $f\oplus g$ максимален в сети G, но поток g не максимален в сети G_f, то есть в этой сети есть допустимый поток g', для которого $M(g')>M(g)$. Рассмотрим тогда допустимый поток $f\oplus g'$: $M(f\oplus g')=M(f)+M(g')>M(f)+M(g)=M(f\oplus g)$, что противоречит предположению о максимальности потока $f\oplus g$.

Обратно, пусть g – максимальный поток в сети G_f, но поток $f\oplus g$ не максимален в сети G, то есть в сети G существует поток f' с $M(f')>M(f\oplus g)$. Рассмотрим допустимый в сети G_f поток $f'\ominus f$: $M(f'\ominus f)=M(f')-M(f)>M(f\oplus g)-M(f)=M(g)$, что противоречит предположению о максимальности потока g.

6.4. Потоки и разрезы

Разрезом $R=(X,Y)$ в сети $G=(V,D,s,t)$ называется разбиение множества вершин V на два класса X,Y: $X\cup Y=V$, $X\cap Y=\emptyset$; так что $s\in X$, $t\in Y$.

По отношению к разрезу $R=(X,Y)$ все дуги сети G распадаются на три группы:

1- *прямые дуги* разреза $D_R^+=\{(a,b)\in D: a\in X \& b\in Y\}$;

2- *обратные дуги* разреза $D_R^-=\{(a,b)\in D: a\in Y \& b\in X\}$;

3- *внутренние дуги* разреза $D_R^0=\{(a,b)\in D: a,b\in X \vee a,b\in Y\}$.

Пропускной способностью разреза R в сети G называется сумма пропускных способностей прямых дуг разреза: $c(R)=\sum\limits_{d\in D_R^+} c(d)$.

Для произвольной функции f на дугах сети G ее *дивергенцией на разрезе* $R=(X,Y)$ называется сумма ее дивергенций в вершинах из класса X: $\mathrm{div}_f(R)=\sum\limits_{v\in X}\mathrm{div}_f(v)$.

Если f - допустимый поток в сети G, то из $\mathrm{div}_f(v)=0$ для $v\notin(s,t)$ следует $\mathrm{div}_f(R)=\mathrm{div}_f(s)=M(f)$. С другой стороны $\mathrm{div}_f(R)=$ $=\sum\limits_{v\in X}\mathrm{div}_f(v)=\sum\limits_{v\in X}\Big(\sum\limits_{(v,v')\in D} f(v,v')-\sum\limits_{(v',v)\in D} f(v',v)\Big)$. Поток в каждой дуге, оба конца которой принадлежат X (внутренней дуге разреза R), входит в правую часть этого выражения дважды, причем с разными знаками. Таким образом, $\mathrm{div}_f(R)=\sum\limits_{d\in D_R^+} f(d)-\sum\limits_{d\in D_R^-} f(d)\leq c(R)$. Тем самым нами доказана следующая лемма.

Лемма. Мощность любого допустимого потока в сети с ограничениями не превосходит пропускной способности любого разреза в этой сети.

На самом деле связь между мощностями допустимых потоков в сети с ограничениями и пропускными способностями разрезов в этой сети более сильная, что устанавливает следующая теорема.

Теорема (о максимальном потоке и минимальном разрезе). Мощность максимального потока в сети с ограничениями равна минимальной пропускной способности разрезов этой сети.

Доказательство. Пусть f - максимальный поток в сети G. В силу предыдущей леммы для доказательства теоремы нам достаточно построить разрез R, такой что $c(R)=M(f)$.

Рассмотрим остаточную сеть G_f и пусть X - множество вершин, достижимых в G_f из источника s. Предположим, что сток $t\in X$. Это означает, что в G_f существует простой путь L из s в t. Построим вдоль этого пути элементарный поток φ_L мощностью $M(\varphi_L)=\min\limits_{d\in L} c_f(d)$. Очевидно, что это допустимый поток в сети G_f, и в силу $c_f(d)>0$, его мощность $M(\varphi_L)>0$. Но тогда $M(f\oplus\varphi_L)>M(f)$, что противоречит предположению о максимальности потока f. Таким образом, $t\notin X$, то есть $R=(X,V\setminus X)$ - разрез. Каждая прямая дуга этого разреза отсутвует в сети G_f (по построению множества X), а это может быть только в случае $c_f(d)=c(d)-f(d)+f(\bar{d})=0$. Так как для допустимого в сети G потока f имеет место $f(d)\leq c(d)$ и $f(\bar{d})\geq 0$, предндущее равенство может выполняться только при $f(d)=c(d)$ и $f(\bar{d})=0$. Таким образом $f(d)=c(d)$ для любой дуги $d\in D_R^+$ и $f(d)=0$ для любой дуги $d\in D_R^-$. Отсюда получаем, что $M(f)=\mathrm{div}_f(R)=\sum\limits_{d\in D_R^+} f(d)-\sum\limits_{d\in D_R^-} f(d)=c(R)$, и теорема доказана.

Из теоремы о максимальном потоке и минимальном разрезе немедленно следует необходимое и достаточное условие максимальности потока.

Критерий максимальности потока. Поток f в сети G максимален тогда и только тогда, когда в сети G_f сток t не достижим из источника s.

Задачу о максимальном потоке можно сформулировать как частный случай задачи линейного программирования: максимизировать линейный функционал $M(f) = \sum_{(s,v)\in D} f(s,v)$ при выполнении линейных равенств $\mathrm{div}_f(v)=0$ для $v\notin\{s,t\}$, и линейных неравенств $0\leq f(d)\leq c(d)$ для всех $d\in D$. Теорема о максимальном потоке и минимальном разрезе устанавливает связь решений этой задачи с решением *двойственной* задачи о минимальном разрезе (которая, кстати, является задачей дискретной оптимизации). Наличие этой связи позволяет построить эффективные алгоритмы решения задачи о максимальном потоке.

6.5. *Алгоритм Форда-Фалкерсона*

Мы рассмотрим несколько итерационных алгоритмов решения задачи о максимальном потоке, в которых на каждой итерации, исходя из некоторого допустимого потока f, строится допустимый поток f' большей мощности.

Опишем итерацию алгоритма Форда-Фалкерсона (АФФ). Пусть f_i - *допустимый поток в сети* G. Построим остаточную сеть G_{f_i} и в ней найдем простой путь L_i из источника s в сток t (*увеличивающий* путь). Определим элементарный допустимый поток вдоль этого пути: $\varphi_{L_i}(d) = \min_{d\in L_i} c_{f_i}(d)$ для $d\in L_i$; $\varphi_{L_i}=0$ для $d\notin L_i$ (*увеличивающий* поток). Положим $f_{i+1}=f_i\oplus\varphi_{L_i}$. Этот допустимый в сети G поток с $M(f_{i+1})>M(f_i)$ *будет исходным для проведения следующей* итерации.

Начав с допустимого в любой сети потока $f_0=0$, последовательными итерациями АФФ получим последовательность допустимых в сети G потоков возрастающей мощности. Если для какой-либо сети G_{f_i} увеличивающего пути не существует, то в соответствии с критерием максимальности, поток f_i максимален в сети G.

Не составляет труда реализовать итерацию АФФ с полиномиальной (по числу вершин и дуг сети G) трудоемкостью. Однако, при произвольных действительных пропускных способностях АФФ может не привести к получению максимального потока за конечное число итераций. Более того, существуют примеры сетей, на которых АФФ при определенном выборе увеличивающих путей строит бесконечную последовательность допустимых потоков, сходящуюся не к максимальному потоку.

В то же время, будучи применен к так называемой *целочисленной* сети, пропускные способности всех дуг которой – целые числа, АФФ находит максимальный поток за конечное число итераций. Действительно, значения f_i, c_{f_i} и φ_{L_i} на всех итерациях АФФ будут целыми. Таким образом, АФФ для целочисленной сети строит последовательность *целочисленных* потоков возрастающей мощности, что в совокупности с ограничением максимального потока, например, величиной $\sum_{(s,v)\in D} c(s,v)$, доказывает конечность числа итераций. Это рассуждение, кроме того, показывает, что справедливо следующее свойство целочисленности.

Утверждение. В целочисленной сети существует целочисленный максимальный поток.

6.6. Алгоритмы кратчайших путей

Количество итераций АФФ даже в случае целочисленной сети ограничено только в терминах пропускных способностей, то есть для сетей с малым числом вершин и дуг количество итераций может быть сколь угодно большим.

Рассмотрим модификацию АФФ - алгоритм кратчайших путей (АКП), количество итераций которого даже без предположения целочисленности ограничено полиномом от числа вершин и дуг сети. В итерации АКП в качестве увеличивающего пути выбирается не произвольный путь из источника в сток, а путь минимальной (по числу входящих в него дуг) длины - *кратчайший* путь.

В основе доказательства конечности числа итераций АКП лежит следующая простая лемма.

Лемма. Пусть в ориентированном графе $\Gamma=(V,D)$ кратчайший путь из вершины s в вершину t проходит через дугу (a,b). Тогда любой путь из s в t, проходящий через дугу (b,a), обратную дуге (a,b) в графе $\Gamma'=(V,D\cup(b,a))$, имеет большую длину.

Доказательство. Из того, что рассматриваемый путь - кратчайший, следует $\rho(s,t)=\rho(s,a)+\rho(b,t)+1$. Длина r любого пути из s в t, проходящего через добавленную дугу (b,a), не меньше $\rho(s,b)+\rho(a,t)+1$. С учетом $\rho(s,b)\geq\rho(s,a)+1$ и $\rho(a,t)\geq\rho(b,t)+1$, получим $r\geq\rho(s,t)+2$, что и требовалось доказать.

С помощью этой леммы легко доказывается следующее утверждение.

Утверждение. Последовательность длин увеличивающих путей в итерациях АКП не убывает.

Доказательство. В сети $G_{f_{i+1}}$ по сравнению с сетью G_{f_i} добавляются только дуги, обратные к дугам из увеличивающего пути L_i в сети G_{f_i}. Так как этот путь кратчайший, в соответствии с леммой любой путь из s в t в сети $G_{f_{i+1}}$, содержащий хотя бы одну из добавившихся дуг, длиннее пути L_i. Таким образом, в сети

$G_{f_{i+1}}$ не появляется путей меньшей длины, чем длина пути L_i, и утверждение доказано.

