# Детерминированная машина Тьюринга (ДМТ)

## Определение алгоритма

Основные атрибуты алгоритма:

* Алгоритм задается последовательностью инструкций
* Алгоритм выполняется детерминировано, т.е. для одинаковых данных выполняются одинаковые действия
* Должен существовать вычислитель, способный выполнить указанные в алгоритме инструкции
* Вычислитель должен иметь средства для хранения и отображения информации

Таким образом алгоритм – это набор инструкций для формальной модели вычислителя.

Результативность алгоритма желательное, но не обязательное условие.

Алгоритм не обязан останавливаться – он может работать бесконечно.

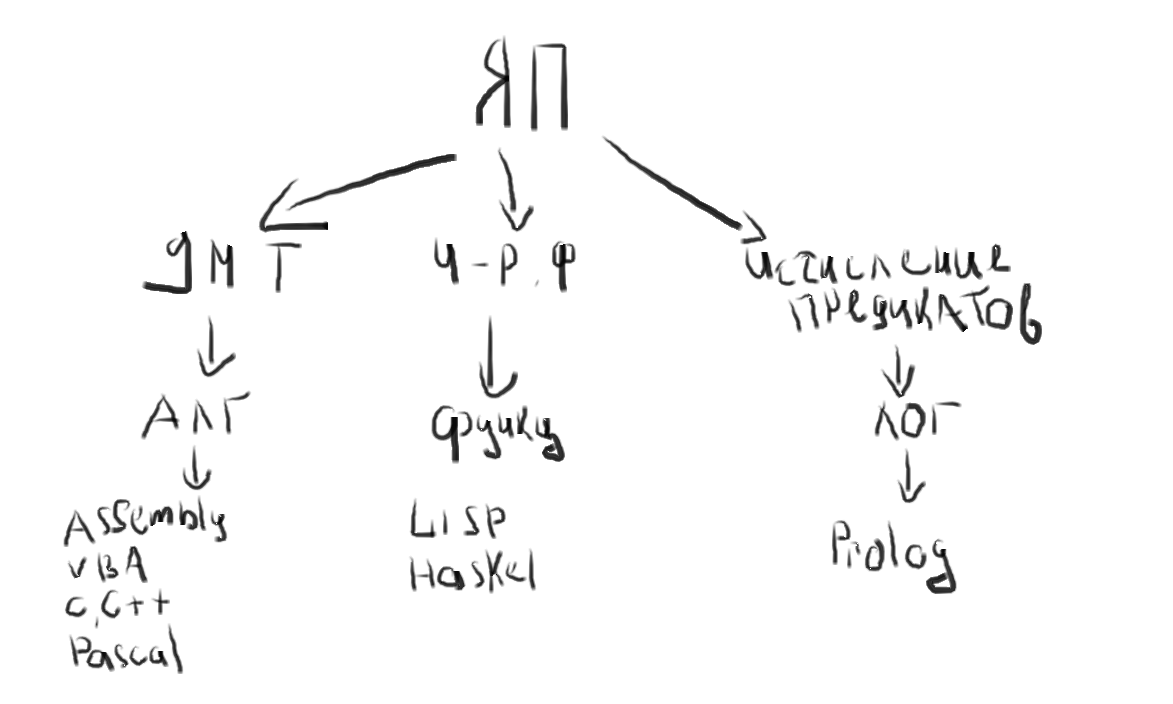
Математическая модель “ДМТ” является вычислителем, способным выполнить все указанные в алгоритме инструкции.

Существует 3 основные математические модели алгоритма:

* Детерминированная машина
* Частично рекурсивная функция
* Исчисление предикатов

Соответственно этим моделям языки программирования делятся на 3 группы:

* Алгоритмические (процедурные)
* Функциональные
* Логические



Программа на логическом языке представляет собой базу знаний, состоящую из истинных фактов и правил вывода этих фактов.

