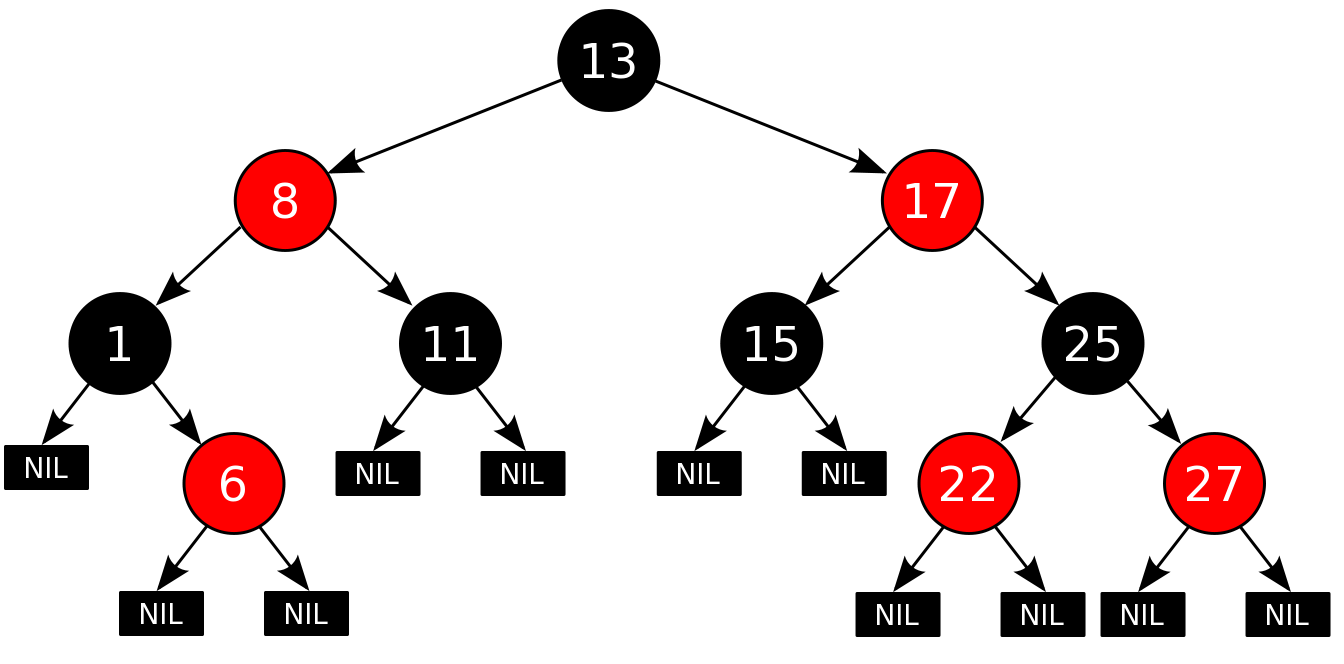
46. Понятие красно-черного дерева. Сравнение АВЛ- и красно-черных деревьев.

Красно-черные деревья: коротко и ясно  
Итак, сегодня хочу немного рассказать о красно-черных деревьях. Рассказ будет кратким, без рассмотрения алгоритмов балансировки при вставке/удалении элементов в красно-черных деревьях.  
  
  
Красно-черные деревья относятся к сбалансированным бинарным деревьям поиска.

Как бинарное дерево, красно-черное обладает свойствами:

1) Оба поддерева являются бинарными деревьями поиска.  
  
2) Для каждого узла с ключом k выполняется критерий упорядочения:

ключи всех левых потомков <= k < ключи всех правых потомков  
(в других определениях дубликаты должны располагаться с правой стороны либо вообще отсутствовать).   
Это неравенство должно быть истинным для всех потомков узла, а не только его дочерних узлов.

Свойства красно-черных деревьев:

1) Каждый узел окрашен либо в красный, либо в черный цвет (в структуре данных узла появляется дополнительное поле – бит цвета).  
  
2) Корень окрашен в черный цвет.  
  
3) Листья(так называемые NULL-узлы) окрашены в черный цвет.  
  
4) Каждый красный узел должен иметь два черных дочерних узла. Нужно отметить, что у черного узла могут быть черные дочерние узлы. Красные узлы в качестве дочерних могут иметь только черные.  
  
5) Пути от узла к его листьям должны содержать одинаковое количество черных узлов(это черная высота).

Ну и почему такое дерево является сбалансированным?

Действительно, красно-черные деревья не гарантируют строгой сбалансированности (разница высот двух поддеревьев любого узла не должна превышать 1), как в [АВЛ-деревьях](https://ru.wikipedia.org/wiki/%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE). Но соблюдение свойств красно-черного дерева позволяет обеспечить выполнение операций вставки, удаления и выборки за время O(logN). И сейчас посмотрим, действительно ли это так.  
  
Пусть у нас есть красно-черное дерево. Черная высота равна bh(black height).  
  
Если путь от корневого узла до листового содержит минимальное количество красных узлов (т.е. ноль), значит этот путь равен bh.  
  
Если же путь содержит максимальное количество красных узлов (bh в соответствии со свойством 4), то этот путь будет равен 2bh.  
  
То есть, пути из корня к листьям могут различаться не более, чем вдвое (h<=2log(N+1), где h — высота поддерева), этого достаточно, чтобы время выполнения операций в таком дереве было O(logN)

Сравнение АВЛ- и красно-черных деревьев:

1) По высоте: при одном кол-ве листьев, КЧД/АВЛ <= 1,4

2) Поиск узла: в КЧД дольше на 39-40%

3) Вставка: До двух поворотов, в КЧД может и больше

4) Удаление: из КЧД до 3 поворотов. Для удаления элемента из АВЛ дерева может потребоваться log(n)

5) Память: в АВЛ дополнительно нужно хранить высоту, а в КЧТ цвет.

С точки зрения балансировки одинаковы.

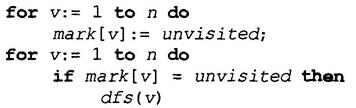
52. Обход ориентированных графов поиском в глубину. Глубинный остовный лес и типы дуг на нем.

При решении многих задач, связанных с ориентированными графами, необходим эффективный метод систематического обхода вершин и дуг орграфов. Таким методом является *поиск в глубину*– обобщение метода обхода дерева в прямом порядке.

