Курс «Вычислительные алгоритмы теории автоматического управления».

Лекция 1. Типовые вычислительные задачи проектирования САУ. Понятие сложности вычислительных алгоритмов.

Цели и задачи дисциплины. Методы принятия технических решений на этапе выбора облика системы. Типовые задачи анализа и моделирования. Основные структуры данных. Понятие о вычислительном процессе как об алгоритме. Типы алгоритмических моделей. Функции сложности алгоритма.

Цель и задачи изучения дисциплины.

Основной целью курса является изложение основных методологических и практических вопросов теории вычислений, возникающих при проектировании систем автоматического управления.

Задачами курса является обучение студентов:

1. Основным принципам рационального выбора технических решений, возникающих при формировании облика системы управления, на основе анализа вычислительных затрат, затраченных на проектирование;
2. Понятиям алгоритма, его вычислительной реализуемости и сложности, принципам представления математических моделей систем в виде вычислительных структур данных;
3. Основным вычислительным процедурам анализа линейных и нелинейных систем управления на базе математических моделей теории автоматического управления.
4. Вычислительным аспектам обработки экспериментальных данных при проектировании, процедурам структурного и параметрического анализа математических моделей систем.
5. Методам и вычислительным алгоритмам оптимизации технических решений при проектировании систем.
6. Вычислительным процедурам параметрического синтеза регуляторов, в том числе при неполной информации об условиях функционирования систем управления.

Задание на проектирование систем автоматического управления. Основные стадии проектирования.

При получении задачи на проектировании системы управления есть обычно три основных варианта Задания на проектирование, отличающихся объемом и степенью определенности информации об объекте и целях проектирования.

При первом варианте имеется схема данных, определяющая структуру будущей системы, которая включает следующее: математическую модель объекта (детерминированную или стохастическую) , определяющую свойства его динамические свойства, перечень измеряемых показателей объекта управления, вид и параметры возмущающих сигналов. В качестве критериев проектирования обычно приводится перечень технических показателей системы, включающий такие показатели, как качество переходных процессов, точность работы системы. При этом часто задаются дополнительные ограничения на весо - габаритные и энерго - емкостные показатели проектируемой системы, также другие параметры. Такое задание определяется Заказчиком, который провел уже достаточно большие предпроектные решения по исследованию как динамических свойств объекта, которым надлежит управлять, так и условий функционирования объекта при различных возмущениях. При таком Задании число вариантов технического решения проектируемой системы обычно невелико, структуру системы можно выбрать на основе небольшого количества критериев, а задача оптимизации сводится к выбору оптимального выбора некоторых технических параметров.

При втором варианте Задания объем информации, предоставляемой Заказчиком, носит менее структурированный характер. Поведение Объекта обычно характеризуется экспериментальными данными в виде временных рядов, отражающими вход - выходные соотношения при различных режимах функционирования. Условия функционирования объекта и возмущающие воздействия задаются часто априорными ограничениями в виде допустимых областей их изменения, а характеристики объекта с изменением режимов функционирования могут значительно изменяться. Перечень измеряемых показателей должен определяться проектировщиком в процессе построения модели системы на основе априорной информации и

заданных ограничениях, либо формироваться апостериорно, в зависимости от уже измеренных значений выбранных выходных сигналов. Критерии проектирования тоже обычно носят неоднозначный характер и зависят от режимов функционирования системы. Нахождение однозначного оптимального технического решения в таких условиях Заказчик обычно не требует, а задает допустимую область их значений.

Третий вариант Задания обычно характеризуется большей неполнотой информации, как об объекте, так и условиях его функционирования. Объект обычно характеризуется областями изменения его параметров для различных режимов функционирования. В некоторых случаях имеется экспертная информация о возможных законах изменения параметров объекта, а также рекомендуемые экспертные заключения о законах управления объектом для различных режимов функционирования. В ряде случаев имеется возможность осуществления экспериментов, позволяющих получить предварительные оценки работоспособности различных вариантов системы управления. При этом наблюдаемая (измеряемая) информация обычно характеризуется наличием значительного уровня шума. При таких условиях технические критерии проектирования часто дополняются критериями эффективности выбора математической модели (обучения).

Процесс проектирования технических систем также принято разделять на три основных стадии:

- стадия НИР или функционального проектирования, которая связана с анализом характеристик объекта, с определением принципов функционирования системы, разработкой ее функциональной и принципиальных схем в соответствии требованиями технического задания, разработкой вычислительных алгоритмов анализа, синтеза и моделирования системы;
- стадия ОКР, которая охватывает вопросы выбора элементной базы, выбор пространственного расположения элементов и монтажной схемы их расположения, разработку программного обеспечения, проверку успешности функционирования опытных образцов, разработку технической документации на систему и прочее;
- стадия технологического проектирования, которая связана с решением комплекса задач, связанных с построением технологических процессов для изготовления и тиражирования проектируемого изделия.

<u>В данном курсе будут рассматриваться только задачи, связанные с разработкой вычислительных алгоритмов в рамках этапа НИР по созданию систем автоматического управления.</u>

Процесс разработки вычислительных алгоритмов и программного обеспечения для систем управления регламентируется целым рядом основных нормативных документов, который может дополняться в ряде прикладных областей другими сопутствующими нормативными материалами.

Определяющими для разработки алгоритмов и программного обеспечения автоматизированных систем являются ГОСТы серии 34, которые частично заменили ранее действующие ГОСТы серии 24. Для организации процесса разработки программного обеспечения и сопутствующей документации в настоящее время также широко применяется ГОСТ ISO 9001-2008. Продолжает широко использоваться ГОСТ 19.402-78, посвященный правилам оформления программного обеспечения и документации к ним. Порядок выполнения работ по НИР регламентирует ГОСТ 15.101-98. Оформление результатов, полученных в процессе НИР, регламентируется ГОСТ 7.32-2001. Для встроенного программного обеспечения, что актуально для цифровых систем управления, действует ГОСТ Р 5194-2002, в котором детально описаны не только требования к структуре программного обеспечения и правилам оформления кода и соответствующей документации, но и методика тестирования программ.

ГОСТ 34.003-90 содержит информацию об основных терминах и определениях предметной области проектирования автоматизированных систем. В частности в пп.1.1-1.3 этого документа содержатся определения автоматизированной системы и ее функциях. Так, в соответствии с этим документом: «Автоматизированная система- это система состоящая из персонала и комплекса средств автоматизации его деятельности, реализующая информационную технологию выполнения установленных функций». Под это определение попадает и пилотируемый ЛА с системой управления, а также БПЛА, управляемый оператором. Полностью автоматизированный БПЛА с собственной интеллектуальной системой, принимающий решения самостоятельно, под это

определение не подпадает. То есть, очевидно, что следует ожидать появления для таких систем своих нормативных документов на условия проектирования.

Другим важным положением ГОСТ 34.003-90 является определение алгоритма функционирования автоматизированной системы – «Это алгоритм, задающий условия и последовательность действий компонентов автоматизированной системы при выполнении ею своих функций». То есть, в описании системы, в соответствии с данным документом, должны быть четко прописаны все функции, выполняемые автоматизированной системой, а также указан полный состав аппаратно-программных средств, позволяющих их выполнить. Очевидно, что при техническом задании 3-го типа, указанном выше, все трудности выполнения этого требования ГОСТ падают на Исполнителя научно-исследовательской работы.