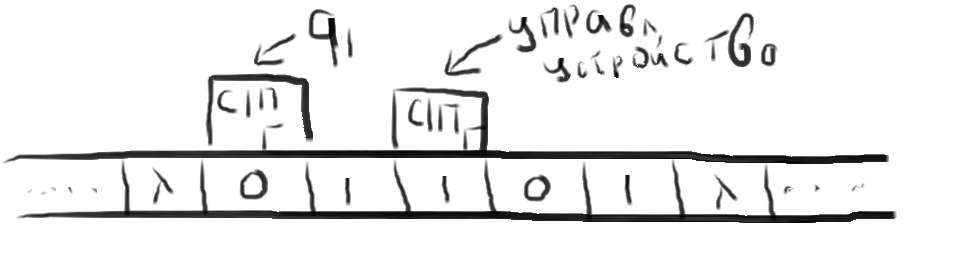
Чтобы запустить прогу нужно сформировать к базе знаний запрос или сформировать цель.

## ДМТ и тезис Тьюринга

### ДМТ как белый ящик или структурная схема

ДМТ – это автомат, который имеет бесконечную ленту, а также управляющее устройство и считывающую – пишущую головку.

Лента разделена на ячейки в каждой из ячеек записан либо пустой символ (λ), либо символ некоторого алфавита А. В каждый момент времени на ленте может быть конечное число не пустых символов.



Управляющее устройство находится в одном из конечного множества состояний. Т.е. количество состояний, как и количество символов конечно. Среди них выделяют начальное – q1 и конечное – qz. Считывающая головка в 1 момент времени обозревает 1 ячейку. Перед началом работы управляющее устройство ДМТ находится в начальном состоянии q1, а сама считывающая головка ДМТ находится в ячейке с 1 непустым символом. По достижению qz ДМТ останавливается.

Один этап работы ДМТ можно описать следующим свойством:

1. считать символ аj с текущей ячейки
2. затереть текущий символ аj и в зависимости от текущего состояния и прочитанного символа записать туда другой символ aj’
3. В зависимости от текущего состояния и прочитанного символа либо остаться на месте, либо сдвинуться на 1 шаг в соседнюю клетку
4. В зависимости от текущего состояния qi и прочитанного символа aj перейти в новое состояние qi’

Машиной Тьюринга называется совокупность <А, λ, Q, q1, δ>, где

А – Алфавит

λ – Пустые значения

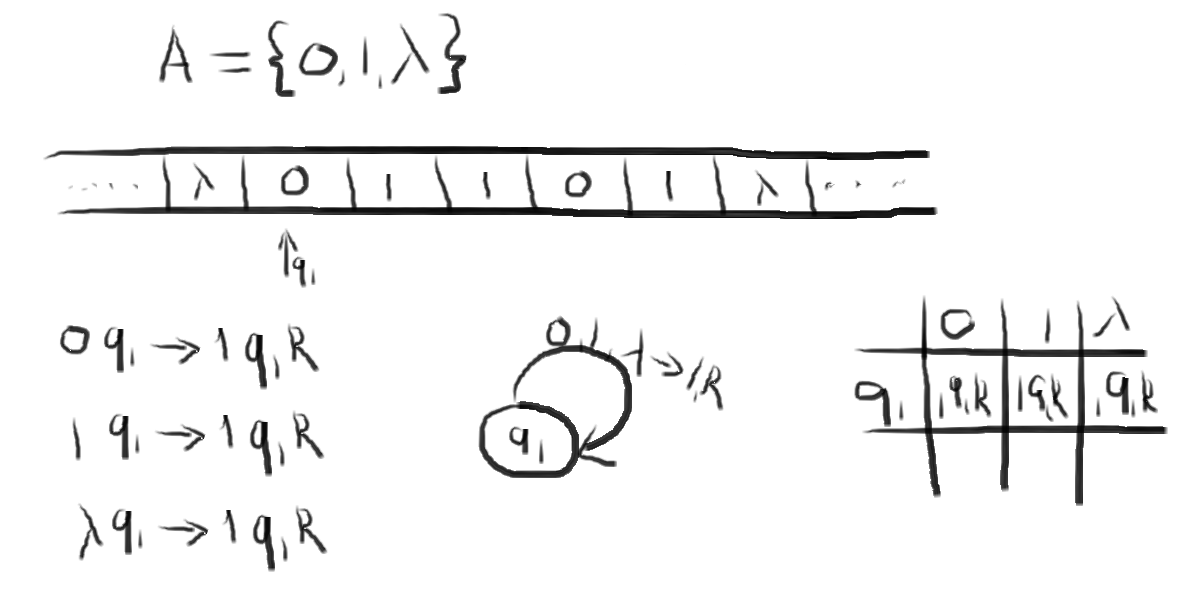
Q – Множество состояний

q1 – Первое состояние

δ – Множество команд

Пример: бесконечно двигаясь вправо заполнить всю ленту единицами.