Из утверждения следует, что итерации АКП с равными длинами кратчайших путей следуют друг за другом. Совокупность всех таких подряд идущих итераций АКП с фиксированной длиной кратчайшего пути назовем *этапом*. Так как в графе с n вершинами длины кратчайших путей могут иметь не более n-1 различных значений, количество этапов АКП не превышает *n*. В сети $G_{f_{i+1}}$ по сравнению с сетью G_{f_i} отсутствует по крайней мере одна дуга *d* из увеличивающего пути L_i, у которой $c_{f_i}(d)=M(\varphi_{L_i})$. Из леммы следует, что эта дуга не может появиться и на последующих итерациях этого этапа. Учитывая, что число дуг в каждой остаточной сети G не превышает 2m, получим, что каждый этап состоит из не более O(m) итераций, а общее число итераций АКП не превышает O(mn).

Оценим трудоемкость выполнения одной итерации АКП. Алгоритм нахождения кратчайшего пути, описанный в п. 5.3, имеет трудоемкость O(m). Так как длина увеличивающего пути L_i не превышает n-1, вычисление увеличивающего потока φ_{L_i} и потока $f_{i+1}=f_i\oplus\varphi_{L_i}$ требует не более O(*n*) действий. Для перехода от сети G_{f_i} к сети $G_{f_{i+1}}$ необходимо пересчитать только пропускные способности дуг, входящих в путь L_i, и обратных к ним. На это также достаточно O(n) действий. Таким образом, трудоемкость одной итерации АКП не превышает O(m), а трудоемкость решения задачи о максимальном потоке с помощью АКП не больше $O(nm^2)$.

Главный член трудоемкости АКП связан с нахождением на каждой итерации кратчайшего пути из источника в сток в соответствующей остаточной сети. Однако мы доказали, что кратчайшие пути для всех итераций одного этапа АКП являются кратчайшими путями в остаточной сети для первой итерации этого этапа. Вся совокупность этих кратчайших путей - подграф кратчайших путей из источника в сток (будем называть его *справочной* кратчайших путей) может быть построена алгоритмом, описанным в п. 5.3, за $O(m)$ действий. При наличии такой справочной нахождение кратчайшего пути в ней может быть выполнено радиальным обходом за $O(n)$ действий. Таким образом, трудоемкость одной итерации этой модификации АКП - алгоритма со справочной (АС) - не превышает $O(n)$. Учитывая, что справочная строится в начале этапа, и на удаление из нее уже использованных увеличивающих путей в течение всего этапа требует не более $O(m)$ действий, получим, что трудоемкость АС не превышает $O(m.n^2)$.

Отметим, что существует модификация АС - алгоритм тупиковых потоков, в котором используется увеличивающий поток вдоль более сложной структуры, чем простой путь, что позволяет снизить количество итераций в этапе до $O(n)$ и общую оценку трудоемкости до $O(n^3)$.

6.7. Комбинаторные сети

В сети G назовем *внутренними* дугами все дуги, не инцидентные полюсам: $D' = \{(a,b) \in D : v, v' \notin (s,t)\}$, и введем *дуговую характеристику* сети: $x_д(G) = \sum\limits_{d \in D'} c(d)$.

Назовем *пропускной способностью вершины* $v \in V$ величину $c(v) = \min\left\{ \sum\limits_{(v,v') \in D} c(v,v'), \sum\limits_{(v',v) \in D} c(v',v) \right\}$ и введем *вершинную характеристику* сети $x_в(G) = \sum\limits_{v \notin (s,t)} c(v)$.

Минимум из дуговой и вершинной характеристик назовем *характеристикой* сети: $x(G) = \min\{x_д(G), x_в(G)\}$.

Теорема (о постоянстве характеристики). Если f – поток в сети G, то $\chi(G_f)=\chi(G)$.

Доказательство. Из определения остаточной сети $c_f(d)+c_f(\bar{d}) =c(d)-\tilde{f}(d)+c(\bar{d})-\tilde{f}(\bar{d})=c(d)+c(\bar{d})$. Суммируя по внутренним дугам, получим $\chi_Д(G_f)=\chi_Д(G)$.

Покажем, что пропускные способности внутренних вершин в сетях G и G_f совпадают. Действительно, $\sum_{(v,v')\in D} c_f(v,v')=$
$= \sum_{(v,v')\in D} c(v,v') - \sum_{(v,v')\in D} \tilde{f}(v,v') = \sum_{(v,v')\in D} c(v,v') - \mathrm{div}_f(v) =$
$= \sum_{(v,v')\in D} c(v,v')$. Аналогично $\sum_{(v',v)\in D} c_f(v',v) = \sum_{(v',v)\in D} c(v',v)$. Отсюда $c_f(v)=c(v)$ при $v\notin\{s,t\}$. Суммируя по всем внутренним вершинам, получим $\chi_В(G_f)=\chi_В(G)$. Равенство дуговых и вершинных характеристик влечет за собой равенство характеристик.

Лемма (о характеристике подсети). Пусть $G'=(V',D',s,t)$ – подсеть сети $G=(V,D,s,t)$, то есть $V'\subseteq V$, $D'\subseteq D$. Тогда $\chi(G')\le\chi(G)$.

Доказательство немедленно следует из определений характеристик.

Целочисленная сеть G называется *комбинаторной*, если $\chi(G)=O(m)$. Для целочисленных сетей удается получить хорошие оценки трудоемкости алгоритмов кратчайших путей в терминах характеристики сети, что для комбинаторных сетей приводит к лучшим оценкам трудоемкости в терминах числа вершин и дуг сети, чем в общем случае.

Утверждение. Трудоемкость этапа АС на целочисленной сети G не превосходит $O\big(\chi(G)+m\big)$.

Доказательство. Рассмотрим этап АС с длиной кратчайшего пути, равной r. Пусть S_r – соответствующая справочная кратчайших путей. На одну итерацию АС (нахождение в S_r увеличивающего пути L, вычисление увеличивающего потока и пересчет пропускных способностей) тратится $O(r)$ действий. В результате итерации пропускные способности каждой дуги из L уменьшаются по крайней мере на единицу. Следовательно, характеристика справочной S_r уменьшается за одну итерацию не менее чем на $O(r)$. Но справочная S_r - подсеть некоторой остаточной сети сети G, поэтому $\varkappa(S_r)\leqslant\varkappa(G)$ в силу теоремы о постоянстве характеристики и леммы о характеристике подсети. Поскольку перед каждой итерацией этапа справочная S_r - связная с $\varkappa(S_r)\geqslant 0$, то количество итераций в этапе не превышает $\varkappa(G)/O(r)$, а трудоемкость их проведения не больше $O\left(\varkappa(G)\right)$. Для построения справочной и ее поддержания в течение этапа достаточно $O(m)$ действий.

Утверждение. Количество этапов АКП на целочисленной сети не превышает $O\left(\varkappa(G)^{1/2}\right)$.

Доказательство. Пусть к началу этапа с длиной кратчайшего пути, равной $r>2$, в сети G построен поток f. Обозначим через Δ_r мощность максимального потока в сети G_f. Поскольку до текущего этапа выполнено не более $O(r)$ этапов и на каждом этапе мощность построенного потока возрастает по крайней мере на единицу, количество этапов $K\leqslant\Delta_r+O(r)$.

Оценим величину Δ_r через характеристику сети G. Обозначим через V_i множество вершин, находящихся в сети G_f на расстоянии i от источника. Рассмотрим в сети G_f систему разрезов $R_i=(X_i, V\setminus X_i)$, $1\leqslant i\leqslant r-2$, где $X_i=\bigcup\limits_{j=0}^{i} V_j$. *Очевидно, что множества* прямых дуг этих разрезов не пересекаются (они исходят из разных вершин) и любая прямая дуга в них - внутренняя. Отсюда

$\sum_{i=1}^{r-2} c_f(R_i) < x_Д(D_f)$. Это значит, что среди этих разрезов найдется разрез R_j с $c_f(R_j) < x_Д(G_f)/O(r)$. Применяя теорему о постоянстве характеристики и теорему о максимальном потоке и минимальном разрезе, получим $\Delta_r < x_Д(G)/O(r)$.

Аналогичные рассуждения для системы разрезов $R_i=(Y_i, V\setminus Y_i)$, $1<i<r-1$, где $Y_i = X_{i-1} \cup \left\{v \in V_i : \sum_{(v,v')\in D} c_f(v,v') = c_f(v)\right\}$, показывают, что $\Delta_r < x_В(G)/O(r)$. Таким образом, имеем $\Delta_r < x(G)/O(r)$ и $K < x(G)/O(r)+O(r)$. Полагая $r=O\left(x(G)^{1/2}\right)$, получим $K<O\left(x(G)^{1/2}\right)$.

В результате для целочисленной сети G имеем оценку трудоемкости АС $O\left((x(G)+m)x(G)^{1/2}\right)$, что для комбинаторной сети дает $O\left(m x(G)^{1/2}\right) < O(m^{3/2})$.

6.8. Нахождение максимального паросочетания в двудольном графе

Пусть $\Gamma=(V_1 \cup V_2, E)$ – неориентированный *двудольный* граф с долями вершин V_1 и V_2, $|V_1 \cup V_2|=n$, $|E|=m$. Подмножество ребер $P \subseteq E$ называется *паросочетанием*, если концы всех ребер в нем попарно различны. Очевидно, что $|P| < |V_1|$ и $|P| < |V_2|$. Задачу нахождения в двудольном графе паросочетания максимальной мощности можно эффективно решить потоковыми алгоритмами.

По графу Г построим сеть $G=(V,D,s,t)$ следующим образом.

1. Добавим к множеству вершин графа Г две вершины s и t: $V=V_1 \cup V_2 \cup \{s,t\}$.

2. Заменим каждое ребро $(v,v'), v\in V_1, v'\in V_2$ графа Г дугой (v,v'). Добавим дуги (s,v) для всех $v \in V_1$ и (v,t) для всех $v\in V_2$. Таким образом $D=\left\{(v,v'):(v,v')\in E\right\} \cup \left\{(s,v):v\in V_1\right\} \cup \left\{(v,t):v\in V_2\right\}$.

3. Для всех $d\in D$ положим $c(d)=1$.

В построенной сети $|V|=n+2=O(n)$, $|D|=m+n=O(m)$. Очевидно, эта сеть комбинаторная и $\chi(G)=O(n)$, так как $c(v)=1$ для $v\notin\{s,t\}$.

