Носов Валентин Александрович

ОСНОВЫ ТЕОРИИ АЛГОРИТМОВ И АНАЛИЗА ИХ СЛОЖНОСТИ

К у р с л е к ц и й

Москва 1992

Настоящее пособие возникло на основе курса лекций, читаемых слушателям, обучающимся по специальности “Прикладная математика”.

В настоящее время имеется много обстоятельных руководств по теории алгоритмов, однако данное пособие отличается тем, что в нем основное внимание уделяется той части теории алгоритмов, которая относится к изучению возможностей вычислительных машин, к сложности вычислений, к нижним оценкам сложности и оптимизации алгоритмов. Перечисленные вопросы имеют важное значение для специалистов, использующих вычислительную технику в своей практической деятельности.

Список литературы указан в конце пособия, использовалась также журнальная литература и результаты автора по сложности алгоритмов.

Автор признателен всем специалистам, прочитавшим рукопись пособия и высказавшим конструктивные замечания.

Электронная версия подготовлена к публикации на web-сервере "Интеллектуальные системы" (http://intsys.msu.ru) кафедры Математической теории интеллектуальных систем механико-математического факультета МГУ имени М.В. Ломоносова

Электронный адрес автора - vnosov@intsys.msu.ru

Оглавление

§ 1. Введение.

Понятие “алгоритм” давно является привычным не только для математиков. Оно является концептуальной основой разнообразных процессов обработки информации. Возможность автоматизации таких процессов обеспечивается наличием соответствующих алгоритмов. С алгоритмами первое знакомство происходит в нача-льной школе при изучении арифметических действий с натуральными числами. В упрощенном понимании “алго-ритм” - это то, что можно запрограммировать на ЭВМ.

Слово алгоритм содержит в своем составе преобразованное географическое название Хорезм. Термин “алгоритм” обязан своим происхождением великому ученому средневекового Востока - Муххамад ибн Муса ал-Хорезми (Магомет, сын Моисея, из Хорезма). Он жил приблизительно с 783 по 850 гг., и в 1983 году отмечалось 1200 летие со дня его рождения в городе Ургенче - областном центре современной Хорезмской области Узбекистана. В латинских переводах с арабского арифметического трактата ал-Хорезми его имя транскрибировалось как *algorismi*. Откуда и пошло слово “алгоритм” - сначала для обозначения алгоритмов цифровых вычислений десятичной позиционной арифметики, а затем для обозначения произвольных процессов, в которых искомые величины решаемых задач находятся последовательно из исходных данных по определенным правилам и инструкциям.

Вплоть до 30-х годов нашего столетия понятие алгоритма оставалось интуитивно понятным, имевшем скорее методологическое, а не математическое значение. Так, к началу XX в. много ярких примеров дала алгебра и теория чисел. Среди них упомянем алгоритм Евклида нахождения наибольшего общего делителя двух натуральных чисел или двух целочисленных многочленов,алгоритм Гаусса решения системы линейных уравнений над полем, алгоритм нахождения рациональных корней многочленов одного переменного с рациональными коэффициентами, алгоритм Штурма определения числа действительных корней многочлена с действительными коэффициентами на некотором отрезке действительных чисел, алгоритм разложения многочлена одного переменного над конечным полем на неприводимые множители. Указанные алгоритмические проблемы решены путем указания конкретных разрешающих процедур. Для получения результатов такого типа достаточно интуитивного понятия алгоритма. Однако, в начале XX в. были сформулированы алгоритмические проблемы, положительное решение которых представлялось маловероятным. Решение таких проблем потребовало привлечения новых логических средств. Ведь одно дело доказать существование разрешающего алгоритма - это можно сделать, используя интуитивное понятие алгоритма. Другое дело - доказать несуществование алгоритма - для этого нужно знать точно - что такое алгоритм.

Задача точного определения понятия алгоритма была решена в 30-х годах в работах Гильберта, Черча, Клини, Поста, Тьюринга в двух формах: на основе понятия рекурсивной функции и на основе описания алгоритмического

процесса. Рекурсивная функция это функция, для которой существует алгоритм вычисления ее значений по произвольному значению аргумента. Класс рекурсивных функций был определен строго как конкретный класс функций в некоторой формальной системе. Был сформулирован тезис (называемый “тезис Черча”), утверждающий, что данный класс функций совпадает с множеством функций, для которых имеется алгоритм вычисления значений по значению аргументов. Другой подход заключался в том, что алгоритмический процесс определяется как процесс, осуществимый на конкретно устроенной машине (называемой “машиной Тьюринга”). Был сформулирован тезис (называемый “тезис Тьюринга”), утверждающий, что любой алгоритм может быть реализован на подходящей машине Тьюринга. Оба данных подхода, а также другие подходы (Марков, Пост) привели к одному и тому же классу алгоритмически вычислимых функций и подтвердили целесообразность использования тезиса Черча или тезиса Тьюринга для решения алгоритмических проблем. Поскольку понятие рекурсивной функции строгое, то с помощью обычной математической техники можно доказать, что решающая некоторую задачу функция не является рекурсивной, что эквивалентно отсутствию для данной задачи разрешающего алгоритма. Аналогично, несуществование разрешающей машины Тьюринга для некоторой задачи равносильно равносильно отсутствию для нее разрешающего алгоритма. Указанные результаты составляют основу так называемой дескриптивной теории алгоритмов, основным содержанием которой является классификация задач по признаку алгоритмической разрешимости, т.е. получение высказываний типа “Задача **П** алгоритмически разрешима” или “Задача **П** алгоритмически неразрешима”. В данном направлении получен ряд фундаментальных результатов. Среди них отрицательное решение Новиковым П.С. в 1952 году классической проблемы тождества для конечно определенных групп, сформулированной Деном в 1912 году. Знаменитая десятая проблема Гильберта, сформулированная им в 1900 году (среди других 23 проблем) формулируется так : “10. Задача о разрешимости диофантова уравнения. Пусть задано диофантово уравнение с произвольными неизвестными и целыми рациональными числовыми коэффициентами. Указать способ, при помощи которого возможно после конечного числа операций установить, разрешимо ли это уравнение в целых рациональных числах”. Алгоритмическая неразрешимость 10-й проблемы Гильберта была доказана в 1970 году Мятиясевичем Ю.В.

В настоящее время теория алгоритмов образует теоретический фундамент вычислительных наук. Применение теории алгоритмов осуществляется как в использовании самих результатов (особенно это касается использования разработанных алгоритмов), так и в обнаружении новых понятий и уточнении старых. С ее помощью проясняются такие понятия как доказуемость, эффективность, разрешимость, перечислимость и другие.

В технику термин “алгоритм” пришел вместе с кибернетикой. Понятие алгоритма помогло, например, точно определить, что значит эффективно задать последовательность управляющих сигналов. Применение ЭВМ послужило стимулом развитию теории алгоритмов и изучению алгоритмических моделей, к

самостоятельному изучению алгоритмов с целью их сравнения по рабочим характеристикам (числу действий, расходу памяти), а также их оптимизации. Возникло важное направление в теории алгоритмов - сложность алгоритмов и вычислений. Начала складываться так называемая метрическая теория алгоритмов, основным содержанием которой является классификация задач по классам сложности. Сами алгоритмы стали объектом точного исследования как и те объекты, для работы с которыми они предназначены. В этой области естественно выделяются задачи получения верхних и задачи получения нижних оценок сложности алгоритмов, и методы решения этих задач совершенно различны. Для получения верхних оценок достаточно интуитивного понятия алгоритма. Для этого строится неформальный алгоритм решения конкретной задачи и затем он формализуется для реализации на подходящей алгоритмической модели. Если показывается, что сложность (время или память) вычисления для этого алгоритма не превосходит значения подходящей функции при всех значениях аргумента, то эта функция объявляется верхней оценкой сложности решения рассматриваемой задачи. В области получения верхних оценок получено много ярких результатов для конкретных задач. Среди них разработаны быстрые алгоритмы умножения целых чисел, многочленов, матриц, решения линейных систем уравнений, которые требуют значительно меньше ресурсов, чем традиционные алгоритмы.

Установить нижнюю оценку - значит доказать, что никакой алгоритм вычисления не имеет сложности меньшей, чем заданная граница. Для получения результатов такого типа необходима точная фиксация рассматриваемой алгоритмической модели, и такие результаты получены только в очень жестких вычислительных моделях. В связи с этим получила развитие проблематика получения “относительных” нижних оценок, так называемая теория **NP**-полноты, связанная с труднорешаемостью переборных задач.

Рассмотрим неформально, что именно в интуитивном понятии алгоритма нуждается в уточнении.

Основные требования к алгоритмам.

1. Каждый алгоритм имеет дело с данными - входными, промежуточными, выходными. Для того, чтобы уточнить понятие данных, фиксируется конечный алфавит исходных символов (цифры, буквы и т.п.) и указываются правила построения алгоритмических объектов. Типичным используемым средством является индуктивное построение. Например, определение идентификатора в АЛГОЛЕ: идентификатор - это либо буква, либо идентификатор, к которому приписана справа либо буква, либо цифра. Слова конечной длины в конечных алфавитах - наиболее обычный тип алгоритмических данных, а число символов в слове - естественная мера объема данных. Другой случай алгоритмических объектов - формулы. Примером могут служить формулы алгебры предикатов и алгебры высказываний. В этом случае не каждое слово в алфавите будет формулой.

2. Алгоритм для размещения данных требует памяти. Память обычно считается однородной и дискретной, т.е. она состоит из одинаковых ячеек,

причем каждая ячейка может содержать один символ данных, что позволяет согласовать единицы измерения объема данных и памяти.

3. Алгоритм состоит из отдельных элементарных шагов, причем множество различных шагов, из которых составлен алгоритм, конечно. Типичный пример множества элементарных шагов - система команд ЭВМ.

4. Последовательность шагов алгоритма детерминированна, т.е. после каждого шага указывается, какой шаг следует выполнять дальше, либо указывается, когда следует работу алгоритма считать законченной.

5. Алгоритм должен обладать результативностью, т.е. останавливаться после конечного числа шагов (зависящего от исходных данных) с выдачей результата. Данное свойство иногда называют сходимостью алгоритма.

6. Алгоритм предполагает наличие механизма реализации, который по описанию алгоритма порождает процесс вычисления на основе исходных данных. Предполагается, что описание алгоритма и механизм его реализации конечны.

Можно заметить аналогию с вычислительными машинами. Требование 1 соответствует цифровой природе ЭВМ, требование 2 - памяти ЭВМ, требование 3 - программе машины, требование 4 - ее логической природе, требования 5, 6 - вычислительному устройству и его возможностям.

Имеются также некоторые черты неформального понятия алгоритма, относительно которых не достигнуто окончательного соглашения. Эти черты сформулируем в виде вопросов и ответов.

7.Следует ли фиксировать конечную границу для размера входных данных ?

8.Следует ли фиксировать конечную границу для числа элементарных шагов ?

9.Следует ли фиксировать конечную границу для размера памяти ?

10.Следует ли ограничить число шагов вычисления ?

На все эти вопросы далее принимается ответ “НЕТ”, хотя возможны и другие варианты ответов, поскольку у физически существующих ЭВМ соответствующие размеры ограничены. Однако теория, изучающая алгоритмические вычисления, осуществимые в принципе, не должна считаться с такого рода ограничениями, поскольку они преодолимы по крайней мере в принципе (например, вообще говоря, любой фиксированный размер памяти всегда можно увеличить на одну ячейку).

Таким образом, уточнение понятия алгоритма связано с уточнением алфавита данных и формы их представления, памяти и размещения в ней данных, элементарных шагов алгоритма и механизма реализации алгоритма. Однако эти понятия сами нуждаются в уточнении. Ясно, что их словесные определения потребуют введения новых понятий, для которых в свою очередь, снова потребуются уточнения и т.д. Поэтому в теории алгоритмов принят другой подход, основанный на конкретной алгоритмической модели, в которой все сформулированные требования выполняются очевидным образом. При этом используемые алгоритмические модели универсальны, т.е. моделируют любые другие разумные алгоритмические модели, что позволяет снять возможное

возражение против такого подхода: не приводит ли жесткая фиксация алгоритмической модели к потере общности формализации алгоритма ? Поэтому данные алгоритмические модели отождествляются с формальным понятием алгоритма. В дальнейшем будут рассмотрены основные типы алгоритмических моделей, различающиеся исходными трактовками, что такое алгоритм.

Первый тип трактует алгоритм как некоторое детерминированное устройство, способное выполнять в каждый момент лишь строго фиксированное множество операций. Основной теоретической моделью такого типа является машина Тьюринга, предложенная им в 30-х годах и оказавшая существенное влияние на понимание логической природы разрабатываемых ЭВМ. Другой теоретической моделью данного типа является машина произвольного доступа (МПД) - введенная достаточно недавно (в 70-х годах) с целью моделирования реальных вычислительных машин и получения оценок сложности вычислений.

Второй тип связывает понятие алгоритма с традиционным представлением - процедурами вычисления значений числовых функций. Основной теоретической моделью этого типа являются рекурсивные функции - исторически первая формализация понятия алгоритма.

Третий тип алгоритмических моделей - это преобразования слов в произвольных алфавитах, в которых операциями являются замены кусков слов другим словом. Основной теоретической моделью этого типа являются нормальные алгоритмы Маркова.

Теория алгоритмов оказала существенное влияние на развитие ЭВМ и практику программирования. В теории алгоритмов были предугаданы основные концепции, заложенные в аппаратуру и языки программирования ЭВМ. Упоминаемые выше главные алгоритмические модели математически эквивалентны; но на практике они существенно различаются сложностными эффектами, возникающими при реализации алгоритмов, и породили разные направления в программировании. Так, микропрограммирование строится на идеях машин Тьюринга, структурное программирование заимствовало свои конструкции из теории рекурсивных функций, языки символьной обработки информации (РЕФАЛ, ПРОЛОГ) берут начало от нормальных алгоритмов Маркова и систем Поста.

Авторы обзора [17] основных достижений теории алгоритмов на стр.230 пишут:

“Алгоритмические концепции играют в процессе обучения и воспитания современного человека фундаментальную роль, сравнимую лишь с ролью письменности”.

§ 2 Машина Тьюринга и функции, вычислимые по Тьюрингу.

1^0) Опишем алгоритмическую модель, предложенную А.Тьюрингом в ЗО-х годах и оказавшую влияние на разработку ЭВМ.

Машина Тьюринга состоит из следующих элементов:

1) Ленты , разбитой на ячейки и бесконечной в обе стороны .В каждой ячейке может быть записан один из символов конечного алфавита A =$\{a_0, a_1, \ldots, a_m\}$, называемого внешним алфавитом Условимся считать,что символ a_0 является пустым символом (также обозначаемым λ).

2)Управляющего устройства ,которое может находиться в одном из конечного числа внутренних состояний Q = $\{q_0, q_1, \ldots, q_n\}$. Число элементов в Q характеризует объем внутренней памяти машины. В множестве Q выделены два специальные состояния : q_0 и q_1 , называемые заключительным и начальным состояниями соответственно. Машина начинает работу в состоянии q_1 ,попав в состояние q_0 , машина всегда останавливается.

3) Считывающей\пишущей головки, которая может перемещаться вдоль ленты и в каждый момент времени обозревает (считывает) одну из ячеек ленты.

Функционирование машины Тьюринга осуществляется в дискретные моменты времени t=0,1,2,... и заключается в следующем. В зависимости от внутреннего состояния машины и считываемого символа на ленте машина Тьюринга:

а) записывает в эту ячейку символ внешнего алфавита;

б) сдвигает считывающую головку на один шаг влево или один шаг вправо или оставляет ее на месте,

в) переходит в новое внутреннее состояние.

Таким образом, работа машины определяется системой команд вида

$$q_i a_j \longrightarrow q_k a_l d \qquad (1)$$

где q_i - внутреннее состояние машины, a_j - считываемый символ, q_k - новое внутреннее состояние a_l - новый записываемый символ, d - направление движения головки), обозначаемое одним из символов L (влево),R (вправо),E (на месте).

Предполагается, что для каждой пары $q_i a_j$, где i=1..n , j=0..m имеется точно одна команда вида (1). Множество этих команд называется программой машины и, значит, в программе имеется n(m+1) команд.

Работа машины заключается в изменении конфигураций. Конфигурация представляет собой совокупность внутреннего состояния, состояния ленты (т.е. размещения букв внешнего алфавита по ячейкам или слова, записанного на ленте), положения головки на ленте.

Предположим, что в начальный момент времени на ленте все ячейки, кроме конечного их числа, содержат пустой символ. Следовательно и в любой другой момент времени лента будет иметь лишь конечное число ячеек, содержащих непустые символы.

Активной зоной конфигурации назовем минимальную связную часть ленты, содержащую обозреваемую ячейку, а также все ячейки, в которых записаны непустые символы.

Конфигурацию можно представить в виде машинного слова в алфавите $A \bigcup Q$ вида

$$\alpha_1 q_i \alpha_2 \qquad (2)$$

где q_i - внутреннее состояние , α_1- слово из символов алфавита А ,находящееся в левой части активной зоны от считывающей головки,Конфигурация К называется заключительной , если $q_i = q_1$.Условимся. что стандартная начальная конфигурация имеет вид $q_1\alpha$, а стандартная заключительная имеет вид $q_0\alpha$.Конфигурацию в момент времени t обозначим K_t.Машина реализует процесс изменения конфигураций в следующем смысле. Если $K_0 = q_i\alpha$ и $\alpha = a_i\alpha'$,$a_i \in A$,то в программе машины имеется точно одна команда вида $q_1 a_i \longrightarrow q_k a_l d$.Тогда следующая конфигурация K_1 определяется так :

$K_1 = q_k a_l \alpha'$,если d=E

$K_1 = a_l q_k \alpha'$,если d=R

$K_1 = q_k a_0 a_l \alpha'$,если d=L

Это обстоятельство записываем в виде: $K_0 \to K_1$ Если теперь конфигурация K_1 , не является заключительной, то в соответствии с системой команд, аналогично предыдущему, определима однозначно следующая конфигурация K_2 т.е. $K_1 \longrightarrow K_2$.Таким образом начальная конфигурация K_0 порождает последовательность конфигураций

$$K_0 \to K_1 \to K_2 \to \ldots \to K_t \to K_{t+1} \to \ldots \qquad (3)$$

Если последовательность (3) конечна , т.е. обрывается в заключительной конфигурации ,то говорят , что машина применима к конфигурации K_0, в противном случае неприменима к K_0.

Если машина применима к конфигурации $K_0 = q_1\alpha$ и $K_t = \alpha' q_0 \alpha''$ - заключительной конфигурация ,то слово $\beta = \alpha'\alpha''$ об`является результатом работы машины на слове α. Т.о. машине Тьюринга соответствует частичная словарная функция с областью определения и областью значения , являющейся конечными словами в алфавите А , которая каждому такому слову ставит в соответствие результат применения машины к данному слову.

Потребуем , чтобы заключительные конфигурации машины находились в стандартной форме. Этого всегда можно добиться , добавляя к машине Т два новых состояния q', q'' и команды

$$q_0 a_i \to q' a_i L \quad i=0...m$$
$$q' a_0 \to q'' a_0 R$$

При этом состояние q'' объявим заключительным. Полученная машина T' эквивалентна машине Т в следующем смысле:

а) Обе машины применимы к одним и тем же начальным конфигурациям.

б) Результаты применения обеих машин совпадают.

в) Заключительные конфигурации у машины T' находятся в стандартной форме.

Определим теперь вычисление функций на машине Тьюринга. Будем рассматривать словарные частичные функции f типа f: $A^* \to A^*$ где А* - множество всех слов конечной длины в алфавите А

Говорят, что машина Тьюринга Т правильно вычисляет частичную функцию f , если для любого $p \in$А* выполнено:

1) Если $f(P)$ определено и $f(P)$=Q, то машина Т применима к начальной конфигурации $q_1 P$ и заключительной конфигурацией является $K_t = q_0 Q$.

2) Если $f(P)$ не определено, то машина Т неприменима к начальной конфигурации $q_1 P$.

Функция f называется правильно вычислимой по Тьюрингу, если существует машина Тьюринга Т , которая ее правильно вычисляет.

Аналогичные определения могут быть сделаны и для функций нескольких переменных. Для этого достаточно множество слов, являющихся аргументами, записать в виде одного слова, введя знак-разделитель. Ограничимся соответствующим определением для числовых функций. Рассмотрим частичную функцию $f(x_1,...,x_n)$ от n переменных, аргументы которой и ее значения принадлежат множеству N_0={0,1,2,...}. Будем считать, что алфавит А машины Т содержит элемент 1 . Условимся произвольное число $x \in N_0$ представлять в виде слов $|^{x+1} = \underbrace{|...|}_{x+1}$1...1, чтобы запись нуля была непустой. Будем говорить, что машина Тьюринга Т правильно вычисляет функцию $f(x_1,...,x_n)$, если

конфигурацию $|q_1|\underbrace{.|}_{x_1+1}*|\underbrace{.|}_{x_2+1}*\ldots*|\underbrace{.|}_{x_n+1}$ она переводит в конфигурацию $q_0\ |\underbrace{.|}_{f(x_1,\ldots x_n)+1}$,если значение $f(x_1,\ldots,x_n)$ определено , и Т неприменима, если значение $f(x_1,\ldots,x_n)$ неопределено. Здесь * - символ-разделитель из А. Класс функций, вычислимых по Тьюрингу обозначим через Т.

2^0) Рассмотрим несколько примеров на построение машин Тьюринга.

1) Пусть A=$\{a_0=\lambda, a_1=1\}$ и Q={q_0,q_1,q_2}

2) Программа машины T_1 :

$$q_1\lambda \to q_2 1E$$
$$q_1 1 \to q_1 1R \qquad (4)$$
$$q_2 1 \to q_2 1L$$
$$q_2\lambda \to q_0\lambda R$$

Пусть $K_0 = q_1 1^{x+1}$.Тогда T_1 порождает следующую последовательность конфигураций :

$$q_1 1^{x+1} \to 1q_1 1^x \to \ldots \to 1^{x+1} q_1\lambda \to 1^{x+1} q_2 1 \to \ldots \to q_2 1^{x+2} \to q_2\lambda 1^{x+2} \to q$$

Таким образом , машина Тьюринга (4) правильно вычисляет функцию

$f(x) = x+1$,$x \in N_0$.

2) Пусть A={λ,1}, Q={q_0,q_1,q_2}

Программа машины T_2 :

$$q_1\lambda \to q_1\lambda R$$
$$q_1\ 1 \to q_2\lambda R$$
$$q_2 1 \to q_2\lambda R \qquad (5)$$
$$q_2\lambda \to q_0 1E$$

Пусть $K_0 = q_1 1^{x+1}$. Тогда T_2 порождает следующую последовательность конфигураций:

$$q_1 1^{x+1} \to q_2 1^x \to \ldots \to q_2\lambda \to q_0 1$$

Значит, машина Тьюринга (5) правильно вычисляет функцию

$f(x) = 0, x \in N_0$

3) Пусть A={λ,1,*}, Q={q_0,q_1,q_2,q_3}

Программа машины T_3 :

$$q_1 1 \to q_2\lambda R$$

$$q_2 1 \to q_2 1R$$
$$q_2 * \to q_2 1R$$
$$q_2 \lambda \to q_3 \lambda L$$
$$q_3 1 \to q_4 \lambda L$$
$$q_4 \lambda \to q_0 \lambda R$$
$$q_4 1 \to q_4 1L$$

(остальные команды не важны для вычисления).

Пусть $K_0 = q_1 1^{x+1} * 1^{y+1}$. Тогда машина T_3 порождает следующую последовательность конфигураций:

$$q_1 1^{x+1} * 1^{y+1} \to q_2 1^x * 1^{y+1} \to ... \to 1^x q_2 * 1^{y+1} \to 1^{x+1} q_2 1^{y+1} \to 1^{x+y+2} q_2$$
$$\to 1^{x+y+1} q_3 1 \to 1^{x+y} q_4 1 \to ... \to q_4 \lambda 1^{x+y+1} \to q_0 1^{x+y+1}$$

Т.е. машина T_3 правильно вычисляет функцию $f(x) = x + y \;\; x, y \in N_0$

3^0) Прямое построение машин Тьюринга для решения даже простых задач может оказаться затруднительным. Однако существуют приемы, которые облегчают данный процесс, если использовать способы сочетания программ нескольких машин в результирующие программы. Дадим некоторое представление об этих приемах, что позволит говорить о существовании тех или иных машин, на деталях же построения конкретных программ останавливаться не будем.

1) Суперпозиция машин. Пусть даны две машины Тьюринга T_1 и T_2 , которые вычисляют соответственно словарные функции $f_1(P)$ и $f_2(P)$ в одном и том же алфавите. Тогда существует машина Тьюринга T , которая вычисляет функцию $f(P) = f_2(f_1(P))$.При этом для любого слова P функция $f(P)$ определена в том и только в том случае, когда $f_1(P)$ определена и $f_2(f_1(P))$ определена. Программа машины T строится так: Состояния машины T_2 переобозначаем так, чтобы они отличались от состояний машины T_1. Начальное состояние q_1^1 Машины T_1 объявляем начальным q_1 для машины T, заключительное состояние ${q_0}^2$ машины T_2 объявляем заключительным q_0 для машины T. Заключительное состояние ${q_0}^1$ машины T_1 отождествляем с начальным состоянием ${q_1}^2$ машины T_2. Полученные команды для обеих машин объединяем в одну программу. Рассмотрим начальную конфигурацию $q_1 P$ Поскольку $q_1 = q_1^1$ - начальное состояние машины T_1 , то вначале T работает как машина T_1 и если T_1 применима

к $q_1^1 P$, то на некотором шаге будет получена конфигурация $q_0^1 f_1(P)$,но $q_0^1 = q_1^2$- начальное состояние для T_2 и теперь Т действует как машина T_2. Если T_2 применима к $q_0^1 f_1(P)$, то на некотором шаге будет получена конфигурация $q_0^2 f_2(f_1(P))$, которая является заключительной для Т , т.к $q_0^2 = q_2$. Если T_1 неприменима к $q_1^1 P$ или T_2 неприменима к $q_0^1 f_1(P)$, то Т неприменима к $q_1 P$. Машина Т называется суперпозицией машин T_1 и T_2 и обозначается T_1T_2. Схематически суперпозиция изображается так:

$$P \xrightarrow[T_1]{} f_1(P) \xrightarrow[T_2]{} f_1(f_2(P))$$

Соединение машин. Пусть даны машины Тьюринга T_1 и T_2 , вычисляющие словарные фунции $f_1(P)$ и $f_2(P)$ соответственно. Тогда существует машина Т , которая начальную конфигурацию $q_1 P$ переводит в заключительную $q_0 f_1(P) * f_2(P)$,если $f_1(P)$ и $f_2(P)$ определены , и неприменима в противном случае.

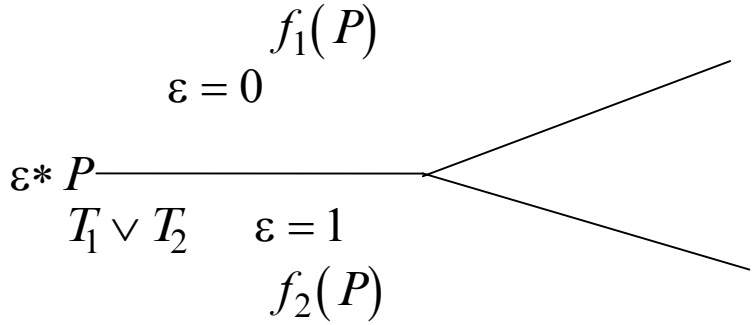
Здесь * - новый символ, не входящий в алфавиты машин T_1 и T_2 . Машина Т называется соединением машин и обозначается T_1*T_2. Существование машины Т вытекает из следующих неформально описываемых конструкций. Лента машины Т- является двухэтажной. В качестве внешнего алфавита Т берутся двухэтажные буквы $\begin{pmatrix} b \\ a \end{pmatrix}$, где a и b - буквы алфавита T_1, T_2 .Каждой букве a ставится в соответствие двухэтажная буква $\begin{pmatrix} \lambda \\ a \end{pmatrix}$. Слову $P = a_{i_1} \ldots a_{i_k}$ ставится в соответствие двухэтажное слово $\begin{pmatrix} \lambda \ldots\ldots \lambda \\ a_{i_1} \ldots a_{i_k} \end{pmatrix}$. Машина Т будет работать так (существование машин Тьюринга для реализации каждого шага ясно)

$$\begin{pmatrix} \lambda \ldots\ldots \lambda \\ a_{i_1} \ldots a_{i_k} \end{pmatrix} \to \begin{pmatrix} a_{i_1} \ldots a_{i_k} \\ a_{i_1} \ldots a_{i_k} \end{pmatrix} \to \begin{pmatrix} a_{i_1} \ldots a_{i_k} \\ f_1(P) \end{pmatrix} \to \begin{pmatrix} f_1(P) \\ f_2(P) \end{pmatrix} \to \begin{pmatrix} \lambda \ldots \lambda \ldots \lambda \\ f_1(P) * f_2(P) \end{pmatrix}$$

3) Ветвление машин. Пусть даны машины Тьюринга T_1 и T_2 вычисляющие словарные функции $f_1(P)$ и $f_2(P)$ соответственно, заданные в одном алфавите. Тогда существует машина Тьюринга Т , которая начальную

конфигурацию $q_1\varepsilon * P$,где $\varepsilon \in \{0,1\}$, переводит в заключительную $q_0 f_1(P)$, если $\varepsilon = 0$ и в $q_0 f_2(P)$, если $\varepsilon = 1$.

Машина Т называется разветвлением машин T_1 и T_2 и обозначается $T_1 \vee T_2$. Схематически разветвление представляется так:



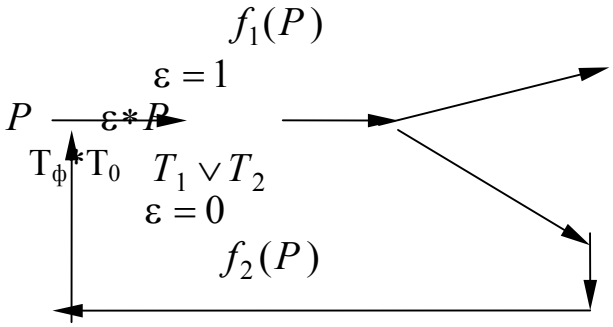
Существование машины Т вытекает из следующих конструкций. Пусть q_1^1 и q_1^2 - начальные состояния машин T_1 и T_2 соответственно. Считаем, что множества внутренних состояний машин не пересекаются. Объединим программы машин T_1 , T_2 , добавим новое начальное состояние q_1 и добавим команды

$$q_1 0 \to q_1^1 \lambda \; R$$
$$q_1^1 * \to q_1^1 \lambda \; R$$
$$q_1 1 \to q_1^2 \lambda \; R$$
$$q_1^2 * \to q_1^2 \lambda \; R$$

Теперь заключительные состояния q_0^1 и q_0^2 машин T_1 и T_2 объединим, а полученное состояние q_0 считаем заключительным для Т. Если $q_1 \varepsilon * P$ - начальная конфигурация, то Т через 2 шага перейдет в конфигурацию q_1^1P , если $\varepsilon = 0$ и в конфигурацию q_1^2P , если $\varepsilon = 1$ а затем будет работать как T_1 или T_2 соответственно.

4) Реализация цикла. Важным приемом в программировании является разбиение решаемой задачи на циклы. После выполнения каждого цикла проверяется выполнимость некоторого условия. Если условие выполнено, то выдается результат, если нет, то цикл повторяется. Точнее, процедура задается так. Пусть имеем словарные функции f_1 и f_2 и некоторый предикат Ф на словах (его значения обозначим 0,1). Для произвольного слова Р проверяемся - верно ли Ф(Р)=1 , если да, то выдается ответ $f_1(P)$. Если Ф(Р)=0 , то вычисляется $P' = f_2(P)$. Затем проверяется-верно ли Ф(P')=1, если да, то ведается ответ $f_1(P')$, если Ф(P')=0 , то вычисляется $P'' = f_2(P')$ и т.д, Существует машина Тьюринга Т , реализующая данную процедуру. Пусть существуют машины Тьюринга для вычисления функций f_1 , f_2 и предиката Ф. Обозначим их T_1 ,

$T_2, T_ф$ соответственно. Пусть T_0-машина, которая оставляет всякое слово P без изменения.Машина T строится в соответствии со схемой:



Пояснение: Заключительные состояния q_0^1 и q_0^2 машин T_1 и T_2 не объединяются, а считаются различными. Состояние q_0^1 объявляется заключительным для T_1 , а q_0^2 отождествляется с начальным состоянием q_1 для T. Заключительное состояние для машины $T_ф * T_0$ объявляется начальным для $T_1 \vee T_2$.Из изложенного следует, что если $T_1 \vee T_2$ работает как T_1, то полученное ею значение является выходом T , если же $T_1 \vee T_2$ работает как T_2 , то полученное ею значение снова подается на вход машины T.

Используя данный прием, можно построить машину Тьюринга для перевода произвольного числа n>0 , заданного в унарной записи, т.е. в виде 1^{n+1}(n+1 палочка) в его двоичную запись, начинающуюся с 1.
Входной алфавит машины есть $\{\wedge,1,1,o)$. Вначале стирается одна палочка, затем работа осуществляется t циклами (1<=t<=n). К концу каждого цикла t машина находится в конфигурации
$q1\alpha_2 ... \alpha_t 11...1$,где $1\alpha_1 ... \alpha_t$ двоичная запись числа t,111...11 n-t палочек.

В течение цикла t стирается одна палочка и к числу t-1 в двоичной записи прибавляется 1 .Формальное описание машины Тьюринга, однако, требует большого числа технических деталей и поэтому мы их опускаем.

Таким образом, язык Тьюрингова программирования содержит основные операторы программирования на алгоритмических языках и позволяет устраивать последовательное выполнение программ, параллельное их соединение, использовать условные переходы ("если Ф , выполнить f_1, иначе f_2 "), реализовывать цикл (“ пока Ф , выполнить f_1, иначе f_2 "). Это является основанием для предположения о том, что для всех процедур, претендующих называться алгоритмическими, существует (при подходящем кодировании) реализующая их машина Тьюринга.Даннное предположение носит название тезиса Тьюринга. Данный тезис доказать нельзя, поскольку здесь используется интуитивное понятие алгоритма. Подтверждением тезису является

математическая практика, а также то, что описание алгоритма в любой другой алгоритмической модели может быть сведено к описанию его в виде машины Тьюринга. Однако принятие тезиса Тьюринга позволяет истолковывать утверждения о несуществовании машин Тьюринга для решения конкретных задач как утверждения о несуществовании алгоритмов вообще.

§ 3 Машины произвольного доступа и вычислимые функции

1^0). Опишем другую алгоритмическую модель, представляющую собой идеализированную ЭВМ и предложенную в 70-х годах с целью моделирования реальных вычислительных машин и анализа сложности вычислений.

Машина произвольного доступа (МПД) состоит из бесконечного числа регистров $R_1, R_2, R_3, \ldots$, в каждом из которых может быть записано натуральное число из N_0. Пусть r_n есть число, записанное в регистре R_n, $n \in N$. Состоянием машины или конфигурацией назовем последовательность чисел ($r_1, r_2, r_3, \ldots$). Функционирование машины заключается в изменении конфигураций путем выполнения команд в порядке их написания.

Машина имеет следующие типы команд:

1) Команды обнуления. Для всякого $n \in N$ имеется команда Z(n) .Действие команды Z(n) заключается в замене содержимого регистра R_n на 0. Содержимое других регистров не меняется. Обозначение действия Z(n)- $r_n := 0$.

2) Команды прибавления единицы. Для всякого $n \in N$ имеется команда S(n). Действие команды S(n) заключается в увеличении содержимого регистра R_n. на 1. Содержимое других регистров не меняется. Обозначение действия S(n) - $r_n := r_n + 1$

3) Команды переадресации. Для всех m, $n \in N$ имеется команда T(m,n). Действие команды T(m,n) заключается в замене содержимого регистра R_n числом r_n - хранящимся в регистре R_m. Содержимое других регистров не меняется (включая и R_m) .Обозначение действия

T(m,n)- $r_n := r_m$ или $r_m \to R_n$.

4) Команды условного перехода. Для всяких $m,n,q \in N$ имеется команда $J(m,n,q)$. Действие этой команды заключается в следующем:

а) Сравнивается содержимое регистров R_m и R_n ,затем :

б) Если $r_n = r_m$, то МПД переходит к выполнению команды с номером (идентификатором) q в списке команд.

в) Если $r_m \neq r_n$, то МПД переходит к выполнению следующей команды в списке команд .

Конечная, упорядоченная последовательность команд данных типов составляет програму МПД.

Пусть зафиксирована начальная конфигурация $K_0 = (a_1, a_2, \ldots)$ чисел и программа $P = I_1 I_2 \ldots I_s$. Тогда однозначно определена последовательность конфигугураций

$$K_0, K_1, K_2, \ldots, K_t, K_{t+1}, \ldots \qquad (1)$$

где K_1 есть конфигурация, полученная из конфигурации K_0 применением команды I_1. Пусть на некотором шаге выполнена команда I_t и получена конфигурация K_t. Тогда, если I_t не есть команда условного перехода,

следующая конфигурация K_{t+1} есть конфигурация, полученная из K_t применением команды I_{t+1}. Если I_t есть команда условного перехода, т.е $I_t = J(m,n,q)$, то K_{t+1} получается из K_t применением команды I_q, если $r_n = r_m$ в конфигурации K_t и команды I_{t+1}, если $r_m \neq r_n$. Последовательность (1) будет обозначаться также $P(a_1, a_2, ...)$ или $P(K_0)$ и называться вычислением.

Вычисление (работа машины) останавли -вается, если:

а) Выполнена последняя команда, т.е.t=s и I_t не есть команда условного перехода.

б) Если $I_t = J(m,n,q)$, $r_n = r_m$ - в конфигурации K_t и $q > s$.

в) Если $I_t = J(m,n,q)$, $r_m \neq r_n$ в конфигурации K_t и t=s.

Если вычисление остановилось) то последова -тельность $(r_1, r_2, ...)$ содержимого регистров $R_1, R_2, ...$ называется заключительной конфигурацией. Если последовательность (1) конечна, то говорим ,что МПД применима к начальной конфигурации $K_0 = (a_1, a_2, ...)$ и пишем $P(a_1, a_2, ...)\downarrow$ или $P(K_0)\downarrow$.

В противном случае говорим, что МПД неприменима к начальной конфигурации $K_0 = (a_1, a_2, ...)$ и пишем $P(a_1, a_2, ...)\uparrow$ или $P(K_0)\uparrow$.

Будем рассматривать только такие начальные конфигурации, в которых имеется конечное число элементов, отличных от нуля. Будем писать $(a_1, a_2, ..., a_n)$ вместо $(a_1, a_2, ..., a_n, 0, ...)$ для таких конфигураций. Ясно, что для любого t конфигурация K_t будет содержать конечное числе отличных от нуля элементов, если этим свойством обладает конфигурация K_0.

Теперь условимся, что понимать под вычислением функций на МПД. Рассматриваем частичные f типа $f: N_0^n \to N_0$. Пусть $P = I_1 I_2 ... I_s$ - фиксированная программа. Пусть $a_1, ..., a_n, b \in N_0$. Будем говорить, что вычисление дает результат b , если $P(a_1, a_2, ...)\downarrow$ и в заключительной конфигурации r_1=b.Обозначение : $P(a_1, a_2, ...)\downarrow b$.

Будем говорить, что программа P вычисляет функцию f, если $\forall a_1, ..., a_n, b \in N_0$ выполнимо $P(a_1, a_2, ...)\downarrow b \Leftrightarrow \{f$ определена на $(a_1, a_2, ..., a_n)$ и $f(a_1, a_2, ..., a_n) = b$.

Назовем функцию f вычислимой (на МПД), если существует программа P , которая вычисляет f. Класс вычислимых функций обозначим **E.**

Заметим, что любая программа P для любого $n \geq 1$ на начальных конфигурациях вида $K_0 = (a_1, a_2, ..., a_n, 0, ...)$ определяет n -местную частичную функцию $f_p^n(x_1, ..., x_n)$ такую, что $\forall a_1, ..., a_n \in N_0$

$$f_p^n(a_1, ..., a_n) = \begin{cases} b \quad , если\ P(a_1, ..., a_n)\downarrow \quad и \quad P(a_1, ..., a_n)\downarrow b \\ неопределена \quad , \quad если \quad P(a_1, ..., a_n)\uparrow \end{cases}$$

Ясно, что разные программы могут вычислять одну и ту же функцию.

2^0) Распространим понятие алгоритмической вычисли-мости на предикаты, заданные на множестве N_0^n. Пусть $\pi(x_1, ..., x_n)$ произвольный такой предикат. Определим характеристическую функцию χ_π предиката π соотношением:

$$\chi_\pi(x_1, ..., x_n) = \begin{cases} 1 \quad , если\ \pi(x_1, ..., x_n) = и \quad (истина) \\ 0 \quad , если \quad \pi(x_1, ..., x_n) = л \quad (ложь) \end{cases}$$

Будем называть предикат π разрешимым, если его характеристическая функция вычислима, и неразрешиным в противном случае.

Это понятие соответствует вопросу о наличии алгоритма для проверка свойства, определяемого предикатом.

Теперь распространим понятие вычислимости на функции, отличные от рассматриваемого типа.

Пусть D - некоторое множество, и $f: D^n \to D$ функция n переменных. Зафиксируем эффективное кодирование множества D натуральными числами N_0, т.е. эада-дим инъективную функцию $\alpha: D \to N_0$. Пусть α^{-1} - ее обратная. Тогда для функции $f: D^n \to D$ можно однозначно определить функцию $f^*: N_0^n \to N_0$, где

$f^* = \alpha f \alpha^{-1}$ или $f^*(a_1, a_2, ..., a_n) = \alpha(f(\alpha^{-1}(a_1), ..., \alpha^{-1}(a_n))$

$\forall a_1, ..., a_n \in N_0$

Будем называть функцию f вычислимой тогда и только тогда, когда функция f^* вычислима.

