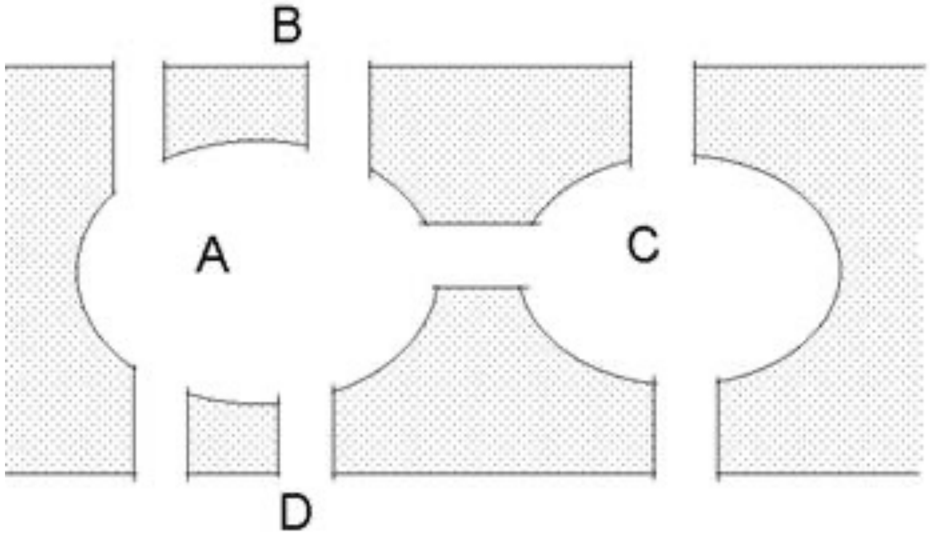
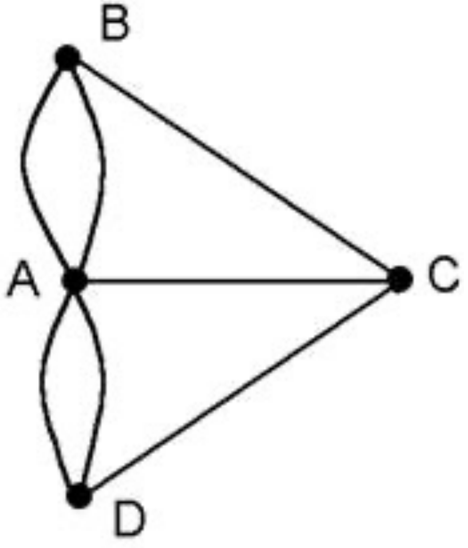
Теория графов. Введение.

Задача о Кёнигсбергских мостах





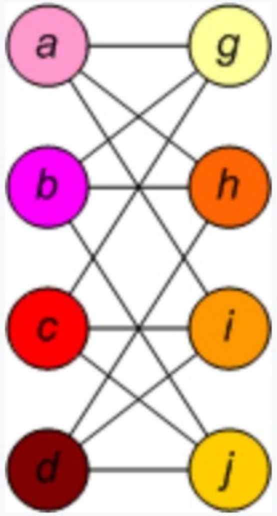
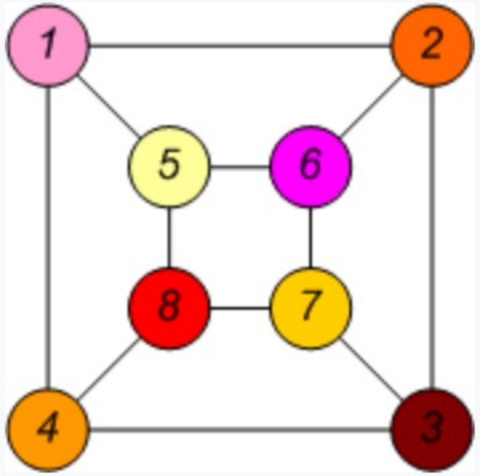
Неориентированный граф

1. Пара (V, E), где V — непустое множество объектов некоторой природы, называемых *вершинами* графа, а E — подмножество двухэлементных подмножеств множества V, называемых *ребрами* графа. Множества вершин и ребер графа G обозначают $V(G)$ и $E(G)$ соответственно. Если $|V(G)| = n$ и $|E(G)| = m$, то говорят о (n, m)-графе G.

Изоморфизм

Графы $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ называют ***изоморфными***, если существует биекция $f: V_1 \to V_2$, такая, что для любых $a, b \in V_1$ $(a, b) \in E_1$ тогда и только тогда, когда $(f(a), f(b)) \in E_2$. Эта биекция f называется ***изоморфизмом*** графа G_1 на граф G_2. Если графы G_1 и G_2 изоморфны, то пишем, что $G_1 \cong G_2$. Отношение изоморфизма графов есть отношение эквивалентности, так как оно рефлексивно, симметрично и транзитивно. Следовательно, множество всех графов разбивается на классы так, что графы из одного класса попарно изоморфны, а графы из разных классов не изоморфны.