Имеется взаимно однозначное соответствие между допустимыми целочисленными потоками в сети G и паросочетаниями в графе Г, по которому эта сеть построена. Если f - допустимый поток в сети G с $f(d)\in\{0,1\}$ для $d\in D$, то множество ребер графа Г, соответствующих внутренним дугам d сети G с $f(d)=1$, образует паросочетание. Действительно, из $c(v)=1$ для $v\notin\{s,t\}$ следует, что начала и концы внутренних дуг с $f(d)=1$ попарно различны. Верно и обратное: любое паросочетание P в графе Г может быть получено таким способом по некоторому целочисленному допустимому потоку f в сети G. Обозначим через $V_1(P)$ и $V_2(P)$ концы ребер из P в долях V_1 и V_2. Положим $D'=\{(v,v')\in D:(v,v')\in P\}\cup\{(s,v):v\in V_1(P)\}\cup\cup\{(t,v):v\in V_2(P)\}$ и построим функцию f: $f(d)=1$, если $d\in D'$; $f(d)=0$, если $d\notin D'$. Легко убедиться, что f - допустимый поток в сети G и $f(d)=1$ для внутренней дуги тогда и только тогда, когда соответствующее этой дуге ребро графа Г принадлежит паросочетанию P. Поскольку $|V_1(P)|=|V_2(P)|=|P|$, мощности потока f и соответствующего ему паросочетания P равны.

Таким образом, паросочетанию максимальной мощности в графе Г соответствует максимальный поток в сети G. Построение сети G и нахождение паросочетания по целочисленному допустимому потоку в этой сети может быть выполнено за O(m) действий. Тем самым АС позволяет решить задачу нахождения максимального паросочетания в двудольном графе с трудоемкостью $O(mn^{1/2})\leq O(n^{5/2})$.

6.9. Нахождение минимальных рассекающих множеств в неориентированных графах

Пусть $\Gamma=(V,E)$ - неориентированный граф, $|V|=n$, $|E|=m$. Для произвольной пары вершин $x,y\in E$ подмножество $X\subseteq E$ называется *реберным* x,y*-рассекающим множеством* (x,y-РРМ), если в графе $\Gamma'=(V,E\setminus X)$ вершины x и y принадлежат разным компонентам связности. Аналогично, подмножество $W\subseteq V\setminus\{x,y\}$ называется *вершинным* x,y*-рассекающим множеством* (x,y-ВРМ), если вершины x и y находятся в разных компонентах связности графа Γ', полученного удалением из Γ вершин W и всех инцидентных им ребер. Понятно, что x,y-ВРМ существует тогда и только тогда, когда $(x,y)\notin E$.

Нахождение x,y-рассекающих множеств минимальной мощности в неориентированном графе можно эффективно произвести потоковыми алгоритмами.

По графу $\Gamma=(V,E)$, в котором нужно найти минимальное x,y-РРМ, построим сеть $G=(V,D,x,y)$, заменив каждое ребро (v,v') графа Γ на две противоположно направленные дуги (v,v') и (v',v), и положив $c(d)=1$ для каждой дуги $d\in D$. Полученная сеть - комбинаторная и $\varkappa(G)<|D|=2m$. Легко видеть, что ребра графа Γ, соответствующие дугам минимального разреза в сети G, образуют минимальное x,y-РРМ. Нахождение максимального потока в сети G с помощью АС требует $O(m^{3/2})$ действий, а на построение сети G по графу Γ и нахождение рассекающего множества в графе Γ по максимальному потоку в сети G достаточно $O(m)$ действий.

Для нахождения минимального x,y-ВРМ в графе $\Gamma=(V,E)$ с $(x,y)\notin E$ построим сеть $G=(V',D,x,y)$ следующим образом.

1. Заменим каждую вершину $v\notin\{x,y\}$ двумя вершинами $(v,0)$ и

$(v,1)$: $V'=\big(V\setminus(x,y)\big)\times(0,1)\cup(x,y)$.

2. Каждое ребро (v,v') при $v,v'\notin(x,y)$ заменим на две дуги $\big((v,1),(v',0)\big)$ и $\big((v',1),(v,0)\big)$. Ребра (x,v) заменим дугами $\big(x,(v,0)\big)$, а ребра (y,v) – дугами $\big((v,1),y\big)$. Кроме того, проведем множество дуг $D'=\big\{\big((v,0),(v,1)\big) : v\in V\setminus(x,y)\big\}$.

3. Положим $c(d)=1$ для $d\in D'$ и $c(d)=\langle\infty\rangle$ для $d\notin D'$.

В построенной сети $|V'|=2n-2=O(n)$ и $|D|<2m+n=O(m)$. Так как $c(v)=1$ для всех внутренних вершин сети, сеть G – комбинаторная и $x(G)=O(n)$. Нетрудно проверить, что минимальный разрез в сети G состоит только из дуг, принадлежащих D', и что множество вершин графа Γ, соответствующих дугам этого разреза, образует минимальное x,y-ВРМ. Максимальный поток в сети G можно найти с помощью АС за $O(mn^{1/2})$ действий, а построение сети G и нахождение разрезающего множества по построенному потоку имеют меньшую трудоемкость – $O(m)$ и $O(n)$, соответственно.

Минимальное по мощности x,y-РРМ среди всех v,v'-РРМ графа Γ называется его *минимальным реберным рассекающим множеством* (МРРМ), а мощность МРРМ называется *реберной связностью* этого графа. Аналогично определяются *минимальное вершинное рассекающее множество* (МВРМ) и *вершинная связность* графа.

МРРМ графа $\Gamma=(V,E)$ можно найти, определив минимальные v,v'-РРМ для фиксированной вершины v и всех вершин $v'\in V\setminus v$, и взяв из них минимальное по мощности. Действительно, если X – МРРМ, то в графе $\Gamma'=(V,E\setminus X)$ имеется как минимум две компоненты связности V_1 и V_2. Пусть $v\in V_1$. Тогда X – минимальное v,v'-РРМ для любой вершины $v'\in V_2$. Таким образом, для нахождения МРРМ достаточно $n-1$ раз найти минимальное v,v'-РРМ, что дает оценку трудоемкости $O(nm^{3/2})$. Существует более экономная модификация этого алгоритма. Найдя v,v'-РРМ в графе Γ, дальнейшие вычисления можно проводить в графе Г', получающемся "склеиванием" вершин v и v'. Систематическое использование этого приема приводит к алгоритму с трудоемкостью $O(nm)$.

Нахождение МВРМ графа $Г=(V,E)$ сводится к выбору минимального по мощности из минимальных v,v'-ВРМ для всех $v,v'\in V$ с $(v,v')\notin E$. Трудоемкость такого алгоритма $O(mn^{5/2})$.

Вопросы для самостоятельной работы

1. Доказать допустимость в сети G_f разности $f'\ominus f$ двух допустимых потоков в сети G.

2. Привести пример нецелочисленного максимального потока в целочисленной сети.

*3. Построить алгоритм поддержания справочной кратчайших путей с трудоемкостью в течение этапа $O(m)$.

4. Воспроизвести полностью рассуждения, доказывающие, что $\Delta_r \leq x_B(G)/O(r)$.

*5. В доказательствах п. 6.7 с целью их упрощения допущена сознательная некорректность в использовании символа $O(\,)$ в знаменателе. Как сделать эти доказательства корректными?

6. Построить потоковый алгоритм решения булевской задачи о назначении (п. 1.2) с трудоемкостью $O\left(n_1 n_2 (n_1+n_2)^{1/2}\right)$, где n_1 и n_2 - количество работ и исполнителей, соответственно.

7. Обосновать описанный в п. 6.9 экономный алгоритм построения МРРМ.

8. Задав на множестве вершин графа $Г=(V,E)$ положительный вес $w: V \to \mathbb{R}^+$, поставим задачу нахождения в графе x,y-ВРМ W минимального веса $w(W)=\sum_{v\in W} w(v)$. Построить потоковый алгоритм решения этой задачи.

7. МАТРОИДНЫЕ АЛГОРИТМЫ.

7.1. Матроид и его свойства

Система подмножеств $S\subseteq 2^A$ множества A называется ***матроидом над A***, если она удовлетворяет следующим аксиомам:

1- $u\in S$, $v\subseteq u$ $\Rightarrow$ $v\in S$ (аксиома наследственности);

2- $u,v\in S$, $|u|>|v|$ $\Rightarrow$ $\exists\alpha\in u\setminus v: v\cup\{\alpha\}\in S$ (аксиома пополнения).

В произвольной системе $S\subseteq 2^A$ можно выделить совокупность $B(S)$ максимальных по включению элементов:

$u\in B(S)$ $\Leftrightarrow$ $\forall\alpha\in A\setminus u$ $\; u\cup\{\alpha\}\notin S$.

Если S - матроид, то элементы $B(S)$ называются его ***базами***. Докажем несколько свойств баз матроида.

Теорема (о равномощности баз). Пусть S - матроид; $u,v\in B(S)$. Тогда $|u|=|v|$.

Доказательство. Предположим противное $|u|>|v|$. Тогда, согласно аксиоме пополнения, найдется элемент $\alpha\in u\setminus v$, такой что $v\cup\{\alpha\}\in S$, что противоречит максимальности v.

Теорема (замены баз). Пусть S - матроид; $u,v\in B(S)$, $u\neq v$. Тогда для любого $\alpha\in v\setminus u$ найдется элемент $\beta\in u\setminus v$, такой что $v\setminus\{\alpha\}\cup\{\beta\}\in B(S)$.

Доказательство. Возьмем произвольный элемент $\alpha\in v\setminus u$ и рассмотрим подмножество $v'=v\setminus\{\alpha\}$, принадлежащее S в силу аксиомы наследственности. По теореме о равномощности баз $|u|=|v|$, следовательно, $|u|>|v'|$. Применяя к u и v' аксиому пополнения, получим утверждение теоремы.

7.2. Примеры матроидов

Очевидно, что системы $S=\emptyset$ и $S=2^A$ удовлетворяют аксиомам матроида. Они называются *пустым* и *полным* матроидами. Мы приведем несколько менее тривиальных примеров матроидов.

Линейный матроид. Рассмотрим n-мерное линейное пространство $\mathbb{R}^n$ над полем действительных чисел. Пусть $u=\{u_1,\ldots,u_s\}$ – подмножество векторов из $\mathbb{R}^n$. Подмножество u называется *линейно-независимым*, если $\lambda_1 u_1+\ldots+\lambda_s u_s=0 \Rightarrow \lambda_1=\ldots=\lambda_s=0$.