Существует 3 способа представления алгоритма Тьюринга



Конфигурация (К) машины Тьюринга – К <α1, qi, α2>

α1 – слово, записанное слева от головки

α2 – слово, записанное справа от головки с учетом текущего символа

qi – текущее состоянии

Слева α1 от и справа α2 находятся пустые символы.

К каждой не заключительной конфигурации применима ровно 1 команда. Текущая конфигурация однозначно определяет все дальнейшее поведение машины Тьюринга.

K1 –> Ki –> Ki+1 –> …–> Kz

Начальной и заключительной конфигурациями считаются q1α и qzβ, где слева от головки МТ находится пустой символ. То есть ДМТ обозревает первый не пустой символ.

### Тезис Тьюринга

Всякий алгоритм может быть реализован ДМТ.

### ДМТ как черный ящик

ДМТ – это модель вычислителя, способного выполнить следующий набор элементарных операций:

{+,-\*,/,&,OR,XOR,WHILE,READ,WRITE,END}

### Функции правильно вычислимые по Тьюрингу

Будем считать, что алфавит A состоит из 3 алфавитов: исходного, промежуточного и конечного.

Промежуточный алфавит образует символы, которые могут появиться в процессе работы и слова, записанные на ленте, будут иметь вид словарных векторов.

α1\* α2 \* ... \* αn

Пусть

V – Это слово или словарный вектор на исходном алфавите

W – Это слово или словарный вектор на результатном алфавите

ДМТ правильно вычисляет функцию f, если:

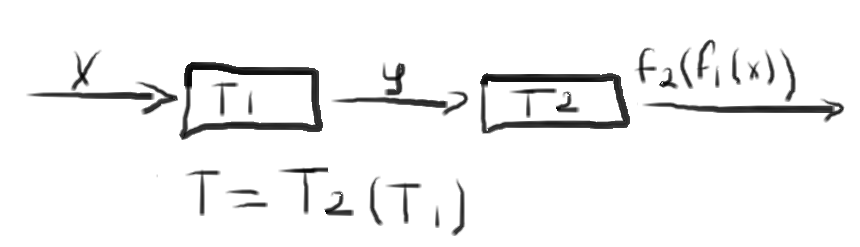
* 1. Для любых v ∈ V и w ∈ W – существует такая функция f(v) = w, то q1v –> qzw
  2. Если f(v) не определена, то ДМТ, запущенная из q1v будет работать бесконечно

Если ДМТ реализует функцию f, которая не определена в точке V, то на входном слове V ДМТ будет работать бесконечно.

### Операции над машинами Тьюринга

Композиция двух функций f1(x) и f2(y) – g(x) = f2(f1(x)). Чтобы g(x) была определена на данном x достаточно чтобы f1 была определена на x и f2 была определена на f1(x).

Теорема. Если f1(x) и f2(y) вычислимы по Тьюрингу, то их композиция так же вычислима по Тьюрингу.

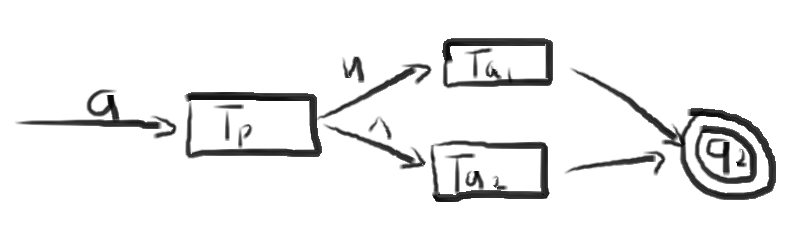


Типа кароче хошь налить супа – сначала приготовь его мудила, а потом наливай

f(a) является условием перехода к g1(a) или g2(a) условию p(a) ,если она задана следующим описанием. Если p(a) истинно, то g1(a), иначе g2(a).