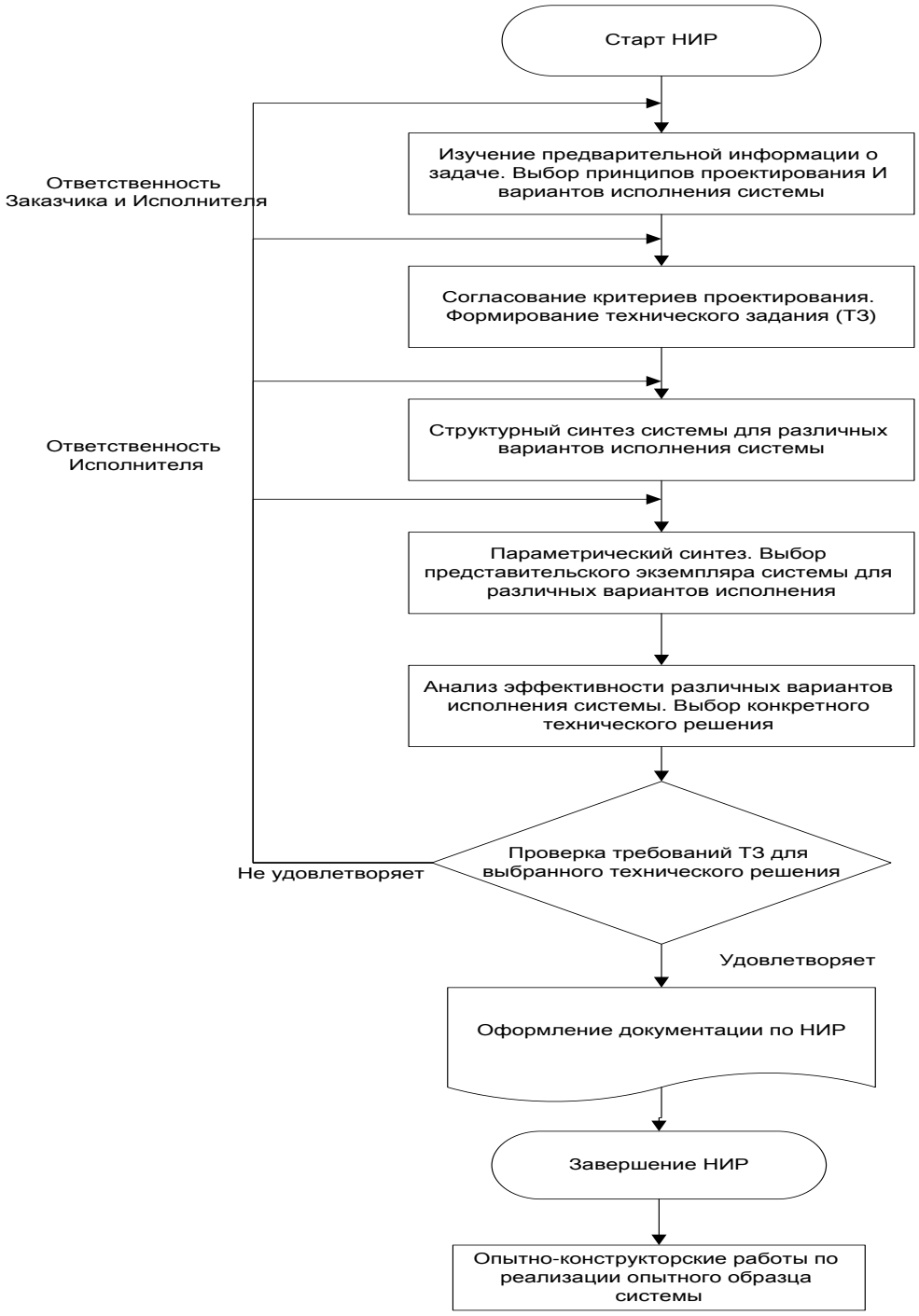
Важным документом является ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы». Данный ГОСТ четко определяет исходные данные, необходимые для начала работ, которые должен предоставить Заказчик, а также основные этапы работ. В соответствии с данным документом техническое задание должно содержать следующие разделы:
- общие сведения;
- назначение и цели создания (развития) системы;
- характеристики объектов автоматизации;
- требования к системе;
- состав и содержание работ по созданию системы;
- порядок контроля и приемки системы;
- требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие;
- требования к документированию;
- источники разработки (ссылочная документация).

Более современный ГОСТ ISO 9001-2008 «Системы менеджмента качества» определяет требования к качеству проектных работ при разработке автоматизированных систем и их документированию. В настоящее время этот ГОСТ широко применяется предприятиями при заключении договоров на разработку систем управления. В соответствии с ГОСТ, качество проводимых работ должен оценивать независимый экспертный орган, проверяющий соответствие всех этапов НИР требованиям данного нормативного документа.

Методы принятия технических решений на этапе выбора облика системы.

При выборе решения обычно осуществляют анализ различных проектных ситуаций, которые должны учитывать такие элементы задачи, как возможные входные и возмущающие сигналы, варианты различных математических моделей объекта, сложности технической реализации, включая выбор основных узлов и составляющих элементов, и прочее. Ситуации принятия решения могут характеризоваться, как единственным критерием оптимальности, так и многими критериями. При многокритериальном решении оценить и сравнить различные варианты в единых универсальных единицах нельзя.

Например, пусть рассматриваются два варианта системы управления. Оба варианта обеспечивают такие необходимые требования к системе, как требования к переходным процессам системы и требуемую точность в установившемся режиме. Но первый вариант системы требует измерения нескольких фазовых координат объекта, а второй вариант на основе использования наблюдателя позволяет обходиться одной фазовой координатой. Однако по критерию помехозащищенности и устойчивости к отклонению параметров модели объекта от расчетных значений первый вариант является более предпочтительным. Для выбора наилучшего, с точки зрения Заказчика, технического решения необходимо предварительно найти для каждого варианта свой оптимальный экземпляр системы с учетом критериев, приведенных в ТЗ, и сравнить их с точки предпочтительности. Процесс проектирования можно изобразить с помощью следующей условной схемы.



Задачи выбора вариантов исполнения системы и структурного синтеза являются наиболее сложными и наименее определенными в схеме проектирования. Решение этих задач осуществляется в условиях неполноты информации, так как предполагают выбор решения только на основе априорных, далеко неполных, данных о поведении объекта. Поэтому при решении этих задач широко используют экспертные оценки специалистов и имеющийся опыт эксплуатации аналогичных систем. Естественно желание Заказчика, чтобы в рамках проектных работ было рассмотрено как можно больше вариантов построения системы с целью получения наиболее подходящего технического решения. Но рамки работ обычно имеют временные и экономические ограничения. Рассмотрим пример формализации задачи оптимизации выполнения проектных работ.

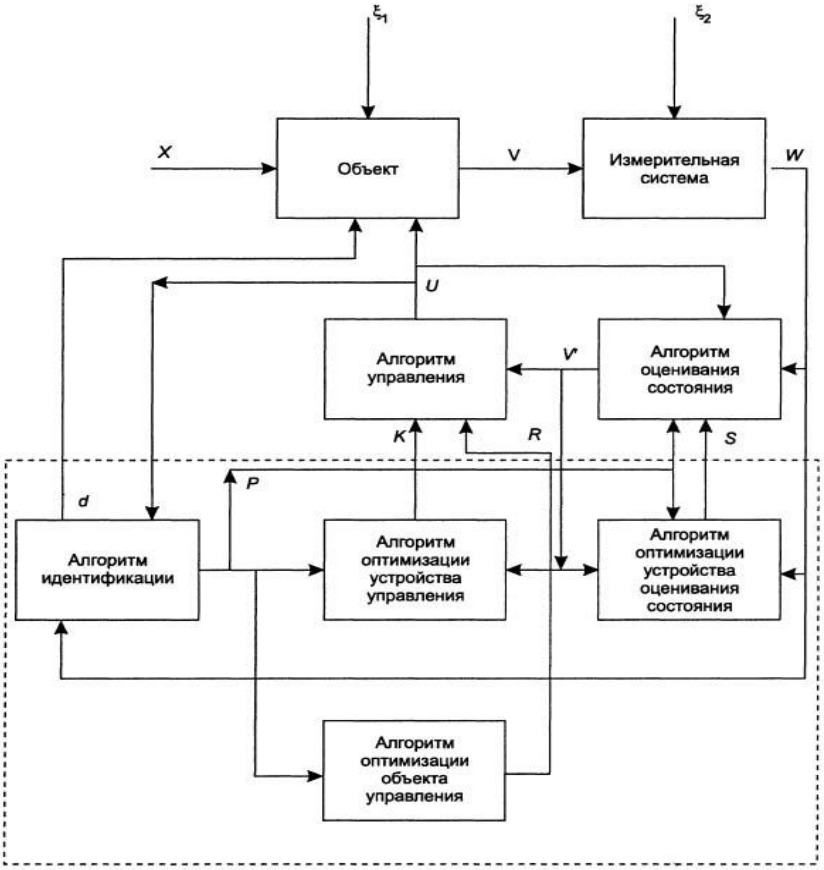
Пусть рассматривается m вариантов выполнения системы, каждый из которых позволяет выполнить Техническое задание на проектирование.. Введем следующие переменные $x_{k,i} \in [0,1], k = 1,...,N; i = 1,...,m$, которые определяют различные затраты: временные,

вычислительные, людские, финансовые и прочее, для i - го варианта проекта. Тогда формальную постановку задачи оптимизации выполнения проектных работ можно сформулировать следующим образом.

Найти такой набор переменных $x_k \in [0,1]$, чтобы: $\sum_{k=1}^{m} \alpha_k \cdot x_k \to \min$, при ограничениях $0 \le x_k \le x_{k,доп}$. Здесь α_k - соответствующие весовые коэффициенты, а $x_k \in \{x_{k1}, x_{k2}, ..., x_{k,m}\}$.