Система S всех линейно-независимых подмножеств векторов из $\mathbb{R}^n$ образует матроид над $\mathbb{R}^n$. Действительно, выполнение аксиомы наследственности очевидно, а для доказательства выполнения аксиомы пополнения достаточно вспомнить, что множество линейных комбинаций $\mu_1 v_1+\ldots+\mu_t v_t$ векторов из линейно-независимого подмножества $v=\{v_1,\ldots,v_t\}$ образует t-мерное линейное подпространство $\mathbb{R}^t$ в $\mathbb{R}^n$; следовательно, при $|u|>|v|$ в u найдется вектор α, не лежащий в $\mathbb{R}^t$, то есть линейно-независимый с подмножеством v. Базами линейного матроида являются всевозможные базисы пространства $\mathbb{R}^n$; все они имеют одинаковую мощность n.

Графовый матроид. Пусть $\Gamma=(V,E)$ – неориентированный граф. Подмножество $u=\{u_1,\ldots,u_s\}$ ребер графа называется *независимым*, если из ребер этого подмножества нельзя образовать ни одного контура. Граф $\Gamma_u=(V,u)$ в этом случае является лесом – объединением деревьев с непересекающимися множествами вершин.

Оказывается, совокупность S всех независимых подмножеств ребер графа – матроид над E. Выполнение аксиомы наследственности очевидно. Для доказательства выполнения аксиомы пополнения нам потребуеися следующая лемма.

Лемма. Пусть $\Gamma=(V,E)$ - неориентированный граф без контуров (лес). Тогда число компонент связности графа Γ равно $|V|-|E|$.

Доказательство. Проведем доказательство индукцией по количеству ребер. При $|E|=0$ утверждение справедливо. Пусть оно справедливо для графа $\Gamma'=(\Gamma,E')$ при $E'\subset E$. Любое ребро $\alpha\in E\setminus E'$ соединяет вершины из разных компонент связности графа Γ', так как в противном случае его добавление привело бы к образованию контура. Следовательно, добавление ребра α к графу Γ' уменьшает число его компонент связности на 1, что и завершает шаг индукции.

Пусть теперь u,v - независимые подмножества ребер графа $\Gamma=(V,E)$ и $|u|>|v|$. Тогда, согласно лемме, граф $\Gamma_u=(V,u)$ имеет меньше компонент связности, чем граф $\Gamma_v=(V,v)$. Это означает, что найдутся вершины a и b, которые в графе Γ_u принадлежат одной и той же компоненте связности, а в графе Γ_v - разным. Таким образом, вершины a и b соединены цепочкой ребер из u, но не соединены цепочкой ребер из v. Следовательно, в соединяющей эти вершины цепочке ребер из u найдется ребро $\alpha\in u\setminus v$, концы которого принадлежат разным компонентам связности графа Γ_v, и добавление этого ребра к Γ_v не приведет к образованию контуров.

Если Γ - связный граф с n вершинами, то базами его графового матроида будут деревья с $n-1$ ребрами, которые называются *остовными деревьями* этого графа.

Трансверсальный матроид. Пусть $\Gamma=(V,E)$ - двудольный граф с долями вершин V_1 и V_2. Напомним, что подмножество ребер $P\subseteq E$ называется *паросочетанием*, если концы ребер из P попарно различны. Обозначим через $V_1(P)$ вершины из доли V_1, являющиеся концами ребер из P. Будем говорить, что эти вершины *покрываются* парососчетанием P.

Совокупность S подмножеств вершин из V_1, покрываемых каким-либо паросочетанием, представляет собой матроид над V_1. Выполнение аксиомы наследственности очевидно. Докажем выполнение аксиомы пополнения. Пусть P и Q – паросочетания в графе Г и $|P|>|Q|$. Для каждой вершины $\alpha\in V_1(P)\setminus V_1(Q)$ рассмотрим начинающуюся в этой вершине цепочку $Z(\alpha)$ максимальной длины, состоящую из чередующихся ребер из паросочетаний P и Q. Очевидно, что цепочки $Z(\alpha)$ и $Z(\alpha')$ при $\alpha\neq\alpha'$ не пересекаются по ребрам и вершинам. Обозначим через $P(\alpha)$ и $Q(\alpha)$ ребра из P и Q, входящие в $Z(\alpha)$, и пусть $\beta(\alpha)$ – конец цепочки $Z(\alpha)$. Легко видеть, что если $\beta(\alpha)\in V_1(Q)\setminus V_1(P)$, то $|P(\alpha)|=|Q(\alpha)|$, в противном случае $|P(\alpha)|=|Q(\alpha)|+1$. Но $|\{\beta(\alpha):\alpha\in V_1(P)\setminus V_1(Q)\}|=|V_1(P)\setminus V_1(Q)|>$ $>|V_1(Q)\setminus V_1(P)|$. Следовательно, найдется вершина $\alpha\in V_1(P)\setminus V_1(Q)$, для которой $|P(\alpha)|=|Q(\alpha)|+1$. При этом множество ребер $Q\setminus Q(\alpha)\cup P(\alpha)$ – паросочетание, покрывающее вершины $V_1(Q)\cup(\alpha)$.

Метод, использованный нами при доказательстве, широко используется при конструировании алгоритмов дискретной оптимизации на двудольных графах и носит название метода *чередующихся цепей*.

7.3. Жадный алгоритм

Пусть $A=\{a_1,\ldots,a_n\}$ – конечное множество; $w:A\to\mathbb{R}^+$ – отображение, ставящее в соответствие элементу $a\in A$ его положительный *вес* $w(a)$.

Рассмотрим следующую задачу дискретной оптимизации: найти подмножество $u\subseteq A$, удовлетворяющее предикату P, определенному на

2^A, и доставляющее максимум функционалу $F(u)=\sum_{a\in u} w(a)$.

В связи с этой задачей опишем *жадный* алгоритм, позволяющий найти подмножество u, удовлетворяющее предикату $P(u)$ и доставляющее функционалу $F(u)$ "не слишком маленькое" значение.

Пусть множество A упорядочено по неубыванию весов его элементов: $w(a_1)\geq w(a_2)\geq\ldots\geq w(a_n)$. В предположении, что $P(\emptyset)$ истинно, положим $u_0=\emptyset$, и выполним n раз следующую рекуррентную процедуру: $x=u_{i-1}\cup\{a_i\}$; $u_i=x$, если $P(x)$ истинно; $u_i=u_{i-1}$, если $P(x)$ ложно.

Подмножество u_n, полученное этим алгоритмом, очевидно удовлетворяет предикату P. Однако далеко не всегда u_n доставляет максимум функционалу F (в частности, u_n может даже не быть максимальным по включению среди подмножеств, удовлетворяющих предикату P). Уместно задать вопрос: при каких условиях жадный алгоритм решает описанную выше задачу оптимизации? Ответ на этот вопрос дает следующая теорема.

Теорема. Если область истинности предиката P - матроид S над A, то полученное жадным алгоритмом подмножество u_n доставляет максимум функционалу F.

Доказательство. Из описания жадного алгоритма видно, что u_n - база матроида S. Вследствие положительности весов максимум F достигается также на некоторой базе матроида. Предположим, что $F(u_n)<\max F$. Пусть v - база матроида, для которой $F(v)=\max F$. Если таких баз максимального веса несколько, выберем в качестве v ту из них, для которой максимально $|v\cap u_n|$. Пусть α - элемент минимального веса из $u_n\setminus v$, а β - элемент минимального веса из $v\setminus u_n$. Рассмотрим два случая.

1. $w(\alpha)<w(\beta)$. Произведем замену базы, удалив из u_n элемент α и добавив некоторый элемент $\gamma\in v\setminus u_n$, так чтобы в соответствии

с теоремой о замене базы $u_n'=u_n\setminus\{\alpha\}\cup\{\gamma\}\in S$. Так как β – элемент минимального веса в $v\setminus u_n$, то $w(\gamma)\geq w(\beta)$, следовательно $w(\gamma)>w(\alpha)$. Пусть $\gamma=a_i$. Тогда в процессе работы жадного алгоритма рассматривалось подмножество $x=u_{i-1}\cup\{a_i\}\subseteq u_n'$. В силу аксиомы наследственности $x\in S$, следовательно $P(x)$ истинно и $a_i=\gamma\in u_n$. Имеет место противоречие с $\gamma\in v\setminus u_n$.

2. $w(\alpha)<w(\beta)$. Произведем замену базы, удалив из v элемент β и добавив элемент $\gamma\in u_n\setminus v$, так чтобы $v'=v\setminus\{\beta\}\cup\{\gamma\}\in S$. Так как α – элемент минимального веса в $u_n\setminus v$, то $w(\gamma)\geq w(\alpha)$, следовательно $w(\gamma)\geq w(\beta)$. Если $w(\gamma)>w(\beta)$, то $F(v')>F(v)$, что противоречит предположению $F(v)=\max F$. Если же $w(\gamma)=w(\beta)$, то $F(v')=F(v)$, но $|v'\cap u_n|>|v\cap u_n|$, что противоречит выбору v по максимальному пересечению с u_n.

Полученные в обоих случаях противоречия опровергают предположение $F(u_n)<\max F$, и тем самым доказывают теорему.

7.4. Задача об остовном дереве максимального веса

В п. 7.2 было показано, что остовное дерево неориентированного графа является базой его графового матроида. Тем самым, остовное дерево максимального веса в графе со взвешенными ребрами может быть найдено жадным алгоритмом.

Для эффективной реализации жадного алгоритма в этом случае необходимо разработать быстрый способ проверки, не приводит ли к образованию контуров добавление ребра a_i к лесу u_{i-1}. Это легко сделать, зная разбиение вершин V графа $\Gamma_{i-1}=(V,u_{i-1})$ на компоненты связности. Именно, подмножество ребер $u_{i-1}\cup\{a_i\}$ не образует контуров тогда и только тогда, когда концы ребра a_i принадлежат разным компонентам связности графа Γ_{i-1}. Если это условие выполнено, то есть $u_i = u_{i-1} \cup \{a_i\}$ не содержит контуров, разбиение на компоненты связности вершин графа $\Gamma_i = (V, u_i)$ получается из предыдущего разбиения объединением компонент, к которым принадлежат концы ребра a_i. Объединение двух компонент связности графа с $n=|V|$ вершинами требует не более $O(n)$ действий, тем самым все $m=|E|$ шагов жадного алгоритма, в течение которых необходимо сделать $n-1$ объединение компонент связности, требуют не более $O(n^2)$ действий. Для использовании жадного алгоритма необходимо предварительно упорядочить ребра по неубыванию весов, что можно сделать за $O(m \log m)$ действий. Таким образом, задача об осговном дереве максимального веса имеет сложность, не превышающую $O(m \log m + n^2)$.