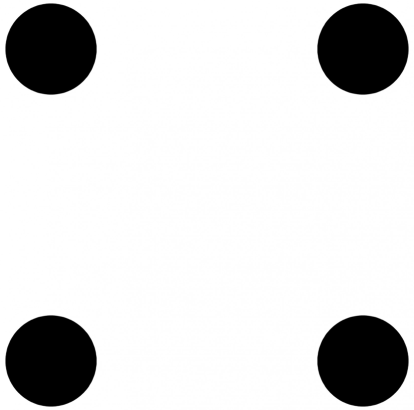
Изоморфизм

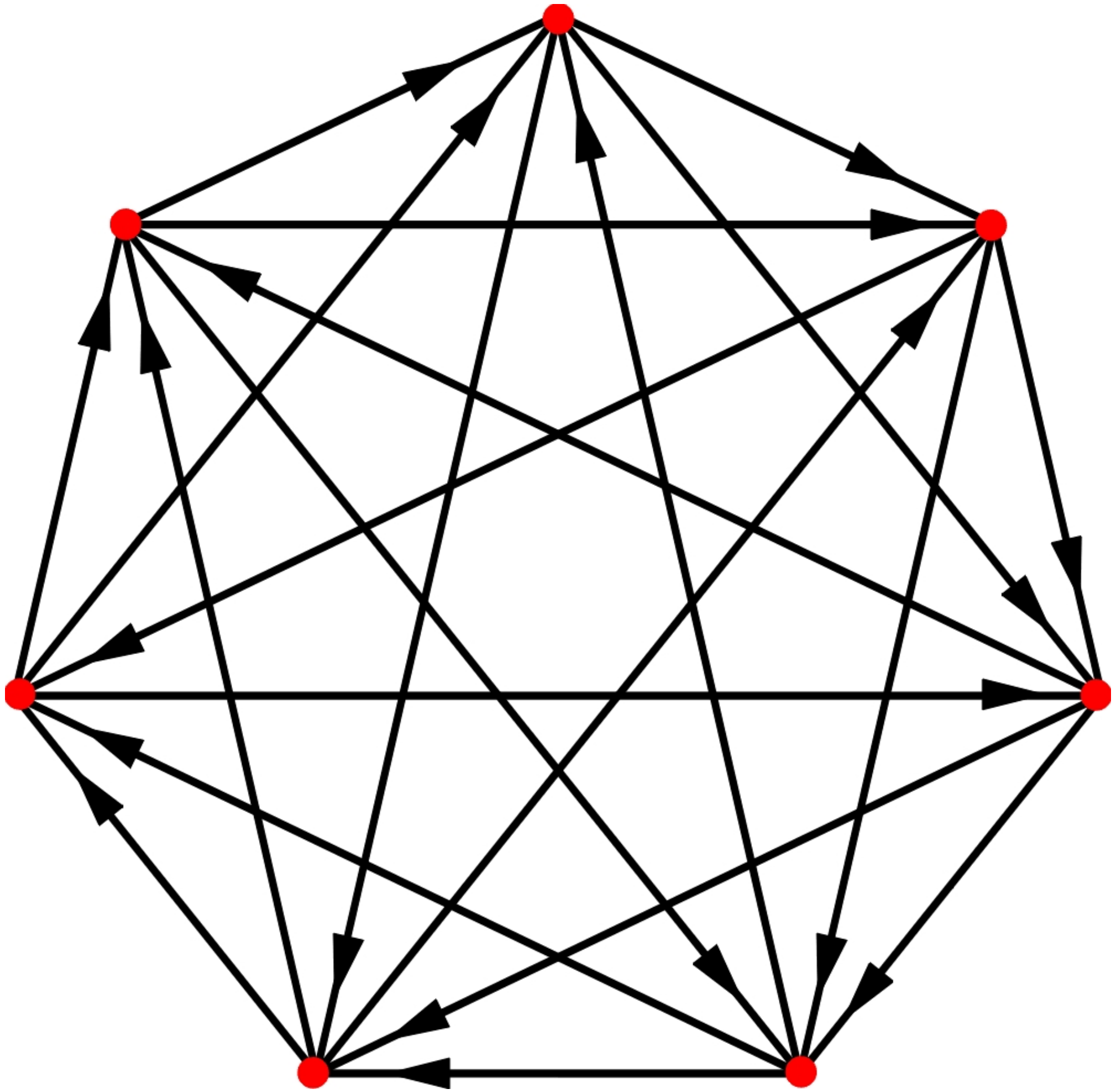
Граф G	Граф H	Изоморфизм между графами G и H	Подстановка изоморфизма f
		$f(a)=1$ $f(b)=6$ $f(c)=8$ $f(d)=3$ $f(g)=5$ $f(h)=2$ $f(i)=4$ $f(j)=7$	$\begin{pmatrix} a & b & c & d & g & h & i & j \\ 1 & 6 & 8 & 3 & 5 & 2 & 4 & 7 \end{pmatrix}$

Мультиграф

- пара *(V,E)*, где *V* - непустое множество вершин, а *E* - семейство подмножеств множества $V \times V$ (ребра); другими словами, мультиграф есть граф с кратными ребрами. (LW: мультимножество (multiset,bag) это множество в котором дозволены кратные вхождения элементов).

- Полный граф из n вершин ($K_n = < V, V \times V >$, где $|V| = n$) - есть ребра между всеми возможными парами вершин.
- Изолированный граф (0-граф) - нет ребер $E = \emptyset$;

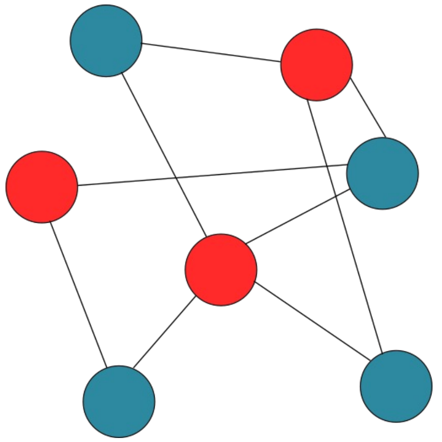


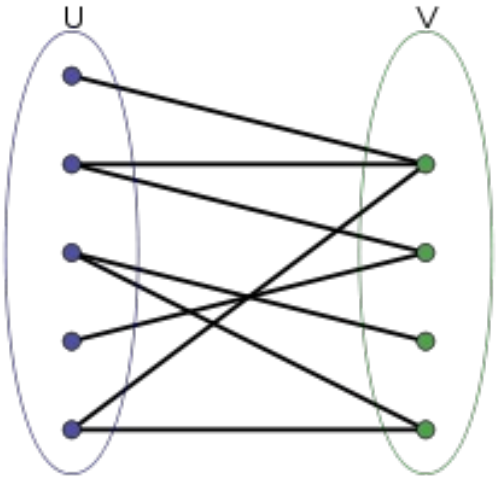


Двудольный граф

Двудольный граф - граф $G = \{V, E\}$ с долями из n и m вершин. Множество вершин V графа состоит из двух непересекающихся подмножеств “долей”. Нет ребер инцидентных двум вершинам одной доли (внутри доли вершины “не дружат”, т.е. не смежны):

1. $V = V_1 \cup V_2$
2. $V_1 \cap V_2 = \emptyset$
3. $|V_1| = n$ и $|V_2| = m$
4. $E \subseteq V \times V \setminus (V_1 \times V_1 \cup V_2 \times V_2)$

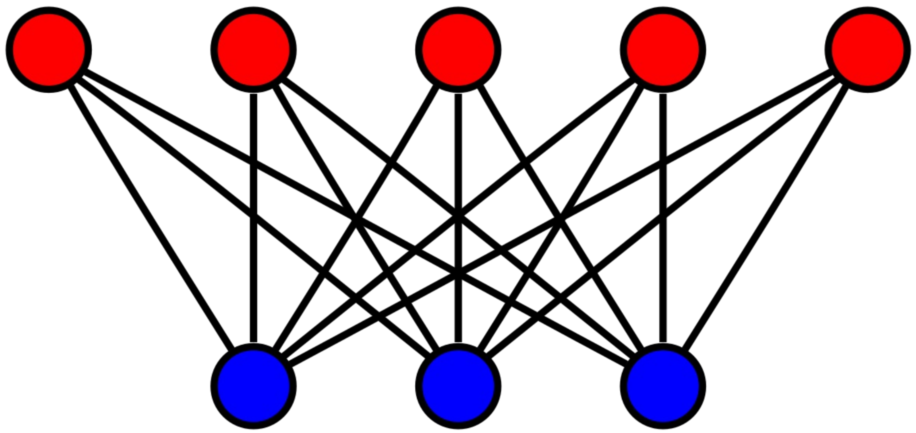




Полный двудольный граф

Полный двудольный граф $K_{n,m} = \{V, E\}$ с долями из n и m вершин - двудольный граф со всеми допустимыми ребрами.