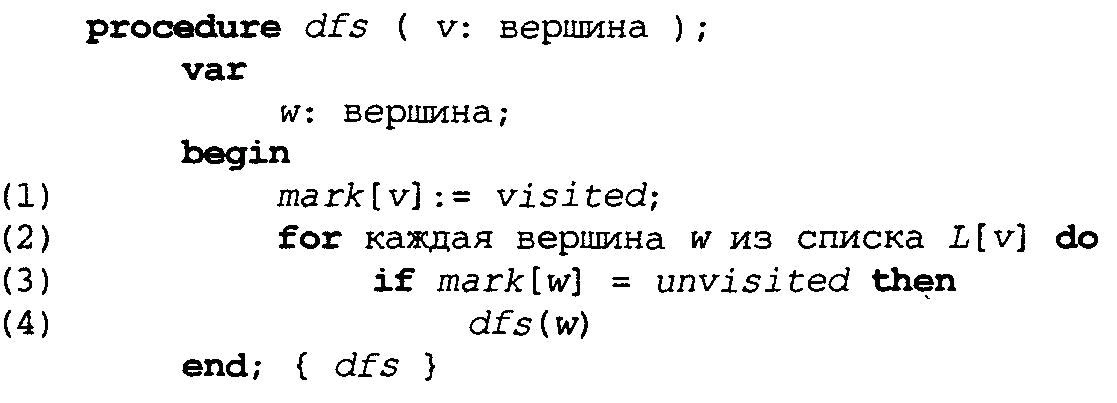
       Предположим, что есть ориентированный граф *G*, в котором первоначально все вершины помечены меткой *unvisited* (не посещались). Поиск в глубину начинается с выбора начальной вершины *v* графа *G*, для этой вершины метка *unvisited* меняется на метку *visited*(посещалась). Затем для каждой вершины, смежной с вершиной *v* и которая не посещалась ранее, рекурсивно применяется поиск в глубину. Когда все вершины, которые можно достичь из вершины *v*, будут посещены, поиск заканчивается. Если некоторые вершины не были посещены, то выбирается одна из них и поиск повторяется. Этот процесс продолжается до тех пор, пока обходом не будут охвачены все вершины орграфа *G*.

       Этот метод обхода вершин орграфа называется поиском в глубину, поскольку поиск непосещенных вершин идет в направлении вперед (вглубь) до тех пор, пока это возможно. Например, пусть *х* – последняя посещенная вершина. Для продолжения процесса выбирается какая-либо нерассмотренная дуга  выходящая из вершины *х*. Если вершина *y* ранее посещалась, то она помечается меткой *visited* и поиск начинается заново от вершины *y*. После того, как пройдены все пути, начинающиеся в вершине *y*, происходит возврат в вершину *х*, т.е. туда, откуда впервые была достигнута вершина *y*. Затем продолжается выбор нерассмотренных дуг, исходящих из вершины *x*, пока не будут исчерпаны все такие дуги.

       Для представления вершин, смежных с вершиной*v*, используется список смежности *L[v]*, а для определения вершин, которые ранее посещались, –   массив*mark*, чьи элементы будут принимать только два значения: *visited*и*unvisited*.  Чтобы применить процедуру поиска в глубину к графу, состоящему из *n* вершин, надо сначала присвоить всем элементам массива *mark*  значения *unvisited*, затем начать поиск в глубину для каждой вершины, помеченной как *unvisited*. Это можно реализовать с помощью следующего кода:



Эскиз рекурсивной процедуры *dfs*, реализующей метод поиска в глубину, представлен ниже. Здесь изменяются только значения массива *mark.*



       Рассмотрим метод поиска в глубину для графа, изображенного на рисунке 18.5., начиная с вершины *А*.

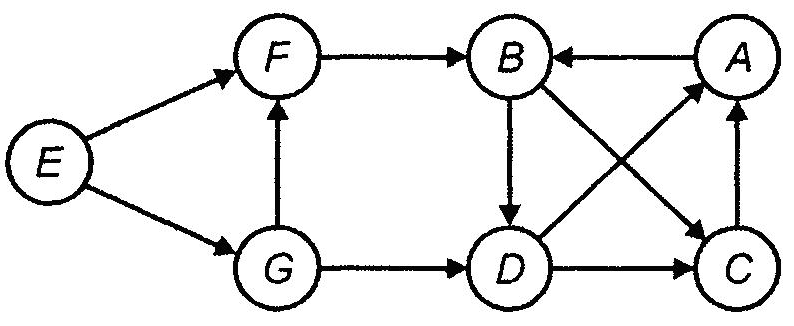


Рисунок 18.5 – Ориентированный граф

       Алгоритм помечает вершину*А* как *visited*и выбирает вершину *В* из списка смежности вершины *А*. Поскольку вершина В помечена как *unvisited*, обход графа продолжается вызовом процедуры  *dfs(B)*. Затем процедура помечает вершину *В* как *visited*и выбирает первую вершину из списка смежности вершины *В*. В зависимости от порядка представления вершин в списке смежности следующей рассматриваемой вершиной может быть или вершина *С*, или вершина *D*. Предположим, что вершина *С* предшествует вершине *D*. Тогда происходит вызов *dfs(С)*. В списке смежности вершины *С* присутствует только вершина *А*, но она уже посещалась ранее. Поскольку все вершины в списке смежности вершины *С* исчерпаны, то поиск возвращается в вершину *В*, откуда процесс поиска продолжается вызовом процедуры *dfs(D)*. Вершины *А* и *С* из списка смежности вершины *D* уже посещались ранее, поэтому поиск возвращается сначала в вершину *В*, а затем в вершину *А*. На этом первоначальный вызов *dfs(А)* завершен, но орграф имеет вершины, которые еще не посещались: *E, F, G*. Для продолжения обхода вершин графа выполняется вызов *dfs(E).*

### Глубинный остовный лес

       В процессе обхода ориентированного графа методом поиска в глубину только определенные дуги ведут к вершинам, которые ранее не посещались. Такие дуги, ведущие к новым вершинам, называются *дугами дерева* и формируют для данного графа *глубинный остовный лес*, построенный методом поиска в глубину. На рисунке 18.6 показан глубинный остовный лес для графа из рисунка 18.5. На рисунке леса сплошными линиями обозначены дуги дерева. Именно они формируют лес, поскольку методом поиска в глубину к любой ранее не посещавшейся вершине можно придти только по одной дуге.

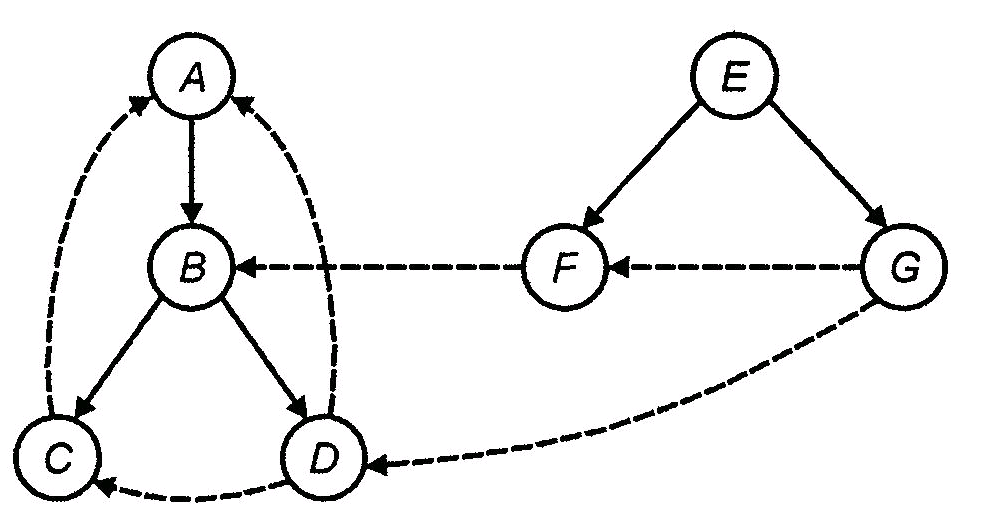


Рисунок 18.6 – Глубинный остовный лес

       Кроме дуг дерева, существуют еще три типа дуг, определяемых в процессе обхода орграфа методом поиска в глубину. Это обратные, прямые и поперечные дуги. *Обратные дуги*, как дуга *C→A*. Они в остовном лесу идут от потомков к предкам. Дуга из вершины в саму себя также является обратной дугой. *Прямыми дугами* называются дуги, идущие от предков к истинным потомкам, но которые не являются дугами дерева. На рисунке  18.6 прямые дуги отсутствуют.