В качестве примера укажем кодирование множества целых чисел Z. Определим

$$\alpha(z) = \begin{cases} 2z \quad , если \quad z \geq 0 \\ -(2z+1), если \quad z < 0 \end{cases}$$

$$\alpha(z)=\begin{cases}\frac{1}{2}m, & если\quad m\quad \div\ ето\\ -\frac{1}{2}(m+1), & если\quad m\quad не\div ето\end{cases}$$

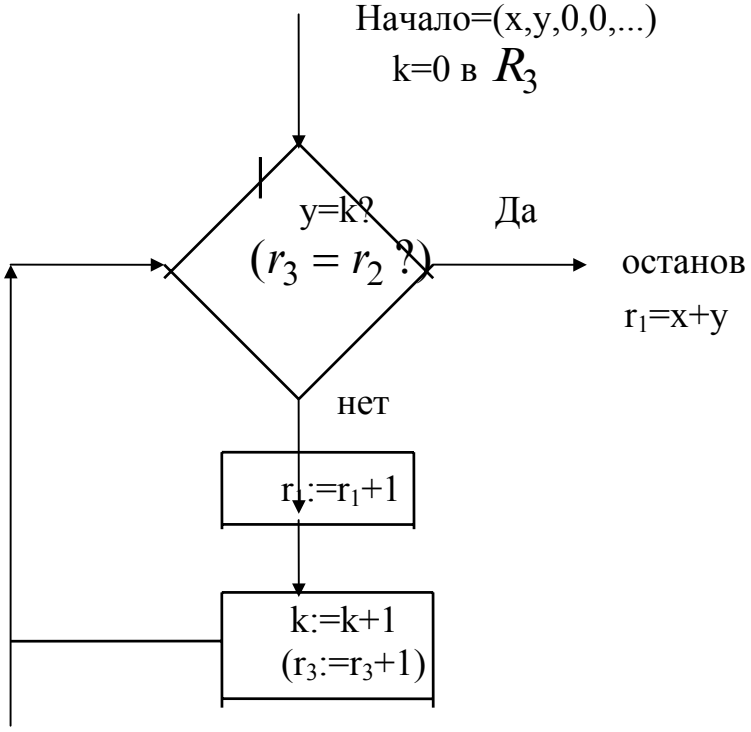
Таким образом, можно считать определенным понятие вычислимости целочисленных функций. Позднее будут рассмотрены эффективные кодирования и других областей.

3^0) Рассмотрим примеры вычислимых функций (на МПД).

а) функция $f(x+y)=x+y$. Эта функция может быть вычислена следующей программой

P: I_1 $J(3,2,5)$
I_2 $S(1)$
I_3 $S(3)$
I_4 $J(1,1,1)$

Данная программа прибавляет 1 к x до тех пор пока r_3 не станет равным y. Работу программн P можно представить блок-схемой:



б)Функция $f(x)=\{\frac{x}{2}$,если x-четное и неопределена

если x-нечетное

Эта функция может быть представлена программой P:

I_1 $J(1,2,6)$

I_2 $S(3)$

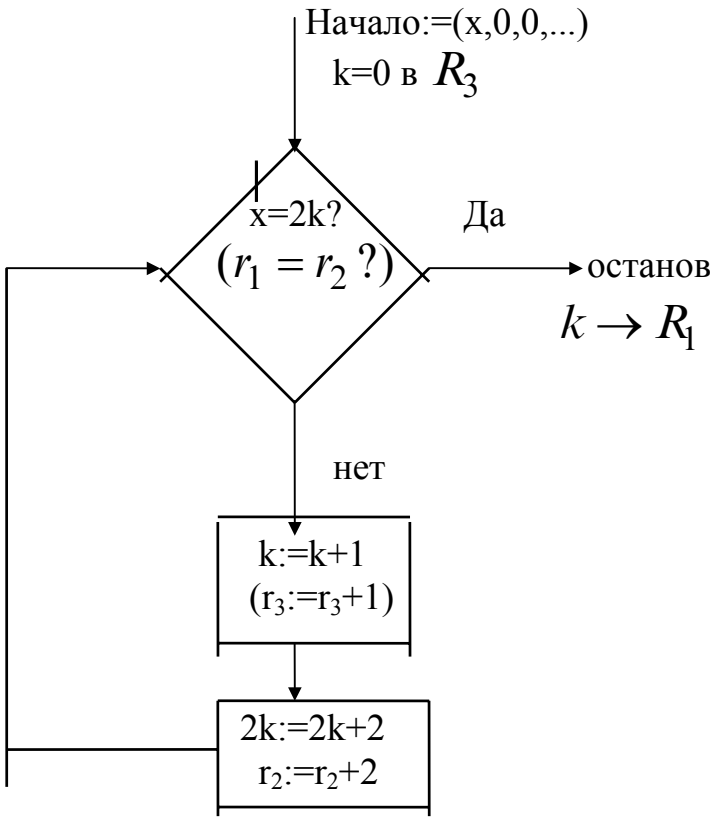
I_3 $S(2)$

I_4 $S(2)$

I_1 $J(1,1,1)$

I_6 $T(3,1)$

Данная программа прибавляет 1 к r_3 и 2 к r_2 до тех пор,пока r_2 не станет равным x , тогда r_3 даст результат. Работу программы можно представить блок-схемой:



4^0) Поскольку доказательства вычислимости конкретных функций связаны с предъявлением конкретных программ, их вычисляющих, то следует ввести некоторые соглашения о составлении и записи про-грамм. Аналогично композиции машин Тьюринга можно ввести компо- зицию программ МПД.

Пусть $P = I_1 I_2 ... I_s$. Будем говорить , что P имеет стандартный вид, если для всякой команды условного перехода $J(m,n,q)$ выполнимо $q \le s+1$.

Две программы P и P' назовем эквивалентными, если они определяют одни и те же n -местные функции, т.е. $f_P^n = f_{P'}^n$ для всех n>0.

Утверждение Для всякой программы P существует эквивалентная ей программа стандартного вида P'.

Док-во.

Пусть $P = I_1 I_2 ... I_s$.Тогда определим $P' = I'_1 I'_2 ... I'_s$, где $\forall\ k \in 1...s$

$$I'_k = \begin{cases} I_k \text{ , если } I_k \text{ не есть программа условного перехода} \\ I_k \quad \text{, если } I_k = J(m,n,q) \text{ и } q \le s+1 \\ J(m,n,q+1) \quad \text{, если } I_k = J(m,n,q \text{ и } q > s+1 \end{cases}$$

Ясно, что P' удовлетворяет нужным требованиям.

Утв док-но.

Пусть теперь даны две программы P и Q стандартного вида. Образуем программу $PQ = I_1 I_2 ... I_s I_{s+1} .... I_{s+t}$,где $P = I_1 I_2 ... I_s$, $Q = I_{s+1} I_{s+2} ... I_{s+t}$ с учетом нумерации, т.е. команды $J(m,n,q)$ заменены на $J(m,n,s+q)$. Тогда результат действия программы PQ совпадает с результатом вычисления по программе P , к которому применена программа Q.

Заметим, что для всякой программы P существует минимальное натуральное число $r(P)$, такое, что для всех $m,n \in N$, входящих в команды из P , т.е. $S(n)$, $Z(n)$, $T(m,n)$, $J(m,n,q)$ выполнено m,n< $r(P)$.

Это число иногда называют ширина, ранг программы P .

Смысл числа $r(P)$ состоит в том, что регистры R_t с t > $r(P)$ в ходе вычисления по программе P не будут менять свое содержание и не будут влиять на содержимое регистров $R_1, R_2, ..., R_r$, поэтому их можно использовать для других вычислений.

Заметим также, что можно организовывать вычисление, используя программу P , в случае, когда входы программы находятся в регистрах $R_{l_1}, R_{l_2}, ..., R_{l_n}$, а результат заносится в R_l. Пусть P вычисляет f в стандартном понимании вычислимости. Тогда программа

$$P[l_1,...,l_n \to l]: \quad \begin{matrix} T(l_1,1) \\ ..... \\ T(l_n,n) \\ Z(n+1) \\ ..... \\ Z(r(P)) \\ P \\ T(1,l) \end{matrix}$$

будет вычислять $f(x_{l_1}, x_{l_2}, ..., x_{l_n})$ и результат запишет в R_l. (Далее считаем, что регистры $R_{l_1}, R_{l_2}, ..., R_{l_n}$ отличны от $R_1, ..., R_n$.

Данную программу обозначим $P[l_1,\ldots,l_n \to l]$.

Приведем еще пример вычислимой функции.

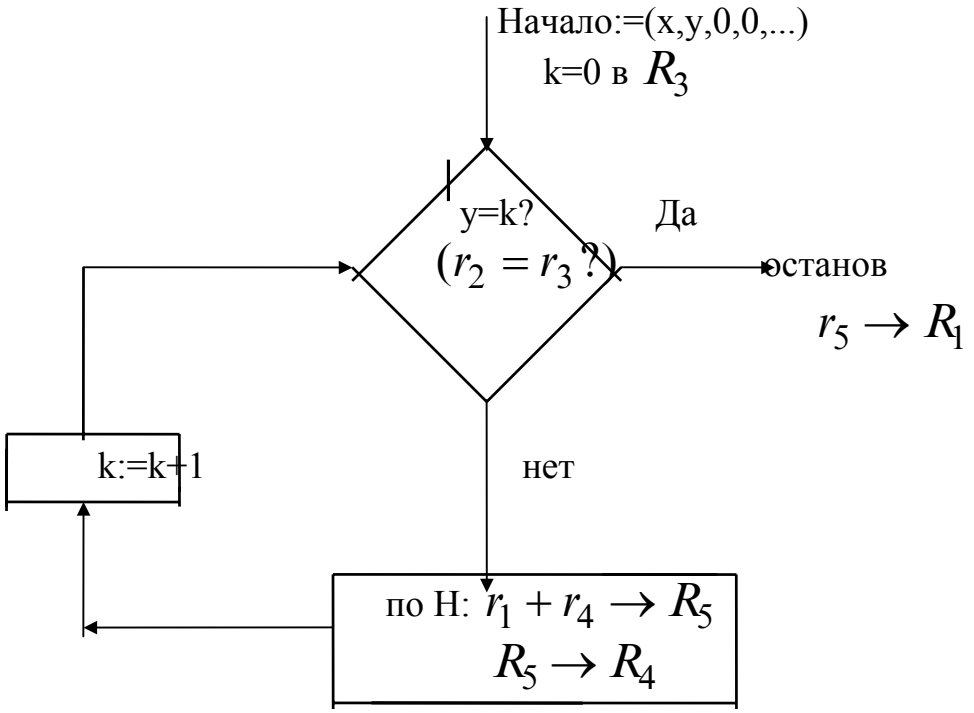
в) Функция $f(x,y) = xy$.

Пусть H - программа, вычисляющая функцию x+y (пример а)). Тогда $f(x,y) = xy$ вычисляется программой

$$P: \quad \begin{array}{l} J(2,3,p) \\ S(3) \\ H[1,4 \to 5] \\ T(5,4) \\ J(1,1,1) \\ I_p: T(5,1) \end{array}$$

Программа P вычисляет xy по правилу : $\begin{cases} x0 = 0 \\ xy = x(y-1) + x \end{cases}$

Работа программы проходит в соответствии со следующей блок-схемой:



Как следует из изложенно, язык програм МПД содержит основные процедуры языков програмирования и позволяет устраивать ком-позицию (соединение) программ и использовать програмны в качестве подпрограм других програм. Это является основаниен для пред-положения о том, что введенный класс вычислимых функций в точности отвечает классу алгоритмически вычислимых функций. Данное предположение называется тезисом Черча (для МПД). Также как и тезис Тьюринга данный тезис доказать

нельзя, однако принятие его позволяет истолковывать утверждения о несуществовании МПД для решения конкретных задач как утверждения о несуществовании алгоритмов вообще.

§ 4.Частично рекурсивные функции и их вычислимость

1^0) Приведем еще один класс вычислимых фукций, пред-ложенный в 30-х годах (Гедель, Клини, Черч) в качестве уточнения понятия алгоритма - класс частично рекурсивных функций. Данный класс определяется путем указания кон-кретных исходных функций и фиксированного множества операций получения новых функций из заданных. Ниже рас-сматриваются функции типа.

В качестве базисных функций беруются следующие:

1) нуль-функция : 0(x)=0 $\forall\ x \in N_0$

2) $f: N_0^n \to N_0$ функция следования: s(x)=x+1 $\forall\ x \in N_0$

3) Функция выбора аргументов: $I_m^n(x_1,...,x_n) = x_m$,
$\forall\ n \in N\ , 1 \le m \le n$

Допустимыми операциями над функциями являются операции суперпозиции (подстановки), рекурсии и минимизации.

Операция суперпозиции. Пусть даны n-местная функция g и n функций $f_1,...,f_n$ Считаем, что функции $f_1,...,f_n$ зависят от одних и тех же переменных $x_1,...,x_m$. Это можно сделать путем введения фиктивных переменных. Суперпозицией (подстановкой) функций g и $f_1,...,f_n$ называется функция

$$h(x_1,...,x_m) = g(f_1(x_1,...,x_m),...,f_n(x_1,...,x_m)) \qquad (1)$$

Если среди заданных функций имеются частичные, то и функция h будет частичной.Функция h на наборе переменных $x_1,...,x_m$ определена тогда и только тогда, когда определены все функции $f_1(x_1,...,x_m)$,..., $f_n(x_1,...,x_m)$ и функция h определена на наборе $f_1(x_1,...,x_m)$,..., $f_n(x_1,...,x_m)$. Операцию суперпозиции обозначают :

$$h = S(g, f_1,..., f_n)$$

Операция рекурсии (точнее: примитивной рекурсии). Пусть заданы n - местная функция $g(x_1,...,x_n)$ и n+2-местная функция $h(x_1,...,x_n,y,z)$. Определим n+1-местную функцию f индуктивным образом с помощью соотношений:

$$f(x_1,...,x_n,0) = g(x_1,...,x_n)$$
$$f(x_1,...,x_n,y+1) = h(x_1,...,x_n,y,f(x_1,...,x_n,y)) \qquad (2)$$

Ясно, что данные соотношения однозначно определяют функцию f .Если функции g и h частичные, то $f(x_1,...,x_n,y+1)$ считается определенной в том и только в том случае, когда определены $f(x_1,...,x_n,y)$ и $h(x_1,...,x_n,y,t)$ при $t = f(x_1,...,x_n,y)$ Значит, если $f(x_1,...,x_n,y_0)$

неопределено, то и $f(x_1,\ldots,x_n,y)$ неопределено при y>y_0 .Про функцию f говорят, что она получена рекурсией из функ-ций g и h и обозначают $f = R(g,h)$.

Операция минимизации .Пусть задана n -местная функция $g(x_1,\ldots,x_{n-1},y)$. Зафиксируем набор $(x_1,\ldots,x_{n-1},x_n)$ и рассмотрим уравнение относительно y :

$$g(x_1,\ldots,x_{n-1},y) = x_n \qquad (3)$$

Будем решать данное уравнение, вычисляя последовательно $g(x_1,\ldots,x_{n-1},0)$. $g(x_1,\ldots,x_{n-1},1), g(x_1,\ldots,x_{n-1},2)$ и сравнивая с x_n. Наименьшее y , для которого выполнено (3) обозначим

$$\mu_y = (g(x_1,\ldots,x_{n-1},y) = x_n) \qquad (4)$$

При этом считаем, что y определено, если $g(x_1,\ldots,x_{n-1},z)$ определено при всех $z \le y$. В противном случае считаем, что y неопределено. Зна-чение y есть функция f от переменных $x_1,\ldots,x_{n-1},x_n$, про которую говорят, что она получена из

$$f = M_y g \qquad (5)$$

Заметим, что определенные выше операции S и R , будучи приме-ненными к всюду определенным функциям, дают всюду определенные функции. Операция M может давать частичные функции даже при при-менении к всюду определенным функциям.

Пример

$$\mu_y(x_2 + y) = x_1 - x_2$$

Здесь $g(x_2,y) = x_2 + y$ всюду определена, но x_1-x_2 определена только при $x_1 \ge x_2$.

Дадим теперь основное определение данного раздела. функция - называется частично рекурсивной, если она может быть получена из базисных функций 0(x) , s(x), $I_m^n(x_1,\ldots,x_n)$ применением конечного числа раз операций суперпозиции, рекурсии и минимизации. Иногда частично рекурсивные функции называют функциями, вычисли-мыми по Черчу. Всюду определенная частично рекурсивная функция называется общерекурсивной .Если рассматривать тот же базис функций, то в качестве допустимых операций брать операции суперпозиции и рекурсии, то получаемые функции называются примитивно рекурсивными.

Обозначим : **Ч**- класс частично рекурсивных функций, $\mathbf{Ч}_0$ - класс общерекурсивных функций, $\mathbf{Ч}_{пр}$ - класс примитивно-рекурсивных функций.

Класс частично рекурсивных функций - одно из главных понятий теории алгорипмов. Это объясняется тем, что какие бы классы точно очерченных

"алгоритмов" до сих пор не рассматривались, во всех случаях оказывалось, что соответствующие числовые функции, вычислимые посредством алгоритмов этих классов, были частично рекурсив-ными. Поэтому общепринятой является гипотеза, формулируемая как

Тезис Черча (для частично рекурсивных функций).

Класс алгоритмически вычислимых функций совпадает с классом всех частично рекурсивных функций. Принятие данного тезиса позволяет истолковывать доказательство, что некоторая функция не является частична рекурсивной, как доказатеяьство отсутствия алгоритма вычисления ее значений.

Сделаем одно замечание. Пусть необходимо доказать, что конкретная функция вычислима. Это можно сделать следующими способами.

1. Написать программу машины Тьюринга или МПД, вычисляющую f , либо показать, что f принадлежит классу функций, вычислимость которых доказана.

2. Написать рекурсивную схему для f , показывающую, что f - частично рекурсивна.

3. Дать неформальное (но достаточно точное) описание алгоритма, вычисляющего f , и затем сослаться на тезис Черча.

Мы будем пользоваться способом 3) как строгим методом доказательства, основанным на тезисе Черча .

2^0) Приведем примеры частично рекурсивных функций и установим частичную рекурсивность основных числовых функций, используемых в арифметике и анализе .

1. Функции-константы.

$f(x) = m = s(s(...s(0(x))...))$_ m раз .

2. Функция $f(x,y) = x + y$

Имеем $x + 0 = x = I_1^2(x,y)$

$x + (y+1) = (x+y) + 1$

Это есть рекурсия с помощью функций g(x)=x и h(x,y,z)=s(z).

3. Функция $f(x,y) = xy$

Имеем $x0 = 0$

$x(y+1) = xy + x$

Это есть рекурсия с помощью функций g(x)=0(x) , h(x,y,z)=x+z

4. Функция $f(x,y) = x^y$

Имеем $x^0 = 1$

$x^{y+1} = xx^y$

Это есть рекурсия с помощью функций g(x)=s(0(x)) , h(x,y,z)=xz.

5. Функция $sg(x) = \begin{cases} 0 & , \quad если \quad x = 0 \\ 1 & , \quad если \quad x > 0 \end{cases}$

Имеем $sg(0) = 0$

$$sg(x+1)=1$$

Это рекурсия, в которой g=0(x) , h=s(0(x))

6. Функция $\overline{sg}(x)=\begin{cases}1 & , \quad если \quad x=0\\ 0 & , \quad если \quad x>0\end{cases}$

Имеем $\overline{sg}(x)=1$

$$\overline{sg}(x+1)=0$$

Это рекурсия, в которой g=1 , h=0.

7. Функция $x\dot{-}1=\begin{cases}0, & если \quad x<y\\ x-1, & если \quad x\geq y\end{cases}$

Имеем $0\dot{-}1=0$

$$(x+1)\dot{-}1=x$$

Это рекурсия, в которой g=x , h=y .

8. Функция $x\dot{-}y=\begin{cases}0, & если \quad x<y\\ x-y, & если \quad x\geq y\end{cases}$

Имеем $x\dot{-}0=x$

$$x\dot{-}(y+1)=(x\dot{-}y)\dot{-}1$$

Это рекурсия, в которой g=x , h= $z\dot{-}1$

---

9. Функция $|x-y|$.

Имеем $|x-y|=(x\dot{-}y)+(y\dot{-}x)$

Замечание. Поскольку функция $|x-y|$ общерекурсивна, то можно заменить определение операции миними-зации, рассматривая вместо

$$f(x_1,\ldots,x_n)=\mu_y(g(x_1,\ldots,x_{n-1},y)=x_n)$$

функции вида

$$f(x_1,\ldots,x_n)=\mu_y(h(x_1,\ldots,x_n,y)=0)$$

Определяемый класс функций при этом будет тем же самым.

10. Функция $\min(x,y)$

Имеем $\min(x,y)=x\,\dot{-}\,(x\,\dot{-}\,y)$

11. Функция $\max(x,y)$

Имеем $\max(x,y)=x+(y\,\dot{-}\,x)$

12. Функция $\left[\frac{x}{y}\right]$-целая часть от деления x на y

По определению полагаем $\left[\frac{x}{0}\right] = x$, чтобы функция была всюду определена.

Имеем $\left[\frac{x}{y}\right] = \mu_{z<x}((x \dot{-} z)((x+1) \dot{-} y(z+1)) = 0)$

Действительноно, при y=o

$$\mu_z((x \dot{-} z)((x+1) \dot{-} y(z+1)) = 0) = \mu_z\big((x \dot{-} z) = 0\big) = x$$

При $y \neq 0$ существует минимальное z, $z \leq x$, при котором $(x+1) \dot{-} y(z+1) = 0$ или $(x+1) \leq y(z+1)$ или $x < y(z+1)$ откуда C=z.

13. Функция $res(x,y)$-остаток от деления x на y.

Имеем $res(x,y)$=x-y$\left[\frac{x}{y}\right]$

14. Функция $d(x,y) = \begin{cases} 1 & , \quad если \quad res(x,y) = 0 \\ 0 & , \quad если \quad res(x,y) \neq 0 \end{cases}$

Имеем $d(x,y) = \overline{sg}(res(x,y))$

15. Двоичная степень числа x.

$\exp_2(x) = t$, *если* $2^t | x$, *но* 2^{t+1} *не делит* x

Имеем $\exp_2(x) = \mu_t(d(x,e^{t+1}) = 0)$ и замечаем , что функция 2^{t+1}-общерекурсивна.

16. Функция, отличная от О в конечном числе точек. Если $f(x) \neq 0$ в точках $x_1,\ldots x_k$, причем $f(x_1) = y_1,\ldots,f(x_k) = y_k$,то имеем

$$f(x) = y_1\overline{sg}|x - x_1| + y_2\overline{sg}|x - x_2| + \ldots + y_k\overline{sg}|x - x_k|$$

Аналогично можно доказать (примитивную) рекурсивность функций

а) $\pi(x) =$ число простых чисел, не превосходящих x.

б) $q(x) = x \dot{-} \left[\sqrt{x}\right]^2$-квадратичный остаток числа x.

в) $p(x) = p_x$ (x+1)-ое -простое число

($p_0 = 2$, $p_1 = 3$, $p_2 = 5$, $p_3 = 7$)

3^0) Покажем теперь вычислимость на МПД частично рекурсивных функций.

Базисные функции $0(n), s(n), I_m^n(x_1,...,x_n)$ вычислимы на МПД командами $Z(n), S(n), T(m,1)$.

Вычислимость суперпозиций

Теорема1. Пусть функции $g(y_1,...,y_n)$, $f_1(x_1,...,x_m),..., f_n(x_1,...,x_m)$ вычислимы на МПД. Тогда вычислима и функция

$$y(x_1,,,x_m) = S(g; f_1,\ldots, f_n)$$

Док-во.

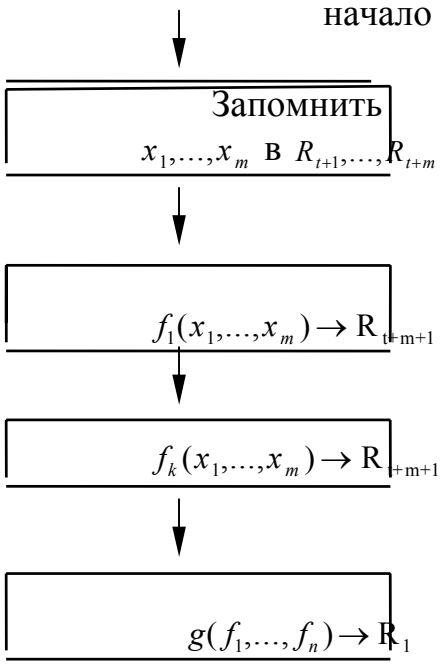
Пусть $G, F_1,\ldots, F_n$ - программа стандартного вида для вычисления функций $g, f_1,\ldots, f_n$ соот-ветственно. Напишем программу H для вычисления функции h.

Положим $t = \max(n, m, r(G), r(F_1),\ldots, r(F_n))$

Запомним $x_1,\ldots,x_m$ в регистрах $R_{t+1},\ldots,R_{t+m}$, в регистрах $R_{t+m+1},\ldots,R_{t+m+n}$ запоминаем значения $F_1(x_1,\ldots,x_m),\ldots,F_n(x_1,\ldots,x_m)$ соответственно. Указанные регистры не затрагиваются вычисле- ниями по программам $G, F_1,\ldots, F_n$.Теперь дадим программу H вычисления h :

$T(1,t+1)$
$T(2,t+2)$
. . . .
$T(m,t+m)$
$F_1[t+1,\ldots,t+m \to t+m+1]$
. . . .
$F_n[t+1,\ldots,t+m \to t+m+n]$
$G[t+m+1,\ldots,t+m+n \to 1]$

Вычисление по программе H происходит в соответ-ствии с блок-схемой



Ясно: вычисление $H(x_1,\ldots,x_m)$ останавливается$\Leftrightarrow$ заканчивается вычисление каждой $F_i(x_1,\ldots,x_m), i\in\overline{1,n}$ и вычисление $G(f_1(x_1,\ldots,x_m),\ldots,f_n(x_1,\ldots,x_m))$

ч.т.д.

Следствие Пусть $f(y_1,\ldots,y_m)$вычислимая на МПД функция и $x_{i_1},\ldots,x_{i_v})$ последовательность m переменных из C(возможно с повторениями). Тогда функция $h(x_1,\ldots,x_n)=f(x_{i_1},\ldots,x_{i_v})$вычислима.

док-во.

Если $\overline{x}=(x_1,\ldots,x_n)$,то $h(x_1,\ldots,x_n)=f(I^n_{i_1}(\overline{x}),\ldots,I^n_{i_m}(\overline{x})$ и потому h вычислима.

ч.т.д .

Вычислимость рекурсии

Теорема 2.

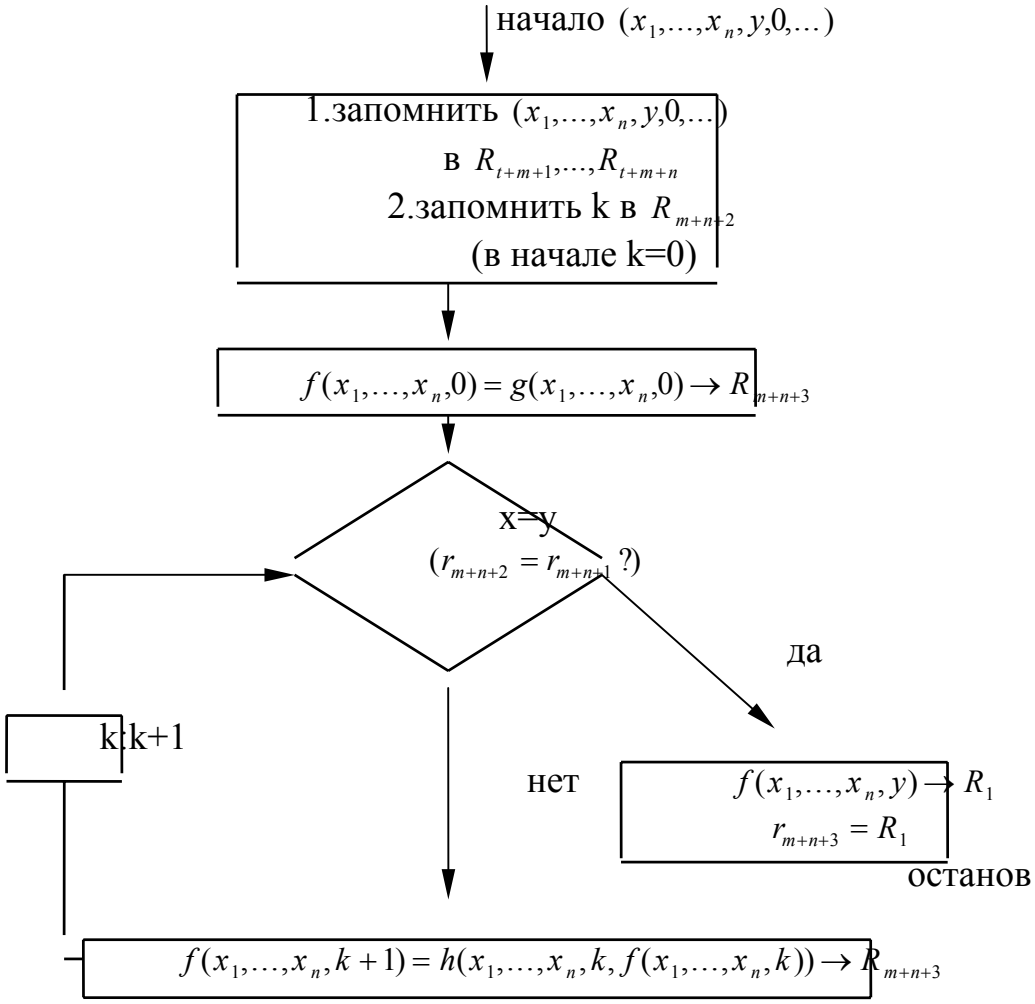
Пусть функции $g(x_1,\ldots,x_n)$, $h(x_1,\ldots,x_n,y,z)$ - вычислимы на МПД. Тогда функция $f=R(g,h)$ - вычислима.

Док-во.

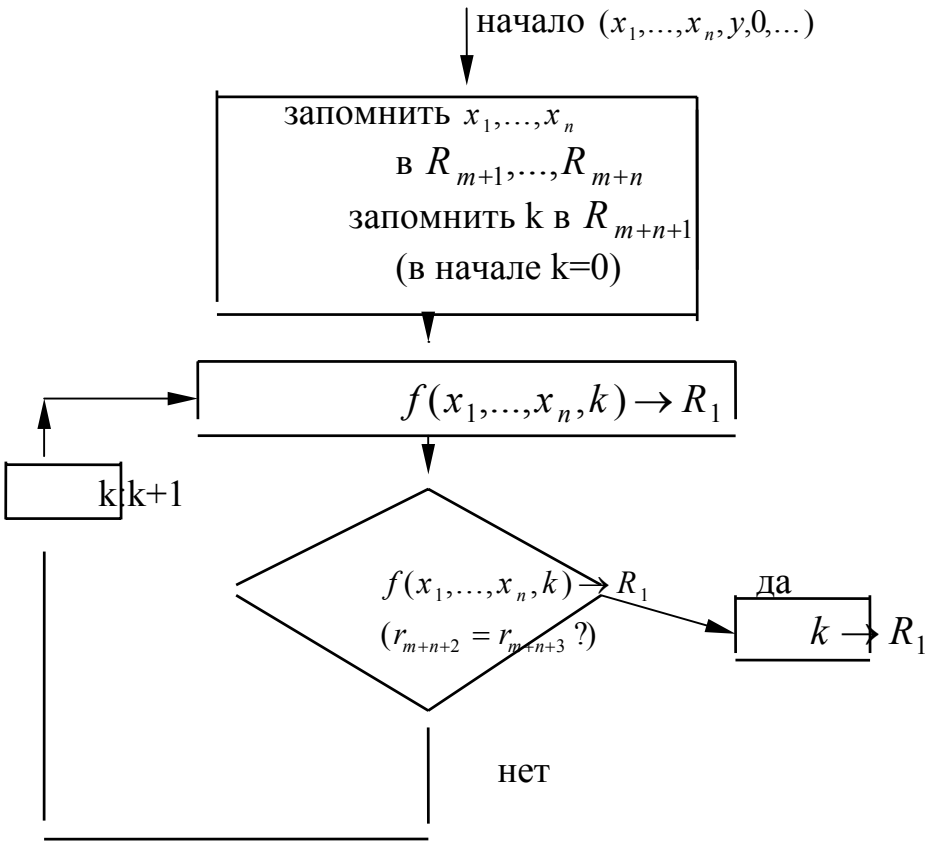
Пусть G и H программы стандартного вида для вычисления функций g и h. Построим программу F вычивляющую функцию $f=R(g,h)$.По начальной конфигурации $(x_1,\ldots,x_n,y,0,\ldots)$ по программе G вычисляется $f(x_1,\ldots,x_n,0)=g(x_1,\ldots,x_n)$. Теперь, если $y\neq 0$ то применяем (многократно) программу Y* для нахождения $f(x_1,\ldots,x_n,1)$, $f(x_1,\ldots,x_n,2)$,..., $f(x_1,\ldots,x_ny)$

Положим $m = \max(n+2, r(G), r(H))$

Запоминаем $x_1,\ldots,x_n,y$ в регистрах $R_{m+1},\ldots,R_{m+n},R_{m+n+1}$.Номер цикла к , где к=0,1,...,у помещаем в R_{m+n+2}. Промежуточное значение $f(x_1,\ldots,x_n,k)$ помещаем в R_{m+n+3}. Вычисление по программе Н происходит в соответствии с блок-схемой.



Помещаем $x_1,\ldots,x_n,k$ в регистры $R_{m+1},\ldots,R_{m+n},R_{m+n+1}$. Полагаем вначале k=0.Промежуточное значиние $f(x_1,\ldots,x_n,k)$ помещаем в R_{m+n+2}. Вычисление функции g происходит в соответствии с блок-схемой:



Программа F для вычисления f : (Здесь t=m+n):

$$
\begin{aligned}
& T(1,m+1) \\
& \quad . \\
& \quad . \\
& \quad . \\
& T(n+1,m+n+1) \\
\mathrm{I_q}: \quad & J(t+2,t+1,p) \\
& H\left[m+1,\ldots,m+n,t+2,t+3 \to t+3\right] \\
& S(t+2) \\
& J(1,1,q) \\
` \quad \mathrm{I_p}: & T(t+3,1)
\end{aligned}
$$

Следовательно, функция f вычислима.

ч.т.д.

<u>Вычислимость минимизации</u>.

<u>Теорема3.</u>

Пусть функция $f(x_1,\ldots,x_n,y)$ вычислима.тогда функция $g(x_1,\ldots,x_n)=\mu_y(f(x_1,\ldots,x_n,y)=0)$ вычислима.

Док-во.

Пусть F-программа стандартного вида,вычисляющая функцию f .Пусть $m=\max(n+1,r(F))$.Построим программу G для вычисления функции g по следующему алгоритму: вычислять $f(x_1,\ldots,x_n,k)$ при k=0,1,... до тех пор, пока не найдется такое k,что $f(x_1,\ldots,x_n,k)=0$ тогда k будет требуемым выходом.

Программа G для вычисления функции g :

$$
\begin{aligned}
& T(1,m+1) \\
& \quad .\ .\ .\ . \qquad\qquad . \\
& T(n,m+n) \\
\mathrm{I_q}: \quad & F\left[m+1,\ldots,m+n \to m+n+2\right] \\
& S(m+n+1) \\
& J(m+n+2,m+n+3,q) \\
& J(1,1,p) \\
` \quad \mathrm{I_p}: & T(m+n+1,1)
\end{aligned}
$$

следовательно , функция g вычислима.

ч.т.д.

Следствие.

Частично рекурсивные функции вычислимы на МПД,т.е.$\mathbf{Ч} \subseteq \mathbf{Е}$.

.

4^0) Покажем теперь частичную рекурсивность вычислимых функций.

Теорема 4. Всякая вычислимая на МПД функция является частично рекурсивной.

Док-во.

Пусть $f(x_1,\ldots,x_n)$ вычислимая на МПД функция и пусть $P = I_1 I_2 \ldots I_s$ - соответствующая программа.Будем называть шагом вычисления выполнение одной команды программы.Для произвольных $\bar{x} = (x_1,\ldots,x_n)$ и $t \in N_0$ определим следующие функции , свезанные с вычислением $P(\bar{x})$:

$$c(\bar{x},t) = \begin{cases} r_1(\text{содержимое } R_1) \text{ после } t \text{ шагов} \\ \text{в } P(\bar{x}), \text{ если } P(\bar{x}) \text{ не остановилось раньше} \\ r_1(\text{содержимое } R_1), \text{ если } P(\bar{x}) \text{ остановилось} \\ \qquad \text{раньше.} \end{cases}$$

$$c(\bar{x},t) = \begin{cases} \text{номер следующей команды, после } t \text{ шагов} \\ \text{в } P(\bar{x}), \text{ если } P(\bar{x}) \text{ не остановилось} \\ \text{после } t \text{ шагов или раньше} \\ 0, \text{ если } P(\bar{x}) \text{ остановилось после } t \text{ шагов} \\ \quad \text{или раньше.} \end{cases}$$

Таким образом, $c(\bar{x},0) = x_1$

$j(\bar{x},0) = 1$

Очевидно, что функции $c(\bar{x},t)$, $j(\bar{x},t)$ всюду определены.

Найдем теперь выражение для $f(\bar{x})$ через введенные функции. Если значение $f(\bar{x})$ определено, то $P(\bar{x})$ останавливается после t_0 шагов, где

$$t_0 = \mu_t(j(\bar{x},t) = 0)$$

поэтому

$$f(\bar{x}) = c(\bar{x},t_0)$$

Если же значение $f(\bar{x})$ неопределено, то $P(\bar{x})$ не останавливается, и тогда $j(\bar{x},t) \neq 0 \quad \forall t \in N_0$, поэтому $\mu_t(j(\bar{x},t) = 0)$ не опредено. Следовательно, во всех случаях $f(\bar{x}) = c(\bar{x}, \mu_t(j(\bar{x},t) = 0))$

Теперь, если убедиться, что функции $c(\bar{x},t)$ и $j(\bar{x},t)$ частично рекурсивны, то таковой будет и функция $f(\bar{x})$. Ясно,что существует неформальный алгоритм вычисления значений функций $c(\bar{x},t)$ и $j(\bar{x},t)$. Для

этого нужно по заданным $\bar{x}$, t написать последовательность конфигураций $K_0 \to K_1 \to \ldots \to K_t$ и выписать содержимое регистра R_1 к номер выполняемой на шаге t+1 команды. По тезису Черча функции $c(\bar{x},t)$ и $j(\bar{x},t)$ частично рекурсивны и, значит, функция $f(\bar{x})$ является частично рекурсивной.

ч.т.д.

Замечание Более точный анализ показывает, что функции $c(\bar{x},t)$ и $j(\bar{x},t)$ является примитивно рекурсивными.

Следствие. Классы функций **Ч** и **Е** совпадают, т.е. **Ч**=**Е** .

5^0) Рассмотрим теперь вопрос о соотношении введенных классов $\mathbf{Ч}_{пр}$,$\mathbf{Ч}_0$,**Ч**.Поскольку класс **Ч** содержит частично определенные функции, то ясно, что $\mathbf{Ч}_0 \subset \mathbf{Ч}$.Кроме того, очевидно,что $\mathbf{Ч}_{пр} \subseteq \mathbf{Ч}$.

Вопрос о том, является ли включение $\mathbf{Ч}_{пр} \subseteq \mathbf{Ч}$ соб- ственным решается несколько сложнее.

Первый пример общерекурсивной функции, не являющейся примитивно рекурсивной,был дан Аккерманом (1928 г.).

Функция Аккермана $\varphi(x,y)$ задается соотношениеми :

$$\varphi(x,0) = y+1$$
$$\varphi(x+1,0) = \varphi(x,1)$$
$$\varphi(x+1,y+1) = \varphi(x,\varphi(x+1,y))$$

Можно доказать, что данные соотношния однозначно определяют функцию $\varphi(x,y)$.

Например,
$$\varphi(0,0) = 1, \varphi(0,1) = 2, \varphi(1,0) = 2$$
$$\varphi(1,1) = \varphi(0,\varphi(1,0)) = \varphi(0,2) = 3$$
$$\varphi(2,0) = \varphi(1,1) = 3$$
$$\varphi(1,2) = \varphi(0,\varphi(1,1)) = \varphi(0,3) = 4$$
$$\varphi(3,0) = \varphi(2,1) = \varphi(1,\varphi(2,0)) = \varphi(1,3) =$$
$$= \varphi(0,\varphi(1,2)) = \varphi(0,4) = 5$$

Результаты вычислений убеждают,что найдется алгоритм вычисления функции $\varphi(x,y)$.

В то же время доказывается, что функция $\varphi(x,y)$ не является примитивно рекурсивной,т.к. растет быстрее, чем любая одноместная примитивно рекурсивная функция.

Доказательство, ввиду его громоздкости, опускается (см. Мальцев (7), стр. 105).

§ 5. Нумерация наборов чисел и слов

1^0) В теории алгоритмов получил распространение прием, позволяющий сводить изучение функций от нескольких переменных к изучению функций одной переменной. Он основан на нумерации наборов чисел так, что имеется биективное соответствие между наборами чисел и их номерами, причем функции, определяющие по набору чисел его номер и по номеру сам набор чисел являются общерекурсивными.

Рассмотрим сначала множество $N_0 * N_0$ - множество пар натуральных чисел. Рассмотрим следующее упорядочение этих пар, называемое канторовским:

(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),... (1)

Здесь в порядке возрастания $n \in N_0$ упорядочиваются пары (x,y)с условием x+y=n в виде последовательности

(0,n),(1,n-1),...,(x,y),...,(n,0) (2)

Пусть c(x,y)- номер пары (x,y)в последовательности (1), причем считаем c(0,0)=0. Если c(x,y)=n , то обозначим через r, l - функции, удовлетворяющие:

$$x = l(n)$$
$$y = r(n)$$

и, следовательно,

$$c(l(n), r(n)) = n$$

Покажем, что функции c, l, r в явном виде выра-жаются через обычные арифметические функции. Произвольная пара (x,y) находится на месте x+1 в последовательности (2). Далее перед последователь-ностью (2) находятся последовательности, отвечающие элементам (x_1,y_1) с условием $x_1+y_1=t$, где t=0,1,2,...,x+y-1 , и каждый из них содержит t+1 элемент.

Следовательно, элемент (x,y) находится в после-довательности (1) на месте 1+2+...+x+y+x+1 . Поскольку нумерация начитается с нуля , то номер элемента (x,y) в последовательности (1) равен

$$c(x,y) = \frac{(x+y)(x+y+1)}{2} + x = \frac{(x+y)^2 + 3x + y}{2} \qquad (4)$$

Пусть теперь c(x,y)=n и найдем $x = l(n)$, $y = r(n)$.

Из (4) следуют равенства:

$$2n = (x+y)^2 + 3x + y$$
$$8n + 1 = (2x + 2y + 1)^2 + 8x$$
$$8n + 1 = (2x + 2y + 3)^2 - 8y - 8$$

Следовательно

$$2x + 2y + 1 \le \left[\sqrt{8n+1}\right] < 2x + 2y + 3$$

или $$x + y + 1 \le \frac{\left[\sqrt{8n+1}\right] + 1}{2} < x + y + 2$$

Это означает, что

$$x + y + 1 = \frac{\left[\sqrt{8n+1}\right]+1}{2} \qquad (5)$$

Теперь, используя (4), определяем x :

$$x = l(n) = n \div \frac{1}{2}\left[\frac{\left[\sqrt{8n+1}\right]+1}{2}\right]\left[\frac{\left[\sqrt{8n+1}\right]\div 1}{2}\right] \qquad (6)$$

Подставляя найденное значение x в (5) получаем y :

$$y = r(n) = \left[\frac{\left[\sqrt{8n+1}\right]+1}{2}\right] + \frac{1}{2}\left[\frac{\left[\sqrt{8n+1}\right]+1}{2}\right]\left[\frac{\left[\sqrt{8n+1}\right]\div 1}{2}\right] \div n \div 1 \quad (7)$$

Заметим, что важен не сам вид полученных функций $c(x,y)$, $r(n)$, $l(n)$, а важен факт их эффективной вы-числимости.