1. $V = V_1 \cup V_2$
2. $V_1 \cap V_2 = \emptyset$
3. $|V_1| = n$ и $|V_2| = m$
4. $|E = V \times V \setminus (V_1 \times V_1 \cup V_2 \times V_2)$

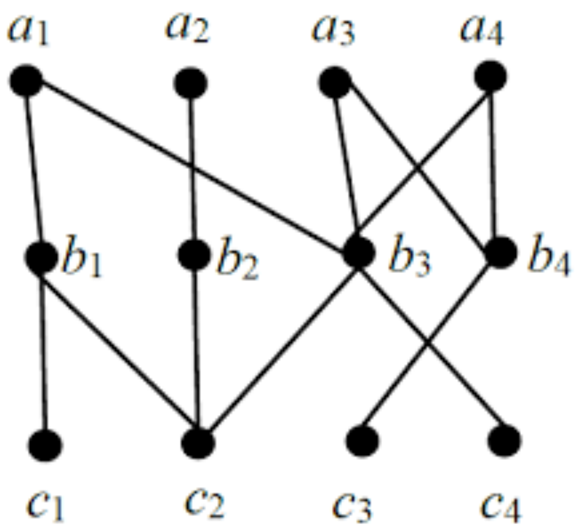


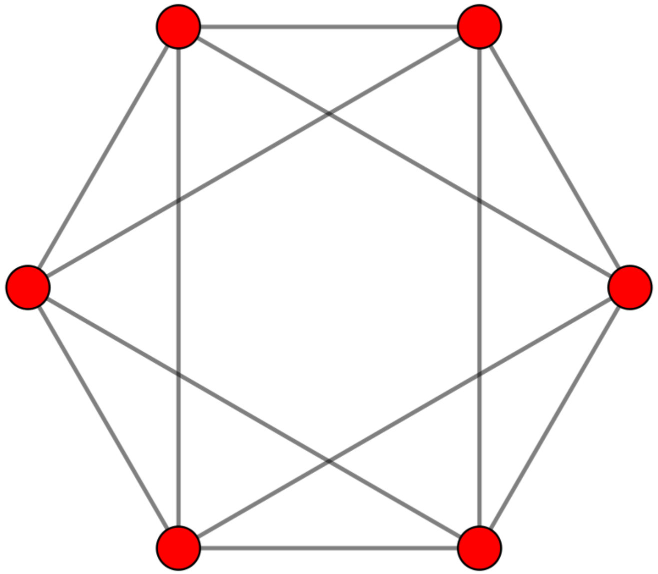
$$\left|E\left(K_{n,m}\right)\right| = n * m$$

Многодольный граф

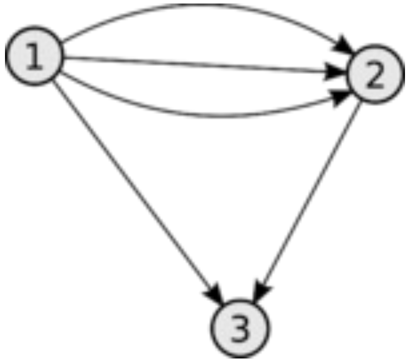
k-дольный граф — граф, множество вершин которого можно разбить на k независимых множеств (доль).