Приближенное решение подобной целочисленной задачи оптимизации, так называемой задачи о ранце, будет рассмотрено далее. Пока лишь отметим, что точное решение подобной задачи возможно лишь с помощью метода перебора различных вариантов.

Рассмотрим теперь типовые задачи исследования и проектирования САУ на этапах ее анализа, параметрического синтеза и моделирования. Типовая алгоритмическая структура системы управления динамическим объектом имеет следующий вид.



Обведенная пунктиром часть системы управления может функционировать вне контура управления и использоваться только на стадии проектирования. Например, если параметры объекта и регулятора не меняются в процессе функционирования, то процедуры идентификации достаточно использовать только на этапе исследования объекта. Аналогично, если условия функционирования системы не будут меняться, то алгоритмы оптимизации регулятора и наблюдателя (устройства оценивания состояния), достаточно использовать только на этапе проектирования.

Типовые задачи анализа и моделирования.

Идентификация модели объекта управления.

При решении задачи идентификации требуется определить некоторую, наиболее соответствующую экспериментальным данным, математическую модель объекта, которая наиболее адекватно описывает временные соотношения между входными и выходными сигналами. Различают два подхода к процессу идентификации.

Пассивная идентификация проводится в режиме обычного функционирования объекта управления без оказания на него специальных пробных сигналов. В этом случае, задача параметрической идентификации сводится к нахождению вектора параметров p модели объекта, при котором будет обеспечиваться минимум некоторого функционала $J(p)$, который зависит от ошибок выходов модели и объекта:

$$J(p) = \sum_{k}^{N} \sum_{i}^{m} (W_{i,p}(t_k) - W_{i,m}(t_k, p))^2 ,$$

Где $W_{i,p}(t_k)$ i - ый выход объекта в момент времени t_k, $W_{i,m}(t_k)$ i - ый выход модели объекта в момент времени t_k.

Активная идентификация предполагает подачу на вход объекта и, соответственно, модели специальных идентифицирующих воздействий, позволяющих выявить наиболее полно динамические характеристики объекта. Целевой функционал, с помощью которого определяются параметры объекта, будет практически такой же, как и при пассивной идентификации. Однако в этом случае необходим дополнительный модуль, обеспечивающий оптимизацию плана построения подачи пробных сигналов (модуль оптимизации плана организации эксперимента). К сожалению, некоторые объекты не допускают активного вмешательства в их работу, что ограничивает сферу применения активной идентификации. С другой стороны весьма часто как активная, так и пассивная идентификация осуществляются непрерывно в процессе непосредственного управления объектом. Это позволяет по мере поступления новой информации об объекте и условиях его функционирования проводить уточнение параметров модели непосредственно в контуре управления.

Выбор структуры измерительной системы.

Процедуры оценивания вектора состояния объекта управления строят наилучшую, в некотором смысле, оценку $\hat{V}$ вектора состояний на основе измеряемых выходов и параметров модели объекта. Для реализации алгоритмов оценивания/наблюдения необходимо предварительно оценить наблюдаемость объекта (в случае нелинейного объекта необходимо также оценить границы области наблюдаемости фазовых координат). Обычно различают три способа получения оценок состояния динамического объекта: сглаживание, фильтрация и прогноз. При решении задач сглаживания, оценка вектора состояний объекта для некоторого момента строится на основе данных наблюдений за выходом объекта на некотором конечном интервале.

Решение о выборе параметров фильтра (наблюдателя) для всех трех способов решается на основе минимизации функционала от невязок между сигналами на выходе устройства оценивания и на выходе объекта. При имеющейся информации о детальной структуре объекта и свойствах ошибок такую задачу иногда можно решить аналитическим способом. Но в большинстве случаев приходится привлекать вычислительные процедуры оптимизации.

Параметрический синтез системы.

Проблема параметрического синтеза системы, то есть определение необходимых настроек регулятора или параметров закона управления, базируется на исследовании динамических свойств замкнутой системы. В рамках данной проблемы приходится решать следующие задачи:

- преобразование структурной схемы системы и вычисление итогового оператора (передаточной функции) замкнутой системы на основе информации о математических моделях звеньев ее составляющих;
- определение точек равновесия и установившихся режимов системы;
- определение областей устойчивости замкнутой системы и оценивание запасов устойчивости;
- определение областей управляемости фазового пространства объекта;
- расчет переходных процессов в системе;
- вычисление ошибок работы системы в установившемся режиме;

- определение параметров регулятора, обеспечивающих необходимые динамические свойства замкнутой системы, и областей их возможного регулирования;
- математическое моделирование динамики замкнутой системы для различных режимов функционирования и воздействия возмущающих сигналов.

Оптимизация настройки параметров системы

Решение задачи параметрического синтеза обычно дополняются условиями выбора наиболее оптимального экземпляра системы с точки зрения некоторых критериев: быстродействие, помехозащищенность, точность и прочее. Рассмотрим процесс формализации такой экстремальной задачи параметрического синтеза. Пусть имеется векторный критерий функционирования системы $W(p) = (W_1(p), W_2(p), ..., W_m(p))$, где $p \in D \subset \Re^n$ – некоторый n - мерный вектор настраиваемых параметров системы, D - допустимая область варьирования вектора p.

Тогда достаточно общую задачу оптимальной настройки системы можно представить в следующем виде:

$$p^* = \arg\{\min_{p \in D}(W(p)\}$$

Решение такой многокритериальной задачи, в общем случае, является неоднозначным и принадлежит некоторому множеству допустимых решений. В пределах этого множества выполняются ограничения на искомые параметры, обычно представленных в виде соответствующей системы неравенств и равенств. Эти ограничения можно рассматривать как формальное представление основных требований технического задания, такие например, как ограничение времени переходного процесса, величины перерегулирования, ограничения на коэффициент усиления прямой цепи системы и прочее.

Задача оптимальной настройки системы не является стандартной из-за наличия векторного критерия оптимальности. Сведение такой задачи к однокритериальному типу, который имеет достаточно эффективные вычислительные алгоритмы решения, не является однозначным и зависит во многом от предпочтений лица принимающего решения (ЛПР) и требований технического задания.

Имеется большое количество задач, не входящих в приведенный выше перечень, которые приходится решать проектировщику систем автоматического управления. Однако, в своей основе, изучение такой схемы позволяет определить базовый уровень знаний, необходимый для разработки современных систем.

Основные структуры данных

Разработка вычислительных алгоритмов для компьютерного решения вышеперечисленных задач проектирования систем автоматического управления проводится обычно в математических или формализованных прикладных терминах, которые в достаточно большой степени не отражают технических возможностей вычислительных средств. Так в настоящее время большинство алгоритмов проектируется для использования в устройствах, обладающих адресуемой памятью. Информация, размещенная в адресуемой памяти, приобретает некоторые новые свойства. Элемент информации в этом случае характеризуется не только своим содержанием, но и адресом, то есть местом расположения в памяти. При этом два элемента данных, соседних в смысле математической модели, не обязательно будут располагаться в соседних ячейках памяти. Проектируя программную реализацию вычислительного алгоритма, приходится проектировать и способ расположения в памяти обрабатываемой информации.

Существуют типы данных, которые используются в математических моделях систем, которые естественным образом вкладываются в адресуемую структуру технической памяти. Примером может служить вектор фиксированной размерности, задаваемый упорядоченным набором своих компонент. Хранение вектора удобно осуществить с помощью массива ячеек, при котором соседние компоненты вектора располагаются в ячейках памяти с подряд идущими номерами. Этот способ позволяет легко программировать такие операции как сложение векторов, вычисление скалярного

произведения и прочее. Однако если размерность вектора в процессе работы алгоритма меняется, то представление в виде массива оказывается неудобным, а в ряде случаев и нереализуемом за требуемый интервал времени. Поэтому обычно одновременно с разработкой вычислительного алгоритма проектируется структура представления данных, позволяющая эффективно реализовать выполнение всех необходимых операций. Для решения этих задач в практике программирования используются определенные способы структурирования информация и разработаны определенные абстрактные типы или структуры данных, которые в дальнейшем удобно реализовывать в программной системе. С помощью этих же абстрактных типов данных удобно также реализовывать большинство известных математических моделей. Место абстрактного типа данных в схеме построения программной реализации вычислительных алгоритмов можно оценить с помощью приведенного ниже рисунка.