       Дуги, таки как *D→C* и *G→D*, соединяющие вершины, не являющиеся ни предками, ни потомками друг друга, называются *поперечными дугами.* Если при построении остовного дерева сыновья одной вершины располагаются слева направо, то все поперечные дуги идут справа налево. Такое расположение вершин и деревьев помогает формировать остовный лес.

       Дуги дерева легко отличить от других, т.к. они получаются в процессе обхода графа как дуги, ведущие к тем вершинам, которые ранее не посещались. Если *v*– вершина орграфа, то всем потомкам вершины*v* присваиваются глубинные номера, не меньшие, чем номер, присвоенный вершине *v*. Фактически вершина *w* будет потомком вершины *v* тогда и только тогда, когда выполняются неравенства:

*dfnumber(v) dfnumber(w)* *dfnumber*(*v*)+количество потомков вершины *v*.

       Очевидно, что прямые дуги идут от вершин с низкими номерами к вершинам с более высокими номерами, а обратные и поперечные дуги – от вершин с высокими номерами к вершинам с более низкими номерами.

53. Понятие сильной связности ориентированного графа. Нахождение компонент сильной связности с помощью поиска в глубину.

[Орграф](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B3%D1%80%D0%B0%D1%84) называется ***сильно связным*** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *strongly connected*), если любые две его вершины сильно связны. Две вершины s и t любого графа сильно связны, если существует ориентированный путь из s в t и ориентированный путь из t в s. **Компонентами сильной связности** орграфа называются его максимальные по включению сильно связные подграфы. **Областью сильной связности** называется множество вершин компоненты сильной связности.

## **Определения**

Орграф, не принадлежащий к классу сильно связных графов, содержит некоторый набор **сильно связных компонент**, и некоторый набор ориентированных ребер, идущих от одной компоненты к другой.

Любая вершина орграфа сильно связна сама с собой.

## **Алгоритмы**

Простейший алгоритм решения задачи о поиске сильно связных компонент в орграфе работает следующим образом:

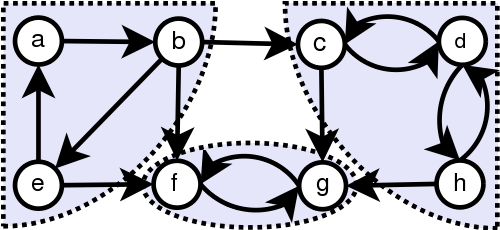
1. При помощи [транзитивного замыкания](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B7%D0%B0%D0%BC%D1%8B%D0%BA%D0%B0%D0%BD%D0%B8%D0%B5) проверяем, достижима ли t из s, и s из t, для всех пар s и t.
2. Затем определяем такой [неориентированный граф](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)#.D0.93.D1.80.D0.B0.D1.84), в котором для каждой такой пары содержится ребро.
3. Поиск компонент связности такого неориентированного графа даст нам компоненты сильной связности нашего орграфа.

Очевидно основное время работы данного алгоритма приходится на реализацию транзитивного замыкания.

Также существует три алгоритма, решающих данную задачу за линейное время, то есть в V раз быстрее, чем приведенный выше алгоритм. Это алгоритмы [Косарайю](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%BE%D1%81%D0%B0%D1%80%D0%B0%D0%B9%D1%8E" \o "Алгоритм Косарайю), [Габова](https://en.wikipedia.org/wiki/Gabow%27s_algorithm) и [Тарьяна](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A2%D0%B0%D1%80%D1%8C%D1%8F%D0%BD%D0%B0" \o "Алгоритм Тарьяна).

## **Пример**

На данном примере изображен орграф, для которого найдены все три компоненты сильной связности (закрашенные области, обведенные пунктиром).

[](https://commons.wikimedia.org/wiki/File:Scc.png?uselang=ru)

[Компоненты сильной связности](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%B2%D1%8F%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D0%B8,_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D1%8B_%D1%81%D0%B2%D1%8F%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D0%B8#.D0.A1.D0.B8.D0.BB.D1.8C.D0.BD.D0.B0.D1.8F_.D1.81.D0.B2.D1.8F.D0.B7.D0.BD.D0.BE.D1.81.D1.82.D1.8C) в графе G можно найти с помощью [поиска в глубину](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D0%B1%D1%85%D0%BE%D0%B4_%D0%B2_%D0%B3%D0%BB%D1%83%D0%B1%D0%B8%D0%BD%D1%83,_%D1%86%D0%B2%D0%B5%D1%82%D0%B0_%D0%B2%D0%B5%D1%80%D1%88%D0%B8%D0%BD) в 3 этапа:

1. Построить граф H с обратными (инвертированными) рёбрами
2. Выполнить в H поиск в глубину и найти f[u] — время окончания обработки вершины u
3. Выполнить поиск в глубину в G, перебирая вершины во внешнем цикле в порядке убывания f[u]

Полученные на 3-ем этапе деревья поиска в глубину будут являться компонентами сильной связности графа G.  
Так как компоненты сильной связности G и H графа совпадают, то первый поиск в глубину для нахождения f[u] можно выполнить на графе G, а второй — на H.

**Общая идея**

Общая идея алгоритма состоит в следующем: для каждой *не пройденной* [вершины](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) необходимо найти все *не пройденные* [смежные вершины](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) и повторить поиск для них.

**Пошаговое представление**

1. Выбираем любую вершину из еще *не пройденных*, обозначим ее как u.
2. Запускаем процедуру dfs(u)
   * Помечаем вершину uu как *пройденную*
   * Для каждой *не пройденной* смежной с u вершиной (назовем ее v) запускаем dfs(v)
3. Повторяем шаги 1 и 2, пока все вершины не окажутся *пройденными*.

### Реализация

В массиве visited[]хранится информация о *пройденных* и *не пройденных* вершинах.

**function** doDfs(G[n]: **Graph**): // функция принимает граф G с количеством вершин n и выполняет обход в глубину во всем графе

visited = array[n, *false*] // создаём массив посещённых вершины длины n, заполненный *false* изначально

**function** dfs(u: **int**):

visited[u] = *true*

**for** v: (u, v) **in** G

**if** **not** visited[v]

dfs(v)

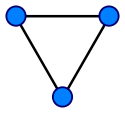
**for** i = 1 **to** n

**if** **not** visited[i]

dfs(i)

54. Основные определения неориентированных графов. Связная компонента неориентированного графа. Матрица смежности.