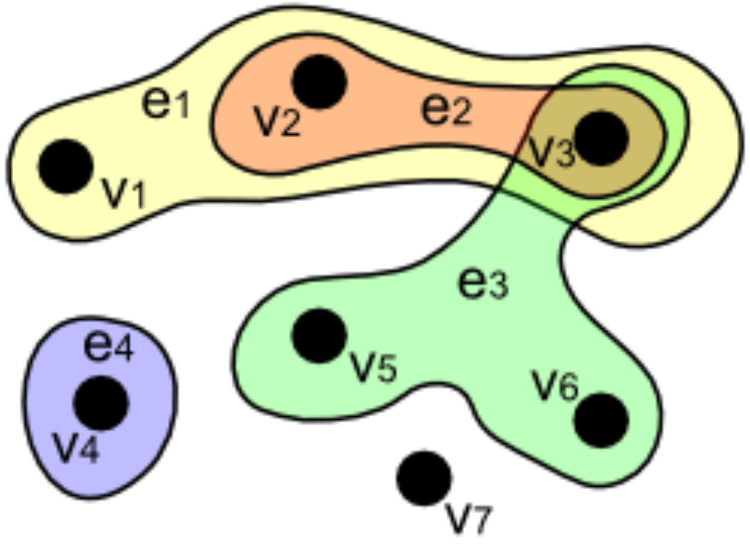
При k = 2:
граф называется двудольным

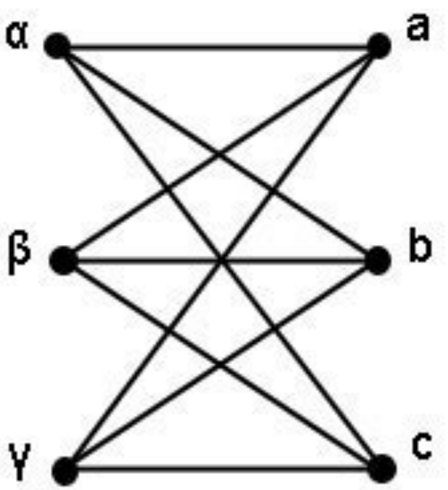


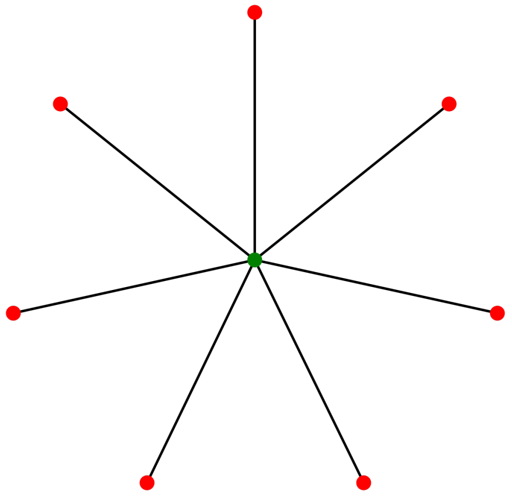


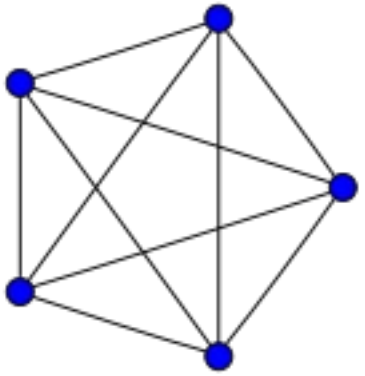
$K_{2,2,2}$

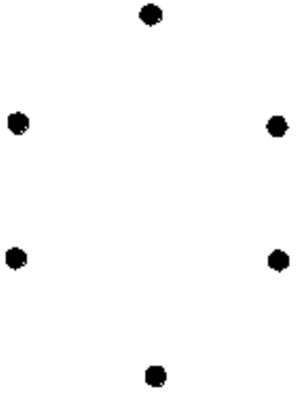

1
2
3

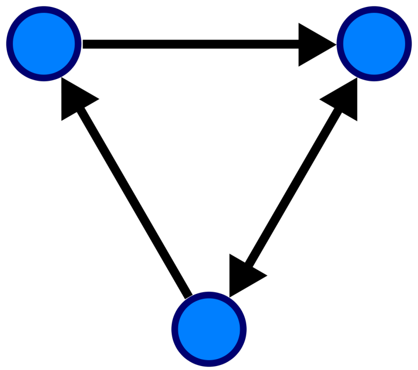

e1
v2
e2
v3
v1
e3
e4
v5
v6
v4
v7

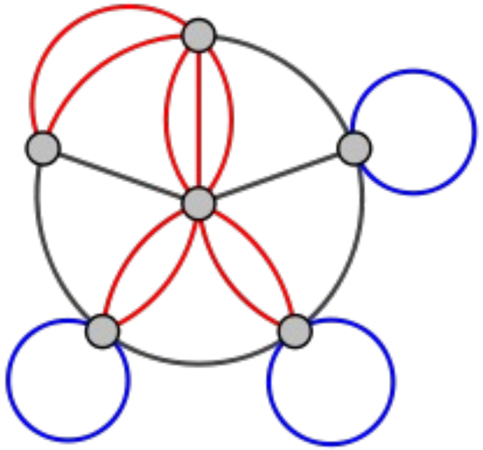

α
a
β
b
γ
c











Основные понятия

Инцидентность*(Incidenty)* - отношение между ребром (дугой) и его концевыми вершинами, т.е. ребро *e = (a,b)* инцидентно вершинам *a* и *b* и вершины *a, b* инцидентны ребру *e = (a,b)*.
Смежность*(Adjacency)* - бинарное отношение Ad на множестве вершин (ребер) графа такое, что *a* Ad *b* тогда и только тогда, когда *a* и *b* соединены дугой или ребром (имеют общую вершину).
Смежные вершины*(Adjacent vertices, joined vertices)* - вершины *a* и *b*, соединенные ребром (в графе), соединенные дугой *(a,b)* (в орграфе), принадлежащие одному ребру (в гиперграфе).

Степень вершины

Полустепень захода вершины $\boldsymbol{d_+(v)}$ (Indegree) - в орграфе число дуг, заходящих в вершину.

Полустепень исхода вершины $\boldsymbol{d_-(v)}$ (Outdegree) - в орграфе число дуг, исходящих из вершины.

Степень вершины $\boldsymbol{d(v)}$ (Degree of a vertex, valency of a vertex) - в графе - число инцидентных вершине ребер; в орграфе - число инцидентных дуг, т.е. сумма полустепеней захода и исхода.

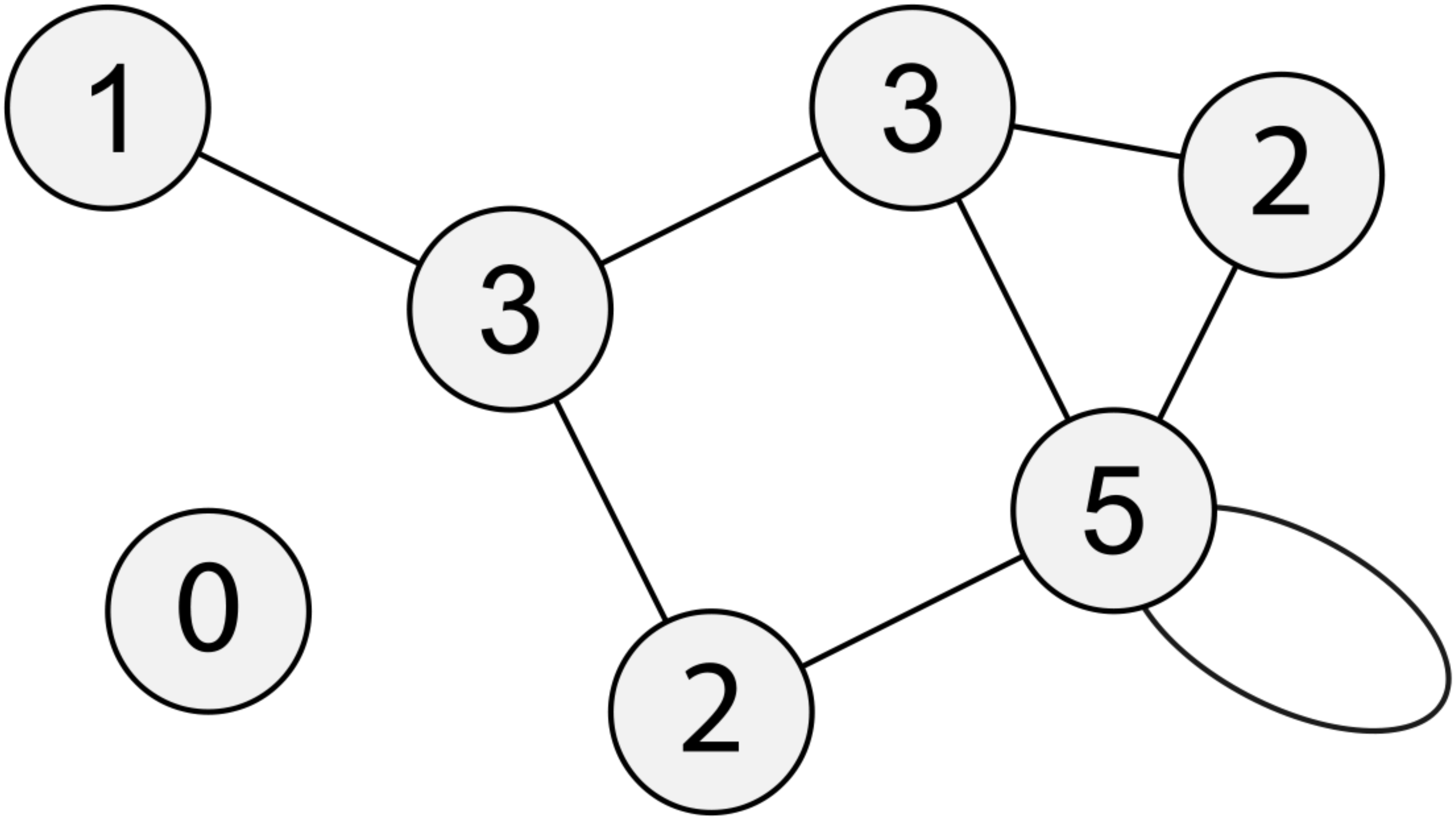
Чтобы сумма полустепеней захода/исхода всех вершин была равна сумме степеней всех вершин, обычно будем считать петли в степени 2 раза. Но это условно, т.к. в конкретных задачах может понадобиться учитывать их другим способом/не учитывать вообще.