Теперь с помощью нумерации пар чисел легко получить нумерацию троек чисел, т.е. элементов множества N_0^3. Определим функции :

$$c^3(x,y,z) = c(c(x,y),z) \qquad (8)$$

Тогда ,если $c^3(x,y,z) = n$

то
$$z = r(n)$$
$$y = r(l(n)) \qquad (9)$$
$$x = l(l(n))$$

Аналогично, для наборов произвольной длины r+1 полагаем

$$c^1(x_1) = x_1 \quad , \quad c^2(x_1,x_2) = c(x_1,x_2) \qquad (10)$$
$$c^{r+1}(x_1,x_2,...,x_{r+1}) = c^r(c(x_1,x_2),x_3,...,x_{r+1})$$

и по определению называем число $c^n(x_1,x_2,...,x_n)$ канторов- ским номером n-kn $(x_1,x_2,...,x_n)$

Если $c^n(x_1,x_2,...,x_n) = m$, то

$$\begin{aligned} x_n &= r(m) \\ x_{n-1} &= r(l(m)) \\ &\cdots\cdots \\ x_2 &= r(l(...l(x)...)) \\ x_1 &= l(l(...l(x)...)) \end{aligned} \qquad (11)$$

Теперь имея нумерацию множеств $N_0^k\,(k \geq 1)$ можно ус-тановить нумерацию множества

$$M = N_0 \cup N_0^2 \cup .... \cup N_0^k \cup ...$$

Положим для любого $n \in N$

$$c(x_1,x_2,...,x_n) = c(c^n(x_1,x_2,...,x_n),n-1) \qquad (12)$$

Ясно, что с - биективное соответствие между М и N_0. Кроме того, если $c(x_1,x_2,...,x_n)=m$, то

$$c^n(x_1,x_2,...,x_n)=l(m)$$
$$n=r(m)+1$$

Отсюда, используя (11) эффективно определяются $x_1,x_2,\ldots,x_n$.

2^0) Приведем еще одну нумерацию наборов чисел. Номер пары (x,y) зададим функцией

$$p(x,y)=2^k(2y+1)\dot{-}1 \qquad (13)$$

Ясно, что это общерекурсивная функция. При этом если p(x,y)=n , то выполнено

$$x=\exp_2(n+1)$$
$$y=\left[\frac{n+1}{2^{\exp_2(n+1)+1}}\right]$$

Следовательно, для данной нумерации

$$l(n)=\exp_2(n+1)$$
$$r(n)=\left[\frac{n+1}{2^{\exp_2(n+1)+1}}\right]$$

Теперь, имея нумерационную функцию для пар чисел, аналогично предыдущему строим нумерационные функции для -kn чисел и множества наборов $M=N_0\cup N_0^2\cup\ldots$.

Другую нумерацию множества М мож но получить так:

$$\tau(x_1,...,x_k)=2^{x_1}+2^{x_1+x_2+1}+...+2^{x_1+...+x_k+k-1}-1 \qquad (14)$$

Ясно , что τ -вычислима. Чтобы установить вычис-лимость τ^{-1} , заметим, что каждое натуральное число имеет единственное представление в двоичной позиционной записи. Т.е. для любого n можно эффек-тивно и однозначно найти $k\geq 1$ и $0\leq b_1<b_2<\ldots,b_k$ такое, что $n+1=2^{b_1}+2^{b_2}+\ldots+2^{b_k}$.Откуда получаем

$\tau^{-1}(n)=(x_1,x_2,\ldots,x_n)$,где $x_1=b_1$,$x_{i+1}=b_{i+1}-b_i-1$ $(1\leq i\leq k)$

3^0) Рассмотрим теперь вопрос о нумерации слов в некотором алфавите и укажем некоторые из применяемых способов такой нумерации.

Пусть $A=\{a_1,\ldots,a_r\}$ - конечный алфавит и пусть A^* - множество всех слов конечной длины в алфавите A , включая и пустое слово λ.

Алфавитная нумерация определяется следующим образом:

$$c(\lambda)=0$$

$$c(a_{i_1},\ldots,a_{i_s}) = i_s + i_{s-1}r + \ldots + i_1 r^{s-1} \qquad (15)$$

Поскольку при фиксированном r каждое положительное число n однозначно представимо в виде

$$n = i_s + i_{s-1}r + \ldots + i_1 r^{s-1} \quad (1 \le i_1 \le r) \qquad (16)$$, то

каждое число есть алфавитный номер одного и только одного слова из множества A^* Разложение (16) называется r-ичным разложением числа n с цифрами

1,...,r в отличии от обычного r-ичного разложения с коэффициентами 1,...,r.

Нумерация слов через нумерационные Функции. Пусть имеется счетный алфавит $A = \{a_0, a_1, \ldots\}$. Тогда нумерация слов определяется так:

$$\nu(\lambda) = 0$$

$$\nu(a_{i_1,\ldots,}a_{i_s}) = c(i_1,\ldots,i_s) + 1 \quad , \quad s \ge 1 \qquad (17)$$

где функция $c(i_1,\ldots,i_s)$ определена соотношениен (12). Ясно, что так определенная функция ν является биек- тивной и вычислимой .

Геделевская _нумерация. Пусть имеем счетный алфавит $A = \{a_0, a_1, \ldots\}$. Определим геделевы номера для каждой буквы $g(a_i) = 2i + 3 \quad , \quad i \in N_0$. Теперь для каждого слова $P = a_{i_0} a_{i_1} \ldots a_{i_k}$ определим геделев номер

$g(P) = 2^{g(a_{i_0})} 3^{g(a_{i_1})} \ldots p_k^{g(a_{i_k})}$,где p_k- k-ое простое число. Кроме того , положим

$$g(\lambda) = 1$$

При этом геделев номер последовательности слов $P_0, P_1, \ldots, P_k$ определяется так:

$$2^{g(P_0)} 3^{g(P_1)} \ldots . p_k^{g(P_k)}$$

4^0) Рассмотрим теперь два применения нумерационных функций.

а) *Утверждение 1* Функция $f(x,y)$, отличная от нуля на конечном множестве пар из N_0^2 общерекурсивна.

Док-во.

Действительно, пусть $f \ne 0$ на парах чисел $(x_1,y_1),\ldots,(x_t,y_t)$ и пусть имеет на них значения $z_1,\ldots,z_t$.На остальных парах $f(x,y)$=0. Положим $u_1 = c(x_1,y_1),\ldots,u_t = c(x_t,y_t)$,где C - нумерационная функция Кантора.

Определи» функцию g так:

$g(u_i) = z_i$, $i = 1,\ldots,t$

$g(u) = 0$ на остальных $u \in N_0$.

Было выше показано,что g -общерекурсивна. По построению выполнено $f(x,y)=g(c(x,y))$ и поэтому f общерекурсивна.

б) Определим сначала понятие совместной рекурсии. В схеме совместной рекурсий функция порождается с помощью нескольких функций.

Пусть для определенности даны функции

$$g_1(\overline{x})$$
$$g_2(\overline{x})$$
$$h_1(\overline{x},y,z,t)$$
$$h_2(\overline{x},y,z,t)$$

здесь обозначено $\overline{x}=(x_1,x_2,\ldots,x_n)$.

Тогда можно определить пару функций $f_1(\overline{x},y)$ и $f_2(\overline{x},y)$ по рекурсии:

$$f_1(\overline{x},y)=g_1(\overline{x})$$
$$f_2(\overline{x},y)=g_2(\overline{x})$$
$$f_1(\overline{x},y+1)=h_1(\overline{x},y,f_1(\overline{x},y),f_2(\overline{x},y))$$
$$f_2(\overline{x},y+1)=h_2(\overline{x},y,f_1(\overline{x},y),f_2(\overline{x},y))$$

Утверждение2 Если g_1,g_2,h_1,h_2-общерекурсивные функции, то f_1,f_2 также общерекурсивны.

Док-во.

Определим функцию , $u(\overline{x},y)=c(f_1(\overline{x},y),f_2(\overline{x},y))$, где

С- нумерационная функция Кантора.

Тогда имеем

$$f_1(\overline{x},y)=l(u(\overline{x},y))$$
$$f_2(\overline{x},y)=r(u(\overline{x},y))$$

Далее имеем

$$u(\overline{x},0)=c(f_1(\overline{x},0),f_2(\overline{x},0))=c((g_1(\overline{x}),g_2(\overline{x})))$$-

частично рекурсивная по условию.

$$u(\overline{x},y+1)=c(f_1(\overline{x},y+1),f_2(\overline{x},y+1))=$$
$$=c(h_1(\overline{x},y,f_1(\overline{x},y),f_2(\overline{x},y)),h_2(\overline{x},y,f_1(\overline{x},y),f_2(\overline{x},y)))$$

функция $u(\overline{x},y+1)$ получается по схеме обычной рекурсии с помощью функций

$$g(\overline{x})=c(g_1(\overline{x}),g_2(\overline{x}))$$
$$h(\overline{x},y,z)=c(h_1(\overline{x},y,l(z),r(z)),h_2(\overline{x},y,l(z),r(z)))$$

Значит функция $u(\overline{x},y)$ частично рекурсивна, а потому частично рекурсивны и функции $f_1(\overline{x},y),f_2(\overline{x},y)$.

§ 6 Вычисление по Тьюрингу частично рекурсивных функций

Во введении отмечалось, что различные уточнения понятия алгоритма определяют один и тот же класс вычислимых функций. Этот факт выше был установлен для класса **Ч** - частично рекурсивных функций и класса **Е**-функций, вычислимых на машинах произвольного доступа. Теперь это будет установлено для класса **Т** - функций, вычислимых по Тьюрингу. В данном разделе будет установлено, что справедливо включение $\mathbf{Ч} \subseteq \mathbf{Т}$, а в следующем разделе - обратное включение $\mathbf{Т} \subseteq \mathbf{Ч}$.

Утверждение.

Всякая частично рекурсивная функция вычислима на подходящей машине Тьюринга.

Док-во.

Поскольку частично рекурсивные функции получаются из базисных функций 0(x),s(x), $I_m^n(x_1,\ldots,x_n)$ с помощью применения конечного числа раз операций суперпозиции (S),рекурсии (R) и минимизации(M), то достаточно доказать вычислимость по Тьюрингу базисных функций и указанных операций.

а) Вычислимость базисных функций .

Функция 0(ч) вычисляется тривиально. Начальная конфигурация $q_1$11..11(x+1 раз) должна переводиться в конфигурацию $q_0$1. Это делает машина

$$\text{T: } q_1 1 \to q_1 \lambda R$$
$$q_1 \lambda \to q_0 1 E$$

функция S(x) также вычисляется тривиально. Начальная конфигурация $q_1$11..11(x+1 раз) должна переводиться в конфигурацию $q_0$11…11(x+2 раз).

Это делает мажина

$$\text{T: } q_1 1 \to q_1 1 L$$
$$q_1 \lambda \to q_0 1 E$$

Для вычисления функции $I_m^n(x_1,\ldots,x_n)$ начальная конфигурация $q_1$11..11*1…1*…*1…1(x_1+1,x_2+1,...,x_m+1 раз соответственно) должна переводиться в конфигурацию $q_0$11…11(x_m+1 раз).Это может выполнить машина, которая стирает все знаки 1 , *,при этом "считает" до m и оставляет m-ую группу без изменения и снова стирает все знаки 1 , * . Ясно, что это может выполнить подходящая машина Тьюринга.

б) Вычислимость операции суперпозиции.

Пусть даны функции $g(y_1,\ldots,y_n)$ и $f_1(x_1,\ldots,x_m),\ldots,f_n(x_1,\ldots,x_m)$ вычислимые по Тьюрингу, Нужно показать вычислимость функции

$$h(x_1,\ldots,x_m) = g(f_1(x_1,\ldots,x_m),\ldots,f_n(x_1,\ldots,x_m))$$

Это можно реализовать в соответствии со схемой:

(слово из x+1 палочек обозначим $\bar{x}$)

$\bar{x}_1 * \ldots * \bar{x}_m \to \bar{f}_1(x_1,\ldots,x_m) * \ldots * \bar{f}_n(x_1,\ldots,x_m) \to \bar{g}(f_1,\ldots,f_n)$ (1)

Пусть $M_1,\ldots,M_n$ - машины, вычисляющие функции $f_1,\ldots,f_n$ соответственно. Тогда первый шаг в схеме (1) может быть выполнен машиной $M_1 * \ldots * M_n$ (в обозначениях раздела 2). Второй шаг в схеме (1) может быть выполнен машиной M_g , вычислявшей функцию g. Следовательно, функция h вычисляется композицией $M_g \circ (M_1 * \ldots * M_n)$

в) Вычислимость операции рекурсии.

Ограничимся для простоты реализацией схемы

f(0)=a (2)

f(x+1)=g(x,f(x))

Общий случай рассматривается аналогично.

Вычисление $f(x)$ осуществляется циклами. Для цикла $i \in [0, x]$ на ленте записано $\bar{x}_1 * \bar{x}_2 * \bar{x}_3 = \overline{x - i} * \bar{i} * f(i)$ по условию $f(i+1) = g(i, f(i))$ и в ходе реализации цикла i+1 данное слово преобразуется в слово $\overline{x_1 \dot{-} 1} * \overline{x_2 + 1} * \overline{g(x_2, x_3)}$, если $x_1>1$. Если же $x_1=0$ то выдается $f(x) = x_3$.Чтобы реализовать эту процедуру,нужно иметь программы следующих машин (их существование очевидно):

- машина $M_Л$ по слову $\bar{x}_1 * \bar{x}_2 * \bar{x}_3$ выдается символ И , если $x_1=0$, и символ Л в противном случае.

- машина M_0 оставляет всякое слово без изменения;

-машина M_1 по слову $\bar{x}_1 * \bar{x}_2 * \bar{x}_3$ выдает $\overline{x_1 \dot{-} 1}$ (нужно стереть $* \bar{x}_2 * \bar{x}_3$ и в слове $\bar{x}_1$, стирает одну палочку);

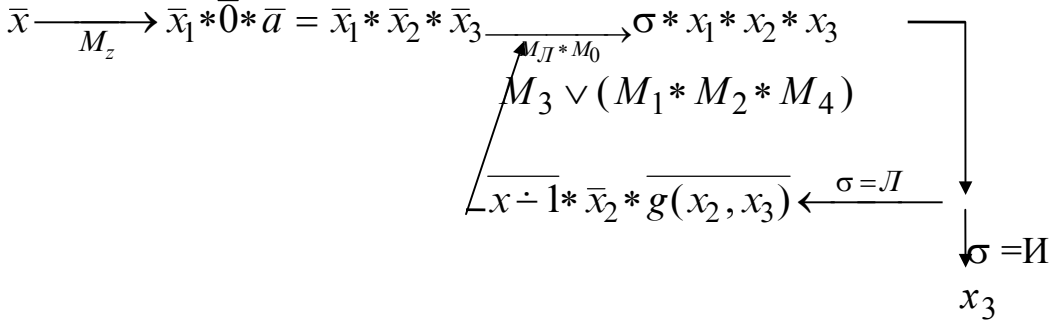
- машина M_2 по слову $\bar{x}_1 * \bar{x}_2 * \bar{x}_3$ выдает $\overline{x_1 + 1}$ (нужно стереть $\bar{x}_1 *$ и $* \bar{x}_1$ и к слову $\bar{x}_2$ приписать одну палочку);

- машина M_3 по слову $\bar{x}_1 * \bar{x}_2 * \bar{x}_3$ выдает $\bar{x}_3$ (нужно стереть $\bar{x}_1 * \bar{x}_2 *$);

- машина M_4 по слову $\bar{x}_1 * \bar{x}_2 * \bar{x}_3$ выдает $\overline{g(x_2, x_3)}$ (нужно стереть $\bar{x}_1 *$ и вычислить $g(x_2, x_3)$);

- машина M_z , осуществляющая вход в цикл, которая по входу $\bar{x}$ выдает $\bar{x}_1 * \bar{0} * \bar{a}$. (нужно дописать постоянное слово $* \bar{0} * \bar{a}$).

Вычисление функции $f(x)$ проходит в соответствии со схемой (в обозначениях раздела 2)



Г) Вычислимость операции минимизации.

Ограничимся для простоты реализацией операции минимизации вида

$$f(x) = \mu_y(g(x,y) = 0)$$

Общий случай разбирается аналогично.

Вычисление $f(x)$ осуществляется циклами. После цикла i на ленте записано $\bar{x}*\bar{i}$. В ходе цикла i+1 вычисляется $g(x,i)$ и проверяется условие $g(x,i) = 0$? Если оно выполнено, то i выдается в качестве ответа. Если оно не выполнено, то слово $\bar{x}*\bar{i}$ преобразуется в слово $\bar{x}*\overline{i+1}$.

Для реализации этой процедуры нужны программы сдедущих машин (их существование очевидно):

-машина $M_Л$ по слову $\bar{x}_1*\bar{x}_2*\bar{x}_3$ выдает И ,если x_1=0, и Л ,else;

-машина M_0 оставляет слово без изменения;

-машина M_1 по слову $\bar{x}_1*\bar{x}_2$ находит $g(x_1,x_2)$;

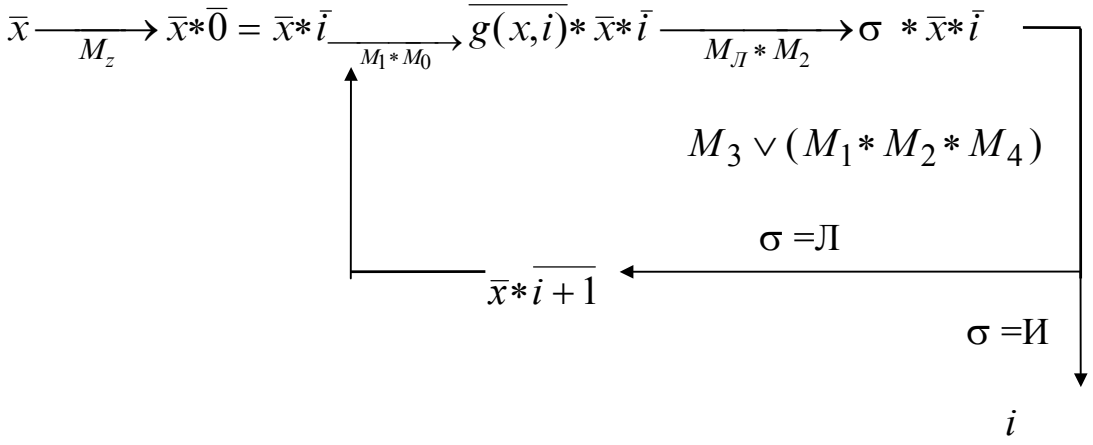
-машина M_2 переводит слово $\bar{x}_1*\bar{x}_2*\bar{x}_3$ в слово $\bar{x}_2*\bar{x}_3$;

-машина M_3 по слову $\bar{x}*\bar{i}$ выдает $\bar{i}$;

-машина M_4 переводит слово $\bar{x}*\bar{i}$ в слово $\bar{x}*\overline{i+1}$;

-машина M_Z осуществляет вход и цикл.Слово $\bar{x}$ она преобразует в слово $\bar{x}*\bar{0}$.

Схема вычисления $f(x)$ осуществляется в соответствии со схемой (в обозначениях раздела 2).



Тем самым установлено, что частично рекурсивные функции вычислимы по Тьюрингу и, следовательно, имеет место включение **Ч⊆Т.**

§ 7.Арифметизация машин Тьюринга и частичная рекурсивность функций, вычислимых по Тьюрингу

В данном разделе будет доказано, что всякая Функция, вычислимая по Тьюрингу, является частично рекурсивной. Это будет сделано на основе приема, распространенного в теории алгоритмов и называемого арифметизацией..

Данный прием заключается в том, что нечисловые объекты -в данном случае слова в конечном алфавите-кодируются натуральными числами, а преобразование этих объектов заменяются арифметическими операциями над их номерами.

Рассмотрим машину Тьюринга Т. Пусть $A = \{a_0, a_1, a_2, \ldots, a_{k-1}\}$ - ее внешний алфавит, причем считаем a_0=Л , a_1=И , т.к. рассматриваем унарные представления чисел. Пусть $Q = \{q_0, q_1, q_2, \ldots, a_{r-1}\}$ внутренний алфавит, причем q_0, q_1 - являются заключительным и начальным состояниями.

Кодирование алфавитов.

$$a_i \leftrightarrow i \in \overline{0, k-1}$$
$$q_j \leftrightarrow j \in \overline{0, r-1} \qquad (1)$$

Поскольку кодирование алфавитов числами проведено, то впредь не будем различать буквы и соответствувщие числа.

Кодирование Конфигураций:

Пусть дана конфигурация машины

…	b_2	b_1	b_0	a_i	c_0	c_1	c_2	…

$$\uparrow$$
$$q_j$$

(2)

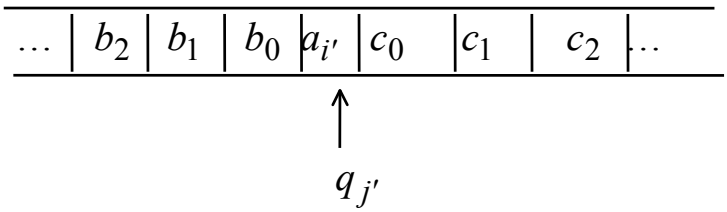
Тогда закодируем ее четверкой чисел $\langle j, a, m, n \rangle$,где

$$j \leftrightarrow q_j$$
$$a \leftrightarrow a_i$$
$$m = \sum_{i=0}^{\infty} b_i k^i \qquad (3)$$
$$n = \sum_{i=0}^{\infty} c_i k^i$$

Поскольку рассматриваются конечные конфигурации, то суммы в (3) содержат конечное число слагаемых, отличных от нуля. Далее, в силу того, что всякое натуральное число однозначно представляется k-ичной записью, то данная четверка чисел однозначно определяет конфигурацию машины.

Преобразование четверок при выполнении команд.

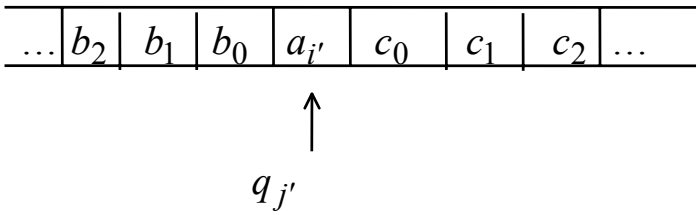
Пусть выполняется команда $q_j a_i \to q_{j'} a_{i'} R$. Тогда конфигурация (2) перейдет в кснфигурацию



Данная конфигурация характеризуется четверкой чисел $\langle j', a', m', n' \rangle$,где

$$a' = c_0 = res(n,k)$$
$$m' = a_{i'} + bk + \ldots = a_{i'} + mk \qquad (4)$$
$$n' = \left[\frac{n}{k}\right]$$

Если выполняется команда $q_j a_i \to q_{j'} a_{i'} L$, то конфигурация(2) перейдет в конфигурацию



Данная конфигурация характеризуется четверкой чисел $\langle j', a', m', n' \rangle$,где

$$a' = b_0 = res(m,k)$$
$$m' = \left[\frac{m}{k}\right] \qquad (5)$$
$$n' = a_{i'} + nk$$

Если выполняется команда $q_j a_i \to q_{j'} a_{i'} E$, то для новой четверки $\langle j', a', m', n' \rangle$ выполнено

$$a' = a_{i'}$$
$$m' = m \qquad (6)$$
$$n' = n$$

Определим теперь числовые функции $S(j,a_j), Q(j,a_j), A(j,a_j)$, которые определяют сдвиг головки, новое состояние и новую букву на ленте. Для произвольной команды $q_j a_i \to q_{j'} a_{i'} S$

положим

$$S(j,a_j) = \begin{cases} 0 \quad , \quad если \quad S = R \quad j \in \{1,\ldots,r-1\} \\ 1 \quad , \quad если \quad S = L \quad a_i \in \{1,\ldots,k-1\} \\ 2 \quad , \quad если \quad S = E \end{cases}$$

на остальных парах положим $S(j,a_j)$=0.

Аналогично

$$Q(j,a_j)=\begin{cases} j' \quad , \quad если \quad j\in\{1,\dots,r-1\} \quad a_i\in\{1,\dots,k-1\} \\ 0 \quad , \quad в \quad остальных \quad слу\div аях \end{cases} \qquad (8)$$

$$A(j,a_j)=\begin{cases} a_{i'} \quad , \quad если \quad j\in\{1,\dots,r-1\} \quad a_i\in\{1,\dots,k-1\} \\ 0 \quad , \quad в \quad остальных \quad слу\div аях \end{cases} \qquad (9)$$

По доказанному выше функции S,Q,A -общерекурсивны ,т.к. они отличны от нуля на конечном числе пар чисел.

Используя данные функции, преобразование четверок чисел можно записать в виде

$$j'=Q(j,a) \qquad (10)$$

$$a'=res(n,k)\overline{sg}S(j,a)+res(m,k)\overline{sg}|S(j,a)-1|+A(j,a)\overline{sg}|S(j,a)-2|$$

$$m'=(mk+A(j,a))\overline{sg}S(j,a)+\left[\frac{m}{k}\right]\overline{sg}|S(j,a)-1|+m\overline{sg}|S(j,a)-2|$$

$$n'=\left[\frac{n}{k}\right]\overline{sg}S(j,a)+(nk+A(j,a))\overline{sg}|S(j,a)-1|+n\overline{sg}|S(j,a)-2|$$

Подчеркнем, что полученные функции j',a',m',n' являются общерекурсивными т.к. они суть суперпозиции общерекурсивных функций.

Определим теперь следующие функции

$Q(t,j,a,m,n)$ -номер состояни, в которое переходит машина из начальной конфигурации с четверкой j',a',m',n' через t тактов.

$\widetilde{A}(t,j,a,m,n)$ - номер считываемой буквы на ленте через t тактов.

$\widetilde{M}(t,j,a,m,n)$ -значение m через t тактов.

$\widetilde{N}(t,j,a,m,n)$ -значение n через t тактов.

Ясно, что при t=0 имеем

$$\begin{aligned} Q(0,j,a,m,n)&=j \\ \widetilde{A}(0,j,a,m,n)&=a \qquad (11) \\ \widetilde{M}(0,j,a,m,n)&=m \\ \widetilde{N}(0,j,a,m,n)&=n \end{aligned}$$

При переходе от t к t+1 справедливы соотношения

$$\widetilde{Q}(t+1,j,a,m,n)=Q(\widetilde{Q}(t,j,a,m,n),\widetilde{A}(t,j,a,m,n))$$

$$\widetilde{A}(t+1,j,a,m,n)=a'(Q'(t,j,a,m,n),.\widetilde{A}(t,j,a,m,n),\widetilde{M}(t,j,a,m,n),\widetilde{N}(t,j,a,m,n)$$

$$\widetilde{M}(t+1,j,a,m,n)=m'(Q'(t,j,a,m,n),\widetilde{A}(t,j,a,m,n),\widetilde{M}(t,j,a,m,n),\widetilde{N}(t,j,a,m,n))$$
$$\widetilde{N}(t+1,j,a,m,n)=n'(Q'(t,j,a,m,n),.\widetilde{A}(t,j,a,m,n),\widetilde{M}(t,j,a,m,n),\widetilde{N}(t,j,a,m,n))$$

(это соотношение 12)

Соотношения (11) и(12) определяют по схеме совместной рекурсии функции $\widetilde{Q},\widetilde{A},\widetilde{M},\widetilde{N}$. Поскольку функции $\widetilde{Q},a',m',n'$ общерекурсивны, то такими будут и функции $\widetilde{Q},\widetilde{A},\widetilde{M},\widetilde{N}$.

Частичная рекурсия функций , вычислимых на машине Тьюринга.

Ограничимся рассмотрением функций одного артумента. Пусть функция $f(x)$ вычислима на машине Тьюринга Т. Произведем арифметизацию данной машины.

Начальная конфигурация $q_1 1^{x+1}$ характеризуется четверкой $\langle 1,1,0,n(x)\rangle$,где

$$n(x)=1+k+k^2+\ldots+k^{x-1}=\frac{k^x-1}{k-1}=\left[\frac{k^x\div 1}{k\div 1}\right] \qquad (13)$$

Элементы четверки через t тактов определяются соотношениями:

$$\begin{aligned} q(t,x)&=\widetilde{Q}(t,1,1,0,n(x))\\ a(t,x)&=\widetilde{A}(t,1,1,0,n(x))\\ m(t,x)&=\widetilde{M}(t,1,1,0,n(x))\\ n(t,x)&=\widetilde{N}(t,1,1,0,n(x)) \end{aligned} \qquad (14)$$

Заметим, что в (13) и (14) мы имеем общерекурсивные фуннкции.

По определению , если $f(x)$ определено, то машина Т останавливается в момент t_0 ,когда выполнено
$q(t,x)=0$,т.е.можно написать

$$t_0(x)=\mu_t(q(t,x)=0) \qquad (15)$$

Если $f(x)$ не определено, то $q(t,x)\neq 0 \quad \forall x$ и тогда значение t_0 в (15) не определено. Заметим, что функция $t_0(x)$ частично рекурсивна, поскольку получена из общерекурсивной применением операции минимизации. Если известно значение $t_0(x)$, то можно определить
все элементы четверки

$$\begin{aligned} q_0(x)&=q(t_0(x),x)\\ a_0(x)&=a(t_0(x),x)\\ m_0(x)&=m(t_0(x),x)\\ n_0(x)&=n(t_0(x),x) \end{aligned} \qquad (16)$$

которые являются частично рекурсивными функциями.

Если значение $f(x)$ определено, то заключительная конфигурация имеет вид $q_0 1^{f(x)+1}$.

Она характеризуется четверкой $\langle 0,1,0,n_0(x)\rangle$

$$n_0(x) = 1 + k + k^2 + \ldots + k^{f(x)-1} = \left[\frac{k^{f(x)} \dot{-} 1}{k \dot{-} 1} \right] \qquad (17)$$

Таким образом можно написать

$$f(x) = \mu_s \left(\left| n_0(x) - \left[\frac{k^s \dot{-} 1}{k \dot{-} 1} \right] \right| = 0 \right) \qquad (18)$$

Если значение $f(x)$ не определено, то не определено и значение $t_0(x)$ и соответственно $n_0(x) = n(t_0(x), x)$ и тогда формула (18) дает неопределенное значение. Из соотношения (18) следует, что функция $f(x)$ является частично рекурсивной.

Результат данного раздела служит серьезным доводом в пользу тезисов Тьюринга и Черча, поэтому доказательства с использованием данных тезисов считаются достаточно строгими и обоснованными.

§ 8 Нормальные алгоритмы

1^0) . В данном разделе дается представление об одном подходе к уточнению понятия алгоритма, предложенном А. А.Марковым и называемом нормальные алгоритмы (в авторской транскрипции - алгорифмы). Данный подход связывает неформальное понятие эффективности с переработкой слов в некотором конечном алфавите в соответствии с определенными правилами. В качестве элементарного преобразования используется подстановка одного слова вместо другого. Множество таких подстановок определяет схему алгоритма. Правила, определяющие порядок применения подстановок, а также правила останова являются общими для всех нормальных алгоритмов. Дадим формальные определения. Пусть $A = \{a_1, \ldots, a_n\}$ - алфавит. Если P,Q - слова в алфавите A (возможно пустые), то выражения $P \to Q$, $P \to (\cdot)Q$ называются Формулами подстановки в алфавите A (предполагается, что знаки $\to$, $\cdot$, не входят в алфавит A). При этом формула $P \to Q$ называется простой, а формула $P \to \cdot Q$ заключительной .Обозначим $P \to (\cdot)Q$ любую из этих формул. Произвольная конечная последовательность таких формул называется схемой S_a нормального алгоритма a. Значит схема нормального алгоритма имеет вид:

$$S_a: \begin{array}{l} P_1 \to (\cdot)Q_1 \\ P_2 \to (\cdot)Q_2 \\ \ldots \\ P_m \to (\cdot)Q_m \end{array} \qquad (1)$$

Схема S_a определяет следующий алгоритм a, перерабатывающий слова в алфавите A (т,е, вычисляющий словарную функцию на словах в алфавите A). Говорим, что слово P входит в слово W , если существуют слова V_1 и V_2 (возможно, пустые)) такие, что

$$W=V_1PV_2 \qquad (2)$$

Если слово V_1 имеет наименьшую длину из всех слов вида (2), то говорят о первом вхождении P в слово W.

Пусть дано произвольное слово K в алфавите A. Возможны следующие случаи:

1) Ни одно из слов $P_1,...,P_m$ не входит в слово K.

В этом случае говорим, что схема S_a неприменима к K_1 и пишем S_a:K .

2) Среди слов $P_1,...,P_m$ существует P_i , входящее в K.

Пусть t-минимальное число, такое, что P_t входит в K , и пусть $K=V_1P_tV_2$,где V_1 имеет минимальную длину (т.е. берется первое вхождение P_t в K.)

Тогда определим слово $W=V_1Q_tV_2$

В этом случае говорим, что схема S_a применима к K и пишем

$$\begin{array}{l} S_a : K \Rightarrow W \\ S_a : K \Rightarrow \cdot W \end{array} \qquad (3)$$

в зависимости от того, применялась простая формула или заключительная соответственно.

Теперь пишем $S_a : K \Rightarrow W$,если существует конечная последовательность слов $W_0, W_1, \ldots, W_k$ в алфавите А такая, что $K = W_0$, $W = W_k$ и выполнено

$$W_0 \Rightarrow W_1, W_2 \Rightarrow W_3, \ldots, W_{k-1} \Rightarrow \cdot W_k \qquad (4)$$

либо $W_{k-1} \Rightarrow W_k$

В первом случае пишем также $S_a : K \Rightarrow \cdot W$.

Говорим, что нормальный алгоритм a со схемой S_a вычисляет словарную функцию $F_s: A^* \to A^*$, где A^*-множество слов в алфавите А, если для любых слов $P, Q \in A^*$ имеем $F_s(P) = Q \Leftrightarrow \begin{cases} \textit{либо} \quad S_a: P \Rightarrow Q \quad \textit{и} \quad S_a: Q \\ \textit{либо} \quad S_a: P \Rightarrow \cdot Q \end{cases} \dashv$

Работа нормального алгоритма может быть описана так. Если дано слово P , то находим в схеме алгоритма S_a первую формулу $P \to (\cdot)Q_t$, такую, что P_t входит в P, и производим замену первого вхождения P_t словом Q_t .Пусть W_1 результат этой подстановки. Если применяемая формула $P \to \cdot Q_t$ - заключительная, то работа алгоритма заканчивается и слово W_1 есть результат работы алгоритма. Если применяемая формула $P \to Q_t$ простая, то, к слову W_1 применяем описанную процедуру. Если на некотором шаге получено слово W_i , к которому неприменима схема алгоритма S_a(т.е. ни одно из $P_j, j \in \overline{1,m}$ не входит в W_i ,то работа алгоритма заканчивается, и W_i есть результат работы алгоритма. Если описанный процесс не заканчивается то, по определению, алгоритм неприменим к слову P.

Словарная функция f в алфавите А (т.е. типа $f: A^* \to A^*$) называется вычислимой по Маркову, если существует схема нормального алгоритма S_a в алфавите $B \supseteq A$, вычисляющая f , т.е. $f = F_s$. Класс функций, вычислимых по Маркову, обозначим **M.** Рассмотрим несколько примеров.

1) $A = \{a_1, a_2\}$

Схема S_a: $a_1 \to \cdot \lambda$
$a_2 \to a_2$

Данный алгоритм оставляет пустое слово λ без изменения и всякое слово P в алфавите А переводит в слово Q , полученное из P путем вычеркивания первого вхождения буквы a_1. Алгоритм a неприменим к словам, не содержащим вхождений буквы a_1.

2) $A = \{a_1, a_2, \ldots, a_n\}$

Схема S_a : $a_1 \to \lambda$
$a_2 \to \lambda$
$\ldots$
$a_n \to \lambda$

Данный алгоритм переводит всякое слово P в алфавите А в пустое слово.

3) $A = \{|\}$

Схема $S_a: \lambda \to \cdot|$

Данный алгоритм переводит всякое слово $P = \overbrace{||\ldots|}^{x}$ в слово $Q = \underbrace{||\ldots|}_{x+1}$.

Если представить натуральное число
n словом $|^{n+1}$, то данный алгоритм вычисляет функцию f(n)=n+1.

4) $A = \{a_1, a_2, \ldots, a_n\}$

Схема $S_a: \lambda \to \cdot\lambda$.

Данный алгоритм вычисляет функцию $F_s(P) = P$, $\forall P$.Если же взять схему $S_a: \lambda \to \lambda$
, то данный алгоритм вычисляет нигде не определенную функцию.

5) $A = \{a_1, a_2, \ldots, a_n\}$.Если $P = a_{i_1} \ldots a_{i_k}$,то обращением слова P назовем слово $P^{-1} = a_{i_k} \ldots a_{i_1}$.

Рассмотрим алфавит $B = A \cup \{\alpha, \beta\}$ и соответственно схему S_a (α, β - новые буквы).

$$1.\ \alpha\alpha \to \beta$$
$$2.\ \beta\alpha \to \alpha\beta, \forall a \in A$$
$$3.\ \beta\alpha \to \beta$$
$$4.\ \beta \to \cdot\lambda$$
$$5.\ \alpha\, ab \to b\alpha\, a, \forall a, b \in A$$
$$6.\ \lambda \to \alpha$$

Покажем, что данный алгоритм a осуществляет обращение слов в алфавите А.

Док-во.

Пусть $P = a_{j_0} \ldots a_{j_k}$ слово в алфавите А. Тогда

$$P \xrightarrow[(n\ o\ 6)]{} \alpha P \xrightarrow[(n\ o5)]{} a_{j_1} \alpha\, a_{j_0} \ldots a_{j_k} \xrightarrow[(n\ o5)]{} a_{j_1} a_{j_2} \alpha\, a_{j_0} a_{j_3} \ldots a_{j_k} \xrightarrow[(n\ o5)]{}$$
$$\to a_{j_1}\, a_{j_2} \ldots a_{j_k} \alpha\, a_{j_0} \xrightarrow[(n\ o6)]{} \alpha\, a_{j_1}\, a_{j_2} \ldots a_{j_k} \alpha\, a_{j_0} \to \ldots \to a_{j_2}\, a_{j_3} \ldots a_{j_k} \alpha\, a_{j_1} \alpha\, a_{j_0}$$

Теперь, повторяя этот процесс,получим :

$$\Rightarrow \alpha\, a_{j_k} \alpha\, a_{j_{k-1}} \ldots \alpha\, a_{j_1} \alpha\, a_{j_0} \xrightarrow[(n\ o\ 6)]{} \alpha\alpha\, a_{j_k} \alpha\, a_{j_{k-1}} \ldots \alpha\, a_{j_1} \alpha\, a_{j_0} \xrightarrow[(n\ o\ 4)]{}$$
$$\Rightarrow \beta\, a_{j_k} \alpha\, a_{j_{k-1}} \ldots \alpha\, a_{j_1} \alpha\, a_{j_0} \xrightarrow[(n\ o\ 2)]{} a_{j_k} \beta\, \alpha\, a_{j_{k-1}} \ldots \alpha\, a_{j_1} \alpha\, a_{j_0} \xrightarrow[(n\ o\ 3)]{}$$
$$\Rightarrow a_{j_k} \beta\, a_{j_{k-1}} \ldots \alpha\, a_{j_1} \alpha\, a_{j_0} \Rightarrow \ldots (n\ o\ 2,3,4) \ldots \Rightarrow a_{j_k} a_{j_{k-1}} \ldots a_{j_0}$$

2^0) Для нормальных алгоритмов разработана техника програмирования, позволяющая осуществлять операции композиции алгоритмов, реализовывать операторы "если Ф , то выполнить F_1 , иначе F_2 ", "пока Ф выполнить F_1 , иначе F_2 ".Следовательно, класс функций **M** достаточно широк. Много конкретных

нормальных алгоритмов и соответствующая техника программирования представлены в книге [8].В связи с этим Марковым А.А.был выдвинут принцип нормализации, который заключается в том, что все алгоритмы исчерпываются нормальными алгоритмами или, что то же самое - всякий алгоритм нормализуем. Принятие данного принципа позволяет истолковывать утверждения о несуществовании нормальных алгоритмов для решения конкретных задач как утверждения о несуществовании алгоритмов вообще. Данный принцип эквивалентен тезисам Черча и Тьюринга поскольку справедлива

Теорема

Класс функций **М** вычислимых по Маркову, совпадает с классом функций **Т** , вычислимых по Тьюрингу, и, следовательно, с классом частично рекурсивных функций **Ч** и с классом МПД-вычислимых функций **Е** .

Доказательство совпадения классов **М** и **Ч** проводится по той же схеме, что и приведенное выше доказательство совпадения классов **Т** и **Ч**. Полностью оно приведено в книге [19], гл. 5.

§ 9 Нумерация алгоритмов

В данном разделе будут приведены эффективные кодирования натуральными числами множества всех алгоритмов для каждой из рассматриваемых моделей алгоритмов: машин Тьюринга , МПД-программ, частично рекурсивных функций. Данные результаты относятся к числу фундаментальных, т.к. они используются для получения многих важных фактов, в частности, для установления невычислимости ряда конкретных функций.

1°) Нумерация машин Тьюринга.

Зафиксируем счетные множества символов $\{a_0, a_1, \ldots, a_i, \ldots\}$ $\{q_0, q_1, \ldots, q_j, \ldots\}$ и будем считать, что внешние алфавиты и алфавиты внутренних состояний всех машин Тьюринга берутся из этих множеств. При этом будем считать, что a_0 принадлежит всем внешним алфавитам машин и интерпретируется как пустой символ, а буквы q_0 и q_1 принадлежат всем внутреним алфавитам машин и всегда означают заключительное и начальное состояния соответственно. Опишем теперь единый способ представления информации о машинах с помощью кодирования. Каждому символу из множества $\{L, R, E, a_0, a_1, \ldots, a_i, \ldots q_0, q_1, \ldots, q_j, \ldots\}$ поставим в соответствие двоичный набор согласно таблице:

	Символ	Код	Число нулей в коде
Символы сдвига	R	10	1
	L	100	2
	E	1000	3
Символы алфавита ленты	a_0	10000	4
	a_1	100000	6
	.	…	…
	a_i	100...00	2i+4
	.	…	…
Символы алфавита состояний	q_0	100000	5
	q_1	10000000	7
	…	…	…
	q_i	1000…000	2i+5
	…	…	…

Таблица кодирования символов машин.

Далее , команде I машины Тьюринга T, имеющей вид $qa \to q'a'd$, ставится в соответствие двоичный набор вида

$$Код(I) = Код(q)Код(a)Код(q')Код(a')Код(d) \qquad (1)$$

в котором коды букв приписаны друг к другу. Пусть машина T имеет алфавиты $A = \{a_0, a_1, \ldots, a_m\}$ $Q = \{q_0, q_1, \ldots, q_n\}$.

Упорядочим команды машины T в соответствии с лексикографическим порядком левых частей команд: $q_1a_0, q_1a_1, q_1a_m, q_2a_0, \ldots, q_2a_m, \ldots, q_na_0, \ldots, q_na_m$.