Абстрактный тип данных обычно определяется как математическая модель с совокупностью операторов, определяемых в рамках этой модели. В качестве примера можно привести множество целых чисел с операторами объединения, пересечения и разности множеств. При этом в конкретной модели абстрактного типа данных операторы могут иметь в качестве операндов не только данные, определяемые в этой модели, но и данные других типов: стандартных типов языка программирования или определенные в других моделях абстрактного типа данных.

Понятие о вычислительном процессе как об алгоритме.

Ранее широко использовались такие термины, как вычислительный алгоритм или просто «алгоритм», реализация алгоритма на ЭВМ. Однако формального определения дано не было. Поэтому рассмотрим более подробно понятие алгоритма и исследуем возможность его реализации с помощью вычислительных устройств.

Слово «алгоритм» содержит в своем составе преобразованное географическое название «Хорезм» и обязано своим происхождением великому ученому Муххамад ибн Муса ал-Хорезми (Магомед, сын Моисея, из Хорезма), жившем с 783 по 850 гг в Ургенче Хорезмского халифата. Имя ал-Хорезми, при переводе его арифметического трактата на латинский язык, транскрибировалось как algorismi. Отсюда и пошло слово «алгоритм», как обозначение последовательности цифровых вычислений в десятичной арифметике. Задача точного определения алгоритма была решена в 30-х годах прошлого столетия в работах Гильберта, Черча, Клини, Поста и Тьюринга в двух формах: на основе понятия рекурсивной функции и на основе описания алгоритмического процесса.

Рекурсивная функция – это конкретная функция, определенная в некоторой формальной системе, для которой существует алгоритм вычисления ее значений по произвольному значению аргумента. Другой подход заключался в том, что алгоритмический процесс определяется как процесс, осуществимый на конкретно устроенной машине, так называемой «машине Тьюринга». Оба данных подхода, а также другие подходы, разработанные Марковым и Постом, привели к одному и тому же классу алгоритмически вычислимых функций. Поскольку понятие рекурсивной функции строгое, то с помощью математической техники можно доказать, что решающая некоторую задачу функция не является рекурсивной, то это эквивалентно отсутствию для данной задачи вычислительного алгоритма. Аналогично, отсутствие разрешающей машины Тьюринга для некоторой задачи, также равносильно отсутствию для нее вычислительного алгоритма. Таким образом появилась дескриптивная теория алгоритмов, содержанием которой стала классификация задач по признаку вычислительной разрешимости, то есть получение высказываний типа «задача П алгоритмически разрешима» или «задача П алгоритмически неразрешима».

Широкое применение алгоритмов с появлением вычислительной техники привело к необходимости классификации их по таким показателям, как число действий или быстродействие, требуемый расход памяти. Это привело к необходимости решения задач оптимизации алгоритмов по указанным критериям. Появилась необходимость указания для сложности (время или память) алгоритмов некоторой верхней границы, которая определяется как функция верхней оценки сложности решения, так и нижней границы, которая показывает, что никакой алгоритм решения конкретной задачи не имеет сложности меньшей, чем определенная граница. В связи с этим получила развитие проблематика получения «относительных» нижних оценок, так называема теория NP-полноты.

Основные требования к алгоритму.
1. Для каждого алгоритма требуются данные – входные, промежуточные или выходные. Поэтому перед разработкой алгоритма всегда уточняется понятие данных, фиксируется алфавит исходных символов и указываются правила построения и преобразования алгоритмических объектов, как это было уже указано на предыдущей лекции для абстрактных типов данных. В качестве алгоритмических объектов могут выступать и формулы, например формулы алгебры предикатов и алгебры высказываний.
2. Алгоритмам для размещения данных требуется память. Память обычно считается однородной и дискретной, то есть она состоит из одинаковых ячеек, причем каждая ячейка может содержать один элемент данных, что позволяет согласовывать единицы измерения объема данных и памяти.
3. Алгоритм состоит из отдельных элементарных шагов, причем множество различно шагов, из которых составлен алгоритм, конечно. Типичный пример множества элементарных шагов – система команд ЭВМ.
4. Последовательность шагов алгоритма детерминирована, то есть после каждого шага указывается, какой шаг следует выполнять дальше, либо указывается, когда следует работу алгоритма считать законченной.
5. Алгоритм должен обладать результативностью, то есть останавливаться после конечного числа шагов 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. При этом алгоритм и механизм его реализации должны быть конечными и результативными.

Имеются еще некоторые требования к неформальному определению алгоритма, по которым еще не достигнуто окончательного соглашения. Эти требования обычно формулируют в виде вопросов.
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?

Таким образом, уточнение понятие алгоритма связано с уточнением алфавита данных и формы их представления, памяти и размещения в ней данных, элементарных шагов алгоритма и механизма реализации алгоритма. В теории алгоритмов для удовлетворения вышеуказанных требований обычно используются конкретные алгоритмические модели, в которых все сформулированные требования выполняются естественным образом. При этом используемые алгоритмические модели универсальны, то есть моделируют любые другие разумные алгоритмические модели, что позволяет снять возможные противоречия меду моделями.

Типы алгоритмических моделей.

Сейчас существуют три основных типа алгоритмических моделей.

Первый тип трактует алгоритм как некоторое детерминированное устройство, способное выполнять в каждый момент лишь строго фиксированное множество операций. Основной теоретической моделью данного типа является машина Тьюринга. Другой теоретической моделью данного типа является машина произвольного доступа, введенная достаточно недавно (в 70-х годах прошлого столетия) с целью моделирования реальных вычислительных машин и получения оценок сложности вычислений.

Второй тип связывает понятие алгоритма с традиционным представлением вычислений, как вычислением числовых функций. Основной теоретической моделью этого типа являются рекурсивные функции.

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

Машина Тьюринга и функции, вычислимые по Тьюрингу.

Машина Тьюринга состоит из следующих элементов.

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

2. Управляющего устройства, которое может находиться в одном из конечного числа внутренних состояний $Q=\{q_0,q_1,...,q_n\}$. Число элементов Q характеризует объем внутренней памяти машины. Выделим в множестве Q два специальных состояния q_0 и q_n. Машина начинает работу в состоянии q_0, а попав в состояние q_n всегда останавливается.

3. Считывающей/пишущей головки, которая может перемещаться вдоль ленты и в каждый момент времени считывает одну из ячеек ленты или записывает в нее новый символ.

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

- записывает в текущую ячейку символ внешнего алфавита;
- сдвигает считывающую головку на один шаг влево или на один вправо или оставляет ее на месте;
- переходит в новое внутреннее состояние.

То есть работа машины определяется системой команд вида:

$$q_i\ a_j \rightarrow q_k\ a_l\ d\,, \qquad (1)$$

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 $q_i\ a_j; i=1,2,...,n; j=0,...,m$ имеется точно одна команда вида (1). Совокупность этих команд называется программой машины и имеется всего $n\cdot(m+1)$ команд. Предполагается, что в начальный момент времени на ленте все ячейки, кроме конечного их числа, содержат пустой символ. Поэтому и в любой другой конечный момент времени лента будет иметь лишь конечное число ячеек, содержащих непустые символы. Конфигурацией машины Тьюринга в конкретный момент времени назовем слово вида $\beta_1\ q_i\ \beta_2$, где q_i внутреннее слово, β_1 - слово из символов алфавита A, находящееся в левой части от считывающей головки, β_2 - слово, находящееся в правой части от считывающей головки. Конфигурацию в момент времени t обозначим K_i. Назовем конфигурацию заключительной, если $q_i = q_n$ и начальной, если $q_i = q_0$.

Если $K_0 = q_0\ \beta$ и $\beta = a_i\ \beta^*$, где $a_i \in A$, то в программе машины имеется точно одна команда вида $q_0\ a_i \to q_k\ a_l\ d$. Тогда следующая конфигурация K_1 определяется так:

$K_1 = q_k\ a_l\ \beta^*$, если d = E

$K_1 = \alpha_l\ q_k\ \beta^*$, если d = R

$K_1 = q_k\ a_0\ a_l\ \beta^*$, если d = L.

Таким образом, начальная конфигурация K_0 порождает последовательность конфигураций $K_0 \to K_1 \to ... \to K_t \to K_{t+1} \to ...$. Если эта последовательность конечна, то говорят, что машина применима к конфигурации K_0, в противном случае неприменима к конфигурации K_0.

Пусть машина применима к начальной конфигурации $K_0 = q_0\ \beta$ и $K_t = \gamma^*\ q_1\ \gamma^{**}$ является заключительной конфигурацией, то слово $\gamma = \gamma^*\ \gamma^{**}$ объявляется результатом работы машины на слове β. Будем рассматривать словарные частичные функции вида $F: A^* \to A^*$, где A^* - множество всех слов конечной длины в алфавите A. Говорят, что машина Тьюринга T правильно вычисляет частичную функцию F, если для любого слова $p \in A^*$ выполнены следующие условия.