Теорема 2:

Если g1(a) g2(a) и p(2) Вычислимы по Тьюрингу, то условный переход по p(a) так же вычислим по Тьюрингу.



Пример: построить машину, суммирующую любые слагаемые

Сумма, полученная на 1 слагаемом – это начальная сумма на 2 слагаемом.

Таким образом существует возможность построить ДМТ, реализующую любое элементарное действие из набора:

{+,-\*,/,&,OR,XOR,WHILE,READ,WRITE,END}

С помощью операции композиции мы можем строить какие-угодно сложные машины Тьюринга, используя в качестве блоков машины Тьюринга для элементарных действий.

Другими словами – для каждого алгоритма можно построить реализующую его ДМТ.

Можем ли мы считать, что у нас построено универсальное устройство – интерпретатор(ДМТ), способный выполнить любой алгоритм? – Нет.

### Универсальная машина Тьюринга или U–машина

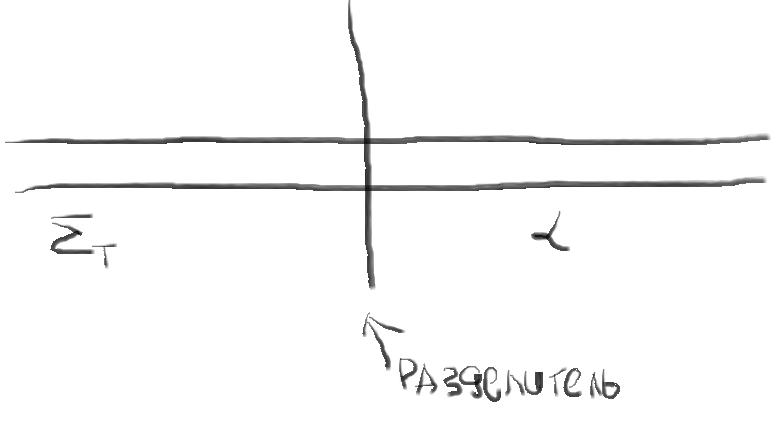
Согласно тезису Тьюринга – любой алгоритм может быть реализован ДМТ, то есть для любого алгоритма можно построить уникальную машину Тьюринга, которая будет выполнять только этот алгоритм.

Другой подход заключатся в создании универсальной машины Тьюринга или U–машины, которая выполняет функции интерпретатора. Для этого у нас должны быть хранимые, модифицируемые и выполняемые программы, а также устройство, которое способно эти программы способно выполнить.

Рассмотрим проблему построение универсального устройства интерпретатора в рамках модели ДМТ. Хранимая, модифицируемая и выполняемая программа – это система команд(aiqi–> ai’qi’αk) МТ. Входные данные для программы – это входное слово α. Результат работы машины Т на слове α – выходное слово Т(α), если машина Т останавливается на слове α.

Получаем, что нужно построить такую универсальную ДМТ, что для любой МТ Т системой команд(∑Т) и входным словом α мы получим следующее U(∑Т, α) = Т(α). Для машины Т входным словом является слово (α), а для машины U входным является система команд (∑Т) и входное слово (α).

Как на ленте U машины разместить входное слово этой U машины? Разделим ленту U машины на две бесконечные полу–ленты. В левой ленте запишем систем команд машины Т. В правой полу–ленте – входное слово.



Машина U читает текущий символ входного слова на правой полу–ленте и ищет подходящую команду в левой полу–ленте. Затем возвращается на правую полу–ленту и переписывает текущий символ, согласно прочитанной команде, а дальше переходит в новое состояние. В результате работы U машины на правой ленте будет записано слово Т(α).

Проблема, возникающая при построении U–машины связана с тем, что U–машина должна иметь фиксированный алфавит. Систему команд и входное слово α машины T нельзя переписать на ленту машины U потому что их алфавиты могут не совпадать. Выход заключается в том, чтобы символы из алфавитов машины Т и U кодировать словами алфавита U (0 и 1).