Пусть $I_1,\ldots,I_{n(m+1)}$ - соответствующая последовательность команд. Тогда машине T поставим в соответствие двоичный набор вида

$$Код\,(T) = Код\,(I_1)Код\,(I_2)\ldots Код\,(I_{n(m+1)}) \qquad (2)$$

полученный приписыванием друг н другу кодов команд. Пример: Пусть дана машина Тьюринга T :

$$A = \{a_0, a_1\}, Q = \{q_0, q_1\}$$

$$\text{T:} \quad \begin{matrix} q_1a_0 \to q_0a_0E \\ q_1a_1 \to q_0a_1E \end{matrix} \qquad (3)$$

Тогда имеем

$$Код\,(T) = 10^710^410^510^410^310^710^610^510^610^3$$

Легко видеть, что машина T переводит конфигурации $q_1a_1{}^x$ в конфигурации $q_0a_1{}^x$ и поэтому, представляя натуральное число n как $a_1{}^{n+1}$, лолучаем, что машина T вычисляет функцию $f(x) = x$

Ясно, что указанное кодирование является алгоритмической процедурой. Имея код машины, однозначно восстанавливается множество ее команд - для этого надо выделить подслова, начинающиеся с единицы с нулями до следующей единицы. Пятерка таких подслов образует команду. Далее, легко видеть, что имеется алгоритмическая процедура, позволяющая по произвольному слову из 0,1 выяснять - будет ли это слово служить кодом некоторой машины Тьюринга. Будем теперь рассматривать код машины Тьюринга как двоичную запись натурального числа. Данное число назовем номером машины Тьюринга. Поскольку все коды начинаются с единицы , то разным кодам соответствуют разные числа.Упорядочим машины Тьюринга по возрастанию чисел , представляемых их кодами , и занумеруем их $T_0,T_1,\ldots,T_n,\ldots$. (4)

Указанное упорядочение является эффективным в том смысле , что существует алгоритм ,который по n выдает код(T_n) и существует алгоритм , который , наоборот ,по код(T_n) выдает n_T .

Если обозначить через $f_n(x)$ -одноместную функцию , которую вычисляет машина Тьюринга T_n ,то в результате получим нумерацию всех одноместных частично рекурсивных функций:

$$f_0(x), f_1(x),\ldots, f_n(x),\ldots \qquad (5)$$

Согластно доказанному ,каждая одноместная частично рекурсивная функция представлена в этой последовательности.Можно доказать ,что каждая такая функция представлена в последовательности (5) бесконечное число раз. Аналогично можно определить нумерацию n-местных функций.

2^0) Нумерация МПД-программ.

Приведем аналогичную конструкцию по нумерации МПД-программ , которая позволит получить нумерацию МПД-вычислимых функций.Сначала определим нумерацию команд.Положим

$$\alpha(Z(n)) = 4(n-1)$$

$$\alpha(S(n)) = 4(n-1)-1$$
$$\alpha(T(m,n)) = 4p(m-1,n-1)+2 \qquad (6)$$
$$\alpha(J(m,n,q)) = 4\pi(m,n,q)+3$$

Здесь $p(x,y) = 2^x(2y+1)-1$ (7)

$$\pi(x,y,z) = p(p(x-1,y-1),z-1)$$

Ясно , что функция α эффективно вычислима.Для вычисления $\alpha^{-1}(x)$ действуем так:

Находим числа u,v ,такие ,что x=4u+v где $0 \le v < 4$.Тогда имеем :

$\alpha^{-1}(x) = Z(u+1)$,если v=0

$\alpha^{-1}(x) = S(u+1)$,если v=1

$\alpha^{-1}(x) = T(l(u)+1, r(u)+1)$,если v=2

$\alpha^{-1}(x) = J(m,n,q)$, если v=3 где $(m,n,q) = \pi^{-1}(u)$

Следовательно , функция α^{-1} также эффективно вычислима.

Пусть теперь дана МПД-программа $P = I_1 \dots I_s$.Определим её номер $\gamma(P) = \tau(\alpha(I_1),\dots,\alpha(I_s))$,где

$$\tau(x_1,\dots,x_s) = 2^{x_1} + 2^{x_1+x_2+1} + 2^{x_1+x_2+x_3+2} + 2^{x_1+\dots+x_s+s-1} - 1 \qquad (8)$$

Ясно , что тем самым определена биекция γ множества программ в множество натуральных чисел , причём функции γ и γ^{-1} эффективно вычислимы ,по программе P эффективно находится её номер $n = \gamma(P)$,а по номеру n можно эффективно найти программу P , такую что $n = \gamma(P)$.Число $\gamma(P)$ называется номером программы P.Например , если $P = I_1 I_2 I_3$,где I_1=T(3,1),I_2=S(4),I_3=Z(6),то $\alpha(T(1,3)) = 18, \alpha(S(4)) = 13, \alpha(Z(6)) = 20$

Следовательно, $\gamma(P) = 2^{18} + 2^{32} + 2^{53} - 1$

Пусть теперь дано n=4127. Поскольку $4127 = 2^5 + 2^{12} - 1$,то $P = I_1 I_2$,где $\alpha(I_2) = 12-5-1 = 6$.Следовательно , I_1=S(2),I_2=T(2,1).

Разумеется , существуют и другие способы нумерации программ , для нас важна лишь эффективная вычислимость функций γ и γ^{-1} .

Таким образом получаем эффективную нумерацию МПД-программ: $P_0, P_1, P_2, \ldots, P_n, \ldots$ (9)

Теперь, имея нумерацию МПД-программ,можно занумеровать МПД-вычислимые функции. Для любого $a \in N \quad и \quad n \geq 1$ определим

$f_a^{(n)} \equiv$ n-местная функция,вычисляемая

программой с номером a.

Это дает нумерацию n-местных МПД-вычислимых функций

$$f_0^{(n)}, f_1^{(n)}, f_2^{(n)}, \ldots \qquad (10)$$

Например, если a =4127,то согласноопределению имеем $f_{4127}^{(1)}(x) = 1$, n=1

$f_{4127}^{(n)}(x_1, \ldots x_n) = x_2 + 1$, $n > 1$.

Поскольку для всякой частично рекурсивной функции $f^{(n)}$ существует вычисляющая её МПД-программа P, то $f^{(n)} = f_a^{(n)}$,где $a = \gamma(P)$.Число a называют номером(индексом) функции $f^{(n)}$.

Приведём одно применение существования нумерации частично рекурсивных функций.

Теорема.

Существуют всюду определенная функция, не являющаяся частично рекурсивной.

Док-во.

построим всюду определенную функцию φ , отличающуюся от каждой частично рекурсивной функции $f_0, f_1, \ldots, f_n, \ldots$ в перечеслении одноместных частачно рекурсивных функций.Полагаем

$$\varphi(x) = \begin{cases} f_x(x) + 1, & если\ f_x(x) \quad определена \\ 0, & else \end{cases}$$

Функция φ всюду определена и отличается от f_n при x=n, $n \in N_0$.Действительно , если $f_n(n)$ определено ,то $\varphi(n) = f_n(n) + 1$, если $f_n(n)$ не определено , то $\varphi(n)$ определено и равно 0.Поскольку φ отличается от f_n при всех $n \in N_0$,то φ не может находиться в перечислении (10) и , значит , она не может быть частично рекурсивной.

ч.т.д.

Замечание. Приведенный метод рассуждения есть пример диагональной конструкции Кантора, с помощью которой им была доназана несчетность множества действительных чисел. Данным методом можно установить нерекурсивность большого класса функций.

Например, функция

$$g(x)=\begin{cases} f_x(x)+2^x \quad , \quad если\ f_x(x) \quad определено \\ p(x) \quad , \quad если\ f_x(x) \quad не \quad определено \end{cases}$$

причем p(x) - целочисленный полином является всюду определенной и не частично рекурсивной.

3°) Универсальные функции.

Пусть F - некоторый класс функций от k переменных. Функцию $U(n,x_1,\ldots,x_k)$ от k+1 переменных называют *универсальной* для класса F , если выполнимы условия:

а) Для всякого $n \in N_0 \quad f_n(x_1,\ldots,x_k)=U(n,x_1,\ldots,x_k) \in F$

б) Для всякой $f\ (x_1,\ldots,x_k) \in F$ существует $n \in N$ такое, что

$f\ (x_1,\ldots,x_k)=U(n,x_1,\ldots,x_n)$.

Теорема 2.

Для класса $\times^1$ - одноместных частично рекурсивных функций существует универсальная частично рекурсивная функция U(n,x).

Док-во.

Определим $U(n,x)=f_{P_n}(x)$,точнее

$$U(n,x)=\begin{cases} y \quad , \quad если\ P_n(x)\downarrow y \\ не \quad определено\ , \quad else \end{cases} \qquad (11)$$

Данное соотношение определяет частичную функцию. Функция U(n,x) является частично рекурсивной. Действительно, для произвольных (n,x) нужно найти программу P_n(эффективная процедура)и применить P_n к начальной конфигурации (x,0,0,…,0,…). Если $P_n(x)\downarrow$, то положить U(n,x)=r_1, где r_1-содержимое регистра R_1 в заключительной конфигурации. Если $P_n(x)\uparrow$, то считать значение U(n,x) неопределенным. По тезису Черча функция U(n,x) частично рекурсивна. Покажем, что функция U(n,x) универсальна. Поскольку U(n,x) частично рекурсивна,то и f_n=U(n,x) для всякого $n \in N_0$ частично рекурсивна, т.к. -f_n получена подстановкой константы вместо первого аргумента. Пусть тепарь $f(x)$ - произвольная одноместная частично рекурсивная функция. Тогда по доказанному она может быть вычислена некоторой МИД-программой P. Пусть $n=\gamma(P)$. Следовательно, $f=f_{P_n}$ и значит $f=U(n,x)$

ч.т.д.

Следствие.

Для всякого $k \geq 1$ существует частично рекурсивная функция $U^k(n,x_1,\ldots,x_k)$ универсальная для всех k-местных частично рекурсивных функций. Это делается с использованием нумерационных функций.

Полагаем $\quad U^2(n,x_1,x_2)=U(n,p(x_1,x_2)) \qquad (12)$

$$U^3(n,x_1,x_2,x_3)=U(n,p(p(x_1,x_2),x_3))$$

и так далее.

Покажем, например, что функция $U^2(n, x_1, x_2)$ удовлетворяет нужным условиям. Во-первых, функция U^2 - частично рекурсивна , т.к. является суперпозицией частично рекурсивных функций. Во-вторых, функция $f_n(x_1, x_2) = U^2(n, p(x_1, x_2))$ - частично рекурсивна, т.к. получается из частично рекурсивной подстановкой константы. Пусть теперь $f\ (x_1, x_2)$ произвольная частично рекурсивная функция. Образуем функцию $g(x) = f(l(x), r(x))$,где l, r - нумерационные функции. Т.к. $g(x)$ - частично рекурсивна, то существует n , такое, что $g(x) = U(n, x)$

Теперь положим $x = p(x_1, x_2)$ и тогда имеем

$f(x_1, x_2) = g(p(x_1, x_2)) = U(n, p(x_1, x_2)) = U^2(n, x_1, x_2)$.

Представляет интерес вопрос о существовании универсальной функции для других рассмотренных выше классов $\mathbf{Ч_0}$ и $\mathbf{Ч_{пр}}$ - общерекурсивных и примитивно рекурсивных функций соответственно.

Теорема 3.

Не существует общерекурсивной функции $U^k(n, x_1, \ldots, x_k)$, универсальной для класса $\times_0^k$ - k -местных общерекурсивных функций при любом $k \geq 1$.

Док-во.

Пусть, наоборот, существует такая функция $U^k(n, x_1, \ldots, x_k)$ для некоторого к. Образуем функцию $f(x_1, x_2 \ldots, x_k) = U^k(x_1, x_1, x_2 \ldots, x_k) + 1$. Согласно свойству универсальности существует n_0, такое, что

$U^k(n_0, x_1, \ldots, x_k) = f(x_1, x_2 \ldots, x_k) = U^k(x_1, x_1, x_2 \ldots, x_k) + 1$

Поскольку данные функции всюду определены, то они определены и при $x_1 = \ldots = x_k = n_0$. Тогда получаем противоречивое равенство

$U^k(n_0, n_0 \ldots, n_0) = U^k(n_0, n_0 \ldots, n_0) + 1$.

Значите предположение о существовании универсальной функции ложно.

ч.т.д.

В то же время справедлива.

Теорема 4.

Для каждого $k \in N$ класс всех k -местных примитивно рекурсивных функций имеет общерекурсивную универсальную функцию.

Доказательство данной теоремы довольно громоздко. Полностью оно приведено в [7] ,§5.

Заметим, что из данной теоремы следует, что класс общерекурсивных функций шире класса примитивно рекурсивных функций, т.к. универсальная функция не может быть примитивно рекурсивной (доказать) и является обшерекурсивной.

Дадим еще одно применение универсальной функции. Пусть $f: N_0 \to N_0$ частичная функция. Функцию f_0 будем называть доопределением f , если f_0

всюду определена и совпадает с f в ее области определения. Покажем, что рассмотрение частичных вычислимых функций вызвано существом дела, а именно - существуют частичные вычислимые функции, любое доопределение которых делает их невычисляемыми.

Теорема 5.

Существует частично рекурсивная функция $f(x)$, которая не может быть доопределена до общерекурсивной.

Док-во.

Рассмотрим функцию $f(x) = \overline{sg}U(x,x)$, где U

универсальная функция. Данная функция частично рекурсивна ,т.к. она получается суперпозицией частично рекурсивных функций. Предположим ,что существует общерекурсивная функция $f_0(x)$,которая является доопределением для $f(x)$.По свойству универсальности U
сушествует n ,такое что $f_0(x) = U(n,x)$.Поскольку $f_0(x)$
всюду определена ,то она определена при x=n, и тогда значение U(n,n) определено и, следовательно ,определено значение $f(n) = \overline{sg}U(n,n)$. Поскольку $f_0(x)$ есть доопределение $f(x)$,то в области определения их значиния должны совпадать , Поэтому имеем $f(n) = \overline{sg}U(n,n) = U(n,n) = f_0(n)$.Однако , последнее равенство дает противоречие , т.к. если $U(n,n) = 0$,то $\overline{sg}U(n,n) = 1$,если $U(n,n) \neq 0$,то $\overline{sg}U(n,n) = 0$.Значит ,допущение о сущесвовании рекурсивного доопределения для функции $f(x)$ приводит к противоречию.

ч.т.д.

§10 Алгоритмически неразрешимые проблемы

1^0) В данном разделе устанавливается алгоритмическая неразрешимость ряда проблем, относящихся к теории алгоритмов. Будем рассматривать так называемые массовые проблемы. Массовая проблема представляет собой бесконечную серию индивидуальных задач. Например, индивидуальной задачей является такая: обладает ли заданным свойством Q конкретная частично рекурсивная функция. Совокупность всех таких задач (для всех частично рекурсивных функций) образует массовую проблему распознавания свойства Q. Мы ограничимся такими массовыми проблемами, в которых все индивидуальные задачи имеют двузначный ответ ("ДА" или "НЕТ").

Массовая проблема П является алгоритмически разрешимой, если соответствующая характеристическая функция f, которая определяется соотношением:

$$f(\pi) = \begin{cases} 1 \Leftrightarrow инд \quad зада \div а \quad \pi \in П \quad имеет \quad ответ \quad "ДА" \\ 0 \Leftrightarrow инд \quad зада \div а \quad \pi \in П \quad имеет \quad ответ \quad "НЕТ" \end{cases} \qquad (1)$$

является вычислимой.

Решая конкретную массовую проблему следует считаться с возможностью, что она может оказаться алгоритмически неразрешимой, поэтому необходимо иметь представление о технике доказательства неразрешимости. Основной метод, применяемый в доказательствах алгоритмической неразрешимости, базируется на следующем рассуждении. Пусть (имеем две массовые проблемы $П_1$ и $П_2$. Пусть имеется алгоритм А , который по всякой индивидуальной задаче $\pi_1 \in П_1$ строит индивидуальную задачу $\pi_2 \in П_2$, такую, что выполнено: π_1 имеет "ДА" $\Leftrightarrow \pi_2$ имеет "ДА" .

В этом случае говорят, что проблема $П_1$ сводится к проблеме $П_2$. Если проблема $П_1$ неразрешима, то проблема $П_2$ также неразрешима. Действительно, пусть это не так, и проблема $П_2$ разрешима. Тогда можно построить разрешающий алгоритм для проблемы $П_1$. Берем произвольную индивидуальную задачу $\pi_1 \in П_1$. Имея алгоритм А ,строим индивидуальную задачу $\pi_2 = A(\pi_1)$. Теперь применяем к задаче π_2 существующий по предположению разрешающий алгоритм для проблемы $П_2$. Если задача π_2 имеет ответ "ДА", то для задачи π_1 полагаем ответ "ДА", в противном случае, для задачи π_1 полагаем ответ "НЕТ". Поскольку, по условию, проблема $П_1$ неразрешима, то получим противоречие. Значит, проблема $П_2$ неразрешима. Данное рассуждение называется методом сводимости, и его применение возможно, если уже имеется запас проблем, алгоритмическая неразрешимость которых уже установлена. Приведем теперь некоторые из таких проблем.

2^0) Рассмотрим так называемую проблему самоприменимости машин Тьюринга. Она заключается в следующем. Рассматриваются, машины Тьюринга, внешние алфавиты которых содержат символы О, 1 (наряду с другими). Для каждой машины Тьюринга Т построим Код (Т) который является (0,1) - словом,

и запустим машину Т в начальной конфигурации $q_1 Код\ (T)$. Если машина Т останавливается (т.е.попадает в заключительное состояние) череэ конечное число шагов, он называется самоприменимой в противном случае - несамоприменимой.

Заметим,что имеются как самоприменимые так и несамоприменимые машины Тьюринга.Например, несамоприменимой будет машина T_1 ,у которой все команды имеют вид $q_i a_j \to q_i a_j E$ (в правых частях команд нет заключительного состояния), самоприменимой будет машина T_2 ,у которой все команды имеют вид $q_i a_j \to q_0 a_j E$ (в правых частях команд имеется заключительное состояние).

Проблема самоприменимости состоит в том , чтобы по любой машине Тьюринга Т определить ,является она самоприменимой или нет.Условимся ,что машина Тьюринга $\mathfrak{m}$ решает проблему самоприменимости ,если для любой машины Т начальную конфигурацию $q_1 Код\ (T)$ она переводит в $q_1 1$,если машина Т самоприменима ,и в $q_1 0$, если машина Т несамоприменима.

Теорема 1.

Не существует машины Тьюринга $\mathfrak{m}$, решающей проблему самоприменимости, т.е. проблема самоприменимости алгоритмически неразрешима.

Док-во.

Предположим противное, т.е. пусть существует машина Тьюринга $\mathfrak{m}$ решающая проблему самоприменимости в указанном выше смысле. Построим новую машину $\mathfrak{m}_0$, добавив новое состояние q_0' и объявив его заключительным, а также добавив новые команды для состояния q_0, которое было заключительным в $\mathfrak{m}$:

$$q_0 1 \to q_0 1 E \qquad (\alpha)$$

$$q_0 0 \to q_0' 0 E \qquad (\beta)$$

Рассмотрим теперь работу машины $\mathfrak{m}_0$. Пусть Т - произвольная машина, если Т - самоприменима, то начальная конфигурация $q_1 Код\ (T)$ перейдет с помощью команд машины $\mathfrak{m}$ через конечное число шагов в конфигурацию $q_0 1$, далее применяется команда (α), и машина, $\mathfrak{m}_0$ никогда не остановится. Если Т - несамоприменимая, то начальная конфигурация $q_0 Код\ (T)$ перейдет с помощью команд машины $\mathfrak{m}$ через конечное число шагов в конфигурацию $q_0 0$, далее применяется команда (β), и машина $\mathfrak{m}_0$ остановится. Значит машина $\mathfrak{m}_0$ применима к кодам несамоприменимых машин Т и неприменима к кодам самоприменимых машин Т. Теперь рассмотрим Код($\mathfrak{m}_0$) и применим машину $\mathfrak{m}_0$ к начальной конфигурации q_1Код($\mathfrak{m}_0$). Сама машина $\mathfrak{m}_0$ является либо

самоприменимой, либо несамоприменимой. Если $\mathfrak{m}_0$ самоприменима, то по доказанному, она никогда не остановится, начав с q_1Код($\mathfrak{m}_0$) и потому она несамоприменима. Если $\mathfrak{m}_0$ несамоприменима, то по доказанному, она останавливается через конечное число шагов, начав с q_1Код($\mathfrak{m}_0$), и потому она самоприменима. Получили противоречие, которое является следствием допущения существования машины $\mathfrak{m}$, реализующей проблему самоприменимости.

ч.т.д.

Приведем еще один пример неразрешимой проблемы. Проблемой останова называют проблему, заключающуюся в том, чтобы по любой машине Тьюринга Т и слову Р в ее внешнем алфавите узнать, применима ли Т к начальной конфигурации $q_1 P$.

Проблема останова алгоритмически неразрешима, т.к. если бы она была разрешимой, то взяв в качестве Р
слово Код(Т), мы получили бы разрешимость проблемы самоприменимости.
2^0) Приведем теперь неразрешимые проблемы, связанные о проверкой свойств частично рекурсивных функций. Пусть $U(n,x)$ - универсальная функция для одноместных частично рекурсивных функций и соответствующая ей нумерация функций

$$f_0(x), f_1(x), \ldots, f_n(x), \ldots \qquad где \quad f_n(x) = U(n,x) \qquad (2)$$

Теорема 2.

Проблема "функция f_n всюду определена", $n \in N_0$ алгоритмически неразрешима.

Док-во.

Определим характеристическую функцию данной проблемы

$$g(x) = \begin{cases} 1, & если \quad f_x \quad всюду \quad определена \\ 0, & else \end{cases} \qquad (3)$$

Определим новую функцию Ф(х)? где

$$\Phi(x) = \begin{cases} f_x(x) + 1, & если \quad f_x \quad всюду \quad определена \\ 0, & else \end{cases} \qquad (4)$$

Заметим,что функция Ф(х) всюду определена. Кроме того

$$\Phi(x) = \begin{cases} f_x(x) + 1, & если \quad g(x) = 1 \\ 0, & если \quad g(x) = 0 \end{cases} \qquad (5)$$

Нам нужно доказать невычислимость функции g. Ясно, что из вычислимости функции g следует вычислимость функции Ф. Предположим, что функция Ф вычислима. Тогда существует число n , такое, что Ф(х)=$U(n,x)$=$f_n(x)$.Поскольку Ф(х)-всюду определена, то должно быть $g(n) = 1$.Значит, имеем Ф(n)=$f_n(x)$ и Ф(n)=$f_n(x) + 1$.Поскольку при х=n

функция Ф(x) определена, то получаем противоречие. Следовательно Ф(x) невычислима, откуда следует, что функция $g(x)$ - невычислима.

ч.т.д.

Обозначим через W_n область определения функции $f_n(x)$ в последовательности (2).

Теорема 3 .

Проблема "$n \in W_n$", $n \in N_0$ алгоритмически неразрешима.

Док-во.

Рассмотрим характеристическую функцию проблемы:

$$f(x) = \begin{cases} 1 \quad , \quad если \quad x \in W_x \\ 0 \quad , \quad если \quad x \overline{\in} W_x \end{cases} \qquad (6)$$

и построим новую функцию $g(x)$, где

$$g(x) = \begin{cases} 0 \quad , \quad если \quad x \overline{\in} W_x \\ не \quad определена \quad , \quad если \quad x \in W_x \end{cases} \qquad (7)$$

Ясно, что из вычислимости функции f следует вычислимость функции g. Кроме того, справедливо соотношение $\forall\, x \in N_0$:

$g(x)$ определена $\Leftrightarrow$ $f_x(x)$ не определена (8)

Предположим теперь, что функция g вычислима. Тогда существует число m , такое, что $g = f_m$. Тогда имеем $m \in W_m \Leftrightarrow g(m)$ определено $\Leftrightarrow f_m(m)$ не определено $\Leftrightarrow m \overline{\in} W_m$. Получили противоречие, и поэтому g - невычислима и, следовательно, f невычислима.

ч.т.д.

Следствие ("Проблема применимости").

Проблема "$f_x(y)$ -определена", $x, y \in N_0$ - неразрешима.

Док-во. Если бы данная проблема была разрешима,то и проблема "$x \in W_x$" была бы разрешима. ч.т.д.

3^0) Рассмотрим теперь алгоритмический приём, позволяющий эффективно работать с номерами частично рекурсивных функций. Пусть $f(x,y)$ - вычислимая функция. Фиксируем x=a и положим $g_a(y) = f(a,y)$. Функция $g_a(y)$ - одноместная и вычислимая. Значит, существует номер l , такой, что $g_a(y) = f_l(y)$ в последовательности (2). Покажем, что данный номер находится эффективно.

Теорема 4.

Для всякой вычислимой функции $f(x,y)$ существует общерекурсивная функция s(x), такая что

$$f(x,y) = f_{s(x)}(y) \qquad (9)$$

Док-во.

Пусть F - программа МПД , вычисляющая $f(x,y)$. Фиксируем x=a , и рассмотрим программу МПД $\mathbf{Q_a}$, где

$$\mathbf{Q_a}: \quad \begin{matrix} T(1,2) \\ Z(1) \\ S(1) \\ \dots \\ S(1) \\ F \end{matrix}$$

Легко убедиться, что данная программа $\mathbf{Q_a}$ вычисляет $f(a,y)$. Пусть s(a) - номер программы $\mathbf{Q_a}$. Согласно построению $f(a,y) = f_{s(a)}(y)$. При этом, поскольку F фиксирована, то s(a) - эффективно вычислима.

ч.т.д.

Теперь, в силу вычисимости $f(x,y)$ существует n , такое, что $f(x,y) = U^2(n,x,y)$, где U^2 - универсальная функция для двухместных частично рекурсивных функций. Тогда в предыдущей конструкции при x=a имеем s=s(n,a) и выполнено $f_{s(n,a)}(y) = U(s(n,a),y)$, где U - универсальная для одноместных частично рекурсивных функций. Таким образом ,используя тезис Черта и приведенную выше конструкцию ,установлена

Теорема 5.(нумерационная)

Существует общерекурсивная функция s(n,x) ? такая,что выполнено

$$U^{(2)}(n,x,y) = U(s(n,x),y) \qquad (10)$$

Данная теорема справедлива и в более общей ситуации, которую отметим без доказательства.

Теорема 6.

Для любых $m,n \geq 1$ существует общерекурсивная функция $S_n^m(k,\bar{x})$, такая, что выполнено

$U^{(m+n)}(k,\bar{x},\bar{y}) = U^{(n)}(S_n^m(k,\bar{x}),\bar{y})$,где $k \in N_0$, $\bar{x} = (x_1,\dots,x_n)$, $\bar{y} = (y_1,\dots,y_n)$

Данное утвертдение называют S-m-n -теорема.

Теорема 7.

Проблема "$f_x = \bar{0}$", $x \in N_0$ - неразрешима. (Здесь $\bar{0}$ означает функцию, таждественно равную нулю).

Док-во.

Рассмотрим следующую функцию

$$f(x,y) = \begin{cases} 0 \quad , \quad если \quad x \in W_x \\ не \quad определена \quad , \quad если \quad x \bar{\in} W_x \end{cases}$$

Легко видеть, что функция $f(x,y)$ вычислима, поскольку универсальная функция $U(n,x)$ вычислима. По теореме существует общерекурсивная функция s(x) такая, что $f(x,y) = f_{s(x)}(y) = U(s(x),y)$. Имеем по определению

$$x \in W_x \Leftrightarrow f_{s(x)}(y) = \bar{0} \qquad (12)$$

Если бы проблема $f_z(y) = \bar{0} \quad , \quad z \in N_0$ была разрешима, то тогда проблема "$x \in W_x$" была бы разрешимой, что противоречило бы теореме 3.

ч.т.д.

Следствие.

Проблема "$f_x \equiv f_y$", $x, y \in N_0$ неразрешима.

Теорема 8. (Клини).

Для любой частично рекурсивной функции $h(x)$ существует число a, такое, что

$$f_{h(a)} = f_a(x) \qquad (13)$$

Док-во.

Пусть задана частично рекурсивная функция $h(x)$. Рассмотрим вспомогательную функцию

$f(y,x) = U(h(s(y,y),x))$,где s - функция из тождества (10), существующая по нумерационной теореме 5. Поскольку функция $f(y,x)$ - частично рекурсивна, то существует число n ,такое,что $f(y,x) = U(h(s(y,y),x)) = U^2(n,y,x)$.

По нумерационной теореме 5 имеем $f(y,x) = U(s(n,y),x)$

или $U(h(s(y,y),x)) = U(s(n,y),x)$. Теперь положим y=n и поскольку функция s общерекурсивна, то s(n,n)определено. Тогда получаем

$U(h(s(n,n),x)) = U(s(n,n),x)$.

Обозначая s(n,n)=a имеем

$$U(h(a),x) = U(a,x)$$

или

$$f_{h(a)}(x) = f_a(x)$$

ч.т.д.

4^0) Теперь используем полученные результаты для установления фундаментального факта, полученного Раисом, показывающего, что проблема проверки любого нетривиального свойства частично рекурсивных функций неразрешима.

Произвольное множество $A \subseteq N_0$ называется рекурсивным, если соответствующая характеристическая функция $\chi_a(x)$,где

$$\chi_a(x) = \begin{cases} 1 & , \quad если \quad x \in A \\ 0 & , \quad если \quad x \overline{\in} A \end{cases} \qquad (14)$$

вычислима. Ясно, что рекурсивное множество A характеризуется тем, что проблема "$x \in A$ " разрешима. Пусть F множество одноместных частично рекурсивных функций, отличных от пустого множества и от множества всех одноместных частично рекурсивных функций. (т.е.F - нетривиальное множество). Обозначим N_F - множество всех номеров функций из F.

Теорема 9. (Раис) Множестео N_F нерекурсивно.

Док-во.

Предположим ,что множество N_F рекурсивно. Тогда его дополнение $\overline{N}_F = N_0 \setminus N_F$ также рекурсивно, т.к.

$$\chi_{\overline{N}_F}(x) = 1 \dot{-} \chi_{N_F}(x)$$

По условию оба множества N_F и $\overline{N}_F$ непустые. Выберем некоторые $\alpha \in N_F$, $\beta \in \overline{N}_F$ и определим функцию

$$g(x) = \begin{cases} \alpha & , \quad если\ x \in \overline{N}_F \\ \beta & , \quad если\ x \in N_F \end{cases} \qquad (15)$$

Функция $g(x)$ общерекурсивна, т.к. выполнено равенство

$g(x) = \alpha\ \chi_{\overline{N}_F}(x) + \beta\ \chi_{N_F}(x)$

По теореме 8 для функции $g(x)$ существует число a, такое, что $f_{g(x)}(x) = f_a(x)$.

Тогда должно быть либо $a \in N_F$ либо $a \in \overline{N}_F$. Если $a \in N_F$,то, по определению, $f_a(x) \in F$ и т.к. $f_{g(a)} = f_a$, то $f_{g(a)}(x) \in F$.Однако,из $a \in N_F$ следует по (15) $g(a) = \beta \in \overline{N}_F$, т.е. $g(a) \in \overline{N}_F$ и тогда $f_{g(a)}(x) \overline{\in} F$. Получено противоречие.

Если $a \in \overline{N}_F$, то, по определению, $f_a(x) \overline{\in} F$ и в силу $f_{g(a)} = f_a$ имеем $f_{g(a)}(x) \overline{\in} F$.Однако, из $a \in \overline{N}_F$ имеем по (15) $g(a) = \alpha \in \overline{N}_F$, т.е. $g(a) \in N_F$ и значит $f_{g(a)} \in F$. Снова получено противоречие. Итак, число a не содержится ни в одном из множеств N_F и $\overline{N}_F$, чего быть не может, т.к. $N_0 = N_F \cup \overline{N}_F$.Противоречие является следствием предположения рекурсивности множества N_F.

Ч.Т.Д.

Пусть Q - некоторое нетривиальное свойство одноместных частично рекурсивных функций,т.е., имеются функции как обладающие своайством Q , так и не обладающие свойством Q.

Теорема 10

Проблема "f обладает свойством Q" $f \in E'$, неразрешима.

Док-во

Пусть проблема "f обладает свойством Q" $f \in E'$ разрешима, т.е. существует МПД, которая для лбой $f \in E'$ начальную конфигурацию (n,0,...), где n - номер функции f, через конечное число шагов переводит в заключительную конфигурацию (1,*,...), если f обладает свойством Q , и в заключительную конфигурацию (0,*,...), если f не обладает свойством Q. Тогда для множества N_{F_Q} номеров функций из E', обладающих свойством Q, можно указать разрешающую процедуру для проблемы $n \in N_{F_Q}$ соотношением $n \in N_{F_Q} \Leftrightarrow$ МПД дает результат 1.Данное соотношение показывает, что множество N_{F_Q} рекурсивно, что противоречит теореме 9.

Ч.Т.Д.

Теорема 9 имеет многочисленные применения в теоретическом программировании и позволяет доказать неразрешимость многих задач, связанных с вычислением на машинах.

Большое число алгоритмически неразрешимых проблем имеется в книге Роджерса Х.[14].

5^0) Приведем одно важное понятие теории алгоритмов, связанное с вычислимостью. Выше была установлена невычислимость характеристической функция предиката "$x \in W_x$". В то же время частичная характеристическая функция, определенная соотношением

$$f(x) = \begin{cases} 1 \quad , \quad x \in W_x \\ не \quad определена \quad , else \end{cases}$$

является вычислимой. Это следует из тезиса Черча. Поэтому говорят, что проблема "$x \in W_x$" полувычислима.

Многие проблемы, будучи неразрешимыми, являются полувычислимыми. Дядим общее

Определение.

Предикат $M(\bar{x})$ на натуральных числах (здесь $\bar{x} = (x_1, \ldots, x_n)$) называется полувычислимым, если частичная характеристическая функция f , определенная соотношением: $f(x) = \begin{cases} 1 \quad , \quad M(\bar{x}) = И \\ не \quad определена \quad , если \quad M(\bar{x}) = Л \end{cases}$ вычислима.

Замечание. В литературе также используются равносильные термины : частичная разрешимость, рекурсивная перечислимость.

Примеры:

1. Любой разрешимый предикат полувычислим.
2. Для любой вычислимой функции $g(\bar{x})$ проблема "$\bar{x} \in D(g)$" является полувычислимой, т.к. вычислима функция $\overline{1}(g(\bar{x}))$ (здесь $\overline{1}(y) = 1$, $\forall y$).
3. Проблема "$x \overline{\in} W_x$" не является полувычислимой. Действительно, если f - соответствующая частичная характеристическая функция, то $\bar{x} \in D(g) \Leftrightarrow x \overline{\in} W_x$.

Значит, $D(f)$ отличается от области определения любой одноместной вычислимой функции. Поэтому f не может быть вычислимой.

Теорема 11. Предикат $M(\bar{x})$ полувычислим тогда и только тогда, когда существует вычислимая функция $g(\bar{x})$, такая, что $M(\bar{x}) = И \Leftrightarrow \bar{x} \in D(g)$.

Док-во.

Если $M(\bar{x})$ полувычислим и f - соответствующая частичная характеристическая функция, то по определению $M(\bar{x}) = И \Leftrightarrow \bar{x} \in D(g)$. Обратное следует из примера 2.

Определение.

Множество А из N_0 называется рекурсивно перечислимым, если частичная характеристическая функция, где $f(x) = \begin{cases} 1 \quad , \quad x \in A \\ не \quad определена \quad , если \quad x \overline{\in} A \end{cases}$ вычислима.

Это равносильно тому, что предикат "$x \in A$" полувычислим. Теорема 11 может быть перефразирована так.

Теорема 12.

Множество является рекурсивно перечислимым тогда и только тогда, когда оно является областью определения одноместной вычислимой функции.

§ 11. Проблема Тождества слов в конечно определенных полугруппах и другие примечательные алгоритмически неразрешимые проблемы

1^0) Пусть даны алфавит $A = \{a_1,\dots a_n\}$ и множество пар слов $S = \{(P_i,Q_i)\}, i \in I$ в алфавите А. Пусть A^* - множество конечных слов в алфавите А. На множестве A^* определим отношение эквивалентности так.

Определим два типа элементарных преобразований (ЭП):

а) Левое Э.П.: замена в $P \in A^*$ любого вхождения слова P_i словом Q_i $(P = W_1P_iW_2 \to W_1Q_iW_2 = Q)$.

б) Правое Э.П.: замена в $P \in A^*$ любого вхождения слова Q_i словом P_i $(P = W_1Q_iW_2 \to W_1P_iW_2 = Q)$.

Если слово Q получено из P одним элементарным преобразованием, то пишем $P \to Q$.

Слова $P,Q \in A^*$ назовем эквивалентными, если существует (конечная) последовательность слов $R_1,\dots,R_k$, такая, что

$$P = R_0 \to R_1 \to \dots \to R_k = Q.$$

Если P,Q эквивалентны, то пишем $P \sim Q$. Ясно, что отношение ~ есть отношение эквивалеютности. Класс эквивалентности, содержащий P обозначим [P].Определим операцию умножения на классах

[P][Q]=[PQ]

Легко доказать, что данное определение корректно. Операция умножения классов ассоциативна ,и множество классов образует полугруппу $\Pi = \langle A, S \rangle$. Она называется
полугруппой ,заданной образующими А и определяющими соотношениями S.Если А-конечный алфавит ,то П конечно определенной.Проблема эквивалкнтности слов в конечно определенной полугруппе ставится так: Для любых двух слов $P,Q \in A^*$ требуется узнать ,эквивалентны они или нет.(т.е. выпелнено ли [P]=[Q],поэтому проблему эквивалентности слов называют проблемой тождества).
Если такой алгоритм есть, то говорят, что проблема тождества разрешима, если нет, то - неразрешима.

Можно указать ряд полугрупп П, в которых проблема эквивалентности слов разрешима.

Примеры:

1. $\Pi = \left\{a_1,\dots,a_n\ ; (a_ia_j = a_ja_i)\ , \forall i,j \in \overline{1,n}\ , i \neq j\right\}$.Ясно, что П - абелева полугруппа. Слова P ,Q эквивалентны $\Leftrightarrow$ они имеют одинаковое число вхождений каждой буквы из $A = \{a_1,\dots a_n\}$. Это дает очевидный алгоритм проверки эквивалентности любых $P,Q \in A^*$.

2. $\Pi = \{a_1, a_2, a_3 ; (a_1a_1, a_1) , (a_1a_2, a_2a_1) , (a_1a_3, a_3a_1) , (a_2a_3, a_3)\}$. Легко показать, что каждое слово из П эквивалентно слову вида $a_1^{\,k}, a_2^{\,m}, a_3^{\,n}$, где k=0,1 $m, n \in N_0$, что также дает очевидный алгоритм проверки эквивалентности слов.

3) $\Pi = \{a_1, \ldots, a_n , b_1, \ldots, b_n ; (a_ib_i, \lambda) , (b_ia_i, \lambda) \ \forall i \in \overline{1,n} \}$

λ - пустое слово. Легко показать, что данная полугруппа будет группой, называемой свободной группой. Наличие алгоритма проверки эквивалентности слов следует из того, что для каждого слова Р имеется единственное эквивалентное несократимое слово $P^{\$}$ (т.е. не содержащее вхождений вида (a_ib_i) или (b_ia_i)).

В 40-х годах было установлено, что существуют конечно определенные полугруппы, в которых проблема эквивалентности слов алгоритмически неразрешима.Первые примеры таких полугрупп указали Марков А.А. и Пост Э. (1947). Данные примеры были довольно громоздкими. Цейтин Г.С. (1958) получил следующий пример такой полугруппы:

A={a,b,c,d,e},S={(ab,da),(ac,ca),(bc,cb),(bd,db),(eca,ce),(edb,de),(cca,ccae)}(т. е. 5 букв, 7 соотношений).

Матиясевич Ю.В. (1967) нашел полугруппу с двумя образующими и тремя определяющими соотношениями с нерезрешимой проблемой эквивалентности слов. В одном из этих соотношений слева стоит слово из 304 букв, справа слово из 608 букв.

2^0) Приведем доказательство существования полугруппы с неразрешимой проблемой эквивалентности слов. Сначала укажем конструкцию полугруппы $\Pi_т$, соответствующей машине Тьюринга Т. Пусть машина Т - имеет алфавиты $Q = \{q_0, \ldots q_m\}, A = \{a_0, \ldots a_n\}$ и система команд $q_ia_j \to q_ka_lS \quad S \in \{R, L, E\}$. Соответствующая полугруппа $\Pi_т$ будет иметь образующие $A_T = \{a_0, \ldots a_n, q_0, \ldots q_m, h\}$,где h - новая буква. Определяющие соотношения получаются так:

Команде $q_ia_j \to q_ka_lS$ соответствует оотношение

(q_ia_j, q_ka_l)

Команде $q_ia_j \to q_ka_lL$ соответствует система оотношений

$(aq_ia_j, q_kaa_l), a \in A$, $(hq_ia_j, hq_ka_0a_l)$

Команде $q_ia_j \to q_ka_lR$ соответствует система оотношений

$(q_ia_ja, a_lq_ka), a \in A$, $(q_ia_jh , a_lq_ka_0h)$.

В качестве системы определяющих соотношений S_T берем объединение всех соотношений, соответствующих всем командам машины Т. Получим полугруппу $\Pi_T = \langle A_T, S_T \rangle$.

Машина Тьюринга Т осуществляет переработку машинных слов вида $P = Wq_jW$, $V, W \in N_0$. Каждому машинному слову Р поставим в

соответствие элемент полугруппы Π_T вида $P' = hWq_jWh$, который назовем обобщенным машинным словом.

Лемма 1.

Если машина Т переводит машинное слово Р в Q, то в полугруппе Π_T имеем $P' \sim Q'$.

Док-во.

Утверждение ясно, поскольку командам машины Т соответствуют элементарные преобразования в $\Pi_т$.

Лемма 2.

Если P', Q' - обобщенные машинные слова из Π_T и слово Q' содержит q_0 - заключительное состояние, причем $P' \sim Q'$ в полугруппе Π_T , то слово Q' может быть получено из P' применением только левых элементарных преобразований.

Док-во.

По условию дано, что $P' \sim Q'$ в Π_T .Значит, слово Q' получается из P' с помощью некоторой цепочки элементарных преобразований

$$P' = R_0 \to R_1 \to \ldots \to R_k = Q'.$$

Если в этой цепочке нет правых преобразований, то доказывать нечего. Пусть правые преобразования есть, тогда правое преобразование не может быть последним, т.к. в Q' есть заключительное состояние q_0 ,а в левых частях определяющих соотношений нет вхождений q_0.Поэтому оно может появиться только в результате левого преобразования.Возьмемпервое справа правое элементарное преобразование: $R_\alpha \to R_{\alpha+1} \to R_{\alpha+2}$

Здесь переход $R_\alpha \to R_{\alpha+1}$ осуществлен правым преобразованием , а переход $R_{\alpha+1} \to R_{\alpha+2}$ левым преобразованием.