**Граф**



**Граф** или **неориентированный граф** *G* — это [упорядоченная пара](https://dic.academic.ru/dic.nsf/ruwiki/1157219) *G*: = (*V*,*E*), для которой выполнены следующие условия:

* *V* это [множество](https://dic.academic.ru/dic.nsf/ruwiki/4759) **вершин** или **узлов**,
* *E* это множество пар (в случае неориентированного графа — неупорядоченных) различных вершин, называемых **рёбрами**.

Вершины и рёбра графа называются также **элементами** графа, число вершин в графе | *V* |  — **порядком**, число рёбер | *E* |  — **размером** графа.

Вершины *u* и *v* называются **концевыми** вершинами (или просто **концами**) ребра *e* = {*u*,*v*}. Ребро, в своюочередь, **соединяет** эти вершины. Две концевые вершины одного и того же ребра называются **соседними**.

Два ребра называются **смежными**, если они имеют общую концевую вершину.

Два ребра называются **кратными**, если множества их концевых вершин совпадают.

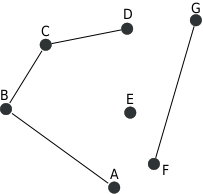
Ребро называется **петлёй**, если его концы совпадают, то есть *e* = {*v*,*v*}.

**Степенью** deg*V* вершины *V* называют количество рёбер, для которых она является концевой (при этом петлисчитают дважды).

Вершина называется **изолированной**, если она не является концом ни для одного ребра; **висячей** (или**листом**), если она является концом ровно одного ребра.

**Компонента связности графа {\displaystyle G}** (или просто **компонента графа** {\displaystyle G}) — максимальный (по включению) [связный](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) [подграф](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%B3%D1%80%D0%B0%D1%84) [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) {\displaystyle G}.

Другими словами, это подграф {\displaystyle G(U)}, [порождённый](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D0%BE%D0%B4%D0%B3%D1%80%D0%B0%D1%84) множеством {\displaystyle U\subseteq V(G)} вершин, в котором для любой пары вершин {\displaystyle u,v\in U} в графе {\displaystyle G} существует {\displaystyle (u,v)}-цепь и для любой пары вершин {\displaystyle u\in U}, {\displaystyle w\notin U} не существует {\displaystyle (u,w)}-[цепи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BF%D1%8C_%D0%B2_%D0%B3%D1%80%D0%B0%D1%84%D0%B5).

[](https://commons.wikimedia.org/wiki/File:Unconnected-graph.svg?uselang=ru)

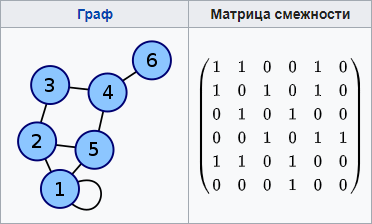
Несвязный граф с тремя компонентами связности

**Матрица смежности** графа *G* с конечным числом вершин *n* (пронумерованных числами от 1 до *n*) — это **квадратная** матрица *A* размера *n*, в которой значение элемента *aij* равно числу рёбер из *i*-й вершины графа в *j*-ю вершину.

Иногда, особенно в случае [неориентированного](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) графа, петля (ребро из *i*-й вершины в саму себя) считается за два ребра, то есть значение диагонального элемента *aii* в этом случае равно удвоенному числу петель вокруг *i*-й вершины.

Матрица смежности [простого графа](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) (не содержащего петель и кратных рёбер) является [бинарной матрицей](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0) и содержит нули на главной диагонали.

Ниже приведён пример **неориентированного графа** и соответствующей ему **матрицы смежности *A***. Этот граф содержит петлю вокруг вершины 1, при этом в зависимости от конкретных приложений элемент{\displaystyle a\_{11}} может считаться равным либо одному (как показано ниже), либо двум.



55. Построение минимального остовного дерева неориентированного графа с помощью алгоритма Прима.

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Изображение** | **Множество выбранных вершин U** | **Ребро (u, v)** | **Множество невыбранных вершин V \ U** | **Описание** |
|  | {} |  | {A,B,C,D,E,F,G} | Исходный взвешенный граф. Числа возле ребер показывают их веса, которые можно рассматривать как расстояния между вершинами. |
|  | {D} | (D,A) = 5 **V**  (D,B) = 9 (D,E) = 15 (D,F) = 6 | {A,B,C,E,F,G} | В качестве начальной произвольно выбирается вершина **D**. Каждая из вершин **A**, **B**, **E** и **F** соединена с **D** единственным ребром. Вершина **A** — ближайшая к **D**, и выбирается как вторая вершина вместе с ребром **AD**. |
|  | {A,D} | (D,B) = 9 (D,E) = 15 (D,F) = 6 **V** (A,B) = 7 | {B,C,E,F,G} | Следующая вершина — ближайшая к любой из выбранных вершин **D** или **A**. **B** удалена от **D** на 9 и от **A** — на 7. Расстояние до **E** равно 15, а до **F** — 6. **F** является ближайшей вершиной, поэтому она включается в дерево **F** вместе с ребром **DF**. |
|  | {A,D,F} | (D,B) = 9 (D,E) = 15 (A,B) = 7 **V** (F,E) = 8 (F,G) = 11 | {B,C,E,G} | Аналогичным образом выбирается вершина **B**, удаленная от **A** на 7. |
|  | {A,B,D,F} | (B,C) = 8 (B,E) = 7 **V** (D,B) = 9 цикл (D,E) = 15 (F,E) = 8 (F,G) = 11 | {C,E,G} | В этом случае есть возможность выбрать либо **C**, либо **E**, либо **G**. **C** удалена от **B** на 8, **E** удалена от **B** на 7, а **G** удалена от **F** на 11. **E** — ближайшая вершина, поэтому выбирается **E** и ребро **BE**. |
|  | {A,B,D,E,F} | (B,C) = 8 (D,B) = 9 цикл (D,E) = 15 цикл (E,C) = 5 **V** (E,G) = 9 (F,E) = 8 цикл (F,G) = 11 | {C,G} | Здесь доступны только вершины **C** и **G**. Расстояние от **E** до **C** равно 5, а до **G** — 9. Выбирается вершина **C** и ребро **EC**. |
|  | {A,B,C,D,E,F} | (B,C) = 8 цикл (D,B) = 9 цикл (D,E) = 15 цикл (E,G) = 9 **V** (F,E) = 8 цикл (F,G) = 11 | {G} | Единственная оставшаяся вершина — **G**. Расстояние от **F** до неё равно 11, от **E** — 9. **E** ближе, поэтому выбирается вершина **G** и ребро **EG**. |
|  | {A,B,C,D,E,F,G} | (B,C) = 8 цикл (D,B) = 9 цикл (D,E) = 15 цикл (F,E) = 8 цикл (F,G) = 11 цикл | {} | Выбраны все вершины, [минимальное остовное дерево](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) построено (выделено зелёным). В этом случае его вес равен 39. |

56. Обход неориентированного графа поиском в глубину.