7.5. Задача о представителях множеств

Пусть $A=\{a_1, \ldots, a_n\}$ - конечное множество, $S=\{s_1, \ldots, s_k\} \subseteq 2^A$ - система его подмножеств. Подмножество $M \subseteq A$ называется *системой представителей* системы подмножеств S, если существует отображение $\varphi : M \to S$, такое что $a \in \varphi(a)$ для каждого $a \in M$ и образы различных элементов из M различны. Задавая на множестве A *вес* $w : A \to \mathbb{R}^+$, приходим к задаче нахождения системы представителей максимального веса $w(M) = \sum_{a \in M} w(a)$.

Построим двудольный граф $\Gamma=(A \cup S, E)$ с долями вершин A и S и множеством ребер $E=\{(a_i, s_j) : a_i \in s_j\}$. Легко видеть, что подмножества множества A, покрываемые паросочетаниями, и только они, являются системами представителей системы подмножеств S, причем покрывающее паросочетание как раз и задает отображение φ. Тем самым, системы представителей образуют трансверсальный матроид графа Г, что позволяет использовать для нахождения системы представителей максимального веса жадный алгоритм.

Для реализации жадного алгоритма на трансверсальном матроиде необходимо иметь эффективную процедуру проверки, покрывается ли паросочетанием в графе Г подмножество $u_i \in A$, полученное из покрываемого паросочетанием подмножества $u_{i-1} \in A$ добавлением вершины a_i. Это удается сделать, если вместе с подмножеством u_{i-1} хранить покрывающее его паросочетание $P(u_{i-1})$.

Обозначим через $A(P)$ и $S(P)$ множество вершин из долей A и S графа Г, покрытых паросочетанием P. Цепь L в графе $\Gamma=(A\cup S,L)$, начинающаяся в вершине $a \in A\setminus A(P)$ и заканчивающаяся в вершине $s \in S\setminus S(P)$, называется *чередующейся цепью* относительно паросочетания P, если ее ребра с нечетными номерами принадлежат $E\setminus P$, а ребра с четными номерами принадлежат P. Очевидно, что длина чередующейся цепи нечетна и ребер из $E\setminus P$ в ней на одно больше, чем ребер из P.

Утверждение. Подмножество $u_i = u_{i-1} \cup (a_i)$, полученное добавлением к подмножеству u_{i-1}, покрытому паросочетанием $P(u_{i-1})$, вершины a_i, покрывается паросочетанием тогда и только тогда, когда в графе $\Gamma=(A\cup S,E)$ существует чередующаяся относительно паросочетания $P(u_{i-1})$ цепь L, начинающаяся в вершине a_i.

Доказательство. Пусть L - чередующаяся цепь относительно $P(u_{i-1})$, начинающаяся в a_i. Тогда $P(u_{i-1})\setminus L \cup L\setminus P(u_{i-1})$ - паросочетание, покрывающее u_i. Обратно, если u_i покрывается паросочетанием $P(u_i)$, то из ребер $P(u_i)\setminus P(u_{i-1}) \cup P(u_{i-1})\setminus P(u_i)$ можно составить чередующуюся относительно $P(u_{i-1})$ цепь с началом в a_i.

Для нахождения чередующейся относительно $P(u_{i-1})$ цепи с *началом в* a_i *построим ориентированный граф* $\Gamma_i=(A\cup S,D_i)$, *получа-*

ющийся из графа $\Gamma=(A\cup S,E)$ ориентацией ребер: ребро $(a,s)\in E$ заменяется дугой $(a,s)\in D_i$, если $(a,s)\notin P(u_{i-1})$, или дугой $(s,a)\in D_i$, если $(a,s)\in P(u_{i-1})$. Легко видеть, что каждая чередующаяся относительно $P(u_{i-1})$ цепь в графе Γ порождает путь в графе Γ_i с началом в вершине $a\in A\setminus A\big(P(u_{i-1})\big)$ и концом в вершине $s\in S\setminus S\big(P(u_{i-1})\big)$. Обратно, каждый такой путь в графе Γ_i снятием ориентации дуг превращается в чередующуюся относительно $P(u_{i-1})$ цепь в графе Γ. Таким образом, если в графе Γ_i есть путь из вершины a_i в какую либо вершину из $S\setminus S\big(P(u_{i-1})\big)$, то u_i покрывается в графе Γ паросочетанием $P(u_i)=P(u_{i-1})\setminus L\cup L\setminus P(u_{i-1})$, где L – соответствующая чередующаяся цепь. В противном случае $u_{i-1}\cup(a_i)$ не покрывается паросочетанием в графе Γ. Нахождение подходящего пути в графе Γ_i можно выполнить фронтальным (п. 5.3) или радиальным (п. 5.4) обходом с корнем a_i за $O(m)$ действий, где $m=|E|=\sum\limits_{s\in S}|s|$. Тем самым трудоемкость решения задачи о представителях множеств жадным алгоритмом равна $O(nm)$.

К задаче о представителях множеств сводится, например, следующая модификация булевской задачи о назначении (п. 1.2): на множестве работ задана положительная весовая функция и требуется найти назначение, максимизирующее суммарный вес выполненных работ. В качестве множества A возьмем множество работ, а в качестве элемента системы S – подмножество работ, которые могут быть выполнены определенным исполнителем.

Вопросы для самостоятельной работы

*1. Доказать, что система подмножеств, для которой выполнена аксиома наследственности и свойство замены максимальных по включению множеств, является матроидом.

2. Доказать, что система всех подмножеств множества A, мощность которых не превосходит фиксированного $k \le |A|$, является матроидом над A (*однородный* матроид).

3. Образуют ли матроид все паросочетания в двудольном графе?

4. Привести пример задачи оптимизации, для которой жадный алгоритм не дает оптимального решения.

*5. Разработать алгоритм нахождения остовного дерева максимального веса в графе с уже упорядоченными ребрами с трудоемкостью $O(m+n\log n)$.

6. Как использовать жадный алгоритм для нахождения в графе остовного дерева минимального веса?

7. Свести к задаче о представителях множеств следующую модификацию булевской задачи о назначении (п. 1.2): на множестве исполнителей задана положительная весовая функция и требуется найти назначение, максимизирующее суммарный вес занятых исполнителей.

8. ПЕРЕБОРНЫЕ АЛГОРИТМЫ

8.1. Дерево полного перебора

Алгоритм полного перебора для решения задачи дискретной оптимизации $R=(X,Y,P,F)$ легко реализовать, производя для заданного входа $x \in X$ последовательную генерацию всех возможных управлений $y \in Y(x)$, для каждого из них проверяя допустимость $P(x,y)$, вычисляя функционал $F(x,y)$ для допустимых управлений и запоминая экстремальное значение из всех уже вычисленных значений функционала вместе с управлением, доставляющим это значение. Недостаток такого подхода состоит в том, что время работы алгоритма при входе x не может быть меньше, чем $|Y(x)|$.

В этой главе мы изложим другой подход к построению переборных алгоритмов, связанный с введением некоторой иерархической структуры на подмножествах множества $Y(x)$. При этом, проведя перебор части множества $Y(x)$, удается отказаться от перебора некоторых других возможных управлений, так как они заведомо не являются допустимыми управлениями с лучшим значением функционала, чем уже рассмотренные. К сожалению, сокращение перебора при таком подходе чаще всего носит *эвристический* характер: для большинства входов время работы оказывается существенно меньше, чем при полном переборе, в тоже время, из-за наличия входов, для которых сокращение перебора незначительно, трудоемкость полученных алгоритмов растет столь же быстро, как и трудоемкость алгоритма полного перебора. Лишь в редких случаях такой подход приводит к алгоритму с полиномиальной трудоемкостью при экспоненциальной мощности множества возможных управлений (один такой пример будет рассмотрен в п..8.2).

Пусть нам нужно найти управление $y \in Y(x)$, удовлетворяющее предикату $P(x,y)$ и доставляющее экстремальное значение функционалу $F(x,y)$. Разобьем множество $Y=Y(x)$ на подмножества $Y_1, Y_2, \ldots, Y_k$. Тогда для решения исходной задачи нам достаточно решить k подзадач нахождения управления $y \in Y_i$, $i=1,\ldots,k$, удовлетворяющего предикату $P(x,y)$ и экстремизирующего функционал $F(x,y)$ и выбрать экстремальное из не более чем k найденных экстремальных значений (решения некоторых подзадач могут не существовать). Для решения каждой подзадачи воспользуемся тем же приемом: множество Y_i разобьем на подмножества $Y_{i,1}, Y_{i,2}, \ldots, Y_{i,k_i}$, сведя подзадачу первого уровня к ряду подзадач второго уровня. Дальше будем действовать точно также: для решения подзадачи s-го уровня с множеством $Y_{i_1,\ldots,i_s}$ разобьем это множество на подмножества $Y_{i_1,\ldots,i_s,1}, \ldots, Y_{i_1,\ldots,i_s,k_{i_1,\ldots,i_s}}$, и так далее до получения одноэлементных подмножеств. Обозначим построенную совокупность подмножеств множества Y через U и рассмотрим ориентированный граф $T=(U,D)$, где $(u,u') \in D$ тогда и только тогда, когда u' является классом разбиения множества u. Очевидно, что T - ориентированное дерево с корнем Y, терминальными вершинами которого являются все одноэлементные подмножества множества Y. Полученное дерево носит название *дерева полного перебора*.

Используя тот или иной обход дерева полного перебора, можно, в принципе, добиться сокращения перебора, элиминируя рассмотрение некоторых поддеревьев в этом дереве. В последующих разделах этой главы мы рассмотрим два таких метода - динамическое программирование и метод ветвей и границ, связанные с фронтальным и радиальным обходами дерева полного перебора.