Пусть переход $R_{\alpha+1} \to R_{\alpha+2}$ осуществлен применением соотношения (q_ia_j, q_ka_l). Значит

$$R_{\alpha+1} = R'q_ia_jR''$$
$$R_{\alpha+2} = R'q_ka_lR''$$

Тогда переход $R_\alpha \to R_{\alpha+1}$ (по правому преобразованию) должен осуществляться применением соотношения содержащеего в левой части вхождение q_ia_j, но такое соотношение единственно и совпадает с (q_ia_j, q_ka_l). Следовательно, $R_\alpha = R_{\alpha+2}$ и слово Q' можно получить из P' с помощью цепочки из k-2 элементарных преобразований , в которой число правых преобразований уменьшено на единицу.

Пусть теперь переход $R_{\alpha+1} \to R_{\alpha+2}$ осуществлен с помощью соотношения (q_ia_j, q_kaa_l). Значит, $R_{\alpha+1} = R'q_ia_jR''$, $R_{\alpha+2} = R'q_kaa_lR''$. Тогда при переходе $R_\alpha \to R_{\alpha+1}$(по правому преобразованию) должно использоваться соотношение с вхождением в левую часть q_ia_j. Но такое соотношение единственно и имеет вид (q_ia_j, q_kaa_l) т.е. снова получили

$R_\alpha = R_{\alpha+2}$ и опять слово Q' можно получить из P' с помощью цепочки из k-2 элементарных преобразований , в которой число правых преобразований уменьшено на единицу.

Остальные случая разбираются аналогично.

ч.т.д.

Следствие.

Если P,Q машинные слова и Q содержит q_0 , то для соответствующих машинных слов Q', P' выполнено $P' \sim Q' \Leftrightarrow$ машина T переводит слово P в слово Q.

Теорема.

Существует конечно определенная полугруппа с неразрешимой проблемой эквивалентности слов.

Док-во.

Рассмотрим функцию $f(x) = \begin{cases} 1, & если \quad U(x,x) \quad определено \\ & не \quad определена, \quad else. \end{cases}$

Здесь $U(n,x)$ - универсальная функция для одноместных частично рекурсивных функций. Поскольку $U(n,x)$ - частично рекурсивна, то функция $f(x)$ - частично рекурсивна. Тогда существует машина Тьюринга T_0 , которая вычисляет функцию $f(x)$ т.е. начальные конфигурации $q_1 \underbrace{|\ldots|}_{x+1}$ переводит в заключительные $q_0||$ в том и только в том случае, когда $f(x)$=1.

По машине Тьюринга T_0 строим конечно определенную полугруппу Π_0 . Это и есть полугруппа с неразрешимой проблемой эквивалентности слов. Пусть это не так, т.е. существует алгоритм проверки эквивалентности слов из Π_0 . Применим его к парам слов вида $P_x = hq_1 \underbrace{|\ldots|}_{x+1} h$ и $Q = hq_0|| \; h$. По доказанному $P_x \sim Q$ тогда и только тогда, когда машина T_0 переводит конфигурацию $q_1 \underbrace{|\ldots|}_{x+1}$ в $q_0||$. Теперь определим алгоритм вычисления функции

$g(x) = \begin{cases} 1, & если \quad U(x,x) \quad определено \\ 0 & , \quad else. \end{cases}$

следующим образом.

Для всякого $x \in N_0$ полагаем $g(x) = 1$,если $P_x \sim Q$ и $g(x) = 0$,если $P_x \not\sim Q$.

Получили противоречие, т.к. согласно теореме 3 предыдущего раздела, функция $g(x)$ невычислима.

ч.т.д.

Заметим, что для полугрупп с одним определяющим соотношением проблема эквивалентности слов разрешима в широком классе случаев и, вероятно, разрешима всегда, но пока доказательство этого факта не получено.(1991)

Для конечно определенных полугрупп можно сформулировать алгоритмическую проблему распознавания свойств. Свойство полугруппы Q назовем марковским, если

1) существует конечно определенная полугруппа, обладающая свойством Q;
2) существует конечно определенная полугруппа, не вложимая ни в какую конечно определенную полугруппу, обладавшую свойством Q.

Для марковских свойств алгоритмическая проблема распознавания для конечно определенных полугрупп неразрешима. (Марков А.А. 1952).

Конечно определенная полугруппа П=(A,S) называется конечно определенной группой, если $A = \{a_1,\ldots,a_n\ , b_1,\ldots,b_n\}$ и среди соотношений S есть соотношения вида $\{(a_ib_i,\lambda)\ ,\ (b_ia_i,\lambda)\}$ λ-пустое слово.

Неразрешимость проблемы эквивалентности слов для конечно определенных групп была установлена Новиковым П.С. (1955). (Данная проблема была сформулирована М.Деном в 1912 г.)

Заметим, что для конечно определенных групп с одним определяющим соотношением проблема эквивалентности слов разрешима.

Неразрешимость алгоритмической проблемы проверки марковских свойств для конечно определенных групп была доказана Адяном С.И.1958).

3^0) Выявлению разрешимых и неразрешимых задач в различных разделах математики посвящено много исследований. Приведем без доказательства некоторые из примечательных таких проблем из алгебры, логики, теоретической кибернетики, которые отличаются простотой формулировки и фундаментальностью.

1. Проблема распознавания истинности формул элементарной арифметики. Формулы строятся с помощью арифмитических действий (сложения и умножения), логических операций (логических связок и кванторов) и знака равенства, а также константы О и натуральнозначных переменных. Проблема состоит в том, чтобы найти алгоритм, который по любой такой формуле определяет, истинна она или нет на натуральном ряду. Неразрешимость этой проблемы установил Гедель К. (1931).

2. Проблема разрешения для логики предикатов первого порядка. Нужно найти алгоритм, который бы проверял общезначимость формулы алгебры предикатов. Норазрешимость этой проблемы установил Черч А. (1936).

3. Проблема сочетаемости Поста. Конечное множество пар слов в некотором алфавите называется сочетаемым, если для некоторых пар $(A_1,B_1),\ldots,(A_s,B_s)$ из V выполнено равенство $A_1\ldots A_s=B_1\ldots B_s$. Нужно найти алгоритм, который по всякому множеству V пар слов узнает сочетаемо оно или нет. Неразрешимость данной проблемы установил Пост Э. (1946).

4. Проблема представимости матриц. Рассматриваются $(n \times n)$-матрицы над кольцом целых чисел **Z**.Пусть даны произвольные матрицы $U_1,\ldots,U_q$ и U . Нужно иметь алгоритм, который бы решал, верно ли $U = U_{i_1}\ldots U_{i_p}$ для

некоторых $i_1,\ldots,i_p$.Неразрешимость данной проблемы, начиная с n=4 , установлена Марковым А.А.(1958).

5. Проблема тождества элементарных функций вещественного переменного. Определим класс термов τ индуктивно: x - переменная, π -число-термы. Если u,v - термы, то $(u+v),(uv),(u/v),\sin u,|u|$ - термы. Нужно иметь алгоритм, который по любым двум термам из τ узнает, задают ли одну и туже функцию одного вещественного переменного x. Неразрешимость данной проблемы установил Матиясевич Ю.В. (1973).

6. Проблема полноты автоматных базисов. Дан набор конечных автоматов (базис) в одним множеством входов и выходами, входящими в множество входов. Строятся схемы с помощью присоединения базисного автомата и введения
обратной связи. Каждая схема реализует автомат. Если схемами в данном базисе может быть реализован любой автомат, в данном алфавите, то базис называется полным, в противном случае - неполным. Проблема полноты заключается в том, чтобы узнавать по заданному базису - является он полным или нет.

Неразрешимость данной проблемы установил
Кратко М.И. (1966).

7. Десятая проблема Гильберта из 23 поставленных им в 1900 году формулируется тая: "Пусть задано диофантово уравнение (т.е. уравнение вида $p(x_1,\ldots,x_n)=0$, p - многочлен) с произвольными неизвестными и целыми рациональными коэффициентами. Указать способ, при помощи которого возможно после конечного числа операций установить, разрешимо ли уравнение в целых рациональных числах".

Неразрешимость 10-й проблемы Гильберта установлена Матиясевичем Ю.В.(1970). Учитывая важность данного результата, приведем его точнее. Пусть $p(x_1,\ldots,x_n,y_1,\ldots,y_m)$ - многочлен над кольцом целых чисел **Z**. Тогда предикат $M(x_1,\ldots,x_n)$ задаваемый формулой

$$M(x_1,\ldots,x_n)=\exists y_1\ldots\exists y_m \quad (p(x_1,\ldots,x_n,y_1,\ldots,y_m)=0)$$

называется диофантовым предикатом. (Область определения кванторов - множество N_0).

Легко убедиться, что диофантовы предикаты являются полувычислимыми. Матиясевич Ю.В.в 1970 году установил:

Теорема.

Каждый полувычислимый предикат диофантов.

Покажем теперь, как из этой теоремы следует неразрешимость десятой проблемы Гильберта. Заметим, что если 10-я проблема Гильберта разрешима, то разрешима и такая проблема: "установить, имеет ли произвольное полиномиальное уравнение $p(x_1,\ldots,x_n)=0$ с целыми коэффициентами решение

в множестве натуральных чисел". Действительно, любое натуральное число может быть представлено как сумма четырех квадратов (по теореме Лагранжа), поэтому, чтобы решать эту проблему достаточно найти целые решения уравнения $p(s_1^2,t_1^2,u_1^2,v_1^2,\ldots,s_n^2,t_n^2,u_n^2,v_n^2)=0$

Возьмем теперь многочлен $p(x,y_1,\ldots,y_m)$, такой,

что $x \in W_x \Leftrightarrow \exists y_1,\ldots,\exists y_m\,(p(x,y_1,\ldots,y_m)=0)$,что можно сделать по теореме Матиясевича. Тогда если бы существовала разрешающая процедура для десятой проблемы Гильберта, то существовала бы и разрешающая процедура для проблемы "$x \in W_x$" : чтобы проверить, верно ли $a \in W_a$, нужно проверить, имеет ли решение в N_0 уравнение $q(y_1,\ldots,y_m)=p(a,y_1,\ldots,y_m)$. Значит, неразрешимая проблема "$x \in W_x$" сводится к проблеме Гильберта, что означает ее неразрешимость.

Укажем еще одно следствие теоремы Матиясевича.

Теорема.

Всякое рекурсивно перечислимое множество является множеством неотрицательных эначенй, принимаемых некоторым многочленом $p(x_1,\ldots,x_n)$ над кольцом целых чисел **Z** (причем $x_1,\ldots,x_n \in N_0$).

Док-во.

Пусть А - рекурсивно перечислимое множество. По теореме Матиясевича существует многочлен $q(x,y_1,\ldots,y_m)$,такой что

$x \in A \Leftrightarrow \exists y_1,\ldots,\exists y_m\,(q(x,y_1,\ldots,y_m)=0)$.

Рассмотрим теперь многочлен $p(x,y_1,\ldots,y_m)$ определяемый формулой

$p(x,y_1,\ldots,y_m)=x-(x+1)(q(x,y_1,\ldots,y_m))^2$

Ясно, что $p(x,y_1,\ldots,y_m)\geq 0 \Leftrightarrow q(x,y_1,\ldots,y_m)=0$ причем в этом случае $p(x,y_1,\ldots,y_m)$=x. Таким образом,

А есть множество неотрицательных значений, принимаемых $p(x,y_1,\ldots,y_m)$ когда $x,y_1,\ldots,y_m$ пробегает множество N_0 .

ч.т.д.

Одно на приложений этого результата - множество простых чисел. Ясно, что множество простых чисел - рекурсивно перечислимо. На основании доказанного множество простых чисел есть множество положительных значений, принимаемых некоторым многочленом с целыми коэффициентами. Данный факт считался ранее маловероятным и даже невозможным.

§ 12 Характеристики сложности вычислений

1^0) В общей теории алгоритмов изучается лишь принципиальная возможность алгоритмического решения задачи. При рассмотрении конкретных задач не обращается внимания на ресурсы времени и памяти для соответствующих им разрешающих алгоритмов. Однако нетрудно понять, что принципиальная возможность алгоритмического решения задачи еще не означает, что оно может быть практически получено. Поэтому возникает потребность уточнить понятие алгоритмической разрешимости, введя характеристики слабости алгоритмов, позволяющие судить об их практической реализуемости. Выше было установлено, что различные алгоритмические модели приводят к одному и тому же классу алгоритмически вычислимых функций. В то же время ясно, что выбор алгоритмической модели, реализующей алгоритмы, существенно влияет на сложность вычислений. Утверждения о трудоемкости вычислений, вообще говоря, не сохраняются при изменении алгоритмической модели. Однако, имеется ряд фактов, которые не зависят от вычислительной модели и относятся к так называемой машинно-независимой теории сложности.

Введем необходимые определения, отправляясь от машины Тьюринга в качестве модели вычислительного устройства. Пусть машина Тьюринга T вычисляет словарную функцию $f(x)$. Определим функцию $t_T(x)$, равную числу шагов машины T , выполненному при вычислении $f(x)$, если $f(x)$ определено. Если $f(x)$ не определено, то значение $t_T(x)$ считается неопределенным. Функция $t_T(x)$ называется временной сложностью.

Активной зоной при работе машины T на слове x называется множество всех ячеек ленты, которые содержат непустые символы, либо посещались в процессе вычисления
$f(x)$ головкой машины T. Определим функцию $s_T(x)$, равную длине активной зоны при работе машины T на слове x , если $f(x)$ определено. Если $f(x)$ не определено, то значение $s_T(x)$ считается неопределенным. Функция $s_T(x)$ называется емкостной (ленточной) сложностью.

Теорема 1.

Пусть машина Тьюринга T имеет внешний и внутренний алфавит мощности k и r соответственно. Тогда для функций сложности $s_T(x), t_T(x)$ справедливы оценки

$$s_T(x) \le |x| + t_T(x) \qquad (1)$$

$$t_T(x) \le r s_T^2(x) k^{s_T(x)}$$

Док-во.

В начальный момент на ленте записано слово x длины $|x|$. На каждом шаге активной становится не более одной новой ячейки, поэтому $s_T(x) \le |x| + t_T(x)$.

Найдем теперь число различных конфигураций

$$K = a^{(1)} \dots a^{(i-1)} q_j a^{(i)} \dots a^{(s')} \quad , \quad s' \le s$$

с активной зоной s' , не превосходящей фиксированной величины s. Имеется $k^{s'} \le k^s$ вариантов записи на ленте, r вариантов выбора положения головки и s вариантов для значения длины s' конфигурации K. Таким образом, число рассматриваемых конфигураций не превосходит rs^2k^s . Далее, если в процессе работы машины встретятся две одинаковые конфигурации, то машина зациклиться, поскольку конфигурация однозначно определяет следующую за ней. Значит, если машина в процессе вычисления использует зону $s_T(x)$, то число ее шагов не превосходит числа различных конфигураций с зоной не превышающей $s_T(x)$.В итоге получаем неравенство $t_T(x) \le rs_T^2(x)k^{s_T(x)}$ и теорема доказана.

ч.т.д.

Введенные функции сложности $s_T(x)$ и $t_T(x)$ являются словарными. Удобно ввести для рассмотрения функции натурального аргумента, положив

$$t_T(n) = \max_{|x| = n} t_T(x) \qquad (2)$$

$$s_T(n) = \max_{|x| = n} s_T(x)$$

Они также называются функциями временной и емкостной сложности (по худшему случаю). Поведение этих функций в пределе при увеличении размера задачи n называется асимптотической временной (соответственно - емкостной) сложностью. Для конкретных задач находятся, как правило, асимптотические функции сложности,

Заметим, что для введенных функций $s_T(x)$ и $t_T(x)$ выполнены свойства:

1) $D(t_T(x)) = D(f_T(x))$

(Здесь $D(g_T(x))$ - область определения функции $g(x)$, f_T - функция, вычисляемая машиной T).

2) Предикат P(x,y) определены соотношением

$P(x,y) \equiv (t_T(x) = y)$ - разрешим. (Аналогично для функции $s_T(x)$).

Подобным образом можно определить функции сложности вычисления на машине МПД. Пусть P - программа МПД. Обозначим через $t_P(x)$ функцию определенную условием $t_P(x) = \mu t(P(x)\downarrow \quad за \quad t \quad шагов.)$ т.е. это есть число шагов вычисления на начальной конфигурации x по программе P , если вычисление заканчивается и неопределено в противном случае.

Ясно, что для введенной функции $t_P(x)$ также выполнены условия 1) и 2).

Дадим теперь определение абстрактной меры вычисления (для одноместных) числовых функций. Пусть $f_1(x), f_2(x), \ldots, f_n(x), \ldots$- нумерация одноместных частично рекурсивных функций. Мерой вычислительной сложности называется любое семейство функций $\Phi_0(x), \Phi_1(x), \ldots, \Phi_n(x), \ldots$, обладающее свойствами

1) $D(\Phi_i) = D(f_i) \quad \forall i \in N_0$

2) Предикат $P_i(x,y) = (\Phi_i(x) = y)$ разрешим $\forall i \in N_0$.

Примером меры вычислительной сложности будет, например, семейство функций $\Phi_i(x), i \in N_0$, где $\Phi_i(x)$ -максимальное число, содержащееся в регистре МПД за все время вычисления $P_i(x)$, если $P_i(x)\downarrow$ и неопределено в противном случае.

В дальнейшем основное внимание будет уделено сложности Тьюринговых вычислений.

2^0) Рассмотрим вопрос о верхней границе сложности вычислений, который формулируется так: существует ли такая общерекурсивная функция $h(x)$, что для любой вычислимой функции $f(x)$ найдется машина Тьюринга для которой $t_T(x) \le h(x)$ соответственно $s_T(x) \le h(x)$ там, 1*де значения $t_T(x)$ определены.

Если допустить, что значения $f(x)$ могут быть сколь угодно большими, то ответ отрицателен т.к. на запись ответа может понадобиться тактов больше, чем $h(x)$. Поэтому предположим что значения $f(x)$ могут быть только О,1. Если допустить, что $f(x)$ может быть частичной, то ответ также отрицателен, т.к. из существования такой общерекурсивной функции $h(x)$ следует существование рекурсивного доопределения для любой рекурсивной функции $f(x)$. Действительно, пусть машина Т вычисляет функцию $f(x)$,для любого х отсчитываем $h(x)$ тактов и если за это время значение $f(x)$ не определено. то полагаем $f(x) = 0$. Эта алгоритмическая процедура дает рекурсивное доопределение. Но это противоречит теореме 6 из раздела 3 §9. Будем теперь рассматривать общерекурсивные (0,1-функции $f(x)$. Следующая теорема показывает, что ответ на поставленный вопрос также отрицательный.

Теорема _2. Для всякой общерекурсивной функции $h(x)$ существует общерекурсивная 0,1-функция $f(x)$, такая, что для любой машины Тьюринга Т , вычисляющей $f(x)$, существует значение аргумента х=n , при котором $t_T(n) < h(x)$.

Док-во.

Рассмотрим нумерацию машин Тьюринга $T_0, T_1, \ldots, T_n, \ldots$ и соответствующую нумерацию частично рекурсивных функций $f_0(x), f_1(x), \ldots, f_n(x), \ldots$. Определим конкретную функцию

$$f(x) = \begin{cases} \overline{sg}\, f_x(x) & , \quad если \quad t_{T_x}(n) \le h(x) \\ & и \quad f_x(x) \quad определено \\ 0 & , \quad else \end{cases}$$

Функция $f(x)$ вычислима с помощью следующей процедуры: для любого х находится значение $h(x)$, затем машина T_x применяется к конфигурации $q_1 \underbrace{\lfloor \ldots \rfloor}_{x+1}$ и считаются такты работы машины T_x .Если в течение $h(x)$ тактов T_x

остановилась в заключительной конфигурации вида $q_0 \underbrace{|\ldots|}_{y+1}$ (это проверяется просмотром активной зоны размера $\le |x| + h(x)$), то полагается $f(x) = \overline{sg}\, y$.В противном случае величина $f_x(x)$ не определена и полагаем $f(x)$=0. Если машина T_x не остановилась в течение $h(x)$ тактов, то полагаем $f(x)$=0. Итак, вычислимость $f(x)$ доказана. Следовательно, существует машина Тьюринга T , вычисляющая $f(x)$. Пусть ее номер n. Докажем, что выполнено $t_T(n) > h(x)$. Если это не так, т.е. $t_T(n) \le h(x)$, то поскольку $f = f_n$ общерекурсивна и $f(n)$ определено, тогда

$$f(n) = \overline{sg}\, f_n(n) = \overline{sg}\, f_T(n)$$

Следовательно, $f(n) \ne f_T(n)$ и это противоречит тому, что f вычисляется машиной T.

ч.т.д.

3^0) Рассмотрим вопрос о существовании сложно вычислимых функций, который формулируется так: существует ли общерекурсивная функция (0,1)-функция $f(x)$ и такая, что для любой машины Тьюринга для всех x выполняется за время, превышающее значение наперед заданной функции $h(x)$. В такой постановке ответ отрицательный т.к. вычисление функций $f(x)$ в любом конечном числе точек можно сделать тривиальным, предварительно занеся эти значения в программу машины. Поэтому в поставленном вопросе потребуем, чтобы нужная оценка выполнилась при всех x , кроме конечного числа значений.

Теорема 3.

Для всякой общерекурсивной функции $h(x)$ существует общерекурсивная (0,1)-функция $f(x)$ и такая, что для любой машины T , вычисляющей $f(x)$, существует x_0 , такое, что выполнено $t_T(x) > h(x)$ при всех $x \ge x_0$.

Док-во.

Нужную функцию $f(x)$ и вспомогательную функцию $\pi(x)$ будем строить последовательно при x=0,1,...

x=0 Если $t_{T_0}(0) \le h(0)$ и $f_0(0)$ определено, то

полагаем $f(0) = \overline{sg}\, f_0(0), \pi(0) = 0$.

В противном случае полагаем $f(0) = 0$ и $\pi(0)$ не определено.

x=n>0 Пусть значения $f(x)$ определены при всех x<n

и определены значения $\pi(x)$.

Для нахождения $f(n)$ поступаем следующим образом: Проверяем выполнимость неравенств

$t_{T_0}(n) \le h(n), t_{T_1}(n) \le h(n), \ldots, t_{T_n}(n) \le h(n)$

Для всех $k \in \overline{0,n}$, для которых выполнены оба условия а) $t_{T_k}(n) \le h(n)$ б) $f_k(n)$ определено

находим наименьшее, которое не является значением функции π. Если такое число есть (обозначим его k_0), тогда полагаем $\pi(n) = k_0$ и $f(n) = \overline{sg}\, f_{k_0}(n)$ Если таких чисел k нет, либо все k уже являются значениями функции π, то считаем, что значение $\pi(n)$ неопределено и $f(n)$=0.

Легко убедиться, что функция f вычислима, всюду определена и принимает значения О,1.Функция $\pi(x)$ принимает различные значения и $\pi(x) \le x$.

Пусть Т - произвольная машина, вычисляющая f и i - номер машины Т. Докажем, что выполнено $t_{T_i}(x) > h(x)$ для всех х , начиная с некоторого x_0 .

Пусть это неравенство нарушается для бесконечного числа значений х. Пусть $n_1, n_2, n_3, \ldots$ такие значения х, что $i \ge n_1 > n_2 > n_3 > \ldots$.Покажем, что по крайней мере в одном из чисел $n_1, n_2, n_3, \ldots, n_i, n_{i+1}$ функция π принимает значение i . Имеем в точках $n_1, n_2, n_3, \ldots, n_i, n_{i+1}$, неравенства $t_{T_i}(n_k) \le h(n_k)$ определено. Если значение i не принимается функций π в точках $n_1, n_2, n_3, \ldots, n_i$, то либо $\pi(n)$ было ранее, что невозможно, т.к. n_1 - первое, начиная с i , число, для которого $t_{T_i}(n_1) \le h(n_1)$, либо функция π принимала значения, меньшие i . Таких значений i штук и при $n \le n_i$ все они являются значениями π (т.к. значения π различны). Тогда в точке n_{i+1} функция π принимает значение i . Это значит, согласно определения функции f , что $f \ne f_i$,т.е. машина Т не вычисляет f_i . Полученное противоречие доказывает теорему.

ч.т.д.

4^0) Выше была определена сложность конкретного алгоритма, вычисляющего некоторую функцию f . Рассмотрим вопрос: можно ли определить сложность вычислимой функции как сложность наилучшего алгоритма, вычисляющего ее.

Приведем без доказательства (ввиду его сложности) результат М.Блюма, показывающий, что, вообще говоря, этого сделать нельзя. Существуют функции, допускающие убыстрение всюду, за исключением конечного числа точек.

Теорема 4.

Для любой общерекурсивной функции $r(x)$ существует общерекурсивная (О,1)-функция $f(x)$,такая, что для любой машины M_i, вычисляющей $f(x)$, существует машина M_j, также вычисляющая $f(x)$, что для всех х , начиная с некоторого выполнено

$$t_{T_i}(x) > r(t_{T_j}(x))$$

Обсудим некоторые следствия из данного результата. 1. Если взять $r(x) = 2^x$, то существует функция $f(x)$ со свойством: если она допускает вычисление со сложностью $t(x)$, то она допускает вычисление со сложностью $t'(x)$, причем $t(x) > 2^{t'(x)}$ или $t'(x) < \log t(x)$.Убыстренное вычисление снова допускает убыстрение-вычисление со скоростью $t''(x)$,

$t''(x) < \log t'(x) < \log\log t(x)$ и т.д. (Неравенства выполняются при всех x, начиная с некоторого).

2. Пусть мы реализуем работы любой машины M со скоростью 1 шаг/сек. и переходим к реализации со скоростью 10^{10} шаг/сек. Тогда вычисление на машине M_i, требующее $t_i(x)$ секунд при старой реализации, требует $t_i'(x) = \frac{t_i(x)}{10^{10}}$ секунд при новой реализации.

Рассмотрим функцию f , определяемую теоремой об ускорении при $r(x) = 10^{10}x$.Пусть f вычисляется M_i. Тогда существует M_j, такая, что $10^{10} t_j(x) < t_i(x)$

начиная с некоторого x_0 $t_j(x) < \frac{t_i(x)}{10^{10}} = t_i'(x)$

Значит, для всех x начиная с некоторого, старая реализация вычисления машины M_j лучше, чем новая реализации машины M_i, т.е. более быстрая реализация вычислений не имеет преимуществ перед медленной реализацией для почти всех входов (для некоторых функций).

§ 13. Нижние оценки временной сложности вычислений на машинах Тьюринга

1^0) В данном разделе излагается техника следов позволяющая получать нетривиальные нижние оценки времени вычисления на машинах Тьюринга. Данная тахника иллюстрируется на задаче распознавания симметрии слов и применяется к задачам распознавания некоторых свойств булевых функций и их систем.

Рассмотрим следующий класс однотипных задач. Пусть дан алфавит $E = \langle 0,1 \rangle$ и для произвольного слова $P = p_1 p_2 \dots p_n$, где $p_i \in E, i \in \overline{1,n}$ требуется узнать симметрично оно или нет , т.е. верно ли равенство

$$P = p_1 p_2 \dots p_n = p_n p_{n-1} \dots p_1 = P'$$

Предполагаем, что соответстветствующая решающая машина Тьюринга на начальной конфигурации $q_1 P$, выдает 1, если слово Р симметрично, и 0, в противном случае.

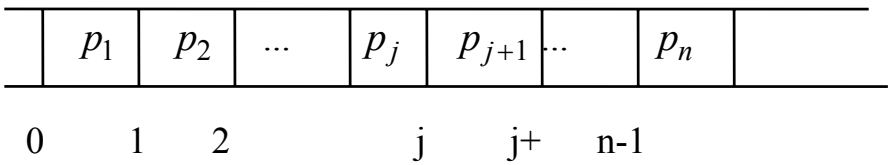
Нетрудно составить программу машины Тьюринга для решения данной задачи, однако для экономии места ограничимся описанием ее работы. Работа машины Тьюринга осуществляется циклами, В течение цикла 1 машина доверяет верно ли $p_1 = p_n$. Для этого запоминается символ p_1 , и считывающая головка смещается вправо, находит символ p_n , и сравнивает его с p_1 . Если эти символы различны, то машина стирает запись на ленте и печатает 0, если эти символы совпадают, то машина реализует цикл 2, в течение которого сравниваются символы p_2 и p_{n-1} и т.д. Если слово Р симметрично, то будет произведено $\left[n/2 \right]$ циклов сравнения символов, в течение которых будет совершено число тактов, равное

$$n + (n-1) + (n-2) + \ldots = O(n^2).$$

Возможны усовершенствования данной машины, приводящие к сокращению времени вычисления, однако не удается уменьшить порядок n^2. В связи с этим возникает проблема - как доказать , что порядок n^2 времени вычисления неустраним для данной машины Тьюринга. Соответствующее доказательство было предложено Барздинем Я.М.

2^0) Рассмотрим последовательность конфигураций $T[P]$, реализуемую машиной Тьюринга Т на слове Р.

Пронумеруем границы между соседними ячейками ленты так, как это показано на рис.



С каждой границей j можно связать последовательность внутренних состояний машины T следующим образом: Если в процессе $T[P]$ головка машины ни разу не пересекала границу j , то эта последовательность пуста. Пусть головка машины T пересекает границу j точно s раз: в первый раз в состоянии q(1) , во второй раз в состоянии q(2) и т.д. Тогда последовательность q(1)q(2)...q(2) , являющаяся словом в алфавите внутренних состояний машины T , называется следом вычисления $T[P]$ в точке j и обозначается $tr_T(P, j)$.Если рассматривать конечные процессы $T[P]$, то и следы будут конечные. Ясно, что пересечение головкой какой-либо границы связано с затратой одного такта работы машины, поэтому справедливо неравенство

$$t_T(P) \geq \sum \left| tr_T(P, j) \right| \qquad (1)$$

где сумма берется по любому множеству границ j. (Через $|Q|$ обозначаем длину слова Q. Неравенство (1) может дать нижнюю оценку времени вычисления, если иметь нижнюю оценку для длин следов.

3^0) Справедлива следующая

Теорема 1.

Пусть T - произвольная машина Тьюринга, решающая проблему симметрии слов, имеющая k внутренних состояний. Тогда для любого $\varepsilon > 0$ справедлива оценка

$$t_T(P) = \Omega\left(\frac{1-\varepsilon}{4 \log_2 k} |P|^2 \right)$$

для почти всех симметричных слов. (Знак Ω означает нижнюю оценку по порядку)

Замечание.

Говорят, что почти все слова, обладающие свойством R, обладают и свойством S , если

$\frac{S_R(n)}{R(n)} \to 1$ при $n \to \infty$, где R(n) - число слов длины n,

обладающих свойством R. $S_R(n)$ - число слов длины n , которые обладают наряду со свойством R также свойством S.

Док-во.

Пусть даны два симметричных слова P_1 и P_2 длины $n = 2\eta$ (в случае нечетного n рассуждения аналогичны) и пусть они имеют разные начала длины ξ. В таком случае следы в точке ξ различны, т.е. $tr_T(P_1, j) \neq tr_T(P_2, j)$.

Допустим противное и пусть $tr_T(P_1, j) = tr_T(P_2, j)$.Для слов P_1 и P_2 обозначим P_1^{ξ} начало слова P_1, длины ξ, ${}^{\xi}P_2$ слово полученное из P_2 удалением начала длины ξ. Образуем слово $P_1^{\xi}\,{}^{\xi}P_2$. По условию слово R не симметрично, Если запустить машину T со словом на ленте, то левее ξ оно будет обрабатываться как P_1 , а правее ξ - как P_2 и в силу симметричности P_1 и P_2,машина T выдаст 1, что противоречит тому, что R не симметрично.

Зафиксируем некоторое $\xi \in \overline{1,n}$ и разобьем множество всех симметричных слов длины $n = 2\eta$ мощности на классы, в каждом из которых все слова имеют одинаковые начала длины ξ, а слова из разных классов имеют разные начала длины ξ. Таких классов, очевидно, 2^{ξ} - по числу различных слов длины ξ в каждом классе имеется $2^{\eta-\xi}$ слов. Слова из разных классов по доказанному, обязаны иметь в точке ξ попарно различные следы. Выберем из каждого класса по одному представителю с наиболее коротким следом в точке ξ. Значит существует 2^{ξ} попарно различных следов,которые являются словами в алфавите из К букв. Пусть дано $\varepsilon > 0$ определим, сколько среди 2^{ξ} следов может быть "коротких" следов, т.е. следов, имеющих длину, не превышающую $(1-\varepsilon)\log_k 2^{\xi} = \dfrac{1-\varepsilon}{\log_2 k}\xi$.Ясно, что "коротких" следов не более, чем

$$k + k^2 + \ldots + k^{\left((1-\varepsilon)/\log_2 k\right)\xi} = \frac{k^{\left((1-\varepsilon)/\log_2 k\right)}+1}{k-1} \le k^{\left((1-\varepsilon)/\log_2 k\right)} + 1 = k2^{(1-\varepsilon)\xi}$$

Отсюда получаем» что число симметричных слов, у которых в точке ξ "короткий' след,не превосходит

$$k2^{(1-\varepsilon)\xi}2^{\eta-\xi} = k2^{\eta-\varepsilon\xi}.$$

Обозначим через $\Delta(\varepsilon,\xi)$ долю симметричных слов, у которых в точке ξ "короткий" след (т.е. след длины $\le \dfrac{1-\varepsilon}{\log_2 k}\xi$).Имеем $\Delta(\varepsilon,\xi) \le \dfrac{k2^{\eta-\varepsilon\xi}}{2^{\eta}} = \dfrac{k}{(2^{\varepsilon})^{\xi}}$.

Обозначая $\alpha = 1/2^{\varepsilon}$, имеем $0 < \alpha < 1$ и $\Delta(\varepsilon,\xi) \le k\alpha^{\xi}$.

Пусть $\omega(\eta)$ - любая функция (например, $\log\eta$), обладающая свойствами

1) $\omega(\eta) \to \infty$ при $\eta \to \infty$

2) $\dfrac{\omega(\eta)}{\eta} \to \infty$ при $\eta \to \infty$

Обозначим через $\Delta(\varepsilon)$ долю тех симметричных слов, для которых существует ξ, такое, что $\omega(\eta) < \xi < \eta$ и в точке ξ - "короткий" след. Имеем

$$\Delta(\varepsilon) \le \Delta(\varepsilon,\omega(\eta)) + \Delta(\varepsilon,\omega(\eta)+1) + \Delta(\varepsilon,\eta) =$$

$$= k(\alpha^{\omega(\eta)} + \alpha^{\omega(\eta)+1} + \ldots + \alpha^{\eta}) = k\frac{\alpha^{\omega(\eta)} - \alpha^{\eta+1}}{1-\alpha}$$

При $\eta \to \infty$ имеем $\Delta(\varepsilon) \to 0$ для любого $\varepsilon > 0$. В этом смысле говорят, что почти все симметричные слова имеют в точках между $\omega(\eta)$ и η "длинные" следы. Сумма длин следов на левой половине почти для всякого симметричного слова больше, чем

$$\frac{1-\varepsilon}{\log_2 k}\left(\omega(\eta) + (\omega(\eta)+1) + \ldots + \eta\right) =$$

$$= \frac{1-\varepsilon}{\log_2 k} \cdot \frac{\eta + \omega(\omega)}{2} (\eta - \omega(\omega)) =$$

$$= \frac{1-\varepsilon}{\log_2 k} \cdot \frac{\eta^2 - \omega(\eta)}{2} \approx \frac{1-\varepsilon}{\log_2 k} \frac{\eta^2}{2} \quad \text{при } \eta \to \infty$$

Это означает, что время нахождения считывающей головки на левой половине слова P по порядку не меньше, чем

$\frac{1-\varepsilon}{\log_2 k} \frac{\eta^2}{2}$ (для почти всех слов P).

Аналогичная картина имеет место для правой половины слова P в итоге получаем:

$$t_T(P) = \Omega\left(\frac{1-\varepsilon}{\log_2 k} \frac{\eta^2}{2} \right) = \frac{1-\varepsilon}{4 \log_2 k} |P|^2$$

ч.т.д.

Поскольку указывалась машина Тьюринга, имеющая верхнюю оценку времени вычисления $t_T(P) = O\left(|P|^2\right)$, то полученная в теореме 1 нижняя оценка является асимптотически оптимальной.

Замечание.

Вместо проверки симметрии слова можно рассмотреть задачу проверки (φ -симметрии, которая ставится так. Пусть задана функция $\varphi(n) \le \frac{n}{2}$, $n \in N$. Будем говорить, что слово P ($|P| = n$ в алфавите $E = \langle 0,1 \rangle$ φ -симметрично, если его концы длины $\varphi(n)$ симметричны. Может быть доказана

Теорема2.

Для любой функции $\varphi(n)$, удовлетворяющей условиям $\log_2 n$ p $\varphi(n)$ p n (неравенство по порядку) и для любой машины Тьюринга T с k состояниями, распознающей φ -симметрию , и для любого $\varepsilon > 0$ имеет место

$$t_T(n) = \Omega\left(\frac{1-\varepsilon}{\log_2 k} n\varphi(n) \right)$$ (неравенство по порядку)

4^0) Техника следов может быть применена для получения нижних оценок временной сложности решения других задач. Рассмотрим некоторые из них.

1) Пусть P - слово в алфавите $E = \langle 0,1 \rangle$. Слово P является точным квадратом, если $P=P_1P_1$ для некоторого слова P_1. Рассмотрим задачу: Для произвольного слова узнать, является ли оно точным квадратом. Легко проверить,что для данной задачи справедлива теорема 1.

2) Для произвольного n рассмотрим множество слов P длины 2^n в алфавите $E = \langle 0,1 \rangle$ и будем их трактовать как таблицы булевых функций $f(x_1,\ldots,x_n)$ при

лексикографическом упорядочении множества аргументов. Рассмотрим задачи:

а) Существенность 1-го переменного: по слову P узнать, является ли переменное x_1 существенным.

б) Существенность n-го переменного: по слову P узнать, является ли переменное x_n существенным.

Легко проверить, что для задачи а) справедлива теорема 1, а задача б) решается за линейное время, т.е.

$$t_T(P) = O(|P|)$$

в) функциональная полнота: по слову P, $|P| = 2^n$ узнать, является ли соответствующая булева функция $f(x_1,\ldots,x_n)$ Шефферовой (т.е. представляет ли она функционально полную систему функций). Можно доказать, что для данной задачи справедлива теорема 1. Более того, может быть доказано существование машины Тьюринга, проверяющей критерий функциональной полноты Поста за время $O(|P|^2)$ и, следовательно, квадратичная оценка является асимптотически оптимальной.

3) Для произвольного n рассмотрим слово P длины $n2^n$ в алфавите $E = \langle 0,1 \rangle$ и будем трактовать его как табличное задание семейства булевых функций $f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n)$ при лексикографическом упорядочении множества переменных. Рассмотрим задачу: по слову P, $|P| = n2^n$ узнать, является ли соответствующее семейство булевых функций биективным. Может быть доказана

Теорема 3.

Пусть T - машина Тьюринга с k внутренними состояниями, которая решает задачу биективности семейства булевых функций. Тогда для любого $\varepsilon > 0$ справедлива оценка

$$t(P) = \Omega\left(\frac{1-\varepsilon}{4\log_2 k} n 2^{2n}\right)$$

для почти всех регулярных семейств $(f_1,\ldots,f_n)$ (Неравенство по порядку)

Заметим, что в данном случае оценка не является квадратичной, т.к. длина входного слова P есть $n2^n$.

§ 14. Классы сложности P и NP и их взаимосвязь

1°. Установление прямых нижних оценок сложности вычислений, о которых шла речь в предыдущем разделе, удается лишь в очень редких случаях. В связи с этим получил распространение подход, связанный с получением косвенных нижних оценок, т.е. установление таких утверждений, в которых существование эффективного разрешающего алгоритма для конкретной задачи влечет за собой существование эффективного алгоритма для многих общепризнанно трудных задач.

Нам необходимо формализовать соответствующий подход. Пусть П - некоторая массовая задача, характеризуемая множеством параметров, $I \in \Pi$ - индивидуальная задача, в которой эти параметры фиксированы. Пусть с массовой задачей П связана и зафиксирована схема кодирования α, которая ставит каждой индивидуальной задаче $I \in \Pi$ в соответствие слово $\alpha(I)$ в некотором алфавите А. При этом под размером задачи I понимается длина слова $\alpha(I)$. Пусть Т - машина Тьюринга, решающая задачу П и

$$t_{\mathrm{T}}(n) = \max_{I,\,|\alpha(I)|=n} t_{\mathrm{T}}(I) \qquad (1)$$

- соответствующая функция временной сложности (по худшему случаю). Говорят, что машина Т решает задачу П за полиномиальное время, если

$$t_{\mathrm{T}}(n) = O(p(n)) \qquad (2)$$

для некоторого полинома р.
В противном случае говорят, что машина Т решает задачу П за экспоненциальное время. Заметим, что при данном определении к экспоненциальным оценкам относятся, например, оценки вида $O(n^{\log_2 n})$. (Некоторые авторы оценки такого вида называют субэкспоненциальными, под которыми понимают такие оценки, которые превосходят любой полином, но меньше, чем $O(2^{n^{\varepsilon}})$ для любого $\varepsilon>0$).

Про задачу П говорят, что она разрешима за полиномиальное время, если существует машина Тьюринга Т, решающая ее за полиномиальное время. Обозначим через **P** класс задач, разрешимых за полиномиальное время. Относительно класса **P** необходимо сделать следующие замечания.

1) В определении класса **P** существенным является фиксация схемы кодирования α. Многие естественные схемы кодирования полиномиально эквиваленты, т.е. позволяют переходить от одного кода задачи к другому коду за полиномиальное время от длины кода. В этом случае принадлежность (или не принадлежность) задачи П классу **P** определяется инвариантно по отношению к схемам кодирования. Однако, это справедливо не всегда и, вообще говоря, класс сложности **P** зависит от схемы кодирования, поэтому там, где схема кодирования не очевидна или может повлиять на класс сложности, ее следует указывать явно.

2) Класс **P** определен через функцию временной сложности машины Тьюринга. Можно сделать соответствующие определения через любую другую алгоритмическую модель.

Однако, имеется ряд фактов о полиномиальной эквивалентности временных функций сложности многих типов вычислительных моделей, что позволяет утверждать, что класс **P** определен однозначно для "разумных" вычислительных моделей. С результатами взаимного моделирования вычислительных моделей можно ознакомиться в [2],[15]. Поэтому без специальных оговорок будут допускаться выражения типа “алгоритм А имеет полиномиальную сложность” или “алгоритм В имеет экспоненциальную сложность”. Обратим внимание, что имеется существенное различие между алгоритмами полиномиальной и экспоненциальной сложности. Ясно, что любой полиномиальный алгоритм более эффективен при достаточно больших размерах входа. Кроме того, полиномиальные алгоритмы лучше реагируют на рост производительности ЭВМ. Рассмотрим такой параметр как размер решаемой задачи на ЭВМ за единицу времени с помощью данного алгоритма. Тогда изменение данного параметра при переходе к ЭВМ в 100 раз и в 1000 раз большей производительности для различных функций временной сложности показан в таблице:

Функция временной сложности	Размер решаемой задачи на современной ЭВМ за сутки	То же на ЭВМ в 100 раз быстрых	То же на ЭВМ в 1000 раз быстрых
n	N_1	$100\,N_1$	$1000\,N_1$
n^2	N_2	$10\,N_2$	$31.6\,N_2$
n^3	N_3	$4.64\,N_3$	$10\,N_3$
2^n	N_4	N_4+6.64	N_4+9.97
3^n	N_5	N_5+4.19	N_5+6.29

Из таблицы видно, что в случае полиномиальных алгоритмов размер решаемой задачи при увеличении производительности ЭВМ увеличивается на мультипликативную константу, тогда как для экспоненциальных алгоритмов имеет место увеличение на аддитивную константу.