***Поиск в глубину*** – это алгоритм обхода вершин графа.  
  
Поиск в ширину производится симметрично (вершины графа просматривались по уровням). Поиск в глубину предполагает продвижение вглубь до тех пор, пока это возможно. Невозможность продвижения означает, что следующим шагом будет переход на последний, имеющий несколько вариантов движения (один из которых исследован полностью), ранее посещенный узел (вершина).  
  
Отсутствие последнего свидетельствует об одной из двух возможных ситуаций:

* все вершины графа уже просмотрены,
* просмотрены вершины доступные из вершины, взятой в качестве начальной, но не все (несвязные и ориентированные графы допускают последний вариант).
* **Применения алгоритма поиска в глубину**
* Поиск любого пути в графе.
* Поиск лексикографически первого пути в графе.
* Проверка, является ли одна вершина дерева предком другой.
* Поиск наименьшего общего предка.
* Топологическая сортировка.
* Поиск компонент связности.

Алгоритм поиска в глубину работает как на ориентированных, так и на неориентированных графах. Применимость алгоритма зависит от конкретной задачи.  
Для реализации алгоритма удобно использовать стек или рекурсию.

57. Обход неориентированного графа поиском в ширину.

***Поиск в ширину*** подразумевает поуровневое исследование графа:

* вначале посещается корень – произвольно выбранный узел,
* затем – все потомки данного узла,
* после этого посещаются потомки потомков и т.д.

Вершины просматриваются в порядке возрастания их расстояния от корня.  
Алгоритм прекращает свою работу после обхода всех вершин графа, либо в случае выполнения требуемого условия (например, найти кратчайший путь из вершины 1 в вершину 6).

**Применения алгоритма поиска в ширину**

* Поиск кратчайшего пути в невзвешенном графе (ориентированном или неориентированном).
* Поиск компонент связности.
* Нахождения решения какой-либо задачи (игры) с наименьшим числом ходов.
* Найти все рёбра, лежащие на каком-либо кратчайшем пути между заданной парой вершин.
* Найти все вершины, лежащие на каком-либо кратчайшем пути между заданной парой вершин.

Алгоритм поиска в ширину работает как на ориентированных, так и на неориентированных графах.  
Для реализации алгоритма удобно использовать очередь.

58. Иерархические системы классификации данных. Их особенности, достоинства и недостатки.

Иерархическая система классификации строится следующим образом:

* исходное множество элементов составляет 0-й уровень и делится в зависимости от выбранного классификационного признака на классы (группировки), которые образуют 1-й уровень;
* каждый класс 1-го уровня в соответствии со своим, характерным для него классификационным признаком делится на подклассы, которые образуют 2-й уровень;
* каждый класс 2-го уровня аналогично делится на группы, которые образуют 3-й уровень, и т.д.

Учитывая достаточно жесткую процедуру построения структуры классификации, необходимо перед началом работы определить ее цель, т.е. какими свойствами должны обладать объединяемые в классы объекты. Эти свойства принимаются в дальнейшем за признаки классификации.

В иерархической системе классификации каждый объект на любом уровне должен быть отнесен к одному классу, который характеризуется конкретным значением выбранного классификационного признака. Для последующей группировки в каждом новом классе необходимо задать свои классификационные признаки и их значения. Таким образом, выбор классификационных признаков будет зависеть от семантического содержания того класса, для которого необходима группировка на последующем уровне иерархии.

Количество уровней классификации, соответствующее числу признаков, выбранных в качестве основания деления, характеризует глубину классификации.

Достоинства иерархической системы классификации:

* простота построения;
* использование независимых классификационных признаков в различных ветвях иерархической структуры.

Недостатки:

* жесткая структура, которая приводит к сложности внесения изменений, так как приходится перераспределять все классификационные группировки;
* невозможность группировать объекты по заранее не предусмотренным сочетаниям признаков.

59. Многоаспектные системы классификации данных. Их особенности, достоинства и недостатки.

**Аспект** — точка зрения на объект классификации, который характеризуется одним или несколькими признаками. **Многоаспектная система** — это система классификации, которая использует параллельно несколько независимых признаков (аспектов) в качестве основания классификации. Существуют **два типа** многоаспектных систем: фасетная и дескрипторная. **Фасет** — это аспект классификации, который используется для образования независимых классификационных группировок. **Дескриптор** — ключевое слово, определяющее некоторое понятие, которое формирует описание объекта и дает принадлежность этого объекта к классу, группе и т.д.

Под фасетным методом классификации понимается "параллельное разделение множества объектов на независимые классификационные группировки". При этом методе классификации заранее жесткой классификационной схемы и конечных группировок не создается. Общий вид фасетной классификационной схемы представлен на рис.

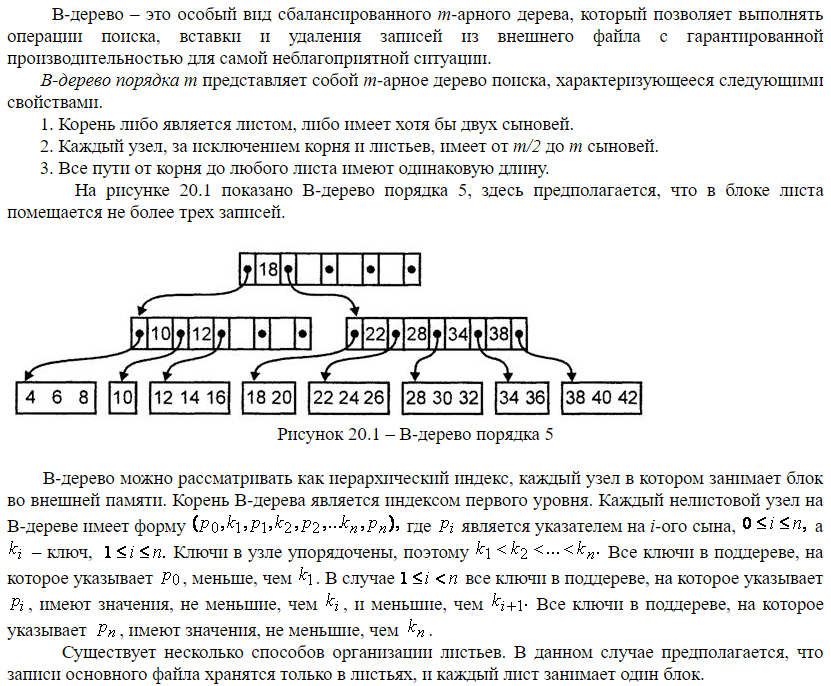
|  |  |
| --- | --- |
|  | К **преимуществам** данной системы следует отнести большую емкость системы и высокую степень гибкости, поскольку при необходимости можно вводить дополнительные фасеты и изменять их место в формуле. |

При изменении характера задач или характеристик объектов классификации разрабатываются новые фасеты или дополняются новыми признаками уже существующие фасеты без коренной перестройки структуры всего классификатора.