1. Если $F(p)$ определено и $F(p) = Q$, то машина T применима к начальной конфигурации $q_0\ p$ и заключительной конфигурацией является $K_t = q_t\ Q$.

2. Если $F(p)$ не определено, то машина Т неприменима к начальной конфигурации $q_0\ p$.

Функция F называется правильно вычислимой по Тьюрингу, если существует машина T, которая ее правильно вычисляет.

Пример.

Рассмотрим работу машины Тьюринга с рабочим алфавитом $A = \{*, |\}$ и тремя внутренними состояниями $Q = \{q_0, q_1, q_2\}$, которая может быть описана следующим образом.

В состоянии q_0, независимо от содержимого рабочей ячейки алгоритм T сдвигает рабочую ячейку вправо и переходит в состояние q_1.

В состоянии q_1 алгоритм T заменяет символ, стоящий в рабочей ячейке символом $|$ и переходит в состояние q_2.

В состоянии q_2 алгоритм T останавливается независимо от содержимого рабочей ячейки.

Работу машины можно описать следующей системой команд (или так называемой T таблицей):

$$q_0\ * \to q_1\ *\ R$$
$$q_0\ * \to q_1\ |\ R$$
$$q_1\ * \to q_2\ |\ E$$
$$q_1\ | \to q_2\ *\ E$$

Пуск машины T происходит в состоянии q_0. Далее алгоритм обнаруживает стартовый символ $*$ в рабочей ячейке, сдвигает читающую головку вправо и переходит в состояние q_1. Затем алгоритм читает символ в рабочей ячейке и заменяет его соответствующим символом $*$ или $|$. В этой ситуации машина T останавливается.

Прямое построение машин Тьюринга для решения даже простых задач может оказаться затруднительным. Поэтому используют некоторые приемы, которые облегчают этот процесс. Например, используют суперпозицию машин, последовательное соединение машин. С помощью объединения программ двух машин реализуют ветвление и организацию цикла.

Рассмотрим, например, суперпозицию машин. Пусть даны две машины Тьюринга T_1 и T_2, которые вычисляют словарные функции $F_1(p)$ и $F_2(p)$, соответственно. Тогда существует машина Тьюринга T, которая вычисляет функцию $F(p) = F_2(F_1(p))$. При этом машина T строится за счет объединения программ машин T_1 и T_2 по определенным правилам.

Язык программирования машин Тьюринга содержит основные операторы программирования на алгоритмических языках. Это является основанием для предположения о том, что для всех процедур, претендующих называться алгоритмическими, существует, при правильном кодировании, соответствующая машина Тьюринга, которая их реализует. Данное предположение носит название тезиса Черча-Тьюринга. Подтверждением этому является математическая практика.

Однако есть вполне определенные задачи, для которых машину Тьюринга построить нельзя. В частности Тьюринг показал, что не существует универсального алгоритма, который позволил бы решить задачу об условии остановки произвольной машины. Таким образом, Тьюринг показал, что не может быть общего алгоритма для решения математических задач. Однако это не означает, что в каждом отдельном случае нельзя выяснить справедливость некоторого математического утверждения или определить, остановится ли данная машина Тьюринга. Во многих частных случаях получить ответ удается. Но не существует, ни одного алгоритма, который позволял бы решать любую математическую задачу или давал ответ на вопрос об остановке любой машины Тьюринга при любых вводимых начальных конфигурациях.

Алгоритмически неразрешимые проблемы.

Рассмотрим предварительно так называемые массовые проблемы. Каждая массовая проблема представляет собой бесконечную серию индивидуальных задач. Например, индивидуальной задачей является такая: обладает ли заданным свойством G конкретная функция из некоторого класса. Совокупность всех таких задач (для всех функций данного класса) образует массовую проблему распознавания свойства G. Ограничимся такими массовыми проблемами, в которых все индивидуальные задачи имеют двузначный ответ: «Да» или «Нет».

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

$f(\pi)=1$, индивидуальная задача $\pi \in P$ имеет ответ «Да»;

$f(\pi)=0$, индивидуальная задача $\pi \in P$ имеет ответ «Нет»;

является вычислимой.

Основной метод, применяемый в доказательствах алгоритмической неразрешимости, базируется на следующем рассуждении. Пусть имеется две массовые проблемы P_1 и P_2. Пусть имеется алгоритм A, который для всякой индивидуальной задачи $\pi_1 \in P_1$ строит индивидуальную задачу $\pi_2 \in P_2$, такую, что выполнено: π_1 имеет ответ «Да» $\leftrightarrow$ π_2 имеет ответ «Да». В этом случае говорят, что проблема P_1 сводится к проблеме P_2. Если проблема P_1 неразрешима, то проблема P_2 также неразрешима.

Рассмотрим проблему самоприменимости машин Тьюринга. Рассматриваются машины Тьюринга внешние алфавиты которых, наряду с другими символами содержат символы 0 и 1. Для каждой машины T построим некоторый код $code(T)$, который является (0,1) словом, и запустим машину T в начальной конфигурации $q_0\, code(T)$. Если машина T останавливается через конечное число шагов, то она называется самоприменимой, в противном случае – нет.

Проблема самоприменимости состоит в том, чтобы по любой машина Тьюринга T определить, является она самоприменимой или нет. Говорят, что машина Тьюринга T решает проблему самоприменимости, если любую начальную конфигурацию $q_0\, code(T)$ она переводит в $q_n\, 1$, если машина самоприменима, и в $q_n\, 0$, если нет.

Теорема. *Не существует машины Тьюринга T решающей проблему самоприменимости, то есть проблема самоприменимости алгоритмически неразрешима.*

Таким образом, если некоторую проблему P удается свести к проблеме самоприменимости машины Тьюринга, то такая проблема будет неразрешимой. Выявлению разрешимых и неразрешимых задач в различных разделах фундаментальной и прикладной математики посвящено много исследований. Приведем некоторые из них.

Проблема представимости матриц. Рассматриваются $(n \times n)$ – матрицы над кольцом целых чисел Z. Пусть даны произвольные матрицы $\Omega_U = \{U_0, U_1, ..., U_q\}$ и U. Нужно найти алгоритм, который бы решал, верно ли равенство $U = U_{i1} \cdot U_{i2} \cdot ... \cdot U_{ip}$, где $U_{ij} \in \Omega_U; j = 1, 2, ..., p$. Неразрешимость данной проблемы, начиная с $n = 4$, установлена А.А. Марковым.

Проблема полноты автоматных базисов. Дан набор конечных автоматов (базис) с одним множеством входов и выходами, входящими в множество входов. Строятся схемы с помощью автоматов, входящих в базис, и введения обратной связи. Каждая построенная схема реализует автомат. Если схемами в данном базис может быть реализован любой автомат в данном алфавите, то базисный набор называется полным, в противном случае нет. Задача состоит в определении полноты заданного базиса. Неразрешимость данной проблемы установил М.И. Кратко.

Десятая проблема Гильберта. Пусть задано диофантово уравнение, то есть уравнение вида $P(x_1, x_2, ..., x_n) = 0$, где 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 была установлена Ю.В. Матиясевичем.

Лямбда-исчисление Черча, рекурсивные функции и вычислимость.

Вычислимость является важнейшим абстрактным понятием для вычислительных задач. Это понятие связано с существованием четко определенных и все же неразрешимых математических операций, таких как, например, проблема остановки машины Тьюринга. Исторически, первым способом выражения вычислимости было проведено А.Черчем, путем применения математических процедур, реализованных с помощью схемы лямбда-исчисления.

В рамках данной схемы рассматривается универсальное множество различных объектов, обозначаемых, например, символами $a,b,c,d,...,z,a',b',...,z',a'',b'',...,z'',...$, каждый из которых представляет собой математическую операцию, или функцию. Аргументы этих функций, то есть объекты, на которые эти функции действуют, в свою очередь, являются объектами той же природы, то есть функциями. Более того, результат действия одной функции на другую также представляет функцию. То есть выражение $a = b \circ c$ означает, что функция b, действуя на функцию c, дает в результате функция a. Важнейшую роль в теории Черча носит операция абстрагирования, для обозначения которой используется греческая буква λ. Непосредственно за ней будет следовать символ одной из функций, например x, который будет рассматриваться как «фиктивная переменная». Каждое появление x в квадратных скобках, следующих сразу за выражением, обозначает теперь просто место, куда подставляется все, что идет за этим выражением. Таким образом, под записью $\lambda x.[fx]$ понимается функция, которая при действии на объект a имеет значения fa, то есть $\lambda x.[fx]a = fa$. Другими словами $\lambda x.[fx] = f$.