Степень вершины

$\sigma(G)$ - минимальная степень вершины

$\Delta(G)$ - максимальная степень вершины

Граф **регулярный** (однородный), если все степени равны.


1
3
3
2
0
5
2

Число ребер/дуг в полном (ор)графе из n вершин

	без петель	с петлями
Неориентированный граф	Число **ребер** =	Число **ребер** =
Ориентированный граф	Число **дуг** =	Число **дуг** =

На n вершинах можно построить различных графов.

Число ребер/дуг в полном (ор)графе из n вершин

	без петель	с петлями
Неориентированный граф	Число **ребер** = $\frac{n*(n-1)}{2}$	Число **ребер** = $\frac{n*(n+1)}{2}$
Ориентированный граф	Число **дуг** = $n*(n-1)$	Число **дуг** = n^2

На n вершинах можно построить $2^{C_n^2}$ различных графов.

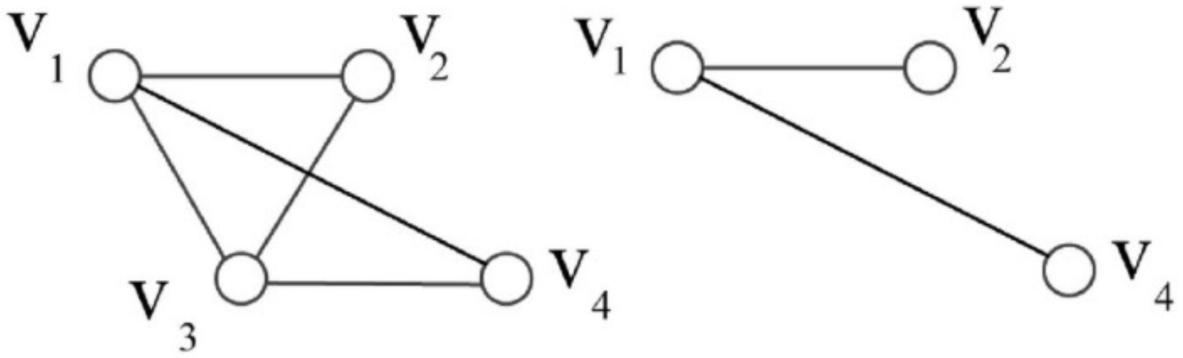
Операции над графами [1]

Удаление вершины *(removal of a vertex)*

- преобразование графа *G* в граф *G* \ v, содержащий все вершины графа *G*, за исключением v , и все ребра графа *G*, не инцидентные v.

Добавление (новой) вершины $\boldsymbol{v}$ (vertex addition).

Исходный граф $G =< V, E >$. Результат операции добавления $G =< V \cup \{v\}, E >$.



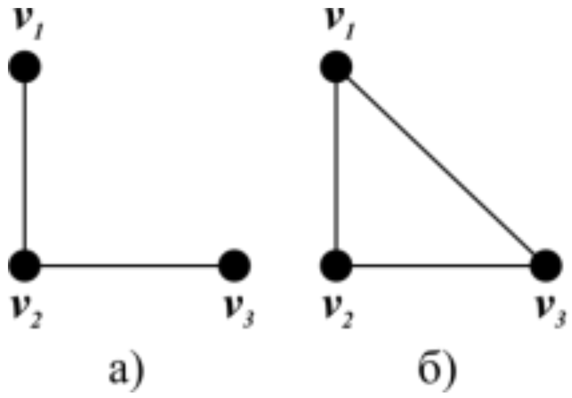
Операции над графами [2]

Удаление ребра *(removal of an edge)*

- преобразование графа *G* в граф *G* \ *e*, содержащий все вершины и все ребра графа *G* за исключением *e*.

Добавление ребра *(edge adding)*

- операция над графами, состоящая в соединении ребром *e* двух несмежных вершин графа *G*, порождая граф *G + e*.



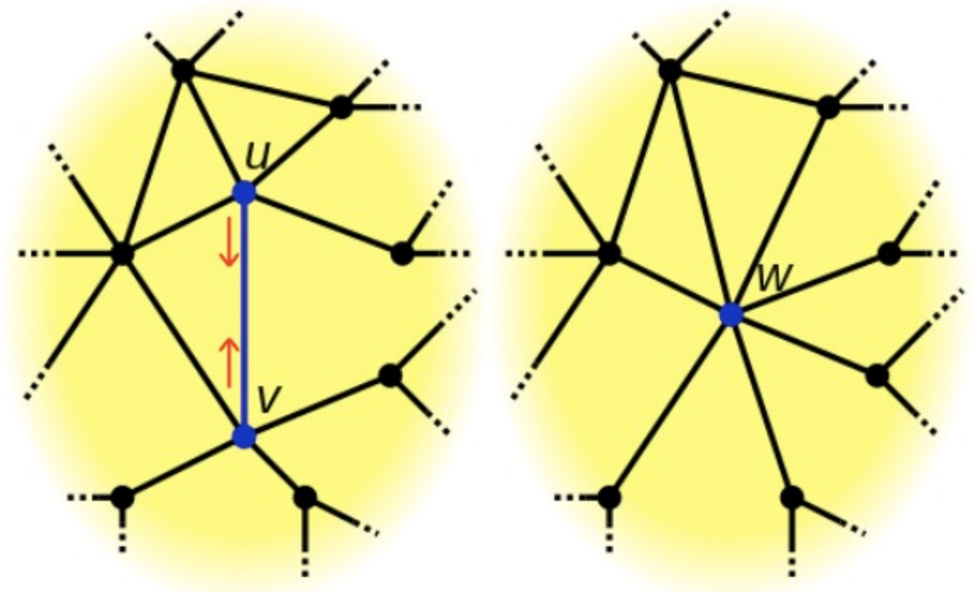
Операции над графами [3]

Стягивание графа $G = \langle V, E \rangle$ по множеству вершин $J \subseteq V$:

1. Вершины J удаляются;
2. Добавляется новая вершина w;
3. Вершины в G, смежные вершинам в J, соединяются ребрами с новой вершиной w.

В результате стягивания могут появиться кратные ребра. Возможны две вариации:

1) в результате стягивания кратные ребра остаются;
2) из результирующих кратных ребер оставляют одно.