### Проблема остановки машины Тьюринга. Алгоритмически неразрешимые проблемы

Рассмотрим проблему разрешимости с точки зрения машин Тьюринга: Машина Тьюринга вычисляет значение некоторой функции натуральных аргументов. Каждая машина представляет собой некую функцию. Верно ли обратное – каждая ли функция вида f:N–>N может быть реализована некоторой машиной Тьюринга. Поскольку каждая машина Тьюринга однозначно задается своей программой, а программу можно записать как последовательность команд, то есть каждую последовательность команд можно записать как строчку конечной длины, в конечном алфавите, то мы имеем счётное количество строк.

То есть множество всех машин Тьюринга или всех алгоритмов – счётно. В тоже время количество функций f:N­–>N несчётно. Следовательно, существуют функции, которые нельзя вычислить за конечное число шагов с помощью алгоритма. Такие функции называются невычислимыми.

Таким образом алгоритмически неразрешимых задач гораздо больше, чем разрешимых – таких задач несчётное количество.

Если про задачу доказано, что не существует алгоритма, который решал бы эту задачу за конечное число шагов, то такая задача считается алгоритмически неразрешимой.

Рассмотрим одну такую задачу, которая называется проблема остановки машины Тьюринга.

Результативность – желательное свойство алгоритма, ДМТ на некоторых входных словах может работать бесконечно.

Построить вместе с универсальным интерпретатором универсальный отладчик. То есть такой алгоритм В, который бы позволял определить для любого алгоритма а и входного слова α – будет ли остановка А на α или он будет работать бесконечно.

В рамках модели ДМТ эта модель формулируется следующим образом: Построить МТ B, такую что для любой МТ А и входного слова α такую, что B(∑A, α) = Истина, если А остановится на α.

Теорема: универсального отладчика не существует, то есть не существует машины Тьюринга В, решающая проблему остановки для произвольной машины Тьюринга Т.

Смысл этой теоремы в том, что не существует одного универсального отладчика, способного отладить любую программу. Для каждого алгоритма проблема отладки должна решаться отдельно.

## Недетерминированная машина Тьюринга А. Классификация задач по степени сложности

### Вычислительная сложность алгоритмов

Сложность задачи – сложность наилучшего алгоритма, известного для ее решения.

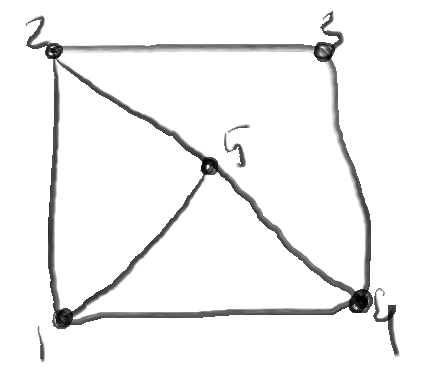
Сложность алгоритма – число элементарных шагов, необходимых для решения наихудшего из всех возможных случаев, допускающих применение алгоритма.

Число элементарных шагов выражается как функция от размерности входных данных задачи.

Эйлеров цикл – цикл, который проход через каждое число неориентированного графа 1 раз.

Для решения задачи о существовании в графе цикла Эйлера имеется алгоритм с вычислительной сложностью O(m), т.е. время его работы линейно зависит от числа ребер в графе.

Существует похожая задача в поиске Гамильтоново цикла. Гамильтонов цикл проходит через вершину графа только 1 раз.



До сих пор нет ни одного простого, необходимого и достаточного условия существования Гамильтоново цикла, и до сих пор не построен алгоритм, который проверял бы существование Гамильтоново цикла в произвольном графе на полиномиальное число шагов (число, ограниченное многочленом фиксированной степени от числа вершин графа). Идеальный случай, когда для решения задачи известна явная математическая формула, тогда сложность задачи не зависит от входных данных и является постоянным. Если же нет не явной формулы, ни рекурсивного выражения приемлемой сложности, то существует всего 2 способа.

1. Построение эффективного алгоритма
2. Перебор всех, или почти всех, вариантов

Асимптотическая сложность – сложность, которая показывает, как быстро растет число шагов алгоритма при неограниченном увеличении размерности входных данных. Если классифицировать задачи по степени сложности, то задачи об Эйлеровом цикле и о Гамильтоновом цикле попадают под разные классы.