Заметим, что так как в ориентированном дереве существует взаимно однозначное соответствие между дугами и некорневыми вершинами, обход дерева можно интерпретировать как бесповторную последовательность его вершин, отличных от корня. Конечно, дерево полного перебора столь велико, что хранить его в памяти целиком невозможно. Конструировать дерево полного перебора нужно так, чтобы можно было генерировать в процессе обхода нужные его части (в идеале - вершины в порядке появления их в обходе).

8.2. Динамическое программирование

Метод сокращения перебора, основанный на фронтальном обходе дерева полного перебора с корнем, совпадающим с корнем дерева, носит название *динамического программирования.*

Легко видеть, что любой фронт обхода с корнем Y дерева полного перебора $T=(U,D)$ отделяет корень дерева от всех его терминальных вершин, что сводит задачу нахождения оптимального управления из множества Y к совокупности подзадач на подмножествах, принадлежащих фронту обхода.

Определим на вершинах фронта обхода отношение эквивалентности, так чтобы из эквивалентности вершин u и u' следовало, что либо обе подзадачи на подмножествах u и u' не имеют решения, либо $\operatorname{extr}_{y\in u} F(x,y)=\operatorname{extr}_{y\in u'} F(x,y)$. Тогда вместо решения подзадач для всех подмножеств фронта достаточно решить по одной подзадаче для каждого класса эквивалентности, за счет чего и получается сокращение перебора в случае, если среди подзадач на подмножествах фронта много "одинаковых".

Проиллюстрируем использование динамического программирования на примере определения оптимального порядка умножения матриц. Пусть нам нужно найти произведение $M=M_1\cdot M_2\cdot\ldots\cdot M_n$ прямоугольных матриц размерами $(m_0\times m_1),(m_1\times m_2),\ldots,(m_{n-1}\times m_n)$, соответственно. В силу ассоциативности умножения матриц порядок действий при вычислении M может быть произвольным, но от принятого порядка действий время выполнения вычислений может зависеть весьма существенно. Например, если обозначить через $t(p,q,r)$ время умножения матриц размерами $(p\times q)$ и $(q\times r)$, то время вычисления выражения $M_1\cdot\left(M_2\cdot(M_3\cdot M_4)\right)$ равно $t(m_2,m_3,m_4)+$ $+t(m_1,m_2,m_4)+t(m_0,m_1,m_4)$, а время вычисления выражения $\left(M_1\cdot(M_2\cdot M_3)\right)\cdot M_4$ равно $t(m_1,m_2,m_3)+t(m_0,m_1,m_3)+t(m_0,m_3,m_4)$. Полагая $t(p,q,r)=2pqr$ (обычный алгоритм умножения матриц) при $m_0=10$, $m_1=20$, $m_2=50$, $m_3=1$ и $m_4=100$ получим, что при первом порядке вычислений мы потратим 250000 действий, а при втором - только 4400!

Пронумеруем символы умножения матриц слева направо числами от 1 до n-1. Тогда порядок выполнения умножений полностью определяется перестановкой $\varkappa$ множества $\{1,\ldots,n-1\}$: $\varkappa(i)$ указывает номер умножения, которое выполняется i-ым по порядку. Таким образом, в этой задаче множество возможных управлений совпадает с множеством всех перестановок множества $\{1,\ldots,n-1\}$ и все управления допустимы. Функционал, равный времени вычисления для заданного порядка действий, легко считается по управлению и числам m_i, $i=0,1,\ldots,n$ при заданной функции $t(p,q,r)$.

Разобьем множество управлений Y на n-1 подмножество $Y_1,Y_2,\ldots,Y_{n-1}$, так что в подмножество Y_i попадут все перестановки $\varkappa$ с $\varkappa(n-1)=i$, то есть те порядки выполнения действий, при которых последним выполняется умножение с номером i. Каждое из этих подмножеств разобьем на $n-2$ подмножества, в каждом из которых будут фиксированы номера двух последних умножений. Действуя так и дальше, получим дерево полного перебора, в котором подмножество k-го уровня $Y_{i_1,\ldots,i_k}$ состоит из перестановок π, у которых $\pi(n-1)=i_1,\ldots,\pi(n-k)=i_k$.

Оказывается, что среди подмножеств одного уровня много эквивалентных. Например при $1<i<n-1$, $1\leq j<i$, $i<k\leq n-1$ эквивалентны подмножества $Y_{i,j,k}$ и $Y_{i,k,j}$, так как безразлично в каком порядке завершать вычисление сомножителей для последнего умножения. Более того, мы покажем, что, несмотря на экспоненциальную зависимость от n числа вершин в дереве полного перебора, число классов эквивалентных вершин в нем зависит от n полиномиально.

Обозначим через $\tau(s,r)$ для $1\leq s\leq r\leq n$ минимальное время вычисления выражения $M_s\cdot\ldots\cdot M_r$. Имеет место очевидное соотношение $\tau(s,r)=\min\limits_{s\leq i\leq r-1}\big(\tau(s,i)+\tau(i+1,r)+t(m_{s-1},m_i,m_r)\big)$. Замечая, что $\tau(i,i)=0$ для $1\leq i\leq n-1$, и что разность аргументов функции τ в правой части выражения меньше, чем в левой, убеждаемся, что $\tau(s,r)$ для всех наборов аргументов $1\leq s\leq r\leq n$ (а их всего $O(n^2)$ штук) можно вычислить в порядке возрастания разности аргументов за $O(n^3)$ действий. Фиксируя при этом значение i, дающее минимум правой части, получим и оптимальные последовательности вычисления всех отрезков искомого произведения, то есть решения всех подзадач в построенном дереве перебора.

8.3. Метод ветвей и границ

Метод сокращения перебора, известный под названием *метода* *ветвей и границ*, основан на радиальном обходе дерева полного перебора. Для каждого появившегося в обходе множества возможных управлений делается попытка доказать, что в этом подмножестве нет ни одного допустимого управления, лучшего, чем уже найденное.

Предикат Q, определенный на вершинах дерева полного перебора, называется *распространением* предиката P, если из ложности предиката Q на каком-либо подмножестве u следует ложность предиката P на любом управлении $y \in u$. Другими словами, истинность предиката $Q(u)$ является необходимым условием существования допустимого управления в подмножестве u возможных управлений.

Рассмотрим для примера построение распространения предиката в задаче *о наибольшей клике* в графе $Г=(V,E)$. *Возможными* управлениями здесь будут подмножества вершин графа, и предикат допустимости P на подмножестве $\{v_1,\dots,v_k\}$ истинен тогда и только тогда, когда подграф графа Г, порожденный вершинами $v_1,\dots,v_k$, полный. Рассмотрим дерево полного перебора для этой задачи, вершина которого $Y_{v_1,\dots,v_k}$ состоит из всех подмножеств, содержащих вершины $v_1,\dots,v_k$ графа Г. Из того, что все порожденные подграфы полного графа также полные, следует, что предикат Q, истинный на $Y_{v_1,\dots,v_k}$ тогда и только тогда, когда вершины $v_1,\dots,v_k$ - клика в графе Г, является распространением предиката P.

Пусть рассматривается задача оптимизации на минимум функционала F. Функционал Ф, определенный на вершинах дерева полного перебора, называется *нижней оценкой* функционала F, если $Ф(u)<F(y)$ для любого допустимого $y \in u$. Аналогично определяется *верхняя оценка* функционала для задачи максимизации.

Регулярный способ построения оценок функционала заключается в расширении множества допустимых управлений. Так, минимальное значение функционала в задаче о назначении доставляет нижнюю оценку функционала в задаче о коммивояжере (смотри вопрос 5 к разделу 1). Такую же роль выполняет обычная задача линейного программирования по отношению к задаче целочисленного линейного программирования.

Заметим, что при фиксированном дереве перебора существует, вообще говоря, много различных распространений предиката P и оценок функционала F. На множестве распространений предиката P введем частичный порядок: будем говорить, что распространение Q *сильнее* распространения Q', если из ложности Q следует ложность Q'. Самым слабым в этом порядке будет тождественно истинный предикат, а самым сильным - предикат, описывающий необходимое и достаточное условие существования допустимого управления в подмножестве возможных управлений. На множестве оценок функционала F также введем частичный порядок: нижняя оценка Φ *сильнее* нижней оценки Φ', если $\Phi(u) \geq \Phi'(u)$ для любого $u \in U$. Самая слабая оценка определяется при этом формулой $\Phi(u) = -\infty$, а самая сильная - это минимум функционала F по всем допустимым управлениям из u, то есть точное решение подзадачи на подмножестве u. Частичный порядок на множестве верхних оценок вводится аналогично.

Метод ветвей и границ состоит в использовании для сокращения перебора распространений предиката P и оценок функционала Φ. Пусть в процессе радиального обхода дерева полного перебора мы фиксируем экстремальное значение f^* функционала F среди всех пройденных терминальных вершин дерева (это значение называется *текущим рекордом*). При рассмотрении появившейся в обходе нетерминальной вершины u вычислим для нее одно или несколько распространений предиката P. Если хотя бы одно из этих распространений ложно, то можно исключить из обхода поддерево с корнем в вершине u, так как все его терминальные вершины не являются допустимыми управлениями. Аналогично, вычислим для вершины u значения одной или нескольких оценок (нижних - для задачи минимизации, верхних - для задачи на максимум). Если хотя бы для одной из оценок окажется $f^* < \Phi(u)$ в случае задачи минимизации (или $f^* > \Phi(u)$ для задачи максимизации), то также можно исключить из рассмотрения поддерево с корнем в вершине u, так среди его терминальных вершин нет допустимых управлений, доставляющих функционалу F лучшее значение, чем текущий рекорд.

Конечно, чем сильнее распространение предиката или оценка функционала, тем большее сокращение перебора дает их применение. Однако вычисление более сильных распространений и оценок, как правило, и более трудоемко. Поэтому при использовании в методе ветвей и границ нескольких распространений предиката и оценок функционала, их следует применять в порядке возрастания силы и трудоемкости вычисления. Решить вопрос о целесообразности использования того или иного распространения предиката (или оценки функционала), то есть убедиться в том, что время, затрачиваемое на его вычисление, окупается сокращением перебора, зачастую удается только путем вычислительного эксперимента.