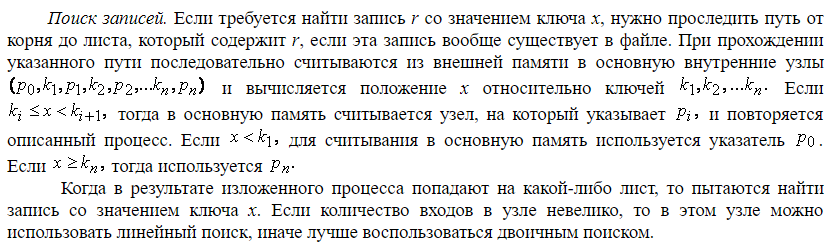
К **недостаткам**, характерным для данной системы, можно отнести сложность структуры и низкую степень заполненности системы.

62. Особенности алгоритмов для внешней памяти. Построение В-дерева.

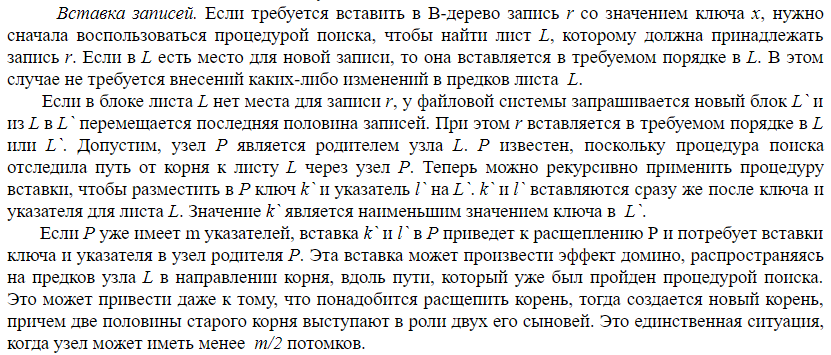
Природа устройств внешней памяти такова, что время, необходимое для поиска блока и чтения его в основную память, достаточно велико по сравнению со временем, которое требуется для относительно простой обработки данных, содержащихся в этом блоке. Оценивая время работы алгоритмов, в которых используются данные, хранящиеся в виде файлов, приходится в первую очередь учитывать количество обращений к блокам, т.е. сколько раз блок считывается в основную память или записывается во вторичную память. Такая операция называется **доступом к блоку**. Поскольку размер блока фиксирован в операционной системе, нет возможности ускорить работу алгоритма, увеличив размер блока и сократив тем самым количество обращений к блокам. Поэтому мерой качества алгоритма, работающего с внешней памятью, является **количество** обращений к блокам. Все ранее рассмотренные алгоритмы предназначены для работы с оперативной памятью. При обработке больших объемов данных их приходится хранить во внешней памяти. В результате усложняется доступ к данным. Основной единицей хранения данных является файл. Его можно рассматривать как связный список блоков. В свою очередь блок состоит из записей. И в каждый блок помещается целое количество значений. **Базовыми операциями** выполняемыми по отношению к файлу является перенос блока в буфер находящийся в оперативной памяти. **Буфер** – зарезервированная область в оперативной, соответствующая размеру блока. По окончанию обращения блок возвращается из буфера во внешний накопитель.



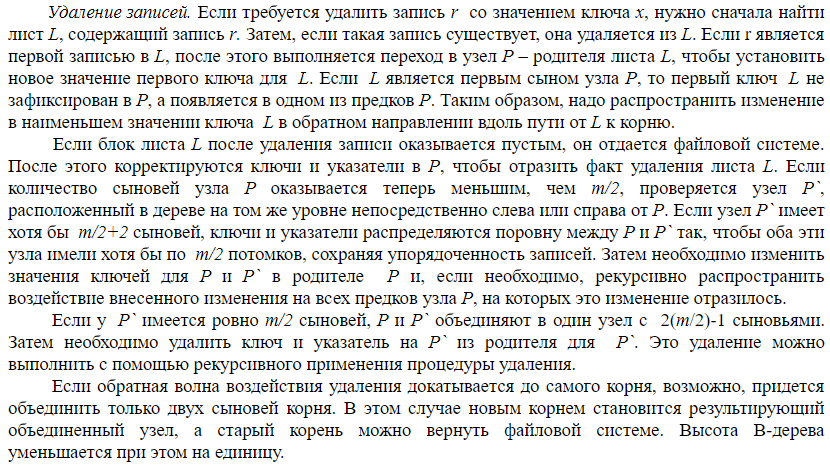
63. Поиск заданного элемента в В-дереве.



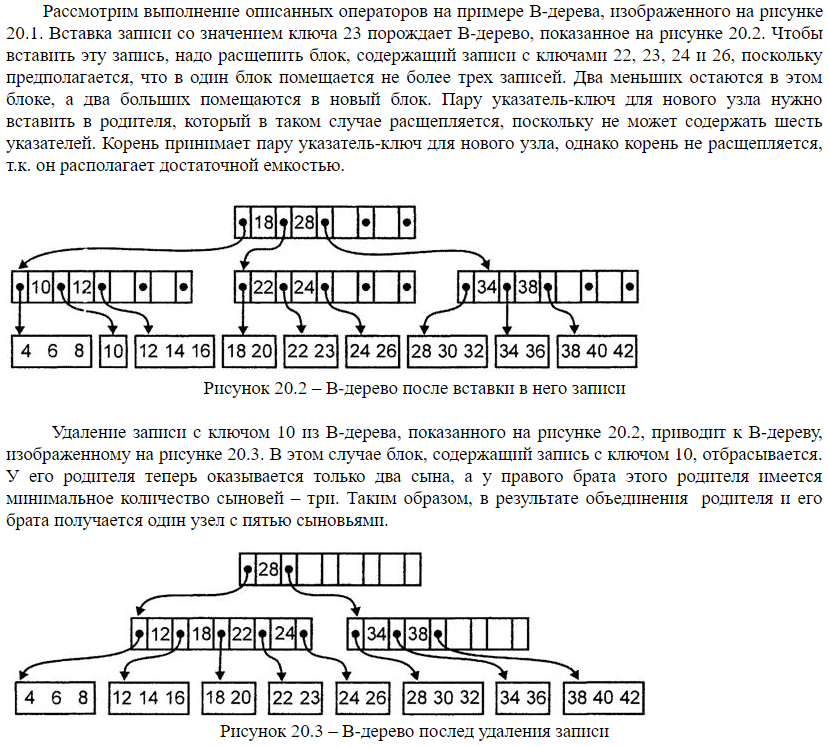
64. Вставка заданного элемента в В-дереве.



65. Удаление заданного элемента в В-дереве.



63-65. Поиск, Вставка и Удаление заданного элемента в В-дереве.



66. Определение, назначение и принципы работы машины Тьюринга.

(Вырезка из конспекта старосты)

**Алгоритм** – решение задач в виде точных указаний. **Свойства**:

- эффективность

- определённость

- конечность

**Эффективность** – возможность выполнения за конечное время.

**Определённость** – возможность точного математического определения или формального описания содержания команд и последующего применения в данной процедуре.