### 3 класса задач

Вычислительная сложность алгоритма – число шагов, выполняемое алгоритмом в самом плохом случае.

Вычислительная сложность зависит от размеров входных данных.

Быстрыми или хорошими являются линейные алгоритмы порядка O(m), где m – размерность входных данных. Другие хорошие алгоритмы имеют полиномиальную сложность. Полиномиальные алгоритмы в литературе называются эффективными, а задачи, решаемые с помощью полиномиальных алгоритмов, относятся к классу P–полиномиальных алгоритмов.

Есть задачи, которые имеют экспоненциальную вычислительную сложность порядка а­n, где а – константа, или полином от n. К таким задачам относятся задача построения всех подмножеств данного множества. Всех перестановок данного множества, всех клик графа.

Кликой неориентированного графа называется подмножество его вершин, любые 2 из которых соединены ребром. Число ответов в этих задачах уже само по себе экспоненциально, поэтому их перечисление требует экспоненциального числа шагов. Такой класс задач называет E или класс экспоненциальных алгоритмов. При n > 20 мы запрашиваем больше информации, чем в состоянии использовать.

Остаются задачи, которые не попали ни в класс P, ни в класс E (NP–полные задачи). В условиях этих задач не содержится экспоненциальные множества, а также для их решения не найден полиномиальный алгоритм. Но и не доказано, что этого алгоритма не существует. К таким задачам относятся задача поиска Гамильтоново цикла. Задача об оптимальном раскрое. Задача об оптимальной загрузке ёмкости. Оптимизация пути Коммивояжёра через сеть городов и т.д. Эти задачи являются модельным и каждой из них соответствует несколько реальных формулировок: размещение персонала, упорядочивание пересовала, управление производством и т.д.

Для большинства этих задач (так называемых NP–полных) была доказана эквивалентность: если для одной из них удастся найти полиномиальный алгоритм автоматически будут решены все остальные.

Если объединить эти задачи с классом P, то получим класс NP–недетерминированных полиномиальных алгоритмов.

Задача называется легко решаемой, если решается за полиномиальное время. Для трудно решаемых задач не известны полиномиальные алгоритмы.

### Класс P детерминированных полиномиальных алгоритмов

Класс P–детерминированных полиномиальных алгоритмов и класс NP–недетерминированных полиномиальных алгоритмов изучается на примере задачи распознавания П.

Формулировка задачи: Дано А, верно ли, что для А выполняется свойство X. Стандартным решение является решение ДА или НЕТ.

Iп (индивидуальная задача распознавания) – содержит один набор исходных данных.

Пример: Найти Гамильтонов цикл минимальной стоимости. Существует ли маршрут стоимости, не превосходящей B (граница стоимости маршрута), проходящий через все города 1 раз и возвращающийся в исходный.

Среди бесконечного множества задач Iп выделяется класс задач Yп (индивидуальные задачи, имеющие ответ да). Для решения таких задач – считаем, что у машины Тьюринга есть 2 состояния qy, qn.

Рассмотрим процесс решения задач P на ДМТ T. Входом для программы является слово α. ДМТ производит вычисление шаг за шагом, согласно своей системе команд. Если текущее состояние ДМТ – qy, то ответ да и процесс вычислений заканчивается, а если qn – то процесс вычислений заканчивается с ответом нет.

Время, требуемое ДМТ программой Т для вычисления на входе α – это число шагов, выполняемой ДМТ до момента остановки.

Если ДМТ программа Т останавливается на всех входах α, то временная сложность ДМТ программы Т можно описать следующим определением.

Время, которое требуется ДМТ программе T для вычисления на входе α такое:

T(n)=max{Существует такое слово α, абсолютное значение которого = n такое, что вычисление по программе T на входе α требует k–шагов}.

Время вычислений является функцией от размерности входных данных. Взятие максимума по времени вычислений на всех словах длины n соответствует “самому плохому случаю”.

ДМТ называется полиномиальной ДМТ программой, если существует такой полином P, что для всех положительных целых n выполняется условие T(n) <= p(n).

Любая задача распознавания П из класса P – решается полиномиальной ДМТ программой.