Эффект сокращения перебора за счет использования оценки функционала оказывается тем больше, чем быстрее текущий рекорд приближается к экстремуму функционала. Поэтому важное значение имеет порядок, в котором рассматриваются вершины, подчиненные какой-либо вершине дерева перебора (напомним, что правила радиального обхода этот порядок не фиксируют). Функция, определенная на вершинах дерева полного перебора, по возрастанию или убыванию которой упорядочивается обход вершин дерева, подчиненных одной и той же вершине, называется *экспресс-оценкой.* В качестве экспресс-оценки чаще всего используют одну из соответствующих оценок (нижнюю или верхнюю) функционала F. Однако упорядочивать вершины в обходе можно и из совсем других соображений. Например, неплохие результаты дает упорядочение по возрастанию экспресс-оценки, равной количеству терминальных вершин в поддереве с корнем в рассматриваемой вершине. В основе такой эвристики лежит следующее правдоподобное рассуждение: маленькие поддеревья с небольшим перебором можно рассматривать и при плохом значении рекорда, перебор же больших поддеревьев следует отложить на потом, в надежде на то, что к этому моменту уже удастся найти достаточно хороший рекорд.

В рамках метода ветвей и границ удобно применять еще один прием сокращения перебора, называемый *форсированием.* Обозначим через $\bar{u}$ множество вершин поддерева с корнем u. Пусть A - отображение, ставящее в соответствие множеству u совокупность его непересекающихся подмножеств из $\bar{u}$, такую что $\operatorname{extr}\limits_{y\in u} F(y) = \operatorname{extr}\limits_{u'\in A(u)} \operatorname{extr}\limits_{y\in u'} F(y)$, где экстремумы берутся только по допустимым управлениям. Тогда обход поддерева с корнем u можно заменить обходом поддеревьев с корнями из $A(u)$. Примеры построения форсирующего оператора A для одной задачи мы приведем в следующем разделе.

Реализация метода ветвей и границ для конкретной задачи дискретной оптимизации - дело совсем не простое, требующее как большой теоретической проработки (доказательства ряда сп цифических утверждений для построения распространений предиката, оценок функционала и операторов форсирования), так и искусства в организации и кодировании данных. Прежде всего это относится к построению дерева перебора, так чтобы, с одной стороны, было возможно эффективно генерировать его вершины в выбранном порядке обхода, и, с другой стороны, чтобы на вершинах этого дерева удалось определить достаточно сильные и легко вычисляемые распространения предиката и оценки функционала. В следующем разделе мы опишем реализацию метода ветвей и границ для задачи нахождения дугового разреза циклов минимального веса в ориентированном графе.

8.4. Нахождение минимального дугового разреза циклов в ориентированном графе

Пусть дан ориентированный граф $\Gamma=(V,D)$, $|D|=m$ с дугами, взвешенными функцией $w: D \to R^+$. Требуется найти *дуговой разрез циклов в этом графе, то есть подмножество дуг $M \subseteq D$ минимального* веса $w(M)=\sum_{d \in M} w(d)$, такое чтобы граф $\Gamma_M=(V,D\setminus M)$ был ациклическим.

В этой задаче множество допустимых управлений $Y=2^D$ - множество всех подмножеств дуг графа; предикат P на подмножестве y истинен тогда и только тогда, когда граф $\Gamma_y=(V,D\setminus y)$ не содержит циклов; функцианал F определяется фомулой $F(y)=\sum_{d \in y} w(d)$.

Построим дерево полного перебора для этой задачи. Множество возможных управлений Y разобьем на классы $Y_1, Y_2, \ldots, Y_m$, так что класс Y_i состоит из всех подмножеств дуг, содержащих дугу d_i и не содержащих дуг $d_1, \ldots, d_{i-1}$. Множество Y_i в свою очередь разобьем на классы $Y_{i,i+1}, \ldots, Y_{i,m}$, что в класс $Y_{i,j}$, $i<j$ попадут все подмножества дуг, содержащие дуги d_i и d_j и не содержащие дуг $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_{j-1}$. Действуя таким же образом и

дальше, получим дерево полного перебора с множеством вершин U, так что вершина k-го уровня $Y_{i_1,i_2,\dots,i_k}$, $i_1<i_2<\dots<i_k$, состоит из всех подмножеств дуг, содержащих дуги $d_{i_1},d_{i_2},\dots,d_{i_k}$ и не содержащих дуги $d_1,\dots,d_{i_1-1},d_{i_1+1},\dots,d_{i_2-1},\dots$ $\dots,d_{i_{k-1}+1},\dots,d_{i_k-1}$. Вершине u построенного дерева соответствует разбиение множества дуг D на три класса: $D^+(u)=\bigcap\limits_{y\in u} y$, $D^-(u)=\bigcap\limits_{y\in u}(D\setminus y)$ и $D^0(u)=D\setminus\left(D^+(u)\cup D^-(u)\right)$. Это соответствие взаимно однозначное: по классам D^+ и D^- любого разбиения множества дуг на три класса строится вершина дерева $u(D^+,D^-)$, состоящая из всех подмножеств y, таких что $D^+\subseteq y$ и $D^-\cap y=\emptyset$. Легко видеть, что это соответствие *антимонотонно* относительно операции включения: из $u\subseteq u'$ следует $D^+(u')\subseteq D^+(u)$ и $D^-(u')\subseteq D^-(u)$, и наоборот. Используя кодировку вершин дерева при помощи двух непересекающихся подмножеств множества D, легко организовать генерацию в лексикографическом порядке вершин, подчиненных вершине u. Первая из этих вершин u_1' определяется разбиением $D^+(u_1)=D^+(u)\cup d^*$, $D^-(u_1)=D^-(u)$, где d^* – дуга с наименьшим номером в $D^0(u)$. Вершина u_{i+1}, лексикографически следующая за вершиной u_i, определяется разбиением $D^+(u_{i+1})=D^+(u)\cup d^*$, $D^-(u_{i+1})=D^-(u_i)\cup$ $\cup\left(D^+(u_i)\setminus D^+(u)\right)$, где d^* – дуга с наименьшим номером в $D^0(u_i)$. Таким образом, при подходящей организации данных можно производить удлинение радиуса обхода дерева перебора этой задачи за константу действий.

Займемся теперь конструированием распространения предиката P на вершины построенного дерева.

Утверждение. В множестве u есть допустимые управления тогда и только тогда, когда граф $\Gamma_u=\left(V,D^-(u)\right)$ – ациклический.

Доказательство. Пусть граф Γ_u – ациклический. Тогда в u

содержится подмножество дуг $y=D\setminus D^-(u)$, для которого граф $\Gamma_y=(V,D\setminus y)$ совпадает с Γ_u. Наоборот, пусть в множестве u есть управление y, для которого граф $\Gamma_y=(V,D\setminus y)$ - ациклический. Так как при этом $D^-(u)\subseteq D\setminus y$, то граф $\Gamma_u=\left(V,D^-(u)\right)$ также ациклический.

Таким образом, предикат Q, истинный в вершине u дерева перебора тогда и только тогда, когда граф $\left(V,D^-(u)\right)$ ациклический, является самым сильным распространением предиката P. Эффективную проверку этого предиката можно выполнить алгоритмом из п. 5.2. Еще более упростить вычисление предиката Q при удлинении радиуса обхода позволяет очевидная лемма.

Лемма. Пусть $\Gamma=(V,D)$ - произвольный ациклический граф. Граф Γ', получающийся из графа Γ добавлением дуги (a,b), $a,b\in V$, является ациклическим *тогда и только тогда*, когда в графе Γ вершина a не достижима из вершины b.

Перейдем теперь к получению нижних оценок функционала F.

Из *положительности* весов дуг графа Γ следует, что $w\left(D^+(u)\right)<w(y)$ для любого $y\in u$. Поэтому функционал Φ_1, определяемый легко вычисляемой формулой $\Phi_1(u)=\sum\limits_{d\in D^+(u)} w(d)$, является нижней оценкой для функционала F. Если дуги графа Γ занумерованы в порядке неубывания весов, то описанный выше алгоритм генерации вершин дерева перебора строит вершины, подчиненные одной и той же вершине, в порядке неубывания функционала Φ_1. Таким образом, Φ_1 при этом служит экспресс-оценкой вершины.

Более сильную оценку функционала F можно получить, вспомнив, что в ациклическом графе обязательно найдется и вершина с нулевой полустепенью захода, и вершина с нулевой полустепенью исхода (смотри п. 5.2 и вопрос 1 к разделу 5). Для $y\in u$ положим

$D(u,y)=y\setminus D^+(u)$. Тогда $w(y)=w\left(D^+(u)\right)+w\left(D(u,y)\right)$ и минимум функционалу F на допустимых управлениях из u доставляет управление $y^*=D^+(u)\cup D^*(u)$, где $D^*(u)=D(u,y^*)$ – дуговой разрез циклов минимального веса в графе $\Gamma_u^0=\left(V,D\setminus D^+(u)\right)$. Оценим снизу вес минимального дугового разреза циклов в произвольном графе $\Gamma=(V,D)$. Для $v\in V$ через $w^+(v)$ и $w^-(v)$ обозначим суммы весов дуг, входящих в вершину v, и исходящих из нее, соответственно. Тогда вес минимального дугового разреза циклов не меньше величины $w(\Gamma)=\max\left(\min_{v\in V} w^+(v),\min_{v\in V} w^-(v)\right)$. Таким образом функционал Φ_2, определяемый формулой $\Phi_2(u)=\Phi_1(u)+w\left(\Gamma_u^0\right)$, представляет собой нижнюю оценку функционала F.

Одно из возможных форсирований в этой задаче основано на использовании следующего утверждения.

Утверждение. Пусть $\Gamma=(V,D)$ – произвольный граф, $M\subseteq D$ – некоторый дуговой разрез циклов в нем, $A\subset M$. Тогда, если $M\setminus A$ содержит дуги, через которые не проходят циклы в графе $(V,D\setminus A)$, то M не является дуговым разрезом минимального веса.

Доказательство. Если $d\in M\setminus A$ – дуга, не являющаяся циклической в графе $(V,D\setminus A)$, то легко видеть, что $M\setminus d$ – дуговой разрез циклов в графе Γ, причем меньшего веса, чем M.

Из этого утверждения следует, что управление, минимизирующее функционал F на множестве u, содержится в подмножестве $u'=\Delta_1(u)$, определяемом разбиением $D^+(u')=D^+(u)$, $D^-(u')=D^-(u)\cup D^R(u)$, где $D^R(u)$ – множество ациклических дуг в графе $\left(V,D\setminus D^+(u)\right)$, то есть оператор Δ_1 является оператором форсирования.