**Конечность** – выполнение за конечное число тактов.

Могут оцениваться числом шагов и памяти:

- конечные автоматы

- машина Тьюринга

- машина Поста

**Маши́на Тью́ринга (МТ)** — абстрактный исполнитель (абстрактная вычислительная машина). Была предложена [Аланом Тьюрингом](https://ru.wikipedia.org/wiki/%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3,_%D0%90%D0%BB%D0%B0%D0%BD_%D0%9C%D0%B0%D1%82%D0%B8%D1%81%D0%BE%D0%BD) в [1936 году](https://ru.wikipedia.org/wiki/1936_%D0%B3%D0%BE%D0%B4) для формализации понятия [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC).

Машина Тьюринга является расширением [конечного автомата](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82) и, согласно [тезису Чёрча — Тьюринга](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%B7%D0%B8%D1%81_%D0%A7%D1%91%D1%80%D1%87%D0%B0_%E2%80%94_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0), *способна имитировать всех исполнителей* (с помощью задания правил перехода), каким-либо образом реализующих процесс пошагового вычисления, в котором каждый шаг вычисления достаточно элементарен.

То есть всякий интуитивный алгоритм может быть реализован с помощью некоторой машины Тьюринга.

## **Устройство**

В состав машины Тьюринга входит неограниченная в обе стороны *лента* (возможны машины Тьюринга, которые имеют несколько бесконечных лент), разделённая на ячейки[[2]](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0#cite_note-_d7e6924ec55c6cc2-2)[[3]](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0#cite_note-_6e97b81c78b18681-3), и *управляющее устройство* (также называется *головкой записи-чтения*(*ГЗЧ*)), способное находиться в одном из *множества состояний*. Число возможных состояний управляющего устройства конечно и точно задано.

Управляющее устройство может перемещаться влево и вправо по ленте, читать и записывать в ячейки символы некоторого конечного алфавита. Выделяется особый *пустой* символ, заполняющий все клетки ленты, кроме тех из них (конечного числа), на которых записаны входные данные.

Управляющее устройство работает согласно *правилам перехода*, которые представляют алгоритм, *реализуемый* данной машиной Тьюринга. Каждое правило перехода предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей клетке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево или вправо. Некоторые состояния машины Тьюринга могут быть помечены как *терминальные*, и переход в любое из них означает конец работы, остановку алгоритма.

Машина Тьюринга называется *детерминированной*, если каждой комбинации состояния и ленточного символа в таблице соответствует не более одного правила. Если существует пара «ленточный символ — состояние», для которой существует 2 и более команд, такая машина Тьюринга называется [*недетерминированной*](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B4%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0).

## **Описание машины Тьюринга**

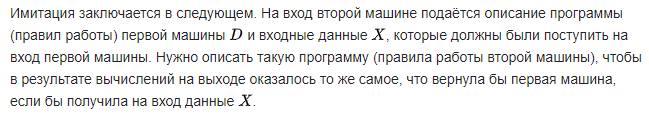
Конкретная машина Тьюринга задаётся перечислением элементов множества букв алфавита A, множества состояний Q и набором правил, по которым работает машина. Они имеют вид: qiaj→qi1aj1dk (если головка находится в состоянии qi, а в обозреваемой ячейке записана буква aj, то головка переходит в состояние qi1, в ячейку вместо aj записывается aj1, головка делает движение dk, которое имеет три варианта: на ячейку влево (L), на ячейку вправо (R), остаться на месте (N)). Для каждой возможной конфигурации <qi, aj> имеется ровно одно правило (для недетерминированной машины Тьюринга может быть большее количество правил). Правил нет только для заключительного состояния, попав в которое, машина останавливается. Кроме того, необходимо указать конечное и начальное состояния, начальную конфигурацию на ленте и расположение головки машины.

## **Полнота по Тьюрингу**

Можно сказать, что машина Тьюринга представляет собой простейшую вычислительную машину с линейной памятью, которая согласно формальным правилам преобразует входные данные с помощью последовательности *элементарных действий*.

Элементарность действий заключается в том, что действие меняет лишь небольшой фрагмент данных в памяти (в случае машины Тьюринга — лишь одну ячейку), и число возможных действий не бесконечно. Несмотря на простоту машины Тьюринга, на ней можно вычислить всё, что можно вычислить на любой другой машине, осуществляющей вычисления с помощью последовательности элементарных действий. Это свойство называется *полнотой*.

Один из естественных способов доказательства того, что алгоритмы вычисления, которые можно реализовать на одной машине, можно реализовать и на другой, — это имитация первой машины на второй.



Как было сказано, на машине Тьюринга можно имитировать (с помощью задания правил перехода) все другие исполнители, каким-либо образом реализующие процесс пошагового вычисления, в котором каждый шаг вычисления достаточно элементарен.

На машине Тьюринга можно имитировать [машину Поста](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_%D0%9F%D0%BE%D1%81%D1%82%D0%B0), [нормальные алгоритмы Маркова](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) и любую программу для обычных компьютеров, преобразующую входные данные в выходные по какому-либо алгоритму. В свою очередь, на различных абстрактных исполнителях можно имитировать Машину Тьюринга. Исполнители, для которых это возможно, называются *полными по Тьюрингу* (Turing complete).

Есть программы для обычных компьютеров, имитирующие работу машины Тьюринга. Но следует отметить, что данная имитация неполная, так как в машине Тьюринга присутствует абстрактная бесконечная лента. Бесконечную ленту с данными невозможно в полной мере имитировать на компьютере с конечной памятью: суммарная память компьютера — оперативная память, жёсткие диски, различные внешние носители данных, регистры и кэш процессора и др. — может быть очень большой, но, тем не менее, всегда конечна.

67. Принципы и общая схема генетического алгоритма. 68. Кодирование информации и оценивание популяции в генетическом алгоритме. 69. Селекция, скрещивание и мутация в генетическом алгоритме.

**Генети́ческий алгори́тм** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *genetic algorithm*) — это [эвристический алгоритм](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных [естественному отбору](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BE%D1%82%D0%B1%D0%BE%D1%80) в природе. Является разновидностью [эволюционных вычислений](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D0%BE%D0%BB%D1%8E%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как [наследование](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%B1%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F)), [мутации](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%82%D0%B0%D1%86%D0%B8%D1%8F), [отбор](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BE%D1%82%D0%B1%D0%BE%D1%80) и [кроссинговер](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%B8%D0%BD%D0%B3%D0%BE%D0%B2%D0%B5%D1%80). Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

## **Описание алгоритма**

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде [вектора](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) («[генотипа](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%BE%D1%82%D0%B8%D0%BF)») генов, где каждый ген может быть [битом](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82), числом или неким другим объектом. В классических реализациях генетического алгоритма (ГА) предполагается, что генотип имеет фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения.