## Переборные методы решения NP–полных задач. Алгоритм с возвратом

Для небольших N (N <= 20) экспоненциальный алгоритм более эффективен, чем полиномиальный, если задача имеет большую размерность, то следует разработать более приближенный или эвристический полиномиальный алгоритм.

### Общая схема алгоритма с возвратом

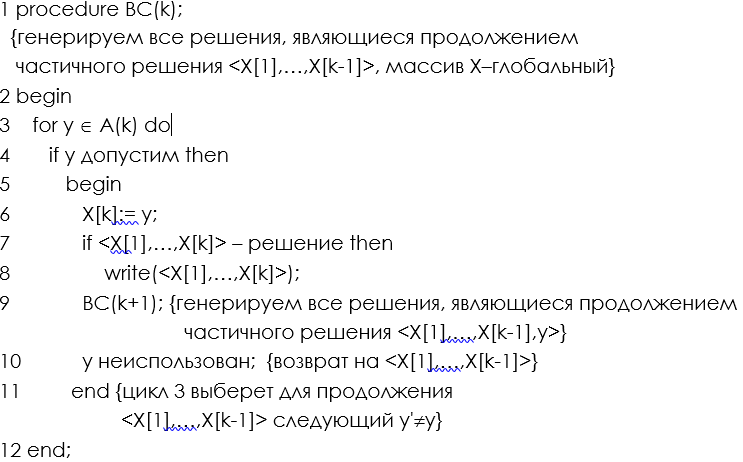
Рассмотрим общую схему на примере задачи существования Гамильтоново цикла. Произведем полный перебор всех возможных путей на графе. Если на графе N вершин, по нам потребуется N! возможных вариантов. При этом для каждой цепочки проверка существования гамильтоново цикла требует еще n шагов. Поэтому сложность алгоритма полного перебора = N \* N!. Пусть решение, которое мы ищем имеет вид последовательности <X(1),…,X(n)> будем строить его, начиная с пустой последовательности 0 длины. Пусть на каком–то этапе уже построено частичное решение длины i <X(1),…,X(i)>, попытаемся продолжить наше решение на 1 шаг, для этого нужно найти допустимое X(i+1). X(i+1) является допустимым если:

1. Или X(1)…X(i+1) уже является решением, тогда X(i+1) будет допустимым.
2. Или относительно цепочки X(1)…X(i+1) нельзя сказать, что его невозможно расширить до полного решения.

Если допустимое X(i+1) существует, то пытаемся продолжить (вызываем рекурсию для частичного решения X(1)…X(i+1)). Если допустимого X(i+1) не существует, то делаем возврат (возвращаемся на шаг назад к частичному решению X(1)…X(i-1) и для него отыскиваем другое X(i) не совпадающее с предыдущим X(i)).

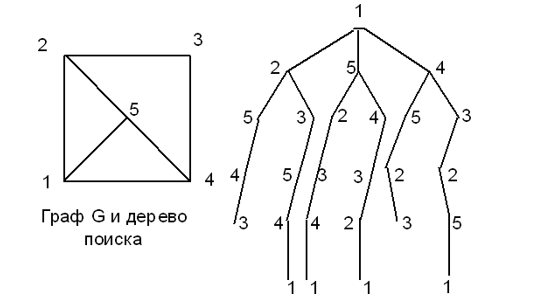
Более точно, пусть для каждого *i* > 0 существует множество A(*i*), из которого будут выбираться X(*i*) – претенденты на заполнение *i*–й позиции частичного решения.

Очевидно, множества A(*i*) должны содержать все X(*i*), занимающие *i*-ю позицию любого решения. Кроме того, A(*i*) всегда будут содержать какие-то лишние элементы, не содержащие в *i*–й координате ни одного решения.



### Задача Гамильтона

Применим общую схему алгоритма с возвратом для генерации всех гамильтоновых циклов в графе G=<V,E>. Каждый такой цикл – последовательность различных вершин графа <X(1), …, X(n+1)> и только X(1) = X(n+1) = *k*, где *k* – произвольная фиксированная вершина; соседние вершины X(*i*) и X(i+1) соединены ребром. A(*i*) = *V* (множество всех вершин) для любого *i*  n+1.