С помощью данного определения можно определять новые функции. Например введем следующее функциональное определения $Q = \lambda x.[\frac{x^3}{6}]$. Тогда, например, получим:

$$Q(a+1) = \frac{(a+1)^3}{6}.$$

В лямбда-исчислении большое значение имеют выражения, составленные просто из элементарных функциональных операций, таких как $\lambda f.[f(fx)]$. Эта функция, которая, действуя на другую функцию, скажем g, дает дважды итерированную g, действующую на x, то есть: $(\lambda f.[f(fx)])g$. Таким образом, данная функция дает «вторую итерацию» g. Черч в своем исчислении ввел следующие формулы классов функций соответствующих итераций:
$0 = \lambda x.[fx];\ 1 = \lambda fx.[fx];\ 2 = \lambda fx.[f(fx)];\ 3 = \lambda fx.[f(f(fx))];$ и так далее.

Значит действие **3** на функцию f, то есть $3f$ равносильно операции «применить f три раза»: $(3f)y = f(f(f(y)))$. Такой метод описания класса функций является бестиповой теорией, в которой объекты (функции) рассматриваются как функции, так и аргументами. То есть теория лямбда-исчисления рассматривает функции, как правила, а не как графики. В частности функция может применяться самой к себе. Черчем был доказан результат, что для натуральных чисел, определимая с точки зрения лямбда-исчисления функция является рекурсивной и сформулировал свой тезис, утверждающий, что рекурсивность является точной формализацией эффективной вычислимости. Тьюринг на основе анализа механической вычислимости в дальнейшем показал, что полученное им понятие эквивалентно лямбда - вычислимости. К сожалению, если функция рекурсивная (частично-рекурсивная), то ее не всегда можно определить с точки зрения лямбда-исчисления (алгоритмическая неразрешимость). Например, если рассмотреть задачу (проблема

Гольдбаха) о том, что всякое четное число большее двух можно представить в виде суммы двух простых чисел и определить функцию h так:

$h(x)=1$, если проблема решается положительно,

$h(x)=0$, если проблема решается отрицательно,

то можно показать, что $h(x)$ является примитивно-рекурсивной. Однако для этой функции неизвестно, каким образом построить верную схему. Следует отметить, что теория лямбда-исчисления, доработанная Скоттом в 1969 году, легла в основу такого алгоритмического языка, как ЛИСП, который является до сих пор основным языком САПР (AutoCad).

Нормальные алгоритмы Маркова.

Подход, разработанный А.А. Марковым, связывает понятие эффективной вычислимости 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.

Пусть $A=\{a_1,a_2,...,a_n\}$ - алфавит. Если $P,Q\in A^*$ слова, построенные на базе алфавита A (возможно пустые), то выражения $P\to Q,\ P\to(.)Q$ называются формулами подстановки в алфавите A. При этом формула $P\to Q$ называется простой, а формула $P\to(.)Q$ заключительной. Произвольную конечную последовательность таких формул будем называть схемой S_a нормального алгоритма a. То есть схема нормального алгоритма имеет вид:

$$S_a=\begin{cases} P_1\to(.)Q_1 \\ P_2\to(.)Q_2 \\ \quad ... \\ P_m\to(.)Q_m \end{cases}$$

Схема S_a определяет алгоритм, перерабатывающий слова в алфавите A. Говорят, что слово P входит в слово W, если существуют слова V_1 и V_2 (возможно пустые) такие, что $W=V_1\cdot P\cdot V_2$. Если слово V_1 имеет наименьшую длину из всех слов вида W, то говорят о первом вхождении P в слово вида W.

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

1. Ни одно из слов $P_1,P_2,...,P_m$ не входит в слово K. В этом случае говорят, что схема S_a неприменима к K. Запишем этот случай в виде: $S_a:K$.

2. Среди слов $P_1,P_2,...,P_m$ существуют такие слова P_i, которые входят в K. Обозначим через t минимальное число, такое, что P_t входит в K. Пусть $K=V_1\cdot P_t\cdot V_2$, где V_1 имеет минимальную длину (то есть берется первое вхождение P_t в K). Определим следующее слово $W=V_1\cdot Q_t\cdot V_2$.

В этом случае говорят, что схема S_a применима к K. Соответственно можно записать формулы преобразования:

$$S_a:\begin{cases} K\to W \\ K\to(.)W \end{cases},$$

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

Работа нормального алгоритма может быть описана так. Если дано слово P, то находим в схеме алгоритма S_a первую формулу $P \to Q$ такую, что P_t входит в P, и производим замену первого вхождения P_t словом Q_t. Пусть W_1 есть результат этой подстановки. Если применяемая формула $P \to (.)Q$ заключительная, то работа алгоритма заканчивается и слово W_1 есть результат алгоритма. Если применяемая формула $P \to Q_t$ простая, то применяем к слову W_1 описанные выше действия. Если на некотором шаге получено слово W_i, к которому неприменима схема S_a, то работа алгоритма заканчивается и W_i есть результат. Если процесс не заканчивается то, по определению, алгоритм неприменим к слову P.

Словарная функция f в алфавите A (то есть $f : A^* \to A^*$) называется вычислимой по Маркову, если существует схема нормального алгоритма S_a в алфавите $B \subseteq A$, вычисляющая f. Класс функций, вычислимых по Маркову, обычно обозначается M.
Имеет место следующая теорема.

Теорема. *Класс функций M, вычислимых по Маркову, совпадает с классом функций T, вычислимых по Тьюрингу, и, следовательно, с классом частично-рекурсивных функций* ***Ч*** *по Черчу.*

Функции сложности алгоритма.

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 алгоритмов. Однако нетрудно понять, что принципиальная возможность алгоритмического решения задачи еще не означает, что оно может быть практически получено. Поэтому возникает потребность уточнения понятия алгоритмической разрешимости, введя характеристики слабости и сложности алгоритмов, позволяющих судить об их практической реализуемости. Выше было показано, что различные алгоритмические модели приводят к одному и тому же классу вычислимых функций. В то же время ясно, что выбор алгоритмической модели, реализующей алгоритм, существенно влияет на сложность вычислений. Однако проведенные исследования показали, что имеется ряд факторов определяющих сложность алгоритма, которые не зависят от алгоритмической модели и относятся к так называемой машинно-независимой теории сложности. Введем некоторые 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, выполненную при вычислении $f(x)$, если $f(x)$ определено. Если значение $f(x)$ не определено, то и значение $t_T(x)$ считается неопределенным. Функция $t_T(x)$ называется временной сложностью. Будем называть активной зоной при работе машины T на слове x 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. *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:*

$$s_T(x) \le |x| + t_T(x)$$
$$t_T(x) \le R \cdot s_T^2(x) \cdot K^{S_T(x)}$$

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

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

Поведение таких функций временной и емкостной сложности в пределе при увеличении размера задачи n называется асимптотической (соответственно – емкостной) сложностью.
Как следствие из теоремы о функциях сложности алгоритма вытекает следующее утверждение.

Теорема. *Для любой машины Тьюринга выполняется неравенство* $s_T(n) \le t_T(n)$*, то есть емкостная сложность не превышает временную.*

Соотношения $t_T(n) = \max_{|x|=n}(t_T(x)); \; s_T(n) = \max_{|x|=n}(s_T(x))$, по сути, определяют сложность алгоритма, реализуемого с помощью машины T. Естественно, хотелось бы на основании данного соотношения определить и сложность задачи – например, как сложность самого эффективного (по времени и емкости) алгоритма, решающего эту задачу. К сожалению это невозможно. Доказано, что есть массовые задачи, для которых не существует самого быстрого алгоритма, потому что любой алгоритм для массовой задачи можно «ускорить», построив более быстрый алгоритм для конкретного экземпляра задачи, принадлежащий классу таких массовых задач.

Теорема Блюма об ускорении. *Если существует такая алгоритмически разрешимая задача* W*, то любой алгоритм* A*, решающий задачу* W*, можно ускорить следующим образом: существует другой алгоритм* A'*, также решающий* W *и такой, что* $t_{A'}(n) \le \log(t_A(n))$ *для почти всех* n*.*

То есть теорема Блюма утверждает, что существование «неудобных» задач не позволяет определить универсальное (применимое ко всем конкретным экземплярам массовой задачи) понятие оптимального алгоритма.

Оценки скорости роста временной сложности алгоритмов.