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «[функции приспособленности](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8)», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет насколько хорошо [фенотип](https://ru.wikipedia.org/wiki/%D0%A4%D0%B5%D0%BD%D0%BE%D1%82%D0%B8%D0%BF), им описываемый, решает поставленную задачу.

При выборе «[функции приспособленности](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8)» (или fitness function в англоязычной литературе) важно следить, чтобы её «рельеф» был «гладким».

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения (обычно лучшие особи имеют большую вероятность быть выбранными), к которым применяются «генетические операторы» (в большинстве случаев «[скрещивание](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B5%D1%89%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)» — crossover и «[мутация](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%82%D0%B0%D1%86%D0%B8%D1%8F)» — mutation), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (*поколений*), пока не будет выполнен критерий остановки алгоритма. Таким критерием может быть:

* нахождение глобального, либо субоптимального решения;
* исчерпание числа поколений, отпущенных на эволюцию;
* исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в многомерных пространствах поиска.

Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Задать целевую функцию (приспособленности) для особей популяции
2. Создать начальную популяцию

* (Начало цикла)

1. Размножение (скрещивание)
2. Мутирование
3. Вычислить значение целевой функции для всех особей
4. Формирование нового поколения (селекция)
5. Если выполняются условия остановки, то (конец цикла), иначе (начало цикла).

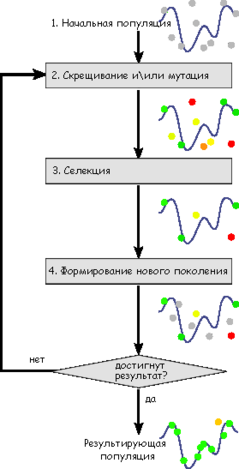
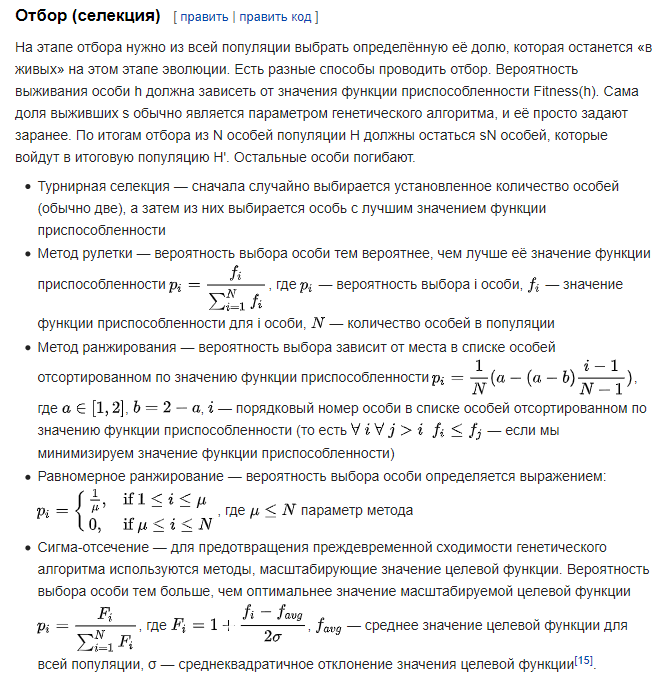
[](https://commons.wikimedia.org/wiki/File:Schema_simple_algorithme_genetique_ru.png?uselang=ru)

Схема работы генетического алгоритма

### Создание начальной популяции

Перед первым шагом нужно случайным образом создать начальную популяцию; даже если она окажется совершенно неконкурентоспособной, вероятно, что генетический алгоритм всё равно достаточно быстро переведёт её в жизнеспособную популяцию. Таким образом, на первом шаге можно особенно не стараться сделать слишком уж приспособленных особей, достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию приспособленности (Fitness). Итогом первого шага является популяция H, состоящая из N особей.



### Выбор родителей

Размножение в генетических алгоритмах требует для производства потомка нескольких родителей, обычно двух.

Можно выделить несколько операторов выбора родителей:

1. Панмиксия — оба родителя выбираются случайно, каждая особь популяции имеет равные шансы быть выбранной
2. Инбридинг — первый родитель выбирается случайно, а вторым выбирается такой, который наиболее похож на первого родителя
3. Аутбридинг — первый родитель выбирается случайно, а вторым выбирается такой, который наименее похож на первого родителя

Инбридинг и аутбридинг бывают в двух формах: фенотипной и генотипной. В случае фенотипной формы похожесть измеряется в зависимости от значения функции приспособленности (чем ближе значения целевой функции, тем особи более похожи), а в случае генотипной формы похожесть измеряется в зависимости от представления генотипа (чем меньше отличий между генотипами особей, тем особи похожее).

### Размножение (Скрещивание)

Размножение в разных алгоритмах определяется по-разному — оно, конечно, зависит от представления данных. Главное требование к размножению — чтобы потомок или потомки имели возможность унаследовать черты обоих родителей, «смешав» их каким-либо способом.

Почему особи для размножения обычно выбираются из всей популяции H, а не из выживших на первом шаге элементов H' (хотя последний вариант тоже имеет право на существование)? Дело в том, что главный недостаток многих генетических алгоритмов — отсутствие разнообразия (diversity) в особях. Достаточно быстро выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция «забивается» копиями этой особи. Есть разные способы борьбы с таким нежелательным эффектом; один из них — выбор для размножения не самых приспособленных, но вообще всех особей. Однако такой подход вынуждает хранить всех существовавших ранее особей, что увеличивает вычислительную сложность задачи. Поэтому часто применяют методы отбора особей для скрещивания таким образом, чтобы «размножались» не только самые приспособленные, но и другие особи, обладающие плохой приспособленностью. При таком подходе для разнообразия генотипа возрастает роль мутаций.

### Мутации

К мутациям относится все то же самое, что и к размножению: есть некоторая доля мутантов m, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать mN особей, а затем изменить их в соответствии с заранее определёнными операциями мутации.

## **Применение генетических алгоритмов**

Генетические алгоритмы применяются для решения следующих задач:

1. [Оптимизация функций](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B9&action=edit&redlink=1)
2. [Оптимизация запросов в базах данных](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2_%D0%A1%D0%A3%D0%91%D0%94)
3. Разнообразные задачи на [графах](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) ([задача коммивояжера](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D0%B5%D1%80%D0%B0), раскраска, нахождение паросочетаний)
4. Настройка и обучение [искусственной нейронной сети](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C)
5. Задачи [компоновки](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0)
6. [Составление расписаний](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%80%D0%B0%D1%81%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D0%B9)
7. [Игровые стратегии](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D1%8F_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B8%D0%B3%D1%80))
8. [Теория приближений](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%B1%D0%BB%D0%B8%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9)
9. [Искусственная жизнь](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B6%D0%B8%D0%B7%D0%BD%D1%8C)
10. [Биоинформатика](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BE%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) ([фолдинг белков](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D0%BB%D0%B4%D0%B8%D0%BD%D0%B3_%D0%B1%D0%B5%D0%BB%D0%BA%D0%B0" \o "Фолдинг белка))
11. Синтез конечных автоматов
12. Настройка ПИД регуляторов