Для построения другого оператора форсирования нам понадобится еще одно утверждение.

Утверждение. Пусть $\Gamma=(V,D)$ - произвольный граф, $A\subseteq D$. Тогда любой дуговой разрез циклов графа Γ, не пересекающийся с A, содержит каждую дугу $(a,b)\in D\setminus A$, такую что вершина a достижима из вершины b в графе (V,A).

Доказательство. Пусть M - дуговой разрез циклов графа Γ, $M\cap A=\emptyset$, $(a,b)\notin M$ и в графе (V,A) есть путь L из вершины b в вершину a. Тогда $A'=A\cup\{(a,b)\}\subseteq D\setminus M$, и в графе A' есть цикл, состоящий из пути L и дуги (a,b), что противоречит ацикличности графа $(V,D\setminus M)$.

Это утверждение позволяет построить оператор форсирования Δ_2, отображающий множество u в его подмножество u', определяемое разбиением $D^+(u')=D^+(u)\cup D^S(u)$, $D^-(u')=D^-(u)$, где $D^S(u)=\{(a,b)\in D^0(u)$: вершина a достижима из вершины b в графе $(V,D^-(u))\}$.

Алгоритмы, необходимые для эффективной реализации операторов Δ_1 и Δ_2, списаны в п.п. 5.3 и 5.4.

Вопросы для самостоятельной работы

1. Доказать, что граф $T=(U,D)$, построенный в п. 8.1, является ориентированным деревом с корнем Y.

2. Построить деревья полного перебора для множеств всех подмножеств, всех k-элементных подмножеств, всех подмножеств мощности не более k, всех перестановок, всех разбиений, всех разбиений на k классов и всех разбиений на не более чем k классов для множества $(1,2,\ldots,n)$.

3. Доказать, что для любого ориентированного дерева его фронтальный обход с корнем, совпадающим с корнем дерева, является и обходом исчерпыванием.

*4. Описать схему решения методом динамического программирования задачи о наибольшей клике в графе Г с использованием его группы автоморфизмов.

5. Пусть $R=(X,Y,P,F)$ и $R'=(X,Y,P',F)$ - две задачи на минимум функционала F, отличающиеся только тем, что предикат задачи R' сильнее предиката задачи R: $P'=P\&P_1$. Для $u\subset Y$ обозначим через u_P и $u_{P'}$ области истинности предикатов P и P', соответственно. Доказать, что для любого $u\subset Y$ $\min\limits_{y\in u_P} F(y)\leq \min\limits_{y\in u_{P'}} F(y)$, то есть минимум функционала F по допустимым управлениям задачи R на подмножестве u является нижней оценкой функционала F в задаче R'.

*6. Построить оператор форсирования для решения методом ветвей и границ задачи о максимальной клике в графе.

*7. Для задачи о минимальном дуговом разрезе циклов (п. 8.4) рассмотрим граф $\Gamma_u^Q=\left(V,D^Q(u)\right)$, где $D^Q(u)$ - множество циклических дуг графа $\left(V,D\setminus D^+(u)\right)$. Доказать, что функционал Φ_3, определяемый формулой $\Phi_3(u)=\Phi_1(u)+w\left(\Gamma_u^Q\right)$, является нижней оценкой функционала F. Сравнить силу оценок Φ_2 и Φ_3.

ЗАКЛЮЧЕНИЕ

Вот и закончился наш краткий экскурс в увлекательный мир дискретных алгоритмов. К сожалению, краткий, так как из-за ограниченности объема курса в него не вошли многие задачи, идеи, методы и алгоритмы, имеющие прямое отношение к рассматриваемой тематике. Здесь будет сделан краткий обзор наиболее интересного и важного материала, оставшегося за пределами курса.

1. Во многих практических приложениях из-за неадекватности математической модели реальному объекту и из-за трудности определения точных значений исходных данных нахождение точного решения задачи оптимизации не имеет смысла. В этих случаях достаточно найти *приближенное* решение задачи оптимизации: такое допустимое управление, значение функционала на котором отличается от оптимального либо на константу (*аддитивное* приближение), либо в константу раз (*мультипликативное* приближение). В некоторых случаях приближенная задача оптимизации оказывается проще исходной. Например, мультипликативное приближение задачи о коммивояжере имеет полиномиальную сложность. В других случаях снижение требований на точность решения не упрощает задачу. Так, даже аддитивное приближение задачи о клике $N\mathcal{P}$-полно.

2. Зачастую в задачах выбора значения функционала принадлежат лишь частично-упорядоченному множеству. В частности, если качество управления оценивается несколькими параметрами, то вектора качества, составленные из значений этих параметров, образуют частично-упорядоченное множество с естественным отношением покоординатного мажорирования. Такого рода задачи выбора

называют задачами *многокритериальной* оптимизации. Подходы к решению таких задач, в том числе и способы сведения задач многокретериальной оптимизации к обычным оптимизационным задачам, изложены в книге О.И.Ларичева "Объективные модели и субъективные решения" - М.: Наука, 1987.

3. В задачах оптимального *структурного* синтеза возможными управлениями являются структуры синтезируемого объекта, описываемые в терминах тех или иных комбинаторных объектов (графов, систем отношений, латинских квадратов и так далее). Задачи такого рода обычно столь трудны, что могут быть решены только переборным алгоритмом: строятся все комбинаторные объекты из заданного класса, удовлетворяющие предикату допустимости и для каждого из них вычисляется значение функционала. В случае, когда на множестве комбинаторных объектов определено отношение эквивалентности, порожденное действием некоторой группы подстановок, не изменяющей значений предиката допустимости и функционала (классы эквивалентности суть орбиты этой группы), сокращение перебора может быть достигнуто путем решения задачи *конструктивного перечисления*: в заданном множестве комбинаторных объектов выделить по одному представителю каждого класса эквивалентности, элементы которого удовлетворяют предикату допустимости. Алгоритм решения задач конструктивного перечисления комбинаторных объектов и многочисленные примеры его использования содержатся в сборнике "Алгоритмические исследования в комбинаторике", - М.: Наука, 1978.

4. Теория матроидов, лишь слегка затронутая в разделе 7, имеет далеко идущие последствия для алгоритмов решения о тимизационных задач. Например, если область истинности предиката допустимости задачи оптимизации, описанной в п. 7.3, является

пересечением двух матроидов, то ее точное решение может быть найдено эффективным алгоритмом типа алгоритма чередующихся цепей. Если же область истинности предиката допустимости описывается пересечением $k>3$ матроидов, то такая задача оптимизации NP-полна, однако жадный алгоритм позволяет получить в этом случае мультипликативное приближение точного решения с мультипликативной константой, зависящей только от k.

5. В разделе 8 мы рассмотрели лишь две крайние возможности в конструировании переборных алгоритмов - динамическое программирование и метод ветвей и границ. Конечно, можно использовать и комбинации этих подходов. В частности, в рамках метода ветвей и границ можно сокращать перебор, элиминируя рассмотрение поддеревьев, корни которых эквивалентны уже рассмотренным вершинам дерева перебора. Именно такая техника используется при решении задач конструктивного перечисления. Например, при решении задачи о клике в графе достаточно рассматривать расширение лишь одной клики из каждой орбиты действия на кликах группы автоморфизмов графа. Наиболее сильные приемы такого рода разработаны при программировании игр двух противников с полной информацией (типа игры в шахматы) - частном случае задачи *многошаговой* оптимизации. Заметим, кстати, что сам метод ветвей и границ возник в игровом программировании и оттуда заимствована часть терминологии (форсирование, экспресс-оценка). С проблематикой и методами, развитыми в этой области, можно познакомиться по книге Г.М.Адельсона-Вельского, А.Л.Арлазарова и М.В.Донского "Программирование игр", - М.: Наука, 1978.

6 В последние годы в связи с появлением многопроцессорных вычислительных систем возник интерес к различным моделям параллельных вычислений, позволяющим оценивать сложность задач с

учетом возможности распараллеливания алгоритмов их решения.

Небольшой объем курса не позволил уделить должного внимания обсуждению реальных задач, возникающих в различных проблемных областях и приводящих к задачам дискретной оптимизации. Некоторым утешением является то, что массовыми источниками задач такого рода, связанных с управлением автоматизированными производствами, системами автоматизации проектирования, информационной технологией, интеллектуальными системами и математическим обеспечением вычислительных комплексов, являются многочисленные курсы, которые вам предстоит изучить в дальнейшем.

Теория и практика комбинаторных алгоритмов, начала которых изложены в этом курсе, находится в поре бурного рассвета. Несмотря на то, что в этой области работают сильные коллективы блестящих математиков во всем мире, многие нерешенные задачи в "горячих" точках допускают элементарные решения и доступны для студентов с хорошей математической и программистской подготовкой в качестве курсовых работ и тем для самостоятельной научно-исследовательской работы. Есть надежда, что некоторым из вас тематика этого курса придется настолько по душе, что они будут так или иначе обращаться к ней на протяжении своей дальнейшей научной жизни. С ними мы еще встретимся на страницах научных журналов и на многочисленных форумах по алгоритмическим аспектам дискретной математики. Итак, до встречи, коллеги!

ЛИТЕРАТУРА

1. А.Ахо, Дж.Хопкрофт, Дж.Ульман. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 536 с.

2. Э.Рейнгольд, Ю.Нивергельт, Н.Део. Комбинаторные алгоритмы. Теория и практика. М.: Мир, 1980. 476 с.

3. С.Гудман, С.Хидетниеми. Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 368 с.

4. М.Гэри, Д.Джонсон. Вычислительные машины и трудно решаемые задачи. М.: Мир, 1982. 416 с.

5. Н.Кристофидес. Теория графов. Алгоритмический подход. М.: Мир, 1978. 432 с.

6. Э.Майника. Алгоритмы оптимизации на сетях и графах. М.: Мир, 1981. 323 с.

7. М.Свами, К.Тхуласираман. Графы, сети и алгоритмы. М.: Мир, 1984. 454 с.

8. В.Липский. Комбинаторика для программистов. М.: Мир, 1988. 213 с.

9. Г.М.Адельсон-Вельский, Е.А.Диниц, А.В.Карзанов. Потоковые алгоритмы. М.: Наука, 1978. 119 с.

10. И.А.Фараджев. Алгоритмы дискретной оптимизации. Учебное пособие для практических занятий. М.: изд. МИСиС, 1987. 60 с.