Для определение функций временной и емкостной сложности при реализации алгоритмов с помощью реальных компьютеров обычно используют модель обобщенной однопроцессорный машины с памятью с произвольным доступом (МПД). Анализ производительности с помощью данной модели неплохо предсказывает значения функций временной и емкостной сложности при выполнении на реальных компьютерах. Однако такой анализ даже для относительно простых алгоритмов требует значительных усилий и затрат времени. С другой стороны реального разработчика алгоритма обычно мало волнует конкретные значения функций сложности для конкретного значения. Зная аппаратные и временные затраты для конкретного значения n модельного примера, разработчика больше интересует скорость или порядок роста затрат. Если время работы алгоритма выражается формулой $a \cdot n^2 + b \cdot n + c$, где a, b, c – некоторые константы, то для оценки скорости роста следует принимать только главный член формулы, поскольку, при больших значениях n, членами меньшего порядка можно пренебречь. То есть, время работы такого алгоритма в наихудшем случае равно $\Theta(n^2)$. Обычно один алгоритм считается эффективнее другого, если время его работы в наихудшем случае имеет более низкий порядок роста. Из-за наличия постоянных множителей и второстепенных членов эта оценка может быть ошибочной, если

входные данные невелики. Однако, если объем входных данных значительный, то например, алгоритм $\Theta(n^2)$ в наихудшем случае исходных данных работает быстрее, чем алгоритм $\Theta(n^2)$.

При рассмотрении входных данных достаточно больших размеров для оценки только такой величины, как порядок времени работы алгоритма, необходимо исследовать асимптотическую эффективность алгоритмов. Введем основные асимптотические обозначения, характеризующие скорость роста функций в пределе при возрастании размера n входных данных.

Запись $\Theta(g(n)) = f(n)$ *обозначает множество функций* $f(n)$ *таких, что существуют положительные константы* C_1, C_2 *и* $n0$ *такие, что* $0 \le C_1 \cdot g(n) \le f(n) \le C_2 \cdot g(n)$ *для всех* $n \ge n0$.

Согласно определению множества $\Theta(g(n))$, необходимо, чтобы каждый элемент $f(n) \in \Theta(g(n))$ этого множества был асимптотически неотрицателен, то есть при достаточно больших значениях n функции $f(n)$ и $g(n)$ были неотрицательными. В Θ - обозначениях функция асимптотически ограничивается сверху и снизу. Если же достаточно определить только асимптотическую верхнюю границу, то используют O обозначения.

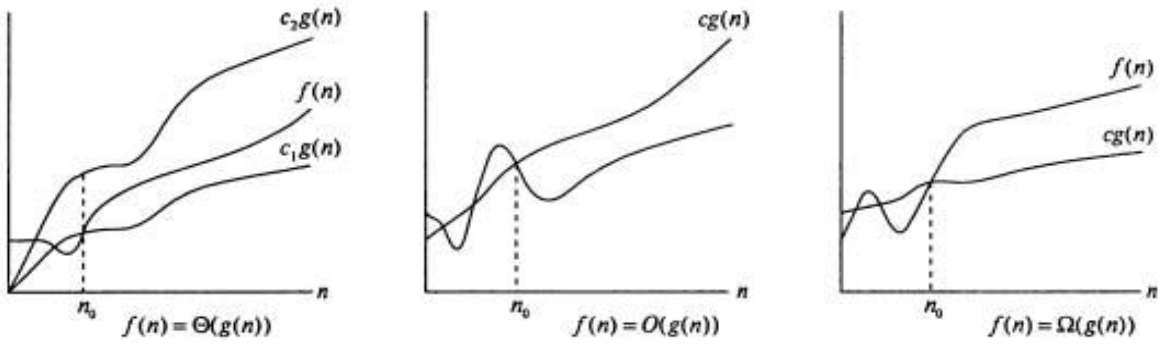
Запись $O(g(n)) = f(n)$ *означает множество функций* $f(n)$ *таких, что существуют положительные константы* $C, n0$ *такие, что* $0 \le f(n) \le C \cdot g(n)$ *для всех* $n \ge n0$.

Обозначения O применяются, когда нужно указать верхнюю границу функции с точностью до постоянного множителя. При этом не играет роли, насколько близко функция $f(n)$ приближается к асимптотически верхнему пределу $C \cdot g(n)$. То есть, когда говорят, что время работы алгоритма равно $O(g(n))$, то подразумевают, что при любом вводе размера n входных данных время решения задачи с данным вводом ограничено сверху значением функции $g(n)$, умноженным на константу. Обозначение $O(1)$ часто используют, чтобы показать, что верхняя оценка роста функции не зависит от размера n входных данных. Для обозначения асимптотической нижней границы функции используется обычно Ω обозначение.

Запись $\Omega(g(n)) = f(n)$ *обозначает множество функций, таких, что существуют положительные константы* $C, n0$ *такие, что* $0 \le C \cdot g(n) \le f(n)$ *для всех* $n \ge n0$.

То есть, когда говорят, что время работы алгоритма равно $\Omega(g(n))$, при этом подразумевается, что независимо от того, какие входные данные выбраны для данного размера n, при достаточно больших n время работы алгоритма представляет как минимум константу, умноженную на $g(n)$.

На рисунке, приведенном ниже, показаны графические примеры Θ, O, Ω обозначений.



Пользуясь введенными определениями можно сформулировать следующую лемму.

***Лемма**. Для любых двух функций $f(n)$ и $g(n)$ соотношение $f(n)=\Theta(g(n))$ выполняется тогда и только тогда, когда $f(n)=O(g(n))$ и $f(n)=\Omega(g(n))$.*

Введем еще два дополнительных обозначения.

Обозначение $o(g(n))=f(n)$ означает, что для <u>любой</u> положительной константы C существует $n0>0$, такое, что $0\le f(n)\le C\cdot g(n)$ для всех $n\ge n0$.

Понятно, что для любой положительной константы это условие будет выполняться только тогда, когда $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=0$

.

Обозначение $\omega(g(n))=f(n)$ определяется, как множество всех функций $f(n)$ таких, что для <u>любой</u> положительной константы C выполняется условие $0\le g(n)\le C\cdot f(n)$ для всех $n\ge n0$.

Соотношение $f(n)=\omega(g(n))$ подразумевает, что $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=\infty$, если этот предел существует. Таким образом, функция $f(n)$ становится сколь угодно большой по сравнению с функцией $g(n)$, если n стремится к бесконечности.

Классификация алгоритмов по сложности.

Введем предварительно следующее определение.

$DTime(f(n))$ *– это класс задач, для каждого экземпляра которого, существует машина Тьюринга, решающая эту задачу с временной сложностью $O(f(n))$.*

Справедлива следующая теорем о классификации алгоритмов по сложности .

***Теорема (об иерархии сложности алгоритмов).** Пусть f и g – две вычислимые, конструируемые по времени функции, и $f(n)=\omega(g(n)\cdot\log(g(n)))$. Тогда класс $DTime(g(n))$ строго вложен в класс $DTime(f(n))$.*

Результаты этой теоремы позволяют упорядочить алгоритмы по сложности и ввести отношение предпочтения, на основании которого можно провести первичную их классификацию. Будем называть алгоритмы, имеющие полиномиальную временную сложность эффективными. Для таких алгоритмов справедливо соотношение $t_T(n)=O(p(n))$, где $p(n)$ - некоторый полином. Задачи, для которых в настоящее время не известны эффективные алгоритмы или для которых доказано отсутствие таких алгоритмов, будем называть труднорешаемыми. В частности, к труднорешаемым относятся задачи, алгоритмы которых реализуются машинами Тьюринга с экспоненциальной функцией сложности, то есть имеющими оценки временной сложности вида $O(n^{\log_2 n})$.

Класс P задач.

Класс задач, решаемых с помощью эффективных (полиномиальных) алгоритмов, обозначим через P. Класс P определен через функцию временной сложности машины Тьюринга. Очевидно, что

можно сделать соответствующие определения и через любую другую алгоритмическую модель. Имеется ряд фактов о полиномиальной эквивалентности функций сложности многих типов вычислительных моделей, что позволяет утверждать, что класс P определен однозначно для различных моделей. Задача называется полиномиальной, то есть относится к классу *P*, если существует константа k и алгоритм, решающий задачу с верхней оценкой $F_a(n) = O(n^k)$. Для существующих алгоритмов обычно $k < 6$. 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 данного алгоритма.