### Условие применимости модифицированной схемы для решения задачи на минимум

1. Припишем каждому решению (и полному, и частичному) стоимость – ее еще называют целевой функцией. Будем обозначать целевую функцию стоимости *cost*.
2. При продолжении решения его стоимость может только увеличиваться: *cost*(<X[1], … , X[*i* –1]>)  *cost*(<X[1], … , X[i-1], *X*[*i*]>) для любого *i*  2.

Для задачи коммивояжера целевой функцией будет стоимость маршрута

*n*

cos*t*(*X* [1],..., *X* [*n*])   *A*[ *X* [*i*  1], *X* [*i*]].

*i*2

### Алгоритм ветвей и границ

1. Находим первое решение и запоминаем его в *OptX*, а его стоимость в *OptCost*.
2. Пусть <X[1], …, X[*i*]> – текущее частичное решение. Найдем допустимое X[*i*+1]. Если *cost*(<X[1], … , X[*i*], X[*i*+1]>)  *OptCost*, то любое его продолжение будет заведомо больше текущего минимального решения. Отбрасываем X[*i*+1] как недопустимое и ищем новое X’[*i*+1].

Таким образом, произведя «досрочный» возврат, мы избавляем себя от генерации всех заведомо неоптимальных продолжений, т.е. обрубаем целое поддерево на дереве поиска.

### Алгоритм с возвратом для решения задач на максимум. Принцип включения\не включения.

Рассмотрим принцип включения\не включения на примере задачи о рюкзаке.

Условие. Заданы конечное множество *A*, положительные целые веса w(*a*), стоимости s(*a*) для каждого *a* ϵ *A* и общее ограничение на вес *K*.

***Вопрос.*** Найти из всевозможных выборок *A'*  *A* такую, чтобы суммарный вес входящих в него элементов не превосходил *K*, а суммарная стоимость была максимальна.

Задача о рюкзаке – запдача об определении оптимальной выборки из *N* объектов, подчиненной некоторому ограничению. Поскольку из *N* объектов возможно сделать 2*N* различных выборок, для решения подобной задачи с помощью алгоритма с возвратом необходимо очень сильное ограничение.

***Вопрос*.** Изменим стоимости предметов на противоположные. Выборка максимальной стоимости превратилась в выборку минимальной стоимости. Почему нельзя эту задачу решить алгоритмом для решения задач на минимум, изложенным в предыдущем разделе?

Однако есть способ значительно сократить количество переборов – в процессе получения возможных решений помнить лучшее из полученных на данный момент решений и не пытаться генерировать те решения, которые будут заведомо хуже известного на данный момент оптимального решения.

Принцип в процессе генерации возможных решений запоминает максимальное из полученных на данный момент и не пытается генерировать те решения, которые будут заведомо хуже текущего максимального.

Каждый *i*-й объект проверяется на допустимость (согласно нашему ограничению). Затем внутри одной процедуры допустимый объект вначале включается в выборку, и с этим включенным объектом рекурсивно генерируются всевозможные решения и лучшее из них запоминается как оптимальное. После этого происходит возврат и *i*-й объект удаляется из выборки.

Классический алгоритм с возвратом стал бы рекурсивно генерировать всевозможные решения без *i*-го объекта – принцип включения\не включения действует иначе.

Прежде, чем пытаться получать решения без *i*-го объекта, проверим, можно ли, не включив этот объект, получить решение лучшее, чем найденное оптимальное с *i*-м объектом. Если нет – то не стоит и пытаться.

Как проверить возможность получения без *i*-го объекта более ценной выборки, чем текущая оптимальная, если мы не собираемся сразу строить эти выборки?

Нужно посчитать стоимость, которую мы, **может быть**, наберем без *i*-го объекта. Для этого просуммируем стоимости всех предметов, **уже вошедших в выборку**, и стоимости всех **еще неисследованных предметов** (разумеется, без *i*-го объекта).

Если эта стоимость, которую, может быть, удастся набрать (а, может, и не удастся – ведь неисследованные объекты могут не пройти ограничение по весу!) будет больше оптимальной – есть смысл генерировать решения без *i*-го объекта.

Опишем рекурсивную процедуру *TRY* с включением\не включением.