Далее, полиномиальные алгоритмы обладают свойством “замкнутости”, - можно комбинировать полиномиальные алгоритмы, используя один в качестве “подпрограммы” другого и при этом результирующий алгоритм будет полиномиальным. В силу приведенных причин используется следующая терминология: полиномиальные алгоритмы называют эффективными, полиномиально решаемые задачи называют легкорешаемые, а экспоненциально решаемые задачи называют труднорешаемыми. Для практики важным является классификация задач по признаку труднорешаемости, хотя следует заметить, что установление легкорешаемости задачи еще не означает ее практическую решаемость. Например, установление полиномиальной оценки $O(n^{1000})$ не гарантирует практической решаемости уже при начальных значениях n.

Аналогичное замечание можно сделать относительно труднорешаемости. Заметим, что труднорешаемость задачи может быть связана с тем, что ее решение настолько велико, что не может быть записано в виде выражения, длина которого была бы ограничена полиномом от длины входа. Чтобы исключить этот тип труднорешаемости, рассматриваются только такие задачи, которые имеют "короткий" ответ.

Рассмотрим несколько примеров.

1. Пусть M=$\{1, 2, \ldots, n\}$ - конечное множество и $R \subseteq$ M×M - бинарное отношение на M.

Рассмотрим задачу проверки - является ли R отношением эквивалентности. Будем задавать индивидуальную задачу матрицей $A_R=(a_{ij})$, $i, j \in \overline{1,n}$, где

$$a_{ij} = \begin{cases} 1, & \text{если } (i,j) \in R \\ 0, & \text{если } (i,j) \notin R \end{cases} \qquad (3)$$

Ясно, что R будет отношением эквивалентности тогда и только тогда, когда матрица A_R имеет единичную диагональ, симметрична и выполнено соотношение

$$b_{ij} \le a_{ij}, \forall \ \ i, j \in \overline{1,n} \qquad (4)$$

где $(b_{ij})=A_{R^2}$ - матрица отношения R^2. Кроме того выполнено $A_{R^2}=A_R \cdot A_R$, где имеется в виду булево умножение матриц. Ясно, что сложность рассматриваемой задачи $O(n^3)$.

Бинарное отношение R называется связным, если для любых $i, j \in \overline{1,n}$ существует k, что выполнено $iR^k j$.

2. Бинарное отношение R называется эйлеровым, если элементы R можно так упорядочить

$$R \ : \ (i_1, j_1), \ (i_2, j_2), \ldots, \ (i_t, j_t), \ t = |R| \qquad (5)$$

что выполнено

$$(j_1, i_2) \in R, \ (j_2, i_3) \in R, \ldots, \ (j_{t-1}, i_t) \in R, \ (j_t, i_1) \in R \quad (6)$$

Ясно, что эйлерово отношение является связным.

Можно доказать, что связное отношение R является эйлеровым тогда и только тогда, когда число единиц в матрице A_R совпадает в i -столбце и в i - строке для каждого $i \in \overline{1,n}$. Это дает алгоритм сложности $O(n^2)$ проверяющий эйлеровость отношения R. Ясно, что связность отношения R проверяется за $O(n^3)$ действий.

Заметим, что здесь главным для нас является установление полиномиальности рассматриваемых задач, а не установление наилучших алгоритмов.

3. Бинарное отношение R называется гамильтоновым, если элементы M можно так упорядочить $i_1, i_2, \ldots, i_n$, что выполнено соотношение

$$(i_1, i_2) \in R, \ (i_2, i_3) \in R, \ldots, \ (i_{n-1}, i_n) \in R, \ (i_n, i_1) \in R \quad (7)$$

В настоящее время неизвестно полиномиального алгоритма от n проверки гамильтоновости произвольного отношения R. Тривиальный алгоритм требует $n!$ упорядочений множества M и проверки условий (7), что, конечно, превосходит по величине любой полином от n.

4. Пусть $f(x_1,\ldots,x_n)$ - формула от булевых переменных $x_1,\ldots,x_n$ в некотором фиксированном базисе B. Напомним, что формула $f(x_1,\ldots,x_n)$ называется выполнимой, если существует набор значений переменных $x_1^0,\ldots,x_n^0$,

такой, что $f(x_1^0,\ldots,x_n^0)=1$.

Формула $f(x_1,\ldots,x_n)$ называется мультиаффинной, если она имеет вид

$$f(x_1,\ldots,x_n)=(x_{i_1}+\ldots+x_{i_{k_1}}+\alpha_1)\ldots(x_{l_1}+\ldots+x_{l_{k_t}}+\alpha_t) \qquad (8)$$

($\alpha_1,\ldots,\alpha_t$ -константы)

т.е. f представляет собой конъюнкцию линейных форм. Рассмотрим задачу проверки выполнимости мультиаффинной формулы. Ясно, что существование выполняющего набора для мультиаффинной формулы вводится к существованию решения линейной системы уравнений над полем F_2. Алгоритм Гаусса дает оценку $O(n^3)$, поэтому рассматриваемая задача полиномиальна.

Пусть формула $f(x_1,\ldots,x_n)$ имеет конъюнктивную нормальную форму, т.е. имеет вид:

$$f(x_1,\ldots,x_n)=(x_{i_1}^{\alpha_1}\vee\ldots\vee x_{i_{k_1}}^{\alpha_{k_1}})\ldots(x_{l_1}^{\gamma_1}\vee\ldots\vee x_{l_{k_t}}^{\gamma_{k_t}}) \qquad (9)$$

($\alpha_i,\ldots,\gamma_i$ - константы).

Рассмотрим задачу проверки выполнимости для формул КНФ. В настоящее время неизвестно алгоритма полиномиальной сложности решения данной задачи. Тривиальный алгоритм требует перебор 2^n наборов значений $(x_1^0,\ldots,x_n^0)$ переменных и вычисления для каждого из них значения формулы.

5. Рассмотрим стандартную задачу линейного программирования: для данных целочисленной $(m\times n)$ -матрицы A, m -вектора b и n -вектора c

а) найти n -вектор x с рациональными координатами, такой, что $x\geq 0$ и $Ax=b$, $c^{\mathrm{T}}\cdot x\to\min$;

либо

б) установить, что не существует n -вектора x, что $x\geq 0$ и $Ax=b$;

либо

в) установить, что множество $\left\{c^{\mathrm{T}}\cdot x\ :\ Ax=b,\ \ x\geq 0\right\}$

не ограничено снизу.

Хачиян Л.Г.(1979) установил, что данная задача принадлежит классу **P** и тем самым разрешил вопрос, долго стоявший открытым.

2°. Определим теперь класс **NP** задач распознавания, т.е. имеющих ответ "ДА" или "НЕТ". Для того, чтобы задача I содержалась в классе **NP** требуется только, чтобы в случае, если I имеет ответ "ДА", то существует слово $c(I)$, длины,

ограниченной полиномом от размера I такое, что задача с начальными данными $c(I),I$ принадлежит **P** . Слово $c(I)$ называется удостоверением или догадкой для задачи I.

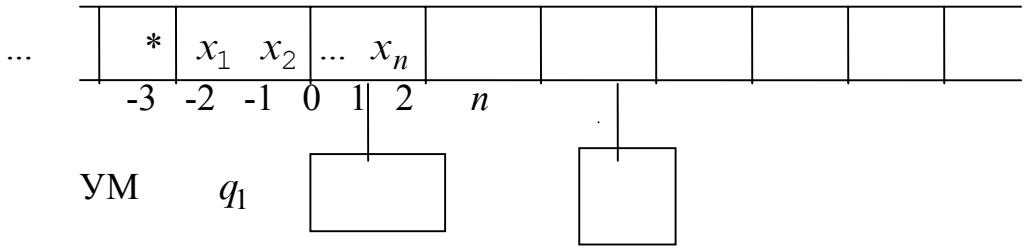
Рассмотрим примеры.

1) Пусть дана задача проверки гамильтоновости бинарного отношения $R \subseteq$ M×M на множестве M=$\{1, 2, \ldots, n\}$. Если R - гамильтоново отношение, то удостоверением этого будет последовательность элементов M $c(R) = i_1 i_2 \ldots i_n$. Имея пару $c(R)$, R, легко убеждаемся, проверяя соотношения (7), верно ли, что R - гамильтоново отношение. Следовательно, задача проверки гамильтоновости бинарного отношения лежит в классе **NP**.

2) Пусть дана задача проверки выполнимости формул КНФ. Если $f(x_1,\ldots,x_n)$ - выполнимая формула, то удостоверением этого будет соответствующий выполняющий набор $(x_1^0,\ldots,x_n^0)$. Имея пару $(x_1^0,\ldots,x_n^0)$, $f(x_1,\ldots,x_n)$, легко убедиться, верно ли, что $f(x_1,\ldots,x_n)$ выполнима.

Формализуем теперь данные идеи. Класс **NP** определяется через понятие недетерминированного алгоритма. Введем понятие недетерминированной машины Тьюринга.

Схема недетерминированной машины Тьюринга:



Отличие от обычной машины Тьюринга заключается в том, что недетерминированная машина (НТ) имеет дополнительно угадывающий модуль (УМ), который может только записывать на ленту слова из внешнего алфавита. Поскольку речь о задачах распознавания, то удобно считать, что машина имеет два заключительных состояния q_Y и q_N, соответствующие ответам "Да" и "НЕТ" соответственно. Работа машины имеет две стадии:

1-я стадия - угадывание. В начальный момент на ленте записано слово $x = x_1,\ldots,x_n$ - код индивидуальной задачи I, начиная с ячейки 1. В ячейке 0 записан знак раздела * . Угадывающий модуль просматривает ячейку -1. Затем УМ пишет произвольное слово $c(x)$ по одной букве за такт в каждой ячейке с номерами -1,-2, -3,... . В итоге 1-ой стадии конфигурацией машины будет $c(x)* q_1 x$.

2-я стадия - решение. На этой стадии недетерминированная машина работает как обычная машина Тьюринга в конфигурации $c(x)* q_1 x$. Если машина через

конечное число шагов приходит в состояние q_Y, то говорим, что она принимает конфигурацию $c(x)^* q_1 x$. Будем говорить, что недетерминированная машина принимает x, если существует слово $c(x)$, что конфигурация $c(x)^* q_1 x$ принимается.

Определим время работы недетерминированной машины, положив

$t_{\text{НТ}}(x)=t_1(x)+t_2(x)$

(если x - принимается)

где $t_1(x)$ - время работы на стадии 1, т.е., по определению, $t_1(x)=|c(x)|$

$t_2(x)$ - время работы на стадии 2, т.е., по определению, $t_2(x)=t_{\text{Т}}(c(x)*x)$

(Т - соответствующая (обычная) машина Тьюринга). Определим

$$t_{\text{НТ}}(n)=\max_{x,|x|=n} t_{\text{НТ}}(x).$$

Недетерминированная машина НТ решает задачу П за полиномиальное время, если

$$t_{\text{НТ}}(n)=O(p(n))$$

для некоторого полинома p.

Заметим, что временная функция определена только для тех индивидуальных задач x, которые принимаются машиной НТ.

Определим класс задач **NP** как множество задач, для которых существует недетерминированная машина Тьюринга, принимающая за полиномиальное время те и только те слова, которые соответствуют индивидуальным задачам с ответом "Да".

Разберем теперь вопрос о взаимоотношении между введенными классами **P** и **NP**. Ясно, что **P** $\subseteq$ **NP .** Имеется много причин считать это включение строгим, однако этот факт пока не доказан (1992).

Теорема. Если задача П$\in$**NP**, то существует такой полином p, что П может быть решена детерминированным алгоритмом со сложностью $O(2^{p(n)})$, n - размер П.

Доказательство. Пусть НТ - недетерминированная машина Тьюринга, решающая задачу П и $q(n)$ - полином, ограничивающий временную функцию сложности НТ. Не нарушая общности, можно считать, что $q(n)=c_1 n^{c_2}$ (c_1, c_2 -константы). По определению класса **NP**, если вход x длины n принимается НТ, то существует слово $c(x)$ длины не более чем $q(n)$, такое, что НТ дает ответ "ДА" не более чем за $q(n)$ шагов. Значит, общее число возможных слов-догадок не более чем $k^{q(n)}$, где k - мощность внешнего алфавита НТ. Считаем, что все догадки имеют длину $q(n)$, в противном случае их можно подравнять. Теперь можно представить детерминированный алгоритм решения задачи П, который на каждом из $k^{q(n)}$ слов-догадок реализует 2-ю стадию работы НТ и работает $q(n)$ тактов. Алгоритм дает ответ "Да", если найдется слово-догадка, приводящая к принимающему вычислению. Время работы данного алгоритма $q(n)\cdot k^{q(n)}$. Ясно, что существует подходящий полином $p(n)$,

что сложность описанного алгоритма не превосходит $O(2^{p(n)})$. Теорема доказана.

Относительно класса **NP** следует сделать несколько замечаний.

1) Класс **NP** один и тот же для различных вычислительных моделей, использующих недетерминированные операции.

2) Класс **P** замкнут относительно дополнения задач. Для класса **NP** этого утверждать нельзя. Дополнением $\overline{A}$ задачи распознавания А называют задачу, в которой коды задач с ответом "Да" в точности соответствуют кодам задач А, которые не имеют ответ "Да". Класс задач, являющихся дополнениями к задачам класса **NP** обозначают **CO- NP.**

§ 15. NP -полные задачи. Теорема Кука

1°. Если **P** ≠ **NP**, то задачи из **NP\P** являются труд- норешаемыми. Цель дальнейших результатов состоит в доказательстве того, что существуют конкретные задачи П, для которых справедливо включение П∈**NP\P**, если **P** ≠ **NP**. Соответствующие результаты основаны на понятии полиномиальной сводимости задач. Пусть Π_1, Π_2 - две задачи распознавания, задаваемые в алфавитах A_1 и A_2 соответственно. Будем говорить, что задача Π_1 полиномиально сводится к задаче Π_2 (Обозначение: $\Pi_1 \propto \Pi_2$), если существует словарная функция $f: A_1^* \to A_2^*$, такая, что выполнены условия :

1) f - полиномиально вычислима;

2) $\forall x \in A_1^*$ выполнено:

x - индивидуальная $\Leftrightarrow$ $f(x) \in A_2^*$ - индивидуальная

задача Π_1 с ответом ДА задача Π_2 с ответом ДА

Утверждение 1. Если выполнено $\Pi_1 \propto \Pi_2$ и $\Pi_2 \in$ **P**, то $\Pi_1 \in$ **P.**

Доказательство. Предложим полиномиальный алгоритм решения задачи Π_1: $x \in \Pi_1$ находим $f(x) \in \Pi_2$ и применяем к $f(x)$ полиномиальный алгоритм, существующий по условию. Если получен ответ ДА, то x имеет ответ ДА. В противном случае x имеет ответ НЕТ. Время работы алгоритма не превосходит $p_f(|x|) + p_2(p_f(|x|))$, где p_f - полином, ограничивающий время вычисления функции f, участвующей в сведении $\Pi_1 \propto \Pi_2$, p_2 - полином, ограничивающий время решения задачи Π_2. Утверждение доказано.

Утверждение 2. Если выполнено $\Pi_1 \propto \Pi_2$ и $\Pi_2 \propto \Pi_3$, то $\Pi_1 \propto \Pi_3$.

Доказательство очевидно, т.к. функция $f_3(x) = f_2(f_1(x))$ осуществляет сведение $\Pi_1 \propto \Pi_3$, если f_1, f_2 дают сведения $\Pi_1 \propto \Pi_2$, $\Pi_2 \propto \Pi_3$ соответственно.

Определение. Задача П называется **NP** -полной, если выполнено

а) П∈**NP**

б) $\Pi_1 \propto$П для любой задачи $\Pi_1 \in$**NP** .

Задача П называется **NP** -трудной, если для нее выполнено условие б).

Обозначим через **NPC** - класс **NP** -полных задач, а через **NPH** - класс **NP** -трудных задач. Согласно определению имеем:

$$\mathbf{NPC} \cap \mathbf{P} \neq \varnothing \Rightarrow \mathbf{P} = \mathbf{NP}$$

$$\mathbf{NPH} \cap \mathbf{P} \neq \varnothing \Rightarrow \mathbf{P} = \mathbf{NP}$$

$$\mathbf{NPC} \cap (\mathbf{NP\backslash P}) \neq \varnothing \Rightarrow \mathbf{NPC} \cap \mathbf{P} = \varnothing$$

Другими словами: Если для какой-то **NP** -полной задачи существует полиномиальный разрешающий алгоритм, то и для любой задачи из класса **NP**

существует полиномиально разрешающий алгоритм. То же высказывание справедливо относительно **NP** -трудной задачи. Если какая-то **NP** -полная задача не лежит в классе **P**, то и все **NP** -полные задачи не лежат в классе **P**.

Утверждение 3. Если задачи $\Pi_1, \Pi_2 \in$ **NP**, $\Pi_1 \propto \Pi_2$ и $\Pi_1 \in$ **NPC**, то $\Pi_2 \in$ **NPC**.

Доказательство. Пусть $\Pi' \in$ **NP** - произвольная задача. Тогда по определению $\Pi' \propto \Pi_1$. Поскольку по условию $\Pi_1 \propto \Pi_2$, то согласно утверждения 2 имеем $\Pi' \propto \Pi_2$, что и доказывает данное утверждение.

Отсюда получаем способ доказательства **NP** -полно-ты конкретных задач, используя полиномиальное сведение к ней другой **NP** -полной задачи. Этот же способ пригоден и для доказательства **NP** -трудности задачи. Напомним, что классы **NPC** и **NPH** определяются только выполнимостью условий а),б) для **NPC** и условия б) для **NPH**.

Докажем теперь важную теорему Кука С. (1971), дающую первый пример **NP** -полной задачи.

2°. *Теорема 4.* Задача проверки выполнимости произвольной КНФ является **NP** -полной задачей.

Доказательство. Пусть ВЫП - идентификатор данной задачи. В предыдущем разделе было показано, что ВЫП$\in$ **NP**.Пусть П - произвольная задача из **NP**. Необходимо показать, что П$\propto$ВЫП. Для этого множество индивидуальных задач L_Π с ответом ДА представим в стандартном виде соответствующей недетерминированной машиной Тьюринга (обозначение: НДМТ), работающей полиномиальное время и принимающей множество L_Π. Такое представление дает общую сводимость задачи, решаемой НДМТ за полиномиальное время.

Пусть распознающая множество L_Π НДМТ имеет алфавиты A,Q и функцию переходов (программу)
$\delta : \mathtt{A} \times \mathtt{Q} \setminus \{q_Y, q_N\} \to \mathtt{A} \times \mathtt{Q} \times \Delta \ \ (\Delta \in \{R, L, S\})$, $p(n)$ - верхняя граница времени вычисления. Функцию f_L, осуществляющую полиномиальное сведение П$\propto$ВЫП опишем в терминах работы НДМТ.

В вычислениях участвуют ячейки ленты с номерами от - $p(n)$ до $p(n)+1$ и при этом требуется учесть не более $p(n)+1$ тактов работы НДМТ. Проверяющее вычисление определяется заданием в каждый момент времени содержания ячеек с указанными номерами, внутреннего состояния машины и положения считывающей головки. Соответствующие вычисления опишем в виде КНФ, использующей полиномиальное число дизъюнкций. Фиксируем нумерацию в алфавитах:

$$\mathtt{Q} : q_0,\ q_1 = q_Y,\ q_2 = q_N,\ q_3, \ldots, q_r$$

$$\mathtt{A} : a_0 = \wedge,\ a_1, \ldots, a_v$$

Условимся, что фраза “в момент времени i ” означает “после выполнения i -го шага работы”. Если вычисление закончилось раньше времени $p(n)$, то конфигурация не меняется во все моменты после остановки. В нулевой момент на ленте записано слово x в ячейках с номерами 1,... ,n. Слово-догадка w пишется

в ячейках с номерами -1,-2,... ,-|w|. Остальные ячейки пусты. Описывая принимающее вычисление необходимо учесть

а) в ячейках пишется точно один символ;

б) машина находится точно в одном состоянии;

в) головка может просматривать точно одну ячейку с номером от - $p(n)$ до $p(n)$ +1;

г) машина работает по программе.

Определим сначала переменные и их смысл с помощью таблицы:

Переменная	Пределы изменения индексов	Смысл
Q(i,k)	$0 \le i \le p(n)$ $0 \le k \le r$	в момент времени i машина находится в состоянии q_k
H(i,j)	$0 \le i \le p(n)$ $-p(n) \le j \le p(n)+1$	в момент времени i головка просматривает ячейку с номером j
a(i,j,k)	$0 \le i \le p(n)$ $-p(n) \le j \le p(n)+1$ $0 \le k \le v$	в момент времени i в j-ой ячейке записан символ a_k

Описание сводящей функции f_L для П $\propto$ ВЫП дадим в виде набора дизъюнкций, конъюнкцией которых и будет f_L. При этом выполняющий набор значений истинности однозначно соответствует принимающему вычислению на x, стадия проверки занимает время $\le p(n)$ шагов, слово-догадка имеет длину $\le p(n)$, причем

$x \in L_{\text{П}}$ $\Leftrightarrow$ на x существует принимающее вычисление

$\Leftrightarrow$ на x существует принимающее вычисление с временем $\le p(n)$ и $|w| \le p(n)$

$\Leftrightarrow$ существует выполняющее значение переменных для задачи $f_L(x)$, заданной КНФ,

при этом $f_L(x)$ вычисляется за полиномиальное время. Определим множества дизъюнкций и их смысл.

G_1 В любой момент времени i машина находится точно в одном состоянии.

G_2 В любой момент времени i головка просматривает точно одну ячейку.

G_3 В любой момент времени i каждая ячейка содержит точно один символ А.

G_4 В момент времени 0 вычисление находится в начальной конфигурации стадии проверки со входом x.

G_5 Не более чем через $p(n)$ шагов машина

переходит в состояние q_Y
(принимает x).

G_6 Для любого времени i, $0 \le i \le p(n)$ конфигурация машины в момент $i+1$ получается из конфигурации в момент i однократным применением команды машины.

Описание дизъюнкций для функции f_L.

G_1 $(\mathtt{Q}(i,0) \vee \mathtt{Q}(i,1) \vee \ldots \vee \mathtt{Q}(i,r)), \quad 0 \le i \le p(n)$

$(\overline{\mathtt{Q}}(i,j) \vee \overline{\mathtt{Q}}(i,j')) \quad 0 \le i \le p(n)$

$0 \le j < j' \le r$

G_2 $(\mathtt{H}(i,-p(n)) \vee \mathtt{H}(i,-p(n)+1) \vee \ldots \vee \mathtt{H}(i,p(n)+1)), \quad 0 \le i \le p(n)$

$(\overline{\mathtt{H}}(i,j) \vee \overline{\mathtt{H}}(i,j')) \quad 0 \le i \le p(n)$

$-p(n) \le j < j' \le p(n)+1$

G_3 $(a(i,j,0) \vee a(i,j,1) \vee \ldots \vee a(i,j,v)), \quad 0 \le i \le p(n)$

$(\overline{a}(i,j,k) \vee \overline{a}(i,j,k')) \quad 0 \le i \le p(n)$

$-p(n) \le j \le p(n)+1$

$0 \le k < k' \le v$

G_4 $(\mathtt{Q}(0,0)), \ (\mathtt{H}(0,1)), \ (a(0,0,0)), \ (a(0,-1,0)), \ldots$

$\ldots(a(0,-p(n),0)), \ (a(0,1,k_1)), \ldots, (a(0,n,k_n))$

$(a(0,n+1,0)), \ldots, (a(0,p(n)+1,0))$

где $x = a_{k_1} a_{k_2} \ldots a_{k_n}$

G_5 $(\mathtt{Q}(p(n),1))$

G_6 *а)* $(\overline{a}(i,j,l), \mathtt{H}(i,j), a(i,j,l)) \quad 0 \le i < p(n)$

$-p(n) \le j \le p(n)+1$

$0 \le l \le v$

Замечание. Дизъюнкция гарантирует, что если головка машины в момент i не просматривает ячейку j, то в момент $i+1$ ее содержимое не меняется.

б) $\forall \ (i,j,k,l) \quad 0 \le i < p(n)$

$-p(n) \le j \le p(n)+1$

$0 \le l \le v$

$(\overline{\mathtt{H}}(i,j) \vee \overline{\mathtt{Q}}(i,k) \vee \overline{a}(i,j,l) \vee \mathtt{H}(i+1, i+\Delta))$

$(\overline{\mathtt{H}}(i,j) \vee \overline{\mathtt{Q}}(i,k) \vee \overline{a}(i,j,l) \vee \mathtt{Q}(i+1,k'))$

$(\overline{\mathtt{H}}(i,j) \vee \overline{\mathtt{Q}}(i,k) \vee \overline{a}(i,j,l) \vee a(i+1,j,l'))$

где Δ,k',l' определены командами машины:

Если $q_k \in \mathtt{Q} \setminus \{q_Y, q_N\}$, то $q_k a_l \to q_{k'} a_{l'} \Delta$.

Если $q_k \in \{q_Y, q_N\}$, то $\Delta = 0$, $k' = k$, $l' = k$.

Заметим, что число дизъюнкций в G_6 - полином от n. Далее, если $x \in L_{\text{П}}$, то существует принимающее вычисление длины $\le p(n)$ и выполнимы все дизъюнкции из $G_1, G_2, G_3, G_4, G_5, G_6$ и $x \in L_{\text{П}} \Leftrightarrow$ существует выполняющий набор для $f_L = G_1 \cdot G_2 \cdot G_3 \cdot G_4 \cdot G_5 \cdot G_6$. Теперь, для любого $x \in L_{\text{П}}$ индивидуальная задача $f_L(x)$ может быть построена за время, ограниченное полиномом от $n=|x|$, при этом длина $f_L(x)$ также ограничена сверху полиномом от n.

Таким образом, преобразование $f_L(x)$ может быть осуществлено за число действий, полиномиально зависящее от n. При этом $f_L(x)$ имеет $O((p(n))^2)$ переменных и $O((p(n))^2)$ - дизъюнкций. Так как П - произвольная задача из **NP**, то тем самым теорема доказана.

§ 16. Основные NP -полные задачи. Сильная NP -полнота

1°. В данном разделе устанавливается **NP** -полнота некоторых известных в различных приложениях задач, Предпочтение отдается графическим задачам как наиболее наглядным. Доказательство получается преобразованием в рассматриваемую задачу другой задачи, **NP** -полнота которой установлена. Для задач с целочисленными параметрами вводится важное понятие - сильная **NP** -полнота.

1. Пусть $f(x_1,\ldots,x_n)$ - формула от булевых переменных $x_1,\ldots,x_n$ в конъюнктивной нормальной форме, где каждая дизъюнкция имеет не более, чем три вхождения переменных. Задача проверки выполнимости таких формул называется задачей 3-выполнимости (идентифика-тор: 3ВЫП).

Утверждение 1. Задача 3ВЫП является **NP** -полной. Доказательство. Достаточно доказать, что ВЫП $\propto$ 3ВЫП. Пусть $F=D_1D_2\ldots D_m$ - индивидуальная задача выполнимость от переменных $x_1,\ldots,x_n$. Пусть $D_i=z_1\vee z_2\vee\ldots\vee z_k$ и $k>3$, $i\in\overline{1,m}$. Положим

$$D_i'=(z_1\vee z_2\vee y_1)(\bar{y}_1\vee z_3\vee y_2)(\bar{y}_2\vee z_4\vee y_3)\ldots$$
$$\ldots(\bar{y}_{k-5}\vee z_{k-3}\vee y_{k-4})(\bar{y}_{k-4}\vee z_{k-2}\vee y_{k-3})(\bar{y}_{k-3}\vee z_{k-1}\vee z_k),$$

где $y_1,y_2,\ldots,y_{k-3}$ - новые переменные.

Покажем, что D_i выполнено $\Leftrightarrow \exists$ значение $y_1,y_2,\ldots,y_{k-3}$ что D_i' выполнено

D_i не выполнено $\Leftrightarrow \forall$ значений $y_1,y_2,\ldots,y_{k-3}$ D_i' не выполнено

Имеем $D_i=0\Rightarrow z_1\vee z_2\vee\ldots\vee z_k=0\Rightarrow D_i'=y_1(\bar{y}_1\vee y_2)(\bar{y}_2\vee y_3)\ldots$
$\ldots(\bar{y}_{k-4}\vee y_{k-3})\bar{y}_{k-3}=0$
$\forall\ y_1,y_2,\ldots,y_{k-3}$

$D_i=1\Rightarrow z_1\vee z_2\vee\ldots\vee z_k=1\Rightarrow \exists\, i \mid z_i=1,\ i\in\overline{1,k}$

Укажем значения переменных $y_1,y_2,\ldots,y_{k-3}$ выполняющие D_i':

	y_1	y_2	y_3	$\ldots$	y_{k-3}
$z_1=1$	0	0	0	...	0
$z_2=1$	0	0	0	...	0
$z_3=1$	1	0	0	...	0
$z_4=1$	1	1	0	...	0

... . . .
z_k=1 1 1 1 ... 1

Проделаем теперь процедуру замены каждой дизъюнкции D_i на D_i' для каждого $i \in \overline{1,m}$ с условием $k>3$. Получим задачу 3-выполнимости за полиномиальное время. По доказанному имеем ВЫП $\propto$ 3ВЫП. Утверждение доказано.

Замечание. Можно доказать, что задача 2-выполни-мости лежит в классе **P**.

2. Рассмотрим графическую задачу: по произвольному графу G(V,E) и числу k узнать, имеется ли в графе G полный подграф с k вершинами (клика). (Граф называется полным, если любые две вершины соединены ребром). (Идентификатор: КЛИКА).

Утверждение 2. Задача КЛИКА является **NP** -полной. Доказательство. Ясно, что КЛИКА $\in$**NP**, т.к. словом-отгадкой для задачи служит список вершин, составляющих клику и детерминированный алгоритм за полиномиальное время проверяет наличие ребра между каждой парой вершин.

Покажем, что ВЫП $\propto$ КЛИКА.

Пусть $F=D_1 D_2 \dots D_k$ - произвольная индивидуальная задача ВЫП. Строим соответствующий граф G_F следующим образом:

Каждому вхождению переменной в F сопоставим вершину графа и присвоим ей обозначение (x^α, i), где x^α - вхождение переменного ($\alpha \in <0,1>$), i - номер соответствующей дизъюнкции. Вершины (x^α, i) и (y^β, j) соединим ребром в том и только в том случае, когда $i \neq j$ и x^α не есть отрицание y^β (т.е. $x \neq y$ или, если $x = y$, то $\alpha = \beta$). Допустим, что F - выполнила и пусть $x_1^0, \dots, x_n^0$ - соответствующий выполняющий набор. В каждом сомножителе F есть вхождение переменной, обратившее его в 1. Выберем по одному такому вхождению из каждой дизъюнкции. Рассмотрим соответствующее множество k вершин графа G_F и покажем, что любые две такие вершины соединены ребром. Действительно, для вершин (x^α, i) и (y^β, j) нет соединяющего ребра лишь в случае $i = j$ либо $x^\alpha = y^{\overline{\beta}}$. Но $i \neq j$, т.к. вхождения переменных взяты из разных дизъюнкций. Если $x^\alpha = y^{\overline{\beta}}$, то одно и то же значение переменного x не может одновременно обратить в 1 x^α и y^β. Значит, из выполнимости F следует наличие клики размера k в G_F.

Обратно, пусть G_F содержит клику размера k. Пусть это набор вершин $(x_{i_1}^{\alpha_1}, j_1), \dots, (x_{i_k}^{\alpha_k}, j_k)$. Покажем, что формула F выполнима. Положим $x_{i_q} = \alpha_q$, $q \in \overline{1,k}$, и тогда $x_{i_q}^{\alpha_q} = 1$. Значения остальных переменных положим

произвольно. Противоречия в выборе значений переменных нет, т.к. если (x^{α}, r) и (x^{β}, s) соединены ребром, то $\alpha = \beta$.

По построению G_F вершинам соответствуют вхождения переменных из разных дизъюнкций и т.к. число вершин равно *k*, то каждая дизъюнкция имеет вхождение переменного, обращающего в 1 при данных значениях переменных, значит и *F* обращается в 1. Построение G_F проводится за полиномиальное время и, следовательно, ВЫП $\propto$ КЛИКА. Утверждение доказано.

3. Говорят, что некоторое множество вершин $V_1 \subseteq V$ графа G(V,E) образует вершинное покрытие графа, если для любого ребра $e \in E$ найдется инцидентная ему вершина $v \in V_1$ этого множества. Задача о вершинном покрытии (Идентификатор: ВП) состоит в том, чтобы по произвольному графу G(V,E) и числу *k* узнать, имеет ли граф вершинное покрытие мощности *k*.

Утверждение 3. Задача ВП является **NP** -полной. Доказательство. Ясно, что ВП $\in$**NP** , т.к. словом-догадкой является список вершин соответствующего вершинного покрытия и правильность ответа проверяется за полиномиальное время. Покажем, что КЛИКА $\propto$ ВП.

Для графа G(V,E) строим граф G', являющийся дополнением G до полного графа (т.е. $G'=(V,E')$, где $e \in E' \Leftrightarrow e \in E$). Покажем, что

A есть полный $\Leftrightarrow$ дополнение $A'=V \setminus A$

подграф в G есть ВП в G'

Действительно, пусть полный подграф с множеством вершин A лежит в G. Тогда если бы для ребра (v_1, v_2) графа G' выполнялось бы $v_1 \in A$ и $v_2 \in A$, то должно быть, что ребра (v_1, v_2) нет в G. Значит, $A'=V \setminus A$ - вершинное покрытие для G'.

Обратно, если A' образует вершинное покрытие графа G', то всякое ребро, оба конца которого находятся в A, не может принадлежать G' и содержится в G, т.е. в G имеется полный подграф с множеством вершин A. Итак, задача о *k* - вершинном полном подграфе сводится к задаче о вершинном покрытии мощности $k' = n - k, \quad n = |V|$. Утверждение доказано.

4. Говорят, что множество вершин $V_1 \subseteq V$ графа G(V,E) независимо, если никакие две вершины из V_1 не связаны ребром. Задача о независимом множестве вершин (Идентификатор: НВ) заключается в том, чтобы для произвольного графа G(V,E) и целого числа *k* выяснить, существует ли в G независимое множество из *k* вершин.

Утверждение 4. Задача НВ является **NP** -полной. Доказательство аналогично предыдущему.

Приведем теперь без доказательства некоторые известные **NP** -полные проблемы. За доказательствами можно обратиться к книге [4].

5. Задача ГАМИЛЬТОНОВ ЦИКЛ (Идентификатор: ГЦ). Для произвольного графа G(V,E) требуется узнать, существует ли перестановка вершин $i_1, i_2, \ldots, i_n$, $n = |\mathrm{V}|$, такая, что выполнено:
$(i_1, i_2) \in \mathrm{E}$, $(i_2, i_3) \in \mathrm{E}, \ldots, (i_{n-1}, i_n) \in \mathrm{E}$, $(i_n, i_1) \in \mathrm{E}$.
(Ясно, что это перефразировка задач о гамнльтоновости бинарного отношения).

6. Задача ЦЕЛОЧИСЛЕННЫЙ РЮКЗАК (Идентификатор: ЦР). Для произвольных натуральных чисел $c_j, j \in \overline{1,n}$ и k требуется узнать, существует ли набор целых чисел $x_j \geq 0, j \in \overline{1,n}$, что выполнено

$$\sum_{j=1}^{n} c_j x_j = k.$$

Вариантом данной задачи является (0,1) -рюкзак, в которой требуется установить существование (0,1) -чисел $x_j, j \in \overline{1,n}$ с условием

$$\sum_{j=1}^{n} c_j x_j = k.$$

7. Множество ребер, разрезающих циклы. Для произвольного графа G(V,E) и целого числа k выяснить, существует ли множество $\mathrm{E}' \subseteq \mathrm{E}$, такое, что $|\mathrm{E}'| = k$ и каждый цикл графа G(V,E) содержит ребро из E'.

8. Множество вершин, разрезающих циклы. То же, но только теперь ищется подмножество множества вершин.

9. Изоморфизм подграфу. Для заданных двух графов $\mathrm{G}(\mathrm{V}_1, \mathrm{E}_1)$ и $\mathrm{H}(\mathrm{V}_2, \mathrm{E}_2)$ выяснить, содержит ли граф G подграф, изоморфный H.

10. Проблема разрешимости диофантовых уравнений (в стандартном двоичном кодировании данных) вида

$$ax^2 + bx + c = 0, \quad a, b, c \in \boldsymbol{Z}.$$

В настоящее время известно большое число **NP** -полных задач из разных областей дискретной математики (несколько тысяч). Мы ограничимся приведением результата, полученного Шеффером Т.И. (1978), дающего бесконечную серию **NP** -полных проблем.

Пусть $\mathrm{S} = \{\mathrm{R}_1, \ldots, \mathrm{R}_m\}$ - любое конечное множество логических отношений. Логическое отношение определяется как некоторое подмножество из $<0,1>^k$ для некоторого целого $k \geq 1$, при этом k называется рангом отношения. Определим S-формулу как произвольную конъюнкцию скобок, каждая вида $\mathrm{R}_i(\xi_1, \xi_2, \ldots)$, где $\xi_1, \xi_2, \ldots$- переменные, число которых соответствует рангу R_i, $i \in \overline{1,m}$. Проблема S-выполнимости это проблема разрешения является ли данная S-формула выполнимой.

Пример, пусть R(x, y, z) - 3-местное логическое отношение с таблицей истинности

$$\{(1,0,0),(0,1,0),(0,0,1)\}.$$

Тогда формула R(x, y, z)R(x, y, u)R(u, u, y) выполнима и (x, y, z, u)=(0,1,0,0) - ее выполняющий набор.

Результат Шеффера состоит в том, что проблема S-выполнимости полиномиально разрешима, если множество удовлетворяет по крайней мере одному из приводимых ниже условий 1)-6). В противном случае проблема **NP** - полна.

1) Каждое отношение R_i, $i \in \overline{1,m}$ из S 0-выполнимо, т.е. $(0,0,...,0) \in R_i$.

2) Каждое отношение R_i, $i \in \overline{1,m}$ из S 1-выполнимо, т.е. $(1,1,...,1) \in R_i$.

3) Каждое отношение R_i, $i \in \overline{1,m}$ из S слабо положительно, т.е. R_i логически эквивалентно формуле КНФ, имеющей самое большее одно переменное с отрицанием в каждой дизъюнкции.

4) Каждое отношение R_i, $i \in \overline{1,m}$ из S слабо отрицательно, т.е. R_i логически эквивалентно формуле КНф, инеющей самое большее одно переменное без отрицания в каждой дизъюнкции.

5) Каждое отношение R_i, $i \in \overline{1,m}$ из S мультиаффинно, т.е. R_i логически эквивалентно формуле, являющейся конъюнкцией линейных форм над полем F_2.

6) Каждое отношение R_i, $i \in \overline{1,m}$ из S биюнктивно, т.е. R_i логически эквивалентно формуле КНФ, имеющей самое большее вхождения 2-х переменных в каждой дизъюнкции.

2°. Установим теперь **NP** -трудность некоторых задач, уже встречавшихся в курсе математической логики. Рассмотрим следующие задачи о булевых функциях.

1) РАВНОВЕРОЯТНОСТЬ БУЛЕВОЙ ФУНКЦИИ. Для данной булевой функции $f(x_1,\ldots,x_n)$, заданной в КНФ, узнать, является ли она равновероятной (т.е. верно ли $\|f\| = 2^{n-1}$).

2) ЛИНЕЙНОСТЬ БУЛЕВОЙ ФУНКЦИИ. Для данной булевой функции $f(x_1,\ldots,x_n)$, заданной в КНФ, узнать, является ли она линейной.

3) СУЩЕСТВЕННОСТЬ ПЕРЕМЕННОГО. Для данных булевой функций $f(x_1,\ldots,x_n)$ в КНФ и целого числа *k* узнать, является ли переменное x_k существенным для *f*.

4) ФУНКЦИОНАЛЬНАЯ ПОЛНОТА. Для данной булевой функции $f(x_1,\ldots,x_n)$ в КНФ выяснить, образует ли *f* функционально полную систему (является ли *f* шефферовой).

Утверждение 5. Задачи 1) - 4) являются **NP** -труд-ными.

Доказательство.

1. Пусть $f(x_1,\ldots,x_n)$ - произвольная индивидуальная задача ВЫП, где $f(x_1,\ldots,x_n) = D_1 D_2 \ldots D_m$, D_i - дизъюнкции. Определим функцию $f^*(x_1,\ldots,x_n,y) = D_1 \ldots D_m \vee y$, где *y* - новое переменное.Имеем

$$f^*(x_1,\ldots,x_n,y) = D_1 \ldots D_m \vee y = (D_1 \vee y)\ldots(D_m \vee y)$$

и, значит, КНФ для функции *f* * строятся по функции *f* за полиномиальное время. Легко видеть, что

$\|f^*\| = \|f\| + 2^n$, где $\|f\|$ - вес функции f. Значит, f^* равновероятна $\Leftrightarrow f$ не выполнима.

Ясно, что условие Задача 1)$\in$**P** $\Rightarrow$ ВЫП$\in$**P**, что означает Задача 1)$\in$**NPH**.

2. Пусть $f(x_1,\ldots,x_n)$ - произвольная индивидуальная задача ВЫП. Определим функцию $f^*(x_1,\ldots,x_n,y_1,y_2) = f(x_1,\ldots,x_n)y_1 \vee y_2$, где y_1, y_2 - новые переменные. Ясно, что КНф для функции f^* строится по f за полиномиальное время. Легко видеть, что f^* линейна $\Leftrightarrow f$ не выполнима и условие Задача 2)$\in$**P** влечет ВЫП$\in$**P**, что означает Задача 2)$\in$**NPH**.

3. Пусть $f(x_1,\ldots,x_n)$ - произвольная индивидуальная задача ВЫП. Образуем функцию $f^*(x_1,\ldots,x_n,y) = f(x_1,\ldots,x_n)y$, где y - новое переменное. Ясно, что y - существенно для $f^* \Leftrightarrow f$ - выполнима. Следовательно, Задача 3)$\in$**P** $\Rightarrow$ ВЫП$\in$**P** и поэтому Задача 3)$\in$**NPH**.