Функция временной сложности	Размер решаемой задачи на ЭВМ за единицу времени	Тоже на ЭВМ в 100 раз быстрее	Тоже на ЭВМ в 1000 раз быстрее
n	N_1	$100 \cdot N_1$	$1000 \cdot N_1$
n^2	N_2	$10 \cdot N_2$	$31.6 \cdot N_2$
n^3	N_3	$4.64 \cdot N_3$	$10 \cdot N_3$
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. Кроме того, полиномиальный алгоритмы обладают важным свойством «замкнутости» - можно комбинировать полиномиальные алгоритмы, используя один в качестве «подпрограммы» другого и, при этом, результирующий алгоритм будет полиномиальным. Следует, однако, 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.

Большинство экспоненциальных алгоритмов – это просто варианты полного перебора, в то время как полиномиальные алгоритмы обычно можно построить лишь тогда, когда удается более глубоко проникнуть в суть решаемой задачи. При этом следует понимать, что различие между полиномиальными и экспоненциальными алгоритмами может принять совсем иной характер, когда размеры решаемых задач невелики. Функция $f(n) = 2^n$ ведет себя лучше, чем $f(n) = n^5$ при $n \le 20.$

Кроме того временная сложность определена для массовой задачи некоторого класса, то есть для наихудшего случая задачи. Однако, может быть, что большинство индивидуальных задач требует для своего решения значительно меньших затрат времени на основании теоремы Блюма. В качестве примера можно привести алгоритм ветвей и границ, с помощью которого успешно решают задачу о «рюкзаке».

Класс NP задач.

Это задачи, для которых, до настоящего времени, не разработаны алгоритмы с полиномиальным временем работы, но и не доказано, что для какой - то из них, таких алгоритмов не существует.

Задачи, принадлежащие классу NP, поддаются проверке в течении полиномиального времени, то есть они допускают «быструю» проверку имеющего решения. Имеется в виду, что если каким-то образом был получен «сертификат» решения, то в течение времени, полиномиальным образом, зависящим от размера входных данных задачи, можно проверить корректность такого решения. Задача проверки «сертификата» решения соответствующей NP задачи называется задачей распознавания. Очевидно, что решение задачи распознавания имеет ответ типа «Да» или «Нет».

Например, если имеется решение о выполнимости формулы от булевых переменных, то в течение полиномиального времени легко проверить конкретный набор, удовлетворяет ли он заданной формуле. Любая задача класса P принадлежит классу NP, поскольку принадлежность классу P означает, что ее решение можно получить в течении полиномиального времени, даже не располагая сертификатом, то есть можно считать $P \subseteq NP$. Однако, тот факт, что класс P является строгим подмножеством класса NP , до сих пор не доказан.

Введем понятие класс NP задач распознавания, то есть имеющих ответ «Да» или «Нет». Задача распознавания J , содержится в классе NP тогда, когда J имеет ответ «Да»

Относительно класса NP можно сделать также следующие замечания.

1. Класс NP один и тот же для различных вычислительных моделей.
2. Класс P является замкнутым относительно дополнения задач. Для класса NP этого утверждать нельзя. Класс задач, являющихся дополнениями к задачам NP класса, обозначают CO-NP.

Под дополнением $\overline{J}$ задачи распознавания J называют задачу, в которой коды задач с ответом «Да» в точности соответствуют кодам задач J, которые не имеют ответ «Да».

Если принять, что P $\neq$ NP, то задачи из NP\P являются труднорешаемыми. Рассмотрим, как можно преобразовать задачу из NP\P в другую, аналогичную задачу из этого же множества.

Пусть Π_1 и Π_2 две задачи распознавания, задаваемые в алфавитах A_1 и A_2 соответственно. Будем говорить, что задача Π_1 сводится к задаче Π_2 (обозначение $\Pi_1 \to \Pi_2$), если существует словарная функция $f : A_1^* \to A_2^*$, такая , что выполнены условия:

1. f полиноминально вычислима;
2. $\forall x \in A_1^*$, где x индивидуальная задача Π_1 с ответом «Да»; $f(x) \in A_2^*$ индивидуальная задача Π_2 с ответом «Да».

Лемма. *Если выполнено* $\Pi_1 \to \Pi_2$ *и* $\Pi_2 \in P$*, то и* $\Pi_1 \in P$.

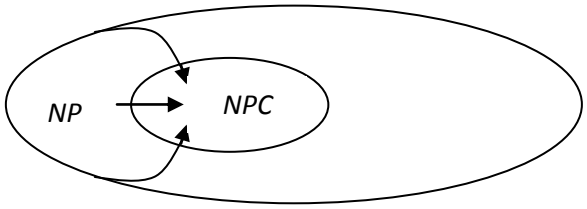
Определение*. Задача* Π *называется NP полной (NPC), если выполнено:*

1. $\Pi \in NP$*;*
2. $\Pi_1 \to \Pi$ *для любой задачи* $\Pi_1 \in NP$.

Обозначим через NPC – класс NP полных задач. Тогда имеем:

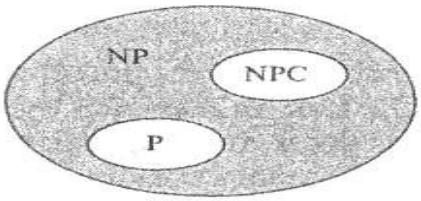
$$NPC \cap P \neq 0 \to P = NP;$$
$$NPC \cap (NP \setminus P) \neq 0 \to NPC \cap P = 0 \ .$$

Очевидно, что определение класса *NPC* (*NP*-complete) или класса *NP*-полных задач требует выполнения следующих двух условий: во-первых, задача должна принадлежать классу *NP (L* $\in$ *NP),* и, во-вторых, к ней полиномиально должны сводиться все задачи из класса *NP* , что схематично представлено на рисунке.



Другими словами: если для какой-то NP полной задачи существует полиномиальный разрешающий алгоритм, то и для любой задачи из класса NP будет существовать полиноминально разрешающий алгоритм.

Приведенные выше соотношения между классами P, NP и NPC изображено на рисунке ниже. Данные соотношения отражают представления большинства специалистов в области вычислительных алгоритмов.



Лемма. *Если задачи* $\Pi_1, \Pi_2 \in NP$, $\Pi_1 \to \Pi_2$ *и* $\Pi_1 \in NPC$, *то* $\Pi_2 \in NPC$.

Отсюда способ доказательства NP полноты конкретных задач сводится к нахождению полиномиального преобразования одной NP полной задачи к другой.

Теорема (Кук С, 1971г). *Задача проверки выполнимости произвольной КНФ (коньюктивно-нормальной формы) является NP полной задачей.*

Перечень некоторых NP полных задач.

1. Задача проверки выполнимости 3 –коньюктивной нормальной формы (3- CNF) является NP полной. (Булева формула выражена в 3-коньюктивной нормальной форме в том случае, если в каждых скобках содержится ровно три различных литерала . Например, булева форма $(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$ принадлежит классу 3-CNF).
2. Задача вершинного покрытия графа является NP полной. (Некоторое множество вершин $V_1 \in V$ 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).
3. Задача целочисленного «рюкзака». Для произвольных натуральных чисел $c_j, j = 1,2,...,n$, и числа K требуется узнать, существует ли набор целых чисел $x_j \geq 0, j = 1,2,...,n$, что выполнено $\sum_j^n c_j \cdot x_j = K$.
4. Задача коммивояжера. Коммивояжер должен посетить n городов ровно по одному разу и завершить путешествие в том же городе, из которого выехал. С каждым переездом из города i в

город j связана некоторая стоимость $c(i, j)$, выражающаяся целым числом. Требуется найти самый дешевый маршрут.

Многие задачи, представляющие практический интерес, являются NP полными и для них трудно найти точное решение. Несмотря на это надежда на нахождение решения остается. Во-первых, если объем входных данных небольшой, алгоритм, временная сложность которого выражается экспоненциальной функцией, вполне может подойти.

Во- вторых, иногда удается выделить важные индивидуальные задачи, разрешимые с помощью полиномиальных алгоритмов. В третьих, остается возможность найти в течении полиномиального времени решение, близкое, в некотором плане, к точному. Часто, при использовании приближенных алгоритмов, ошибка аппроксимации возрастает с ростом размера входных данных n. Однако некоторые NP полные задачи допускают наличие приближенных алгоритмов, коэффициент аппроксимации которых можно уменьшать за счет увеличения их работы. То есть допускается компромисс между временем вычисления и качеством приближения. В этом случае схема аппроксимации включает в себя не только входные параметры экземпляра задачи, но и параметр приближения ε > 0, который определяет качество приближения к точному решению. В ряде случаев удается найти приближенное решение с полностью полиномиальным временем работы. Например, время работы такой схемы можно оценить как $O((\frac{1}{\varepsilon})^2 \cdot n^3)$.