Пример стягивания для подмножества вершин $J = \{u, v\}$

Маршрут

Маршрут *(Sequence)* - чередующаяся последовательность $\sigma = \langle a = v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n = b \rangle$ вершин и ребер графа такая, что $e_i = \langle v_{i-1}, v_i \rangle$ или $e_i = \langle v_i, v_{i-1} \rangle$, *для* $1 \leq i \leq n$. Говорят, что маршрут соединяет вершины *a* и *b* - концы маршрута. Очевидно, что маршрут можно задать перечислением лишь его вершин $a = v_0, v_1, \ldots, v_{n-1}, v_n = b$ или его ребер $a = e_1, \ldots, e_n$.

Длина маршрута - число ребер.

Цепь

Цепь *(Chain, trail)*
В неориентированном графе маршрут, все ребра которого различны.

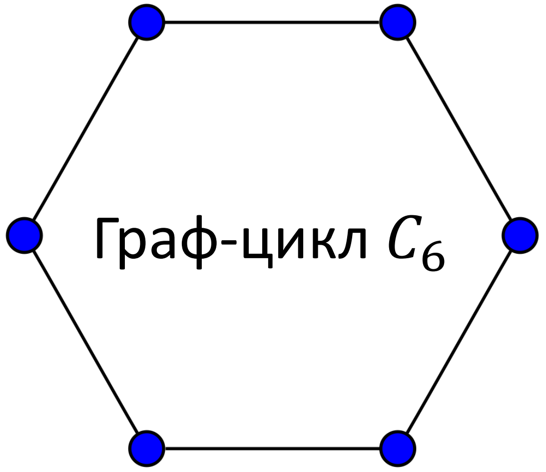
Простая цепь *(Simple chain)*
- цепь, в которой ни одна вершина не встречается дважды.

Цикл

Циклический маршрут (замкнутый маршрут) *(Cyclic sequence)*
- маршрут, концы которого совпадают.

Цикл *(Loop, Circuit, Cycle)*
- цепь, концы которой совпадают.
LW: граф-цикл из n вершин обозначается C_n.

Простой цикл *(Simple circuit)*
- цикл, в котором ни одна вершина не встречается дважды.
Иначе, замкнутая простая цепь, с оговоркой про повторяющиеся начальную и конечную вершину.



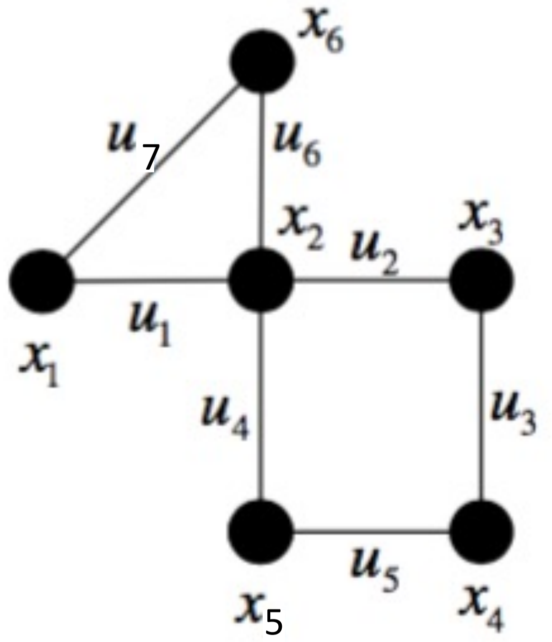


Рис. 1. Пример неориентированного графа.

1) $x_1 u_1 x_2 u_2 x_3 u_2 x_2 u_4 x_5 u_5 x_4$ –
2) $x_1 u_1 x_2 u_2 x_3 u_2 x_2 u_4 x_5 u_5 x_4 u_5 x_5 u_4 x_5 u_4 x_2 u_1 x_1$ –
3) $x_1 u_1 x_2 u_2 x_3 u_3 x_4 u_5 x_5 u_4 x_2$ –
4) $x_1 u_1 x_2 u_2 x_3 u_3 x_4 u_5 x_5 u_4 x_2 u_6 x_6 u_7 x_1$ –
5) $x_1 u_1 x_2 u_2 x_3 u_3 x_4$ –
6) $x_2 u_2 x_3 u_3 x_4 u_5 x_5 u_4 x_2$ –

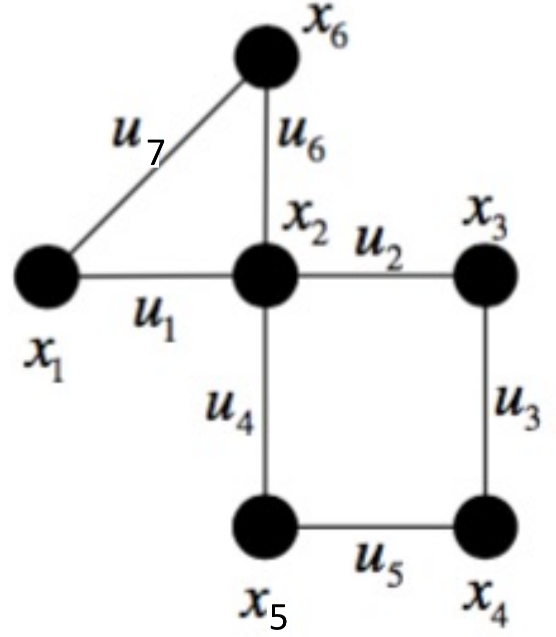


Рис. 1. Пример неориентированного графа.

1) $x_1 u_1 x_2 u_2 x_3 u_2 x_2 u_4 x_5 u_5 x_4$ – маршрут;

2) $x_1 u_1 x_2 u_2 x_3 u_2 x_2 u_4 x_5 u_5 x_4 u_5 x_5 u_4 x_5 u_4 x_2 u_1 x_1$ – циклический маршрут;

3) $x_1 u_1 x_2 u_2 x_3 u_3 x_4 u_5 x_5 u_4 x_2$ – цепь

4) $x_1 u_1 x_2 u_2 x_3 u_3 x_4 u_5 x_5 u_4 x_2 u_6 x_6 u_7 x_1$ – цикл;

5) $x_1 u_1 x_2 u_2 x_3 u_3 x_4$ – простая цепь

6) $x_2 u_2 x_3 u_3 x_4 u_5 x_5 u_4 x_2$ – простой цикл.

Лемма о рукопожатии

Формула суммы степеней:

$$\Sigma_{v \in V} d(v) = 2|E|$$

Следствие леммы о рукопожатии

Формула суммы степеней:

$$\Sigma_{v \in V} d(v) = 2|E|$$

Следствие: **количество нечетных вершин четно.**

Докажите, что

В любом простом графе есть 2 вершины с одинаковой степени.