4. Пусть $f(x_1,\ldots,x_n)$ - произвольная индивидуальная задача ВЫП. Образуем функцию $f^*(x_1,\ldots,x_n,y_1,y_2,y_3) =$
$= f \cdot \bar{y}_1\bar{y}_2 \vee \bar{y}_3$. Ясно, что КНФ для f^* строится по f за полиномиальное время. Функция f^* образует функционально полную систему $\Leftrightarrow f$ выполнима. Действительно, если f не выполнима, то $f^* \equiv \bar{y}_3$ и f^* не является функционально полной. Если f выполнима, то пусть $x_1^0,\ldots,x_n^0$ - выполняющий набор. Тогда имеем

$$f^*(x_1^0,\ldots,x_n^0,0,0,1) = f^*(\bar{x}_1^0,\ldots,\bar{x}_n^0,1,1,0) = 1$$

$$f^*(x_1^0,\ldots,x_n^0,0,0,0) = f^*(x_1^0,\ldots,x_n^0,0,0,1) = 1$$

Отсюда следует не самодвойственность и не линейность функции f^*. Очевидно, что f^* не сохраняет нуль, не сохраняет единицу и не монотонна. Значит, функция f^* удовлетворяет критерию Шеффера функциональной полноты. Следовательно, условие Задача 4)$\in$**P** $\Rightarrow$ ВЫП$\in$**P** и поэтому Задача 4)$\in$**NPH**. Утверждение доказано.

Замечание. Легко убедиться, что отрицание задачи 2), задача 3) лежат в классе **NP** и поэтому они **NP** -полны. Неизвестно, верно ли это для задач 1) и 4). Очевидно, что при табличном задании булевых функций рассмотренные задачи имеют полиномиальную сложность.

3°. Разберем еще одно важное понятие, относящееся к обсуждаемому кругу вопросов. Рассмотрим задачу ЦЕЛОЧИСЛЕННЫЙ РЮКЗАК, которая, как отмечалось выше, является **NP** -полной. Пусть $c_1,\ldots,c_n,K$ - натуральные числа и спрашивается, существуют ли такие целые $x_1,\ldots,x_n \geq 0$, что выполнено

$$\sum_{j=1}^{n} c_j x_j = K.$$

Приведем один алгоритм решения данной задачи. Для индивидуальной задачи ЦР($c_1,\ldots,c_n,K$) построим ориентированный граф G($c_1,\ldots,c_n,K$)=(V,E), где $V = \{0, 1, \ldots, K\}$,

$\mathrm{E} = \left\{(m,k)\ ,\ 0 \le m < k \le K \ \text{ и } \ k - m = c_j \ \text{ для некоторого } j \le n\right\}$.

Значит, граф G имеет K+1 вершин и $O(nK)$ дуг.

Утверждение 6. В графе G($c_1,\ldots,c_n,K$) имеется путь из 0 в K тогда и только тогда, когда индивидуальная задача ЦР($c_1,\ldots,c_n,K$) имеет решение.

Доказательство. Пусть $(0 \equiv i_0, i_1,\ldots,i_m \equiv K)$ - нужный путь в графе G. Рассмотрим набор чисел $(s_1,\ldots s_m) =$
$= (i_1 - i_0,\ldots,i_m - i_{m-1})$. Все эти числа содержатся среди чисел $\{c_1,\ldots,c_n\}$ согласно определению графа G. Кроме того, имеем $\sum_{i=1}^{n} s_i = K$. Отсюда следует, что уравнение

$$\sum_{j=1}^{n} c_j x_j = K$$

разрешимо в неотрицательных числах, причем x_j равно числу появлений c_j в последовательности $(s_1,\ldots s_m)$. Обратно, если $\sum_{j=1}^{n} c_j x_j = K$ для неотрицательных целых чисел $x_1,\ldots,x_n$, то можно восстановить некоторый путь из 0 в K в графе G, если положить

$$(s_1,\ldots s_m) = (\underbrace{c_1, c_1, \ldots, c_1}_{x_1 \text{ раз}}, \underbrace{c_2, c_2, \ldots, c_2}_{x_2 \text{ раз}}, \ldots, \underbrace{c_n, c_n, \ldots, c_n}_{x_n \text{ раз}})$$

и пусть из 0 в K имеет вид $(0 \equiv i_0, i_1,\ldots,i_{m-1}, i_m \equiv K)$, где $i_1 = s_1$, $i_2 = s_1 + s_2,\ldots$, $i_m = s_1 + \ldots + s_m$. Утверждение доказано.

Утверждение 7. Любая индивидуальная задача ЦР может быть решена за $O(nK)$ действий.

Доказательство. По данным $c_1,\ldots,c_n,K$ строим граф G за $O(nK)$ действий. Затем за $O(nK)$ действий проверяем существует ли путь из 0 в K, используя способ пометок: вершину 0 помечаем 0, вершины, достижимые из 0 за 1 шаг, помечаем 1 и т.д. Если K получает пометку, то задача разрешима, если нет, но неразрешима. Утверждение доказано.

Приведенный результат показывает, что **NP** -полная задача ЦЕЛОЧИСЛЕННЫЙ РЮКЗАК решается с помощью алгоритма с временной сложностью $O(nK)$ - полиномиального и, следовательно, доказано, что **P = NP** и можно считать ненужным предыдущее и последующее обсуждение теории **NP** - полноты. Дело в том, что оценка $O(nK)$ не является полиномиальной функцией от длины входа, т.к. целые числа в экономном кодировании должны задаваться в двоичной системе счисления. В то же время приведенный результат важен, т.к. он показывает, что **NP** -полные задачи имеют разную "сложность".

Для задач с числовыми параметрами введем следующие определения. Пусть I - индивидуальная вычислительная задача, т.е. с числовыми параметрами. Обозначим через $num\ (I)$ - наибольшее целое число, появляющееся в I.

Определение. Пусть A - вычислительная задача и f:**N**$\to$**N** - числовая функция. Обозначим через A_f подзадачу задачи A, в которой берутся индивидуальные задачи I, для которых выполнено

$$num(I) \le f(\ |I|\ )$$

Говорят, что задача А сильно **NP** -полна, если для некоторго полинома $p(n)$ задача $\mathtt{A}_p$ является **NP** -полной.

Замечание. Можно показать, что задачи КЛИКА, ГАМИЛЬТОНОВ ЦИКЛ являются сильно **NP** -полными, а задачи (0,1)-РЮКЗАК и ЦЕЛОЧИСЛЕННЫЙ РЮКЗАК не являются таковыми.

Определение. Алгоритм *a* для задачи А называют псевдополиномиальным, если он решает любую индивидуальную задачу $I \in$ А за время, ограниченное полиномом (двух переменных) от $|I|$ и $num\ (I)$. Значит, алгоритм со сложностью $O(nK)$ для задачи ЦЕЛОЧИСЛЕННЫЙ РЮКЗАК является псевдополиномиальным (Ясно, что для индивидуальной задачи I ЦР $num\ (I)=K$).

Отметим, что сильная **NP** -полнота задачи делает маловероятным существование псевдополиномиального алгоритма точно также, как **NP** -полнота задачи делает маловероятным существование полиномиального алгоритма.

Утверждение 8. Если **P = NP**, то ни для одной сильно **NP** -полной задачи не существует псевдополиномиального алгоритма.

Доказательство. Пусть А - сильно **NP** -полная задача. Значит, для некоторого полинома $p(n)$ задача $\mathtt{A}_p$ является **NP** -полной. Далее, пусть для А существует псевдополиномиальный алгоритм *a*, который решает любую индивидуальную задачу $I \in$ А за время $q(|I|, num\ (I))$ для некоторого полинома q от двух переменных. Тогда очевидно, что алгоритм *a* решает **NP** -полную задачу $\mathtt{A}_p$ за время $q(n,p(n))$ - что является полиномиальной оценкой. Получено противоречие при **P = NP.** Утверждение доказано.

§ 17. Сложность алгоритмов, использующих рекурсию

1°. Рекурсия является важным и очень общим алгоритмическим приемом для построения эффективных алгоритмов. Данный прием заключается в решении задачи путем сведения ее к одной или нескольким подзадачам за счет разбиения исходной задачи. Используя рекурсию, часто можно достаточно просто представить и записать алгоритмы. Многие языки программирования (Алгол, PL/1, Паскаль, но не ФОРТРАН) допускают рекурсивные процедуры. Для многих практически важных задач лучшие оценки сложности дают алгоритмы, использующие рекурсию. Рассмотрим несколько примеров.

1. Сортировка чисел. Дана последовательность $x_1, x_2, \ldots, x_n$ натуральных чисел. Требуется путем попарных сравнений чисел упорядочить ее, т.е. представить в виде

$$x_{i_1}, x_{i_2}, \ldots, x_{i_n}, \qquad (1)$$

причем $x_{i_1} \le x_{i_2} \le \ \ldots \ \le x_{i_n}$.

Ясно, что данная задача может быть решена с помощью алгоритма, который попарными сравнениями находит наименьший элемент последовательности и ставит его в начало результирующей последовательности и затем повторяет данный шаг. Легко видеть, что такой алгоритм требует $O(n^2)$ попарных сравнений в худшем случае. Предложим для данной задачи рекурсивный алгоритм. Пусть $n=2^k$.

При k=1 алгоритм упорядочивает последовательность одним сравнением.

Пусть для k алгоритм определен. Тогда при k +1 алгоритм работает так:

1. Последовательность $x_1, x_2, \ldots, x_{2^{k+1}}$ разбивается на две длины 2^k: $x_1, x_2, \ldots, x_{2^k}$ и $x_{2^k+1}, \ldots, x_{2^{k+1}}$.

2. К обеим последовательностям длины 2^k применяется построенный алгоритм и получаем две упорядоченные последовательности: $x'_1, x'_2, \ldots, x'_{2^k}$ и $x'_{2^k+1}, \ldots, x'_{2^{k+1}}$.

3. Осуществляется слияние двух полученных упорядоченных последовательностей сравнением их наименьших элементов x'_1 и x'_{2^k+1} и помещением наименьшего в начало результирующей последовательности.

Пусть $n \ne 2^k$. Тогда последовательность дополняется нулями, так, чтобы ее длина стала степенью двойки.

Пусть T(n) - число попарных сравнений, используемых в данном алгоритме для произвольной последовательности длины n. Тогда получаем соотношение

$$\mathrm{T}(n)=2\mathrm{T}\left(\frac{n}{2}\right)+n-1 \qquad (2)$$

T(2)=1.

Утверждение 1. Справедлива формула

$$\mathrm{T}(2^k) = 2^k \cdot k - 2^k + 1 \qquad (3)$$

Доказательство. При k=1 имеем $\mathrm{T}(2^1) = 1$ и начальные условия выполнены. Пусть для k формула (3) есть решение соотношения (2). Тогда при $k+1$ имеем

$$\mathrm{T}(2^{k+1}) = 2\mathrm{T}(2^k) + 2^{k+1} - 1 = 2(2^k \cdot k - 2^k + 1) + 2^{k+1} - 1 =$$

$$= 2^{k+1} \cdot k - 2^{k+1} + 2^{k+1} + 1 = (k+1)2^{k+1} - 2^{k+1} + 1.$$

Значит при $k+1$ формула (3) есть решение соотношения (2). Утверждение доказано.

Ясно, что для произвольного n справедливо

$$\mathrm{T}(n)=O(n \ log \ n) \qquad (4)$$

Таким образом, представленный рекурсивный алгоритм лучше по порядку исходного "наивного" алгоритма. Можно доказать, что данный алгоритм асимптотически оптимален.

2. Умножение n - разрядных двоичных чисел.

Даны два n - разрядных двоичных числа и требуется найти их произведение. "Наивный" алгоритм умножения "столбиком" требует $O(n^2)$ битовых операций. Предложим рекурсивный алгоритм для данной задачи. Пусть x и y - два n - битовых числа и пусть $n=2^k$, в противном случае дополняем числа слева нулями. Разобьем числа x и y на две равные части в виде $x = \underline{a \mid b}$, $y = \underline{c \mid d}$ и будем рассматривать каждую часть как $\frac{n}{2}$ - разрядное двоичное число.

Тогда произведение $x \cdot y$ можно представить в виде

$$x \cdot y = (a \cdot 2^{\frac{n}{2}} + b)(c \cdot 2^{\frac{n}{2}} + d) = ac2^n + (ad + bc)2^{\frac{n}{2}} + bd \qquad (5)$$

Равенство (5) дает способ вычисления произведения $x \cdot y$ с помощью четырех умножений $\frac{n}{2}$ - разрядных чисел и некоторого числа сложений и сдвигов (умножений на степень числа 2).

Действительно, находим

$$u = (a + b)(c + d)$$

$$v = a \cdot c$$

$$w = b \cdot d \qquad (6)$$

$$z = v \cdot 2^n + (u - v - w)2^{\frac{n}{2}} + w$$

Ясно, что операции сложения и сдвига имеют сложность $O(n)$.

Заметим, что $a+b$ и $c+d$ имеют, вообще говоря, $\frac{n}{2}+1$ разрядов. Тогда запишем

$$a + b = a_1 \cdot 2^{\frac{n}{2}} + b_1$$

$$c + d = c_1 \cdot 2^{\frac{n}{2}} + d_1$$

$$\text{и } (a+b)(c+d) = a_1 c_1 2^n + (a_1 d_1 + b_1 c_1) 2^{\frac{n}{2}} + b_1 d_1 .$$

Произведение $b_1 d_1$ находится с помощью рекурсивного алгоритма на задачах размера $\frac{n}{2}$. Остальные вычисления можно выполнить за время $O(n)$, поскольку они содержат один из аргументов либо единственный бит a_1 или c_1, либо степень числа 2.

Таким образом число используемых операций T(n) ограничено сверху

$$\mathrm{T}(n)=\begin{cases} k, & \text{при } n=1 \\ 3\mathrm{T}\left(\frac{n}{2}\right)+kn, & \text{при } n>1 \end{cases} \qquad (7)$$

где k - постоянная, отражающая число сложений и сдвигов в алгоритме (6).

Утверждение 2. Справедлива формула

$$\mathrm{T}(n) = 3kn^{\log_2 3} - 2kn \qquad (8)$$

Доказательство. При n=1 начальные условия выполнены. Пусть для n формула (8) есть решение соотношения (7). Тогда имеем

$$\mathrm{T}(2n) = 3\mathrm{T}(n) + 2kn = 3\left[3kn^{\log_2 3} - 2kn\right] + 2kn = 3k(2n)^{\log_2 3} - 2k(2n)$$

(Здесь использовано равенство $3 = 2^{\log_2 3}$). Таким образом, формула (8) справедлива для 2n. Утверждение доказано.

Ясно, что T(n)=$O(n^{\log_2 3})$ и данный алгоритм лучше по порядку исходного "наивного" алгоритма.

Для данной задачи известны и более лучшие алгоритмы, с которыми можно ознакомиться, например, в [2].

2а. Возведение в степень. Фиксируется элемент a некоторой алгебраической системы с операцией умножения и натуральное число n. Требуется найти a^n. Тривиальный алгоритм требует n - 1 умножение. Следующий рекурсивный алгоритм требует существенно меньшее число умножений. Алгоритм работает так:

Пусть $u(n)=a^n$. Если n=1, то $u(1)=a$. Если n >1, то находим $k=\left[\frac{n}{2}\right]$ и $l=res(2,n)$ - остаток от деления n на 2. Тогда справедливо

$$u(n) = u\left(\frac{n}{2}\right) \cdot u\left(\frac{n}{2}\right), \quad \text{если} \quad l = 0$$

$$u(n) = u\left(\left[\frac{n}{2}\right]\right) \cdot u\left(\left[\frac{n}{2}\right]\right) \cdot a, \quad \text{если} \quad l = 1$$

Нетрудно убедиться, что в данном алгоритме используется не более $2\lceil \log_2 n \rceil$ умножений.

3. Умножение матриц. Даны две (n x n) матрицы с элементами из некоторого кольца K и требуется найти их произведение. Стандартный алгоритм требует n^3 умножений и $n^3 - n^2$ сложений элементов K. На первый взгляд кажется, что невозможно сколько-нибудь существенно снизить данное число операций. Однако, был предложен следующий алгоритм (Штрассен В., 1969):

Пусть n=1. Тогда произведение матриц находится одним умножением.

Пусть n=2 и даны матрицы

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

и пусть $A \cdot B = C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$.

Тогда матрица C может быть вычислена так:

Пусть $p_1=(a_{11} + a_{22})(b_{11} + b_{22})$

$p_2=(a_{21} + a_{22})b_{11}$

$p_3=a_{11}(b_{12} - b_{22})$

$p_4=a_{22}(-b_{11} + b_{21})$ (7)

$p_5=(a_{11} + a_{12})b_{22}$

$p_6=(-a_{11} + a_{21})(b_{11} + b_{12})$

$p_7=(a_{12} - a_{22})(b_{21} + b_{22})$

Тогда находим

$c_{11} = p_1 + p_4 - p_5 + p_7$

$c_{12} = p_3 + p_5$

$c_{21} = p_2 + p_4$

$c_{22} = p_1 + p_3 - p_2 + p_6$

Заметим, в данном алгоритме использовано 7 умножений и 18 сложений и не использована коммутативность умножения.

Пусть теперь $n = 2^{k+1}$. Тогда алгоритм умножения матриц работает так: матрицы A и B порядка $2^{k+1} \times 2^{k+1}$ представляются как (2 x 2)-матрицы над кольцом матриц порядка $2^k \times 2^k$ и применяется изложенный выше алгоритм умножения (2 x 2)-матриц. Если $n \neq 2^{k+1}$, то находится такое k, что

$2^k < n \le 2^{k+1}$, и к матрицам добавляется нужное число нулевых строк и столбцов.

Пусть T(2^k) - число арифметических операций, используемых в алгоритме на матрицах размера $2^k \times 2^k$. Тогда справедливо соотношение

$$\begin{aligned} &T(2^{k+1}) = 7T(2^k) + 18(2^k)^2 \\ &T(2^0) = 1 \end{aligned} \qquad (9)$$

Утверждение 3. Справедлива формула

$$T(2^k) = 7^{k+1} - 6 \cdot 2^{2k} \qquad (10)$$

Доказательство. При k=0,1, формула (10) справедлива. Пусть для k формула (10) есть решение соотношения (9). Имеем при k+1:

$$T(2^{k+1}) = 7T(2^k) + 18(2^k)^2 = 7(7^{k+1} - 6 \cdot 2^{2k}) + 18 \cdot 2^{2k} =$$
$$= 7^{k+2} - 42 \cdot 2^{2k} + 18 \cdot 2^{2k} = 7^{k+2} - 6 \cdot 2^{2(k+1)},$$

т.е. формула (10) справедлива и для k+1. Утверждение доказано.

Ясно, что выполняется

$$T(n) = O(7^{\log_2 n}) = O((2^{\log_2 7})^{\log_2 n}) = O(n^{\log_2 7}) \qquad (11)$$

Значит, представленный алгоритм лучше по порядку исходного алгоритма (заметим что $\log_2 7 = 2.807$).

Для умножения (2x2)-матриц Виноград предложил алгоритм, требующий 7 умножений и 15 сложений:

$$\begin{aligned} s_1 &= a_{21} + a_{22} & m_1 &= s_2 s_6 \\ s_2 &= s_1 - a_{11} & m_2 &= a_{11} b_{11} \\ s_3 &= a_{11} - a_{21} & m_3 &= a_{12} b_{21} \\ s_4 &= a_{12} - s_2 & m_4 &= s_3 s_7 \\ s_5 &= b_{12} - b_{11} & m_5 &= s_1 s_5 \\ s_6 &= b_{22} - s_5 & m_6 &= s_4 b_{22} \\ s_7 &= b_{22} - b_{12} & m_7 &= a_{22} s_8 \\ s_8 &= s_6 - b_{21} & & \\ t_1 &= m_1 + m_2 & t_2 &= t_1 + m_4 \\ c_{11} &= m_2 + m_3 & c_{21} &= t_2 - m_7 \\ c_{12} &= t_1 + m_5 + m_6 & c_{22} &= t_2 + m_5 \end{aligned}$$

Использование данного алгоритма в качестве базового и рекурсии дает следующее рекуррентное уравнение для сложности соответствующего алгоритма.

$$\begin{aligned} &T(2^{k+1}) = 7T(2^k) + 15 \cdot 2^{2k} \\ &T(2^0) = 1 \end{aligned}$$

Легко убедиться, что решением данного уравнения является

$$T(2^k) = 6 \cdot 7^k - 5 \cdot 2^{2k}.$$

Обратимся теперь к задаче обращения матриц. Для применимости предлагаемого алгоритма требуется предположить не только обратимость матрицы, но и обратимость матриц, появляющихся на промежуточных этапах алгоритма. (Аналогичные предложения имеются и в алгоритме Гаусса).

Рекурсивный алгоритм обращения матрицы следует из приводимых ниже легко проверяемых тождеств и трактовки (n x n)-матриц как (2x2)-матриц над кольцом $\left(\frac{n}{2} \times \frac{n}{2}\right)$ -матриц.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} E & 0 \\ A_{21}A_{11}^{-1} & E \end{pmatrix} \begin{pmatrix} A_{11} & 0 \\ 0 & D \end{pmatrix} \begin{pmatrix} E & A_{11}^{-1}A_{12} \\ 0 & E \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} E & -A_{11}^{-1}A_{12} \\ 0 & E \end{pmatrix} \begin{pmatrix} A_{11}^{-1} & 0 \\ 0 & D^{-1} \end{pmatrix} \begin{pmatrix} E & 0 \\ -A_{21}A_{11}^{-1} & E \end{pmatrix}, \quad D = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

Для нахождения обратной (n x n)-матрицы используется 2 обращения, 6 умножений и 2 сложения $\left(\frac{n}{2} \times \frac{n}{2}\right)$ -матриц. Отсюда легко получить (используя "быстрый" алгоритм умножения матриц), что сложность I(n) обращения матриц удовлетворяет соотношению

$$\mathrm{I}(n) = O(n^{\log_2 7}).$$

Далее, пусть T(n) - сложность умножения (n x n)-матриц и пусть T($2n$) $\geq (2+\varepsilon)$T(n) для некоторого $\varepsilon>0$ и всех n. Тогда из приведенного алгоритма обращения матрицы следует, что I(n)=O(T(n)). Заметим, что произведение (n x n)-матриц можно найти, обращая ($3n$ x $3n$)-матрицы, что следует из тождества

$$\begin{pmatrix} E & A & 0 \\ 0 & E & B \\ 0 & 0 & E \end{pmatrix}^{-1} = \begin{pmatrix} E & -A & AB \\ 0 & E & -B \\ 0 & 0 & E \end{pmatrix}$$

Отсюда имеем I($3n$) $\geq$ T(n) . Поскольку выполнено $\mathrm{T}\left(\left[\frac{n}{4}\right]\right) \geq k\mathrm{T}(n)$ для некоторого $k>0$, то имеем I(n) $\geq k$ T(n) .

Наконец, отправляясь от тождества

$$\det A = \det A_{11} \cdot \det(A_{22} - A_{21}A_{11}^{-1}A_{12})$$

можно показать, что сложность вычисления определителя матриц с точностью до константы совпадает со сложностью обращения матрицы. Итак, сложность таких практически важных задач как решение системы линейных уравнений, обращение невырожденной матрицы, вычисление определителя матрицы имеет тот же порядок, что и сложность умножения матриц.

2°. Рассмотрим теперь в общем виде вопрос о сложности алгоритмов, использующих рекурсию. Из приведенных выше примеров, ясно, что сложность рекурсивных алгоритмов выражается рекуррентными соотношениями. Если в рекурсивном алгоритме используется одна процедура, то значение сложности

T(N) для задачи размера N определяется значениями T(m) для аргументов m, где $m < N$. Однако возникающие при этой рекуррентные соотношения решаются в конечном виде в очень редких случаях. При равномерном разбиении задача размера N разбивается на k подзадач размера $\frac{N}{k}$, k - некоторая фиксированная константа в рекурсивном алгоритме, при этом функция сложности T(N) определяется рекуррентным соотношением вида:

$$\begin{aligned} &\mathrm{T}(N) = \mathrm{A}(N)\mathrm{T}\left(\frac{N}{k}\right) + \mathrm{B}(N) \\ &\mathrm{T}(1) = c \end{aligned} \qquad (12)$$

где c - константа, A(N),B(N) неотрицательные, целочисленные, монотонные функции, характеризующие сложность получения решения исходной задачи размера N из k решений подзадач размера $\frac{N}{k}$. Ясно, что соотношение (12) определяет однозначно функцию T(N) только при значениях аргументов N вида k^{p}, P=0,1,... . Для практических приложений представляет интерес выделение случаев, когда решение T(N) соотношения (12) ограничено сверху полиномом от N.

Легко доказывается следующее

Утверждение 4. Пусть функция A(N) удовлетворяет условию: существует $\varepsilon > 0$, такое, что справедливо соотношение

$$\mathrm{A}(kN) \geq (1+\varepsilon)\mathrm{A}(N) \qquad (13)$$

для всех натуральных N.

Тогда решение T(N) соотношения (12) не мажорируется сверху никаким полиномом от N.

Таким образом в дальнейшем ограничимся случаем A(N)=a, B(N)=bN^t, где a,b,t - фиксированные целые числа.

Справедливо

Утверждение 5. Пусть дано рекуррентное соотношение

$$\begin{aligned} &\mathrm{T}(N) = a\mathrm{T}\left(\frac{N}{k}\right) + bN^t \\ &\mathrm{T}(1) = c \end{aligned} \qquad (14)$$

где a,b,c,k,t - фиксированные константы. Тогда при $N=k^{\mathrm{p}}$, p=0,1, ... решением соотношения (14) является выражение

$$\mathrm{T}(k^{\mathrm{p}}) = \begin{cases} a^{\mathrm{p}} \cdot c + b \cdot a^{\mathrm{p}} \cdot \mathrm{p}, & \text{если } a = k^t \\ a^{\mathrm{p}} \cdot c + b \cdot k^{\mathrm{p}t} \dfrac{\left(\frac{a}{k^t}\right)^{\mathrm{p}} - 1}{\frac{a}{k^t} - 1}, & \text{если } a \neq k^t \end{cases} \qquad (15)$$

Доказательство. Докажем утверждение индукцией по p. При p=0 имеем T(k^0)=c в обоих случаях, и выражение (15) совпадает с начальными условиями. Пусть $a=k^t$ и при $N=k^p$ формула (15) является решением соотношения (14). Покажем справедливость этого при $N_1=k^{p+1}$. Имеем

$$T(N_1) = aT(k^p) + bk^{(p+1)t} =$$

$$= a\left[a^p \cdot c + b \cdot a^p \cdot p\right] + b \cdot k^{(p+1)t} = a\left[a^p \cdot c + b \cdot a^p \cdot p\right] + b \cdot a^{p+1} =$$

$$= a^{p+1} \cdot c + b \cdot a^{p+1}(p+1), \quad \text{т.е. при } N_1 = k^{p+1}$$

формула (15) является решением (14).

Пусть теперь $a\neq k^t$ и при $N=k^p$ формула (15) есть решение (14). Тогда имеем для $N_1=k^{p+1}$

$$T(N_1) = aT(k^p) + b \cdot k^{(p+1)t} = a\left[a^p \cdot c + b \cdot k^{pt} \frac{\left(\frac{a}{k^t}\right)^p - 1}{\frac{a}{k^t} - 1}\right] + b \cdot k^{(p+1)t} =$$

$$= a^{(p+1)} \cdot c + b \cdot k^{(p+1)t}\left[\frac{a\left(\frac{a}{k^t}\right)^p - a}{k^t\left(\left(\frac{a}{k^t}\right) - 1\right)} + 1\right] = a^{(p+1)} \cdot c + b \cdot k^{(p+1)t} \frac{\left(\frac{a}{k^t}\right)^{p+1} - 1}{\left(\frac{a}{k^t}\right) - 1}$$

и, значит, формула (15) является решением (14) при $N_1=k^{p+1}$, что и доказывает утверждение.

Следствие. В предположениях предыдущего утверждения справедливы соотношения

$$T(N) = \begin{cases} O(N^t \log_k N), & \text{если } a = k^t \\ O(N^t), & \text{если } a < k^t \\ O(N^{\log_k a}), & \text{если } a > k^t \end{cases} \qquad (16)$$

В некоторых случаях рекурсию для задачи размера N удается организовать только путем аддитивного уменьшения исходного размера задачи N на некоторую константу k. Сложность T(N) такого типа рекурсивных алгоритмов определяется рекуррентным соотношением вида

$$\begin{aligned} T(N) &= aT(N-k) + bN^t \\ T(0) &= c \end{aligned} \qquad (17)$$

где a,b,c,t - константы. Например, такая ситуация возникает при решении графических задач.

Справедливо

Утверждение 6. Для функции T(N), являющейся решением уравнения (17) при $N=k$p, p=0,1, ... справедлива оценка

$$\mathrm{T}(N) \le \begin{cases} a^{\frac{N}{k}} \cdot c + bN^t \dfrac{a^{\frac{N}{k}} - 1}{a-1}, & a \ne 1 \\ c + b\dfrac{N^{t+1}}{k}, & a = 1 \end{cases} \qquad (18)$$

В некоторых случаях для организации рекурсии используется "квадратичное" разбиение, при котором исходная задача размера N разбивается на $\sqrt{N}$ подзадач размера $\sqrt{N}$. Для сложности T(N) рекурсивного алгоритма возникает следующее рекуррентное уравнение

$$\begin{aligned} &\mathrm{T}(N) = a\sqrt{N}\,\mathrm{T}(\sqrt{N}) + bN \\ &\mathrm{T}(2) = c \end{aligned} \qquad (19)$$

где a,b,c - фиксированные константы.

Данное соотношение определяет однозначно функцию T(N) при $N=2^{2^n}$, n=0,1, ...

Утверждение 7. Решением рекуррентного соотношения является выражение

$$\mathrm{T}(N) = \begin{cases} n \cdot 2^{2^n} \cdot b + c \cdot 2^{2^n - 1}, & a = 1 \\ a^n \cdot c \cdot 2^{2^n - 1} + b\dfrac{a^n - 1}{a-1}, & a \ne 1 \end{cases} \qquad (20)$$

Доказательство. Докажем утверждение индукцией по n. При n=0 имеем $\mathrm{T}(2^{2^0}) = 1$ в обоих случаях и выражение (20) совпадает с начальными условиями. Пусть a=1 и при n формула (20) дает решение соотношения (19).
Имеем при n +1

$$\mathrm{T}(2^{2^{n+1}}) = 2^{2^n} \cdot \mathrm{T}(2^{2^n}) + b \cdot 2^{2^{n+1}} = 2^{2^n}\left[n \cdot 2^{2^n} \cdot b + c \cdot 2^{2^n - 1} \right] + b \cdot 2^{2^{n+1}} =$$

$$= n \cdot 2^{2^{n+1}} \cdot b + c \cdot 2^{2^{n+1} - 1} + b \cdot 2^{2^{n+1}} = (n+1) \cdot 2^{2^{n+1}} \cdot b + c \cdot 2^{2^{n+1} - 1}$$

и для n +1 формула (20) является решением соотношения (19).

Пусть $a \ne 1$ и при n утверждение справедливо. Имеем

$$\mathrm{T}(2^{2^{n+1}}) = a\cdot 2^{2^n}\,\mathrm{T}(2^{2^n}) + b\cdot 2^{2^{n+1}} = a\cdot 2^{2^n}\left[a^n c\cdot 2^{2^n-1} + b\frac{a^n-1}{a-1}2^{2^n}\right] + b\cdot 2^{2^{n+1}} =$$

$$= a^{n+1}c\cdot 2^{2^{n+1}-1} + ab\frac{a^n-1}{a-1}2^{2^{n+1}} + b\cdot 2^{2^{n+1}} =$$

$$= a^{n+1}c\cdot 2^{2^{n+1}-1} + \left[ab\frac{a^n-1}{a-1}+b\right]2^{2^{n+1}} = a^{n+1}c\cdot 2^{2^{n+1}-1} + b\frac{a^{n+1}-1}{a-1}2^{2^{n+1}}$$

и при n +1 утверждение справедливо.

3°. Данные результаты имеют практический интерес. В качестве примера заметим, что для задачи сортировки можно предложить рекурсивные алгоритмы, использующие как разбиение задачи на произвольное фиксированное число подзадач так и квадратичное разбиение. Эти алгоритмы дают более лучшие оценки, чем приведенные выше.

Для задачи умножения матриц использование базового алгоритма умножения (r x r) -матриц с a умножениями для построения рекурсивного алгоритма, сводящего умножение (N x N) -матриц к умножению (N/r)x(N/r) -матриц приводит к оценкам для сложности T(N) :

$$\mathrm{T}(N) = \begin{cases} O(N^{\log_r a}) & ,\text{если } a > r^2 \\ O(N^2 \log_r N) & ,\text{если } a = r^2 \\ O(N^2) & ,\text{если } a < r^2 \end{cases}$$

(В алгоритме Штрассена r = 2, a = 7) В.Пан (1978) использовал алгоритм с параметрами r = 70 с a = 143640 и получил

$$\mathrm{T}(N) = O(N^{2\cdot 796})$$

В настоящее время известны и более лучшие алгоритмы.
Усилиями ряда математиков (Пан, Виноград, Романи, Шеихаге, Штрассен, Копперсмит) экспонента сложности матричного умножения последовательно принимала значения:

$$\omega = 2.6054$$

$$\omega = 2.523$$

$$\omega = 2.5166$$

$$\omega = 2.495364$$

$$\omega = 2.40364$$

$$\omega = 2.376$$

Основная проблема - нахождение доказательства равенства ω = 2 еще не решена (1991).

Укажем на другие имеющиеся результаты по нахождению эффективных базовых алгоритмов умножения (r x r) - матриц.

Для $r = 3$ предложен алгоритм, использующий 23 умножения, но он не приводит к улучшению оценки, т.к. $\log_3 21 < \log_2 7 < \log_3 22$.

Для $r = 5$ найден алгоритм, использующий 100 умножений.

Рассмотрим еще один пример применения полученных результатов. Вернемся к задаче умножения чисел в двоичной записи. Зафиксируем натуральное число $r \geq 2$ и рассмотрим два $r \cdot n$ -битовых числа u и v, где

$$u = u_{r \cdot n-1} \ldots u_1 u_0$$

$$v = v_{r \cdot n-1} \ldots v_1 v_0$$

Разобьем эти числа на r частей

$$u = u_{r-1} 2^{(r-1)n} + \ldots + u_1 2^n + u_0$$

$$v = v_{r-1} 2^{(r-1)n} + \ldots + v_1 2^n + v_0$$

Здесь каждое u_i и v_i являются n - битовыми числами, $i \in \overline{1, r-1}$.

Рассмотрим многочлены

$$u(x) = u_{r-1} x^{r-1} + \ldots + u_1 x + u_0$$

$$v(x) = v_{r-1} x^{r-1} + \ldots + v_1 x + v_0$$

и образуем произведение

$$w(x) = u(x) \cdot v(x) = w_{2(r-1)} x^{2(r-1)} + \ldots + w_1 x + w_0$$

Ясно, что выполнено

$$u = u(2^n)$$

$$v = v(2^n)$$

$$u \cdot v = w(2^n)$$

Поэтому, знание многочлена $w(x)$ позволяет вычислить произведение $u \cdot v$ за число действий, пропорциональное n . Чтобы найти коэффициенты многочлена $w(x)$ находим значение $w(x)$ в $2(r-1)$ точках $0,1, \ldots, 2(r-1)$:
$u(0)v(0)=w(0)$, $u(1)v(1)=w(1)$,... , $u(2(r-1))\, v(2(r-1)) = w(2(r-1))$.
Разрядность чисел $u(i)$ (соответственно $v(i)$) не превышает $n + t$, где t - некоторая константа, зависящая от r. Ясно, что если $T(n)$ - сложность умножения n - битовых чисел, то сложность умножения $(n + t)$ - битовых чисел есть $T(n)+c'n$ для некоторой константы c'.

Если $T(r \cdot n)$ - сложность умножения $r \cdot n$ - битовых чисел, то справедливо соотношение

$$T(r \cdot n) = (2(r-1)+1)T(n) + c''n$$

(c'' - константа).
Отсюда, согласно (16), получаем

$$T(n) = O(n^{\log_r (2(r-1)+1)})$$

Поскольку имеем $\log_r (2(r-1)+1) < 1 + \log_r 2$ и, обозначая $\varepsilon = \log_r 2$ можно записать

$$T(n) = O(n^{1+\varepsilon})$$

Таким образом, доказано

Утверждение 8. Дня любого ε >0 существует алгоритм умножения n - битовых чисел, для которого сложность удовлетворяет соотношению

$$T(n) = O(n^{1+\varepsilon})$$

Имеются и более эффективные алгоритмы умножения чисел, но они используют уже не цифровые операции.

§ 18. Алгоритм быстрого преобразования Фурье и его приложения.

1° Преобразование Фурье имеет многочисленные применения в математике и кибернетике. Задача дискретного преобразования Фурье возникает в обработке сигналов. Алгоритм дискретного преобразования Фурье используется как подпрограмма в различных арифметических и алгебраических задачах, включая вычисление и интерполяцию полиномов и умножение чисел и полиномов. Дискретное преобразование Фурье можно рассматривать как переход от представления полинома его коэффициентами к представлению его значениями в специально выбранных точках. Этими точками являются корни из единицы, и быстрое преобразование Фурье (БПФ) есть эффективный метод выполнения этого преобразования, а также ему обратного.

В данном разделе дается алгоритм БПФ и приводится верхняя оценка сложности его реализации. В качестве приложения рассматривается проблема умножения двух полиномов n -ой степени, для которой БПФ требует $O(n \log n)$ арифметических операций, в то время как обычный алгоритм требует $O(n^2)$ операций. Шенхаге и Штрассен (1971) применили БПФ для построения алгоритма умножения двух n -битовых чисел, который требует $O(n \log n \log\log n)$ операций.

2° Для дискретного преобразования Фурье будем называть прямым преобразованием переход от представления полинома степени n коэффициентами к представлению его значениями в точках, т.е.

$$< a_0, \ldots, a_n > \quad \rightarrow \quad << x_0, y_0 >, \ldots, < x_s, y_s >>,$$

где $s \geq n+1$, n - степень полинома. Соответственно обратным преобразованием - переход от представления значениями в точках к представлению коэффициентами.

Утверждение 1. Пусть имеется алгоритм со сложностью $O(n \log n)$ для выполнения прямого и обратного преобразования Фурье. Тогда имеется алгоритм той же сложности для умножения двух полиномов n -ой степени.

Доказательство. Пусть требуется перемножить полиномы $p(x)=\sum_{j=0}^{n} a_j x^j$ и $q(x)= \sum_{j=0}^{n} b_j x^j$. Умножение осуществляем следующим образом:

Шаг 1. Осуществляем прямое преобразование Фурье и

вычисляем $p(x)\Big|_{x=x_i}$; $0 \leq i \leq 2n$

$q(x)\Big|_{x=x_i}$, $n = \deg p = \deg q$.

Шаг 2. Вычисляем $w(x_i) = p(x_i)q(x_i)$, $0 \leq i \leq 2n$.

Шаг 3. Осуществляем обратное преобразование Фурье и

по парам $\{< x_i, w(x_i) >\}$, $0 \leq i \leq 2n$ находим полином,

т.е. его коэффициенты, степени $\leq 2n$, который

в данных точках принимает данные значения.

Определим сложность данного алгоритма. По предположению сложность шага 1 есть O($n\ log\ n$), шага 2 - $2n$+1, шага 3 - O($n\ log\ n$) и общее число операций, следовательно, равно O($n\ log\ n$). Утверждение доказано.

Заметим, что обычный алгоритм требует O(n^2) операций.

Задача теперь заключается в том, чтобы выбрать точки $\{x_i\}$, в которых возможно быстрое преобразование Фурье.

Будем предполагать, что операции выполняются в поле комплексных чисел ***C***.

Напомним, что число ω называется примитивным корнем степени k из единицы, если

$$\omega^k = 1 \text{ и } \omega^j \neq 1 \text{ при } j \in [1, k-1].$$

Множество корней степени k из 1 образует группу по умножению и примитивные корни это образующие данной группы.

Утверждение 2. Пусть ω - примитивный корень степени n +1 из 1. Тогда выполнено:

$$\sum_{s=0}^{n} \omega^{s\alpha} = \begin{cases} n+1, \text{ если } \alpha \equiv 0(\mathrm{mod}(n+1)) \\ 0, \text{ если сравнение неверно} \end{cases}.$$

Доказательство. Если $\alpha \equiv 0(\mathrm{mod}(n+1))$,то $\omega^{s\alpha} = 1$ по определению.

Если сравнение неверно, то $\sum_{s=0}^{n} \omega^{s\alpha} = \sum_{s=0}^{n} t^s \Big|_{t=\omega^\alpha} = \dfrac{t^{n+1}-1}{t-1}\Big|_{t=\omega^\alpha} = 0$, т.к. $t = \omega^\alpha \neq 1$ и $t^{n+1} = \omega^{\alpha(n+1)} = 1$. Утверждение доказано.

Утверждение 3. Выберем точки $x_i = \omega^i,\ 0 \le i \le n$, где ω - примитивный корень степени n +1 из 1. Тогда прямое и обратное преобразования полинома степени n требует O($n\ log\ n$) операций.

Доказательство. Используем данные точки $x_0, \ldots, x_n$ для преобразования Фурье. Пусть дан многочлен $p(x)=\sum_{j=0}^{n} a_j x^j$ и выполним прямое преобразование $p(x)=\sum_{j=0}^{n} a_j x^j \Big|_{x=x_j}$ следующим образом:

Считаем, что n +1=2^r. Имеем $p(x) = a_0 + a_1 x + \ldots + a_n x^n =$

$$= (a_0 + a_2 x^2 + \ldots + a_{n-1} x^{n-1}) + (a_1 x + a_3 x^3 + \ldots + a_n x^n) =$$
$$= (a_0 + a_2 x^2 + \ldots + a_{n-1} x^{n-1}) + x(a_1 + a_3 x^2 + \ldots + a_n x^{n-1}).$$

Положим $y = x^2$. Тогда имеем

$$p(x) = (a_0 + a_2 y + \ldots + a_{n-1} y^{\frac{n-1}{2}}) + x(a_1 + a_3 y + \ldots + a_n y^{\frac{n-1}{2}}) = p_1(y) + x p_2(y),$$

где $\deg p_1 = \deg p_2 = \dfrac{n-1}{2} = 2^{r-1} - 1$.

Вычисление $p(x)$ в точках $x_0,\ldots,x_n$ сводится к $p_1(y)$ и $p_2(y)$ в точках $y_0,\ldots,y_n$, где $y_i = x_i^2$, среди которых $\frac{n+1}{2}$ различных.

Данный процесс организуем рекурсивно, решая 2 задачи половинной размерности.

Оценим сложность $T(2^r)$ - число арифметических операций для вычисления многочлена $\sum_{i=0}^{2^r-1} a_i x^i$ в выбранных 2^r точках $\left\{\omega^t \,\middle|\, 0 \le t \le 2^r - 1\right\}$, где ω - примитивный корень степени 2^r из 1. Тогда имеем

$T(2^r) = 2T(2^{r-1}) + 2\cdot 2^r$,

где последний член представляет одно сложение и одно умножение для каждого x^i.