Слабая и сильная достижимости

Слабая достижимость (сильная достижимость) (*Reachability*) — бинарное отношение R на множестве вершин графа (**орграфа**) такое, что aRb тогда и только тогда, когда в графе существует маршрут (**ормаршрут**) из a в b. От одной вершины к другой может быть несколько различных маршрутов (**ормаршрутов**), кратчайший из них называется геодезической линией. Множество вершин, достижимых (**сильно достижимых**) из данной вершины v, называется множеством достижимости (**множеством сильной достижимости**) вершины v.

Слабую и сильную достижимость обозначим $\mathfrak{R}$ и $\overline{\mathfrak{R}}$ соответственно.

Слабая и сильная взаимные достижимости

Слабая взаимная достижимость (связанность) вершин v_1 и v_2: $v_1 \mathfrak{R} v_2 \wedge v_2 \mathfrak{R} v_1$.

Сильная взаимная достижимость (сильная связанность) вершин v_1 и v_2: $v_1 \overline{\mathfrak{R}} v_2 \wedge v_2 \overline{\mathfrak{R}} v_1$.

Слабая (сильная) связность подграфа - все вершины подграфа попарно слабо (сильно) взаимно достижимы **в подграфе**.

Слабая и сильная достижимости

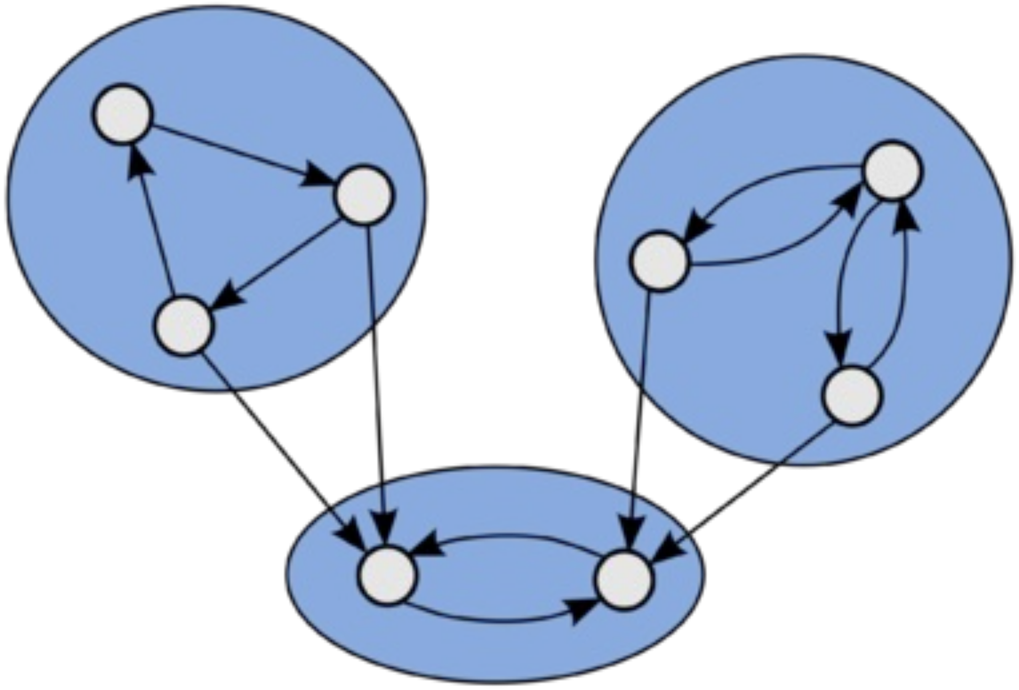
Компонента (слабой) связности - CC.

Компонента сильной связности - SCC.

Возможные определения:

1) CC (SCC) - максимальный подграф по свойству “слабая взаимная достижимость” (“сильная взаимная достижимость”).
2) CC (SCC) - класс эквивалентности по отношению “слабая взаимная достижимость” (“сильная взаимная достижимость”).

Компоненты слабой и сильной связности



Связность

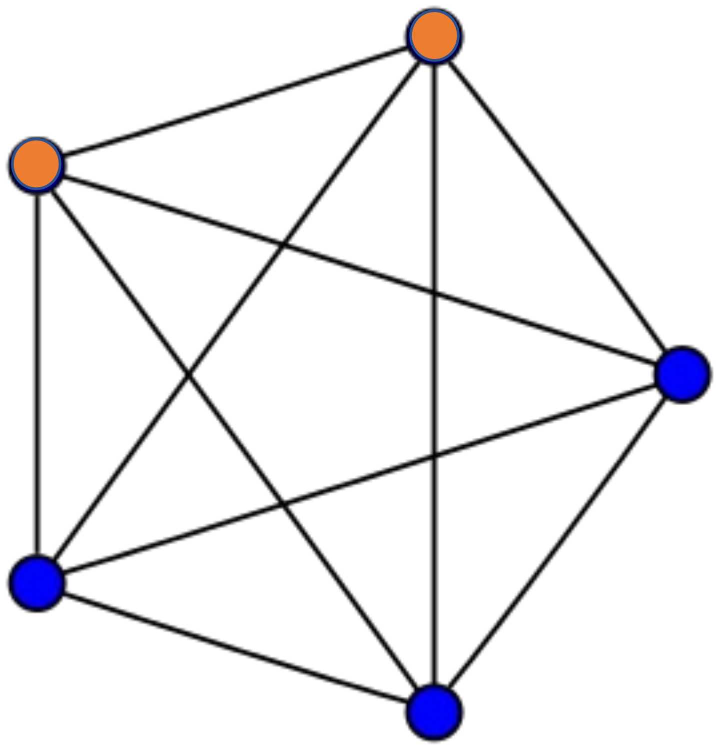
Неориентированный граф называется связным, если в нем для любых двух вершин имеется маршрут, соединяющий эти вершины.
Ориентированный граф называется сильно-связным, если в нем для любых двух вершин имеется маршрут, соединяющий эти вершины.
Ориентированный граф называется слабо-связным, если в его неориентированном изображении для любых двух вершин имеется маршрут, соединяющий эти вершины.

Подграфы

Максимальный (подграф/объект) по свойству P (по порядку $\leq$) – подграф/объект со свойством P, такой что, среди всех рассматриваемых подграфов/объектов со свойством P, не существует подграфа/объекта большего по $\leq$. В качестве $\leq$ для подграфов, обычно берется включение вершин и ребер.

Наибольший (подграф/объект) по свойству P - максимальный по мощности среди всех объектов со свойством P в заданном универсуме.