Теперь заметим, что можно уменьшить число умножений в два раза, используя соотношение $x_{i+2^{r-1}} = -x_i$.

Значит можно записать $T(2^r) = 2T(2^{r-1}) + \frac{3}{2}\cdot 2^r$, $T(1) = 0$. Отсюда получаем $T(2^r) = \frac{3}{2}\cdot 2^r \cdot r$. Значит T(*n*)=O(*n log n*).

Найдем обратное преобразование: $\{< x_i, y_i >\} \to \{a_i\}$, где $x_i = \omega^i$, где ω - примитивный корень степени *n* +1 из 1.

Организуем вычисление как умножение вектора на матрицу. Положим

$$V = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^n \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2n} \\ & & \cdot & \cdot & \cdot \\ 1 & \omega^n & \omega^{2n} & \ldots & \omega^{n^2} \end{pmatrix}$$

Тогда $(a_0,\ldots,a_n)V = (y_0,\ldots,y_n)$ и $(a_0,\ldots,a_n) = (y_0,\ldots,y_n)V^{-1}$.

Определим $(n+1\times n+1)$-матрицу $\widetilde{V} = (\widetilde{v}_{ij})$, где $\widetilde{v}_{ij} = \frac{\omega^{-ij}}{n+1}$. Имеем согласно утверждению 2 $\left(V\cdot\widetilde{V}\right)_{ij} = \begin{cases} 1, & \text{если } i = j, \\ 0, & \text{если } i \ne j. \end{cases}$

и поэтому $V^{-1} = \widetilde{V}$.

Следовательно, $(a_0,\ldots,a_n) = (y_0,\ldots,y_n)V^{-1} =$

$$= (y_0,\ldots,y_n)\begin{pmatrix} 1 & 1 & \ldots & 1 \\ 1 & \omega^{-1} & \ldots & \omega^{-n} \\ & \cdot & \cdot & \cdot \\ 1 & \omega^{-n} & \ldots & \omega^{-n^2} \end{pmatrix} \frac{1}{n+1} = \sum_{i=0}^{n} y_i t^i \Big|_{t=\omega^{-j}} \cdot \frac{1}{n+1}, \quad (0 \le j \le n).$$

Если ω - примитивный корень степени $n+1$ из 1, то это верно и для ω^{-1}, поэтому обратное преобразование Фурье эквивалентно прямому преобразованию и, следовательно требует O($n \log n$) арифметических операций.
Если n - не степень двойки, то заменяем ее ближайшей сверху степенью двойки. Ясно, что это сохраняет порядок оценки.

Следствие. Умножение двух многочленов степени n можно выполнить за O($n \log n$) операций в поле комплексных чисел.

Заметим, что имеются алгоритмы БПФ с операциями в конечных полях. Для знакомства с ними следует обратиться к журнальной литературе.

§ 19. Сложность алгоритмов выбора на частично упорядоченном множестве и их оптимальность.

Построение эффективных алгоритмов выбора элементов конечного множества в соответствии с некоторым свойством возможно лишь в тех случаях, когда исходное множество наделено некоторой структурой, а соответствующие алгоритмы опираются на данную структуру. В данном разделе изучается сложность выбора элементов из частично упорядоченного множества с помощью алгоритмов, опирающихся на отношение частичной упорядоченности и выясняется их оптимальность. К задачам такого типа сводятся многие переборные задачи, представляющие практический интерес,

Пусть M - конечное множество, на элементах которого определено некоторое свойство P. Пусть f_P - характеристическая функция тех элементов из M, которые обладают свойством P, т.е. двоичная функция, для которой имеем

$$f_P(a) = \begin{cases} 1, \text{ если } a \text{ обладает P}, \\ 0, \text{ если } a \text{ не обладает P}. \end{cases}$$

Ясно, что задача перечисления элементов M, обладающих свойством P, равносильна задаче выделения двоичной функции f_P или задаче нахождения разбиения $(\mathtt{M}_1, \mathtt{M}_2)$, где $\mathtt{M}_1 = \{a \in \mathtt{M} \mid f_P(a) = 1\}$.

Пусть M(≤) - частично упорядоченное множество, а свойство P согласовано с отношением частичного порядка. Свойство $\mathtt{P}_1$ на множестве M называется наследственным влево, если из $b \le a$, $f_{P_1}(b) = 1$ следует $f_{P_1}(a) = 1$.

Соответственно, свойство $\mathtt{P}_2$ называется наследственным вправо, если из $b \le a$, $f_{P_2}(a) = 1$ следует $f_{P_2}(b) = 1$. Легко видеть, что наследственным влево свойствам P соответствуют монотонные функции f_P, для которых справедливо соотношение:

$$b \le a \Rightarrow f_P(b) \le f_P(a), \quad \forall \ a, b \in \mathtt{M}.$$

Наследственным вправо свойствам P соответствуют инверсно монотонные функции f_P, для которых справедливо соотношение:

$$b \le a \Rightarrow f_P(a) \le f_P(b), \quad \forall \ a, b \in \mathtt{M}.$$

Ясно, что достаточно ограничиться рассмотрением наследственных влево свойств P и соответственно задачей выделения монотонной функции f_P на M , поскольку двойственный случай наследственного вправо свойства сводится переходом от множества M к инверсно изоморфному множеству M'. Нас будет интересовать вопрос алгоритмической сложности выделения монотонной двоичной функции f_P на частично упорядоченном множестве M. Сложность алгоритма определяется положенной в основу вычислительной интерпретацией алгоритма, т.е. элементарным шагом алгоритма и его логической схемой.

Под сложностью алгоритма A понимается число элементарных шагов, необходимое для получения окончательного результата. Пусть f - произвольная

монотонная двоичная функция на M, заданная в виде отображения $f\colon\ \mathtt{M} \to \{0,1\}$. Пусть a={A} - множество алгоритмов, решающих поставленную задачу, т.е. для произвольной монотонной функции на M любой алгоритм A$\in a$ с помощью отображения f определяет соответствующее f разбиение $(\mathtt{M}_1,\mathtt{M}_2)$ множества M. Работа любого алгоритма A$\in a$ протекает следующим образом. Алгоритм A производит выбор некоторого элемента $a \in$M и вычисляет значение $f(a)$.

Если $f(a)$=1, то в список $\mathtt{M}_1$ заносится a и все элементы b, $b \ge a$. Если $f(a)$=0, то в список $\mathtt{M}_2$ заносится a и все элементы b, $b \le a$. Затем по некоторому правилу выбирается другой элемент из M и процесс повторяется. Это продолжается до тех пор, пока разбиение $(\mathtt{M}_1,\mathtt{M}_2)$ не будет определено полностью.

В качестве элементарного шага алгоритма A возьмем вычисление значения функции $f(a)$ для некоторого $a \in$M с помощью отображения $f\colon\ \mathtt{M} \to \{0,1\}$.

Для произвольного $a \in$M определим множества $\mathtt{V}_a = \{b,\ \ b > a\}$ и $\mathtt{N}_a = \{b,\ \ a > b\}$. Тогда за один шаг алгоритма на элементе a значение f определяется либо на всем $\mathtt{V}_a$, либо на всем $\mathtt{N}_a$. Любой паре - алгоритму A и монотонной функции f можно поставить в соответствие число $\omega(A,f)$ - количество обращений к отображению $f\colon\ \mathtt{M} \to \{0,1\}$ в процессе определения разбиения $(\mathtt{M}_1,\mathtt{M}_2)$ для функции f при работе по алгоритму A. Сложность фиксированного алгоритма A оценим функцией

$$\omega(\mathtt{A},\mathtt{M}) = \max_f\ \ \omega(\mathtt{A},f)$$

(максимум по всем монотонным функциям типа $f\colon\ \mathtt{M} \to \{0,1\}$).

Сложность самой задачи выделения произвольной монотонной функции на M будем характеризовать функцией

$$\omega(\mathtt{M}) = \min_{\mathtt{A}}\ \ \max_f\ \ \omega(\mathtt{A},f)$$

(минимум по всем алгоритмам A$\in a$, решающим данную задачу).

Функция ω(M) для частично упорядоченного множества оценивает возможности для сокращения числа опробований в переборных алгоритмах, использующих только структуру частичной упорядоченности на опробуемых элементах.

По аналогии с терминологией, используемой в теории монотонных булевых функций, введем следующие определения.

Пусть F - множество всех монотонных двоичных функций на M. Совокупность элементов $\mathtt{N}_f \subseteq$ M называется определяющей совокупностью для $f\in$F, если по значениям функции f на множестве $\mathtt{N}_f$ однозначно определяются значения f на всем M. Другими словами, определяющая совокупность $\mathtt{N}_f$ характеризуется тем, что для любой функции $g\in$F из того, что $f(a)=g(a)$, $\forall a \in \mathtt{N}_f$ следует $f=g$.

Определяющая совокупность называется тупиковой для $f \in F$, если никакое ее собственное подмножество не является определяющей совокупностью для f.

Заметим, что для определения разбиения (M_1, M_2) множества M для $f \in F$ необходимо и достаточно определить значения f на некоторой ее определяющей совокупности.

Некоторый элемент $a \in M$ называется:
-верхним нулем для f, если $f(a)=0$ и $f(b)=1$, $\forall\, b \in V_a$,
-нижней единицей для f, если $f(a)=1$ и $f(b)=0$, $\forall\, b \in N_a$. Обозначим через Z_f - множество верхних нулей, а через U_f - множество всех нижних единиц для функции f.

Теорема 1. Множество $D_f = U_f \cup Z_f$ является единственной тупиковой определяющей совокупностью для монотонной функции f.

Доказательство. Действительно, по определению множества D_f, по значениям функции f на множестве D_f однозначно определяются значения f на множестве M и, значит, D_f - определяющая совокупность для функции f.

Далее, согласно определению, $a \in D_f$ в том и только в том случае, когда значение $f(a)$ однозначно определяет значение f для всех $b \in M$, таких, что $b \in V_a$ и $b \in N_a$. Поэтому никакое подмножество $D_f' \subset D_f$ не может быть определяющей совокупностью для f.

Пусть теперь R - некоторая определяющая совокупность для функции f и $z \in Z_f$ - некоторый верхний нуль для функции f.

Предположим, что $z \notin R$. Тогда имеем по определению $f(z)=0$. Поскольку R - определяющая совокупность, то по значениям f на множестве R однозначно определяются значения f на всем M, в том числе и в точке z. Но $f(z)=0$ по монотонности может быть выведено только в том случае, когда имеется нулевое значение для элемента, большего z. Значит, существует $z_1 \in R$, $z < z_1$ и $f(z_1)=0$. Но z - верхний нуль для f, значит $f(z_1)=1$, что противоречит допущению $z \notin R$. Следовательно, $Z_f \subset D_f$. Аналогично доказывается, что $U_f \subset D_f$. Значит, любая определяющая совокупность для функции $f \in F$ содержит множество D_f. Теорема доказана.

Обозначим $d_M = \max\limits_f \left|D_f\right|$, где максимум берется по всем $f \in F$.

Теорема 2. Для любого частично упорядоченного множества M имеют место соотношения

$$i \le d_M \le 2i,$$

где i - максимальное число попарно несравнимых элементов M.

Доказательство. Пусть $\{a_1, \ldots, a_i\}$ - множество попарно несравнимых элементов множества M. Определим двоичную функцию f следующим образом:

$$f(a_1) = f(a_2) = \ldots = f(a_i) = 1$$

$$f(a)=1,\ \text{если}\ a\in \mathrm{V}_{a_k}\ \text{для некоторого}\ k\in\overline{1,i}\ ,$$

$$f(a)=0,\ \text{если}\ a\in \mathrm{N}_{a_k}\ \text{для некоторого}\ k\in\overline{1,i}\ .$$

Функция *f* имеет, очевидно, *i* нижних единиц, откуда и следует нижняя оценка.

Для доказательства верхней оценки заметим, что любые две нижние единицы попарно несравнимы. Значит, число элементов M, являющихся нижними единицами, не более, чем *i* - максимальное число попарно несравнимых элементов в M. То же имеет место и для множества верхних нулей функции *f*. Значит $\mathrm{d}_{\mathrm{M}} \le 2i$. Теорема доказана.

Из теоремы 2 следует, что для выделения произвольной монотонной функции на M достаточно знать ее значения не более, чем в 2*i* элементах. Однако для оценивания сложности задачи необходим алгоритм, который бы находил эти элементы.

Теорема 3. Для функции ω(M) справедливы оценки

$$i \le \omega(\mathrm{M}) \le \left[i\ \log_2 \frac{|\mathrm{M}|}{i} \right] + i\ ,$$

где $\left[\mathrm{x}\right]$ - целая часть x.

Доказательство. Необходимо доказать только верхнюю оценку, поскольку из $\omega(\mathrm{M}) \ge \mathrm{d}_{\mathrm{M}}$ и $\mathrm{d}_{\mathrm{M}} \ge i$ следует нижняя оценка.

Опишем конкретный алгоритм A, позволяющий выделять произвольную двоичную функцию *f* на M. Если *i* - максимальное число попарно несравнимых элементов, то по теореме Дилуорса [11, параграф 3] существует разбиение множества M на *i* непересекающихся цепей. Пусть это цепи $\mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_i$, $|\mathrm{C}_p| = l_p$, $p = \overline{1,i}$. Будем считать, что это разбиение задано, т.е. для любых *p*,*r* можно указать элемент с номером *r* в цепи C_p. Пусть $a_1, a_2, \ldots, a_{l_1}$ - цепь C_1 в порядке убывания. Пусть $r = \left]\log_2 l_1\right[$ - ближайшее сверху к $\log_2 l_1$ целое число. На первом шаге алгоритм A определяет значение $f(a_{2^{r-1}})$. Если $f(a_{2^{r-1}})$=0, то $f(a_t)$=0 при $t > 2^{r-1}$ в силу монотонности *f* и на втором шаге алгоритм A применяется к цепи $a_1, \ldots, a_{2^{r-1}-1}$. Если $f(a_{2^{r-1}})$=1, то $f(a_t)$=1 при $t < 2^{r-1}$ и на втором шаге алгоритм A применяется к цепи $a_{2^{r-1}+1}, \ldots, a_{l_1}$ и т.д. Число обращений к отображению $f\colon\ \mathrm{M} \to \{0,1\}$, достаточное для определения значения *f* на цепи C_1 не превосходит $r = \left]\log_2 l_1\right[$.

Проделаем эту процедуру со всеми цепями $\mathrm{C}_2, \ldots, \mathrm{C}_i$ независимо. Тогда имеем

$$\omega(\mathrm{M}) \le \sum_{t=1}^{t=i} \left]\log_2 l_t\right[\le i + \sum_{t=1}^{t=i} \log_2 l_t \le i + \log(l_1 \ldots l_i) \le i + \left[i \log_2 \frac{|\mathrm{M}|}{i} \right],$$

поскольку справедливо неравенство $l_1 \ldots l_i \le \left(\frac{|\mathbb{M}|}{i}\right)^i$, если $l_1 + \ldots + l_i = |\mathbb{M}|$.

Теорема доказана.

Применим теперь полученные результаты к конкретному частично упорядоченному множеству. Пусть $\mathbb{E}_n$ - множество двоичных наборов длины n с обычным отношением частичного порядка:

$$x \le y \Leftrightarrow x_i \le y_i, \quad i \in \overline{1,n}, \quad x = (x_1, \ldots, x_n), \quad y = (y_1, \ldots, y_n).$$

Множество $\mathrm{L} \subset \mathbb{E}_n$ называется цепью, если любая пара элементов из L сравнима.

Теорема 4. Множество $\mathbb{E}_n$ может быть представлено в виде объединения $\binom{n}{\left[\frac{n}{2}\right]}$ попарно непересекающихся цепей, обладающих свойством: число цепей длины n-2p+1 равно $\binom{n}{p} - \binom{n}{p-1}$, $0 \le p \le \left[\frac{n}{2}\right]$, при этом минимальный элемент цепи имеет вес p, максимальный - вес n-p.

Доказательство. Доказательство индукцией по n. При n=1,2 утверждение проверяем непосредственно. Разобьем множество $\mathbb{E}_n$ на два подмножества $\mathbb{E}^1_{n-1}$ и $\mathbb{E}^2_{n-1}$, получаемые фиксацией первой координаты 0 и 1 соответственно. Пусть утверждение верно для $\mathbb{E}_{n-1}$ и множества $\mathbb{E}^1_{n-1}$ и $\mathbb{E}^2_{n-1}$, покрыты $\binom{n-1}{\left[\frac{n-1}{2}\right]}$ цепями с указанными свойствами. Образуем покрытие множества $\mathbb{E}_n$ следующим образом: Пусть $\mathbb{C}_1$ - цепь множества $\mathbb{E}^1_{n-1}$ и y_1 - ее максимальный элемент. Тогда удлиним цепь $\mathbb{C}_1$, добавив к ней максимальный элемент y_2 цепи $\mathbb{C}_2$, изоморфной $\mathbb{C}_1$ в множестве $\mathbb{E}^2_{n-1}$, удаляя при этом из $\mathbb{C}_2$ элемент y_2. Проделаем эту операцию со всеми цепями из $\mathbb{E}^1_{n-1}$. Ясно, что цепи, покрывающие $\mathbb{E}_n$, не будут пересекаться. Число цепей длины n-2p+1 будет равно

$$\left[\binom{n-1}{p} - \binom{n-1}{p-1}\right] + \left[\binom{n-1}{p-1} - \binom{n-1}{p-2}\right].$$

Действительно, цепи длины n-2p+1 получаются из цепей длины n-2p удлинением и из цепей длины n-2p+2 укорочением. Однако, n-2p= n-1-2p+1 и по индукции

число таких цепей равно $\binom{n-1}{p}-\binom{n-1}{p-1}$. Далее, n-2p+2= n-1-2(p-1)+1 и по индукции имеем число таких цепей $\binom{n-1}{p-1}-\binom{n-1}{p-2}$. Отсюда получаем

$$\binom{n-1}{p}-\binom{n-1}{p-1}+\binom{n-1}{p-1}-\binom{n-1}{p-2}=\binom{n}{p}-\binom{n}{p-1}.$$

Тем самым получено требуемое число цепей длины n-2p+1. Общее число цепей равно, очевидно, $\binom{n}{\left[\frac{n}{2}\right]}$.

Теорема доказана.

Следствие. Максимальное число попарно несравнимых элементов в E_n равно $\binom{n}{\left[\frac{n}{2}\right]}$.

Доказательство. Пусть i - максимальное число попарно несравнимых элементов в E_n. Рассмотрим в E_n множество наборов веса $k=\left[\frac{n}{2}\right]$. Таких наборов, очевидно, $\binom{n}{\left[\frac{n}{2}\right]}$. Ясно, что наборы одного веса из E_n несравнимы.

Тогда $i \ge \binom{n}{\left[\frac{n}{2}\right]}$. С другой стороны, если взять покрытие E_n цепями, то несравнимые наборы не могут лежать на одной цепи. Поэтому по доказанному, $i \ge \binom{n}{\left[\frac{n}{2}\right]}$. Следствие доказано.

Теорема 5. Справедливо соотношение

$$\omega(\mathrm{E}_n)=\binom{n}{\left[\frac{n}{2}\right]}+\binom{n}{\left[\frac{n}{2}\right]+1}.$$

Доказательство. Оценку снизу устанавливаем предъявлением конкретной функции

$$f(x_1,\ldots,x_n)=\begin{cases} 0, \text{ если вес } (x_1,\ldots,x_n)\le\left[\dfrac{n}{2}\right] \\ 1, \text{ если вес } (x_1,\ldots,x_n)>\left[\dfrac{n}{2}\right] \end{cases}.$$

Функция $f(x_1,\ldots,x_n)$ имеет $\binom{n}{\left[\frac{n}{2}\right]}$ верхних нулей (это наборы веса $\left[\frac{n}{2}\right]$) и $\binom{n}{\left[\frac{n}{2}\right]+1}$ нижних единиц (это наборы веса $\left[\frac{n}{2}\right]+1$). Следовательно, выполнено

$$\omega(\mathrm{E}_n)\ge\binom{n}{\left[\frac{n}{2}\right]}+\binom{n}{\left[\frac{n}{2}\right]+1}.$$

Оценку сверху получим предъявлением конкретного алгоритма, выделяющего произвольную монотонную функцию.

1.Определяются значения *f* на цепях длины 1 (если *n* четно) или цепях длины 2 (если *n* нечетно).

2.Если значения *f* определены на цепях длины *n*-2*p*-1, то рассматривается цепь C длины *n*-2*p*+1: $\alpha_1,\alpha_2,\ldots,\alpha_{n-2p+1}$. Рассматриваем элементы $\alpha'_2,\ldots,\alpha'_{n-2p}$, такие, что $\alpha_1<\alpha'_2<\alpha_3,\ \alpha_2<\alpha'_3<\alpha_4,\ldots,\ \alpha_{n-2p-1}<\alpha'_{n-2p}<\alpha_{n-2p+1}$. Эти элементы лежат на цепях длины *n*-2*p*-1 и поэтому там значения *f* определены. По монотонности значения *f* определены на всей цепи C кроме, быть может, двух точек. Если $f(\alpha'_q)=1$ для $q=2,\ldots,n-2p$, то требуется найти $f(\alpha_1)$ и $f(\alpha_2)$. Если $f(\alpha'_q)=0$ для $q=2,\ldots,n-2p$, то требуется найти $f(\alpha_{n-2p})$ и $f(\alpha_{n-2p+1})$. Если для некоторого *i* имеем $f(\alpha'_i)=0,\ f(\alpha'_{i+1})=1$, то требуется найти $f(\alpha_i),\ f(\alpha_{i+1})$. Тем самым алгоритм А определен.

Число элементарных операций (обращений к функции *f*) удовлетворяет неравенству $\omega(\mathrm{E}_n)\le c(1)+2(c(2)+\ldots+c(n+1))$,

где $c(i)$ - число цепей длины i, $i\in\overline{1,n+1}$. Тогда имеем при $n=2k$

$$\omega(\mathrm{E}_n)\le\binom{n}{k}-\binom{n}{k-1}+2\left[\binom{n}{k-1}-\binom{n}{k-2}+\ldots\right]=\binom{n}{k}+\binom{n}{k-1}.$$

При $n=2k+1$ имеем

$$\omega(\mathrm{E}_n)\le 2\left[\binom{n}{k}-\binom{n}{k-1}+\binom{n}{k-1}+\ldots\right]=\binom{n}{k}+\binom{n}{k}=\binom{n}{k}+\binom{n}{k+1}.$$

Тем самым теорема доказана.

Данный результат имеет различные приложения в теоретической кибернетике. Укажем два из них.

1. Для булевой функции $f(x_1,\ldots,x_n)$ переменные $x_1,\ldots,x_t$ называются существенными (в совокупности), если

$\sum_{\alpha_1,\ldots,\alpha_t} f(\alpha_1,\ldots,\alpha_t,x_{t+1},\ldots,x_n) \neq 0$ (сумма по модулю 2).

Пример:Булева функция $f(x_1,x_2,x_3,x_4) = x_1x_2x_3 + x_1x_4$ имеет области существенных переменных $\{x_1,x_2,x_3\}$, $\{x_1,x_2\}$, а также области, в них содержащиеся.

Доказано, что задача нахождения всех областей существенных переменных произвольной булевой функции *n* переменных равносильна задаче выделения произвольной монотонной функции.

2. Для частичной булевой функции $F(x_1,\ldots,x_n)$ переменные $x_1,\ldots,x_t$ называются существенными, если существует частичная булева функция $f(x_1,\ldots,x_t)$, такая, что $f(x_1,\ldots,x_t)=F(x_1,\ldots,x_n)$ в области определения функции F.

Пример:Частичная булева функция $F(x_1,x_2,x_3,x_4,x_5,x_6)$, заданная таблицей

x_1	x_2	x_3	x_4	x_5	x_6	F
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	1	0	0	1
1	1	1	1	1	1	1

имеет области существенных переменных $\{x_1,x_2,x_3\}$, $\{x_4,x_5,x_6\}$, а также области, содержащие по крайней мере одну из них.

Доказано, что задача нахождения всех областей существенных переменных произвольной частичной булевой функции *n* переменных равносильна задаче выделения произвольной монотонной функции.

Рассмотрим теперь задачу поиска максимального верхнего нуля монотонной булевой функции. Верхний нуль α булевой функции *f* называют максимальным верхним нулем, если для любого верхнего нуля β функции *f* выполнено $\|\beta\| \le \|\alpha\|$ ($\|x\|$ - означает число единиц в наборе *x*).

Аналогично предыдущему определяем функцию сложности

$$\mu(E_n) = \min_{B\in B} \max_f \mu(B,f),$$

где *B* - класс алгоритмов, решающих данную задачу,
$\mu(B,f)$ - число обращений к функции *f*, достаточное для нахождения максимального верхнего нуля функции *f* при помощи алгоритма $B\in B$.
Имеет место

Теорема 6.[1] Справедливо равенство

[1] АН СССР, 1975, т.224, №3, Катериночкина Н.Н.

$$\mu(\mathrm{E}_n) = \binom{n}{\left\lceil \frac{n}{2} \right\rceil} + 1 .$$

В данном разделе мы ограничились рассмотрением основных результатов, связанных с доказательством оптимальности алгоритмов. Для знакомства с развитием данных результатов следует обратиться к журнальной литературе.

§ 20.Оптимальность жадного алгоритма

Предметом данного раздела является выяснение условий оптимальности определенного класса алгоритмов. Предварительно рассмотрим два примера.

Пусть дана матрица А с действительными элементами, где

$$A = \begin{pmatrix} 10 & 8 & 4 \\ 6 & 7 & 6 \\ 5 & 6 & 4 \end{pmatrix}$$

и сформулируем следующую оптимизационную задачу.

α)Найти такое подмножество S элементов матрицы, для которого выполнены условия:

1) каждый столбец содержит не более одного элемента.

2) сумма выбранных элементов максимальна.

Будем решать поставленную задачу с помощью, так называемого, жадного алгоритма, т.е. алгоритма, который на каждом шаге выбирает наибольший элемент из возможных.

Очевидно, что данный алгоритм даст решение задачи α) $S=\{a_{11}, a_{12}, a_{23}\}=\{10,\ 8,\ 6\}$, причем 10+8+6=24 и данное решение действительно наибольшее.

Рассмотрим другую оптимизационную задачу относительно матрицы A.

β)Найти такое подмножество S элементов матрицы A, для которого выполнены условия:

1) каждый столбец и каждая строка матрицы содержит не более одного элемента множества S.

2) сумма выбранных элементов максимальна.

Применяя жадный алгоритм к матрице A получим решение задачи β) $S=\{a_{11}, a_{22}, a_{33}\}=\{10,\ 7,\ 4\}$, причем 10+7+4=21. Но это не максимальное решение, т.к. решение $S'=\{a_{11}, a_{23}, a_{32}\}=\{10,\ 6,\ 6\}$ дает 10+6+6=22.

В связи с приведенными примерами возникает вопрос: когда жадные алгоритмы дают оптимальные решения? В данном разделе дается ответ на этот важный в практическом отношении вопрос в терминах матроида. Понятие матроида возникло в ЗО-х годах нашего столетия, и соответствующая теория, лежащая на стыке алгебры и геометрии, получила глубокое развитие, однако рассматривалась как весьма абстрактная и далекая от практического применения.

Нам понадобятся предварительные определения.

Системой подмножеств $S=(E,J)$ будем называть конечное множество E вместе с семейством J подмножеств множества E, замкнутое относительно включения (т.е. если $A\in J$, $A'\subseteq A$, то $A'\in J$). Элементы семейства J называются независимыми.

Комбинаторная задача оптимизации для системы подмножеств $S=(E,J)$ состоит в следующем. Для каждого $e\in E$ задан его вес $w(e)\geq 0$ - действительное число. Требуется найти независимое множество $A\in J$, имеющее наибольший вес

$$W(A)=\sum_{e\in A} w(e)$$

Ясно, что рассмотренные выше задачи α) и β) принадлежат данному классу оптимизационных задач. Здесь E множество позиций матрицы (i, j), w ставит в соответствие каждой позиции (i, j) число $a_{i,j}$. При этом для задачи α) $A \in J \Leftrightarrow$ каждый столбец содержит не более одного элемента из A; для задачи β) $A \in J \Leftrightarrow$ каждый столбец и каждая строка содержит не более одного элемента A.

Сформулируем теперь для системы подмножеств $S= =(E,J)$ жадный алгоритм решения оптимизационной задачи.

1.Положить $I:=\varnothing$.

2.Найти $e \in E$ - элемент с наибольшим весом.

3.Положить $E':=E\setminus\{e\}$. Если $I \cup \{e\} \in J$, положить $I':=I \cup \{e\}$.

4.Перейти к шагу 2.

5.Если $E'=\varnothing$, стоп.

Определение. Пусть $M=(E,J)$ - система подмножеств. M называется матроидом, если жадный алгоритм корректно решает любую индивидуальную комбинаторную задачу оптимизации для системы M.

Аксиомы матроидов отражают характерные свойства независимых множеств векторов линейных пространств.

Существует много эквивалентных определений матроида, но мы ограничимся следующим.

Утверждение 1. Пусть $M=(E,J)$ - система подмножеств. Тогда следующие утверждения эквивалентны:

1) M - матроид.

2) Если I_p, $I_{p+1} \in J$, где $|I_p| = p$, $|I_{p+1}| = p$, то существует такой элемент $e \in I_{p+1} \setminus I_p$, что $I_p \cup \{e\} \in J$.

3) Если A - подмножество множества E и I,I' - максимальные по включению подмножества A, то $|I| = |I'|$.

Доказательство. 1)⇒ 2). Пусть M - матроид, но 2) не выполняется. Это значит, что существуют такие два независимые подножества I_p, I_{p+1}, что $|I_p| = p$, $|I_{p+1}| = p+1$ и для всех $e \in I_{p+1} \setminus I_p$ подмножество $I_p \cup \{e\}$ не является независимым. Рассмотрим следующую функцию весов w на E:

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

Имеем $w(I_{p+1}) \geq (p+1)^2 > p(p+2) = w(I_p)$. Жадный алгоритм в этом случае начнет с выбора всех элементов множества I_p, поскольку эти элементы имеют максимальный вес. После этого жадный алгоритм не сможет увеличить общий вес, т.к. для всех остальных элементов е либо $I_p \cup \{e\} \in J$ (если $e \in I_{p+1}$),

либо $w(e)=0$. Следовательно, жадный алгоритм дает неоптимальное решение I_p, и поэтому M не является матроидом.

2)$\Rightarrow$ 3). Пусть выполнено 2) и рассмотрим I,I' - два максимальных по включению независимых подмножества множества $A \subseteq E$. Допустим, что $|I| < |I'|$. Исключим из I' $|I'| - |I| - 1$ элементов и получим множество I'', где $|I''| = |I| + 1$. Поскольку J замкнуто относительно включения, то $I'' \in J$. По свойству 2 существует элемент $e \in I'' \setminus I$ такой, что $I \cup \{e\} \in J$. Следовательно, I не является максимальным по включению независимым подмножеством множества A.

3)$\Rightarrow$ 1). Пусть выполнено 3) и пусть для некоторого множества весов $w(e)$, $e \in E$ жадный алгоритм не дает оптимального решения, т.е. приводит к независимому множеству $I=\{e_1, e_2, \ldots, e_i\}$ тогда как существует множество $I_1 = \{e_1^1, e_2^1, \ldots, e_j^1\} \in J$ такое, что $w(I_1) > w(I)$. Пусть элементы множеств I и I_1 упорядочены так, что $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_i)$ и $w(e_1^1) \geq w(e_2^1) \geq \ldots \geq w(e_j^1)$.

Можно считать, что I_1 - максимальное по включению независимое множество в E. По построению I - максимально по включению, поэтому из свойства 3) (при $E=A$) следует, что $i=j$. Покажем, что выполнено $w(e_t) \geq w(e_t^1),\ \forall\, t \in \overline{1,i}$, что будет противоречить допущению, что $w(I_1) > w(I)$. Доказываем это индукцией по t. При t=1 это следует из жадности алгоритма. Пусть теперь $w(e_m) < w(e_m^1)$ для некоторого m>1 и $w(e_s) \geq w(e_s^1)$ для $s \in \overline{1, m-1}$. Рассмотрим множество $A = \left\{ e \in E \middle| w(e) \geq w(e_m^1) \right\}$. Множество $\{e_1, \ldots, e_{m-1}\}$ является максимальным по включению независимым множеством в A, т.к. если $\{e_1, \ldots, e_{m-1}, e\} \in J$ и $w(e) \geq w(e_m^1) > w(e_m)$, то жадный алгоритм выбрал бы e вместо e_m в качестве следующего элемента I. Это противоречит 3), т.к. $\{e_1^1, \ldots, e_m^1\}$ - другое независимое множество в A большей мощности. Противоречие показывает законность индукции. Утверждение доказано.

Сделаем теперь несколько замечаний.

1.Оптимальное множество, являющееся решением оптимизационной задачи, зависит только от упорядочения весов.

2.В матроиде нельзя выбрать независимое множество, состоящее из меньшего числа больших по величине элементов.

3.При обращении упорядочения элементов жадный алгоритм находит минимальное по весу множество.

4.Жадные алгоритмы эффективны. Они имеют слож-ность полиномиально зависящую от размера задачи.

Пусть $M=(E,J)$ - матроид и $A \subseteq E$. Рангом $r(A)$ множества A в M называется мощность максимальных по включению независимых подмножеств множества A.

Согласно свойству 3 утверждения 1 все такие подмножества в A имеют одну и ту же мощность и, значит, ранг определяется корректно. Ранг $r(E)$ называется рангом матроида. Максимальные по включению независимые подмножества для E называются базисами матроида.

Из утверждения 1 следует, что каждые два базиса матроида имеют одинаковую мощность.

Отметим следующие свойства рангов.

Утверждение 2. Для произвольных $A,B \subseteq E$ и $e,f \in E$ справедливо:

1) $0 \le r(A) \le |A|$;

2) если $A \subseteq B$, то $r(A) \le r(B)$;

3) $r(A \cup B) + r(A \cap B) \le r(A) + r(B)$;

4) $r(A) \le r(A+e) \le r(A)+1$, где $A+e$ означает $A \cup \{e\}$;

5) если $r(A+e) = r(A+f) = r(A)$, то $r(A+e+f) = r(A)$.

Доказательство. Свойства 1) и 2) очевидны. Докажем 3).

Пусть $\{e_1,\dots,e_p\}$ - максимальное независимое множество в $A \cap B$. Расширим его до максимального независимого множества $\{e_1,\dots,e_p,f_1,\dots,f_q\}$ в A и до максимального независимого множества $\{e_1,\dots,e_p,f_1,\dots,f_q,g_1,\dots,g_t\}$ в $A \cup B$. Имеем $p = r(A \cap B)$, $p+q = r(A)$, $p+t \le p+q+t = r(A \cup B)$, следовательно $r(A \cup B) + r(A \cap B) = p+q+t+p \le r(A) + r(B)$.

Свойство 4) очевидно, свойство 5) следует из свойства 3):

$r(A) \le r(A+e+f) = r((A+e)+f) \le$

$\le r(A+e) + r(A+f) - r((A+e) \cap (A+f)) = r(A) + r(A) - r(A) = r(A)$.

Утверждение доказано.

Множества семейства J называются независимыми, а подмножества D множества E, не входящие в J, называются зависимыми. Минимальное по включению зависимое подмножество C в E называется циклом. Оболочкой множества A называется максимальное по включению множество S, содержащее A и удовлетворяющее условию $r(S)=r(A)$.(Обозначение:$sp(A)$).

Приведем два полезных свойства матроидов.

Утверждение 3. Пусть $I \in J$ и $e \in E$. Тогда либо $I+e \in J$, либо $I+e$ содержит единственный цикл.

Доказательство. Допустим, что $I+e \notin J$. Пусть $C=\{c \mid I+e-c \in J\}$. Покажем, что C - цикл. Действительно, это множество зависимо, т.к. в противном случае его можно увеличить до базиса в $I+e$, поскольку $C \subseteq I+e$ и который должен иметь мощность $|I|$ и, следовательно, иметь вид $I+e-d$, что приводит к противоречию, поскольку это означает, что $d \in C$. Далее, множество C - минимально, т.к. удаляя любой его элемент, скажем c, получаем подмножество $C-c$, которое содержится в независимом подмножестве $I+e-c$. Покажем теперь единственность. Пусть D - другой цикл в $I+e$ и пусть $c \in C \setminus D$. Тогда D является подмножеством $I+e-c$ и, значит, независимо, что противоречит предположению. Утверждение доказано.

Утверждение 4. Любое подмножество $A \subseteq E$ имеет единственную оболочку.

Доказательство. Если S - оболочка подмножества A и $e \in S$, то $r(A+e)=r(A)$. В противном случае, если $r(A+e)>r(A)$, то $r(S) \geq r(A+e)>r(A)$ и получили бы противоречие. Значит $S \subseteq sp(A)$.

Покажем, что $r(sp(A))=r(A)$. Легко видеть, что общий базис двух множеств является базисом их объединения. Поэтому базис множества A является базисом $sp(A)$, т.к. он является базисом в $A+e$ для каждого $e \in sp(A)$. Утверждение доказано.

Следствие 1. $sp(A)$ есть объединение A и всех циклов, все элементы которых, кроме одного, содержатся в A.

Следствие 2. Если $I \in J$, $I+e \notin J$ и c принадлежит циклу в $I+e$, то $sp(I)=sp(I+e-c)$.

Рассмотрим теперь примеры матроидов.

1.Для конечного множества E пара $(E,B(E))$, $B(E)$- булеан E, удовлетворяет условиям утверждения 1 и является матроидом. Данный матроид называется свободным матроидом.

2.Для конечного множества E рассмотрим разбиение $E = E_1+\ldots+E_k$. Будем называть $I \subseteq E$ независимым, если никакие два элемента из I не лежат в одном блоке разбиения, т.е. $\left| I \cap E_j \right| \leq 1$, $j \in \overline{1,k}$. Пусть J - семейство независимых множеств. Если $A, B \in J$, причем $\left| B \right| = \left| A \right| + 1$, то существует $i \in \overline{1,k}$ такой, что $E_i \cap A = \varnothing$ и $E_i \cap B \neq \varnothing$. Возьмем $e \in E_i \cap B$ и образуем $A \cup \{e\}$. Ясно, что $A \cup \{e\} \in J$. Согласно утверждения 1 пара (E,J) образует матроид, называемый матроидом разбиения.

Функция ранга определяется так:

Пусть $i(A) = \{ j \leq k \mid E_j \cap A \neq \varnothing \}$. Легко проверить, что $r(A) = \left| i(A) \right|$ есть требуемая функция ранга, т.к. для данного A можно построить максимальное по включению независимое подмножество в A, выбирая по одному элементу множества A из каждого E_j, с которым A пересекается. Например, пусть

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$.

$П = \{ \{e_1\}, \{e_2, e_3\}, \{e_4, e_5\}, \{e_6, e_7, e_8\} \}$. Тогда ранг множества $A = \{e_1, e_2, e_3, e_6, e_7\}$ равен 3. Оболочка множества A определяется формулой $sp(A) = \bigcup_{j \in i(A)} E_j$.

3. Пусть $A = (a_{ij})$- матрица размера m×n над некоторым полем. Пусть E - множество столбцов матрицы A,

$I \in J$ в том и только в том случае, когда множество столбцов I линейно независимо. Ясно, что пара (E,J) образует матроид, называемый матроидом матрицы А. Функция ранга есть обычный ранг системы векторов. Матроид называется матричным, если он изоморфен матроиду некоторой матрицы. (Два матроида

(*E,J*) и (*E',J'*) называются изоморфными, если существует биекция $f: E \to E'$ с условием $A \in J \Leftrightarrow f(A) \in J'$).

4. Пусть *G* =(*V,E*)- неориентированный граф. Определим *M*(*G*)=(*E,J*), где *J*={$A \subseteq E$ и граф (*V,A*) не имеет циклов. Нетрудно показать, что *M*(*G*) является матроидом, называемым матроидом графа G. Матроид называется графовым, если он изоморфен матроиду некоторого графа.

Изучение графовых матроидов может быть сведено к матричным матроидам. Можно показать, что каждый графовый матроид можно трактовать как матричный матроид, соответствующий матрице инцидентности графа, рассматриваемой как матрица над $F_2 = \{ 0 , 1 \}$.

5. Пусть $A = (A_1, \ldots, A_n)$ - семейство подмножеств конечного множества *E*. Напомним, что множество $S \subseteq E$ является частичной трансверсалью семейства *A*, если существует инъективное отображение $f: S \to \{1,2,\ldots,n\}$, такое, что $e \in A_{f(e)}$ для каждого $e \in S$. Пусть *J* - семейство всех частичных трансверсалей множества *E*. Можно доказать (этот факт есть теорема Эдмонса-Фалкерсона), что пара (*E,J*) есть матроид, называемый трансверсальным матроидом.

В данном разделе лишь кратко затронута теория матроидов. На русском языке с данной теорией можно ознакомиться в работе [13], где рассмотрены также и другие алгоритмические вопросы, касающиеся матроидов.

Литература

1. Агафонов В.Н. Сложность алгоритмов и вычислений: спецкурс для студентов НГУ, ч.2, Новосибирск, 1975.
2. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М., 1979.
3. Глухов М.М. Математическая логика. М., в/ч 33965, 1982.
4. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М., 1982.
5. Катленд Н. Вычислимость. Введение в теорию рекурсивных функций. М., 1983.
6. Кузнецов О.П., Адельсон-Вельский Г.М. Дискретная математика для инженера. М., 1988.
7. Мальцев А.И. Алгоритмы и рекурсивные функции. М., 1965.
8. Марков А.А., Нагорный Н.М. Теория алгоритмов. М., 1984.
9. Мятисевич Ю.В. Диофантовы множества. УМН., т.27, вып. 5(167), с.185-222, 1972.
10. Носов В.А. Специальные главы дискретной математики. М., в/ч 33965, 1990.
11. Носов В.А. Основы комбинаторной теории для инженеров. М., в/ч 33965, 1990.
12. Носов В.А., Сачков В.Н., Тараканов В.Е. Комбинаторный анализ (Неотрицательные матрицы, алгоритмические проблемы) Итоги науки и техники. ВИНИТИ. Сер. Теория вероятн., Мат.статист. Теорет. киберн., 1983, 21, 120-178.
13. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. М., 1985.
14. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. М., 1972.
15. Трахтенброт Б.А. Сложность алгоритмов и вычислений: спецкурс для студентов НГУ, Новосибирск, 1967.
16. Трахтенброт Б.А. Алгоритмы и вычислительные автоматы. М., 1974.
17. Успенский В.А., Семенов А.Л. Теория алгоритмов: основные открытия и приложения. М., 1987.
18. Шоломов Л.А. Основы теории дискретных логических и вычислительных устройств. М., 1980.
19. Мендельсон Э. Введение в математическую логику. М., 1971.

Задачники.

1. Глухов М.М., Шапошников В.А. Задачи и упражнения по математической логике. М., в/ч 33965, 1984.
2. Лавров И.А., Максимова Л.А. Задачи по теории множеств, математической логике и теории алгоритмов. М., 1975.
3. Гаврилов Г.П., Сапоженко А.А. Сборник задач по дискретной математике. М., 1977.