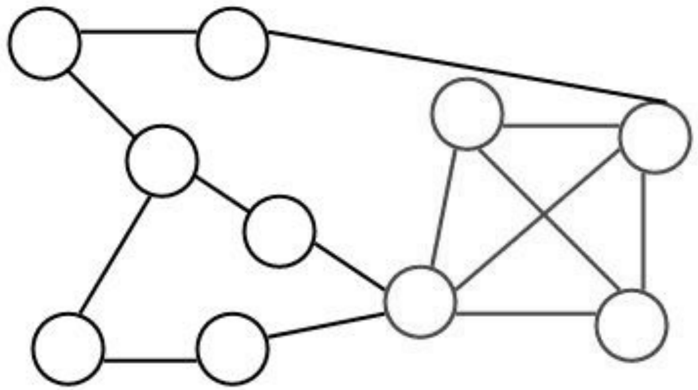
Подграфы

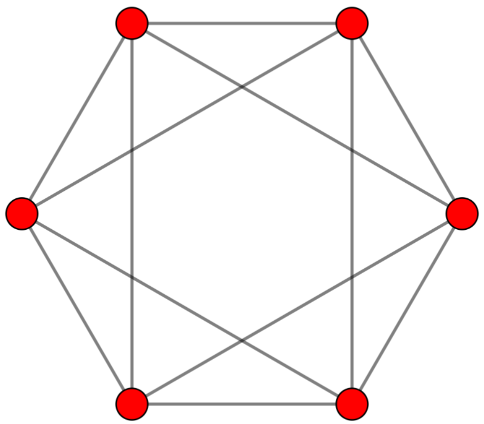


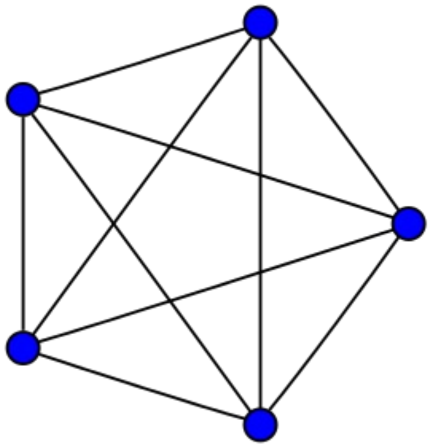
Клика

Клика (*Clique*) — подмножество V' графа G , в котором любые две вершины смежны, т.е. порожденный ими подграф $G(V')$ является полным.

Клика из 2 вершин - ребро. Из 3 вершин - треугольник (C_3 или K_3).







Расстояния

Пусть G – связный неориентированный граф. Так как любые две вершины графа a и b связаны, то существуют простые цепи с концами a и b. Таких цепей может быть несколько. Их длины являются неотрицательными целыми числами. Следовательно, между вершинами a и b должны существовать простые *цепи наименьшей длины*. Длина цепи наименьшей длины, связывающей вершины a и b, обозначается символом $d(a, b)$ и называется *расстоянием* между вершинами a и b. По определению $d(a, a) = 0$.

Нетрудно убедиться, что введенное таким образом понятие расстояния, удовлетворяет аксиомам метрики:

1. $d(a, b) \geq 0$;

2. $d(a, b) = 0$ тогда и только тогда, когда $a = b$;

3. $d(a, b) = d(b, a)$;

4. справедливо неравенство треугольника:

$$d(a, b) + d(b, c) \geq d(a, c).$$

Расстояния

Для фиксированной вершины графа a расстояние до наиболее удаленной от нее вершины: $e(a) = \max_{v \in V} d(a, v)$, называют *эксцентриситетом* (*максимальным удалением*) вершины a.

Диаметром графа G называют число $d(G)$, равное расстоянию между наиболее удаленными друг от друга вершинами графа:

$$d(G) = \max_{a, v \in V} d(a, v).$$

Простая цепь, длина которой равна $d(G)$, называется *диаметральной цепью*. Очевидно, что диаметр графа равен наибольшему среди всех эксцентриситетов вершин графа. Вершина v называется *периферийной*, если $e(v) = d(G)$.

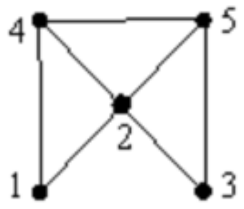
Минимальный из эксцентриситетов вершин связного графа G называют его *радиусом* и обозначают $r(G)$:

$$r(G) = \min_{a \in V} e(a) = \min_{a \in V} \max_{b \in V} d(a, b).$$

Расстояния

Так как диаметр графа равен наибольшему из эксцентриситетов вершин, а радиус – наименьшему, то радиус графа не может быть больше его диаметра. Вершина v называется *центральной*, если $e(v) = r(G)$. Множество всех центральных вершин графа называют его *центром*. Центром графа может быть одна вершина или несколько вершин. Есть графы, центр которых совпадает с множеством всех его вершин. Например, центр простой цепи состоит из двух вершин при четном числе ее вершин и из одной – при нечетном, а у любого цикла все вершины являются центральными.

Расстояния



Для иллюстрации обратимся к графу на рис.

$d(1, 2) = d(1, 4) = 1$, $d(1, 3) = d(1, 5) = 2$,

$d(2, 3) = d(2, 4) = d(2, 5) = 1$, $d(3, 4) = 2$,

$d(3, 5) = 1$ $d(4, 5) = 1$. Поэтому

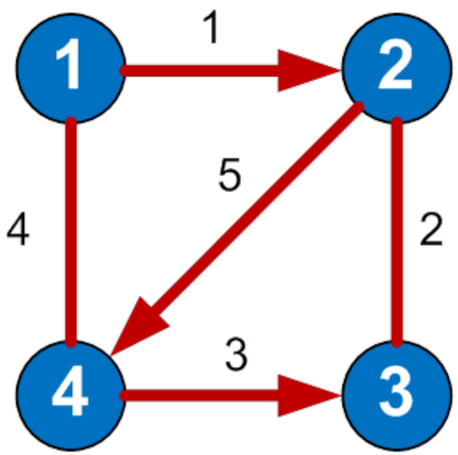
$$e(1) = e(3) = e(4) = e(5) = 2, \; e(2) = 1, \; d(G) = 2, \; r(G) = 1.$$

Вершина 2 является центром графа, а все остальные его вершины – периферийные. Цепь 1, 2, 3 – одна из диаметральных цепей.

Для связного орграфа расстояние $d(a, b)$ между вершинами a и b определяется как расстояние между вершинами a и b в неориентированном дубликате этого графа.

Представление графа в памяти компьютера

- Матрица смежности представляет собой таблицу, в которой номера столбцов и строк означают номера вершин графа. На пересечении строк и столбцов ставится 1, если вершины соединены ребром в графе, и 0, если не соединены.
- В матрице инциденций номера строк – номера вершин, а номера столбцов – номера ребер (дуг). На их пересечении ставится 1, если ребро и вершина инцидентны (т. е. ребро соединено с данной вершиной).
- Список смежности. Каждой вершине графа соответствует список, состоящий из «соседей» этой вершины.
- Список дуг (массив дуг/ребер). В первой строке которого хранится информация, из какой вершины начинается дуга, во второй - в какой кончается, а в третьей строке - вес дуги.


1
2
3
4
1
5
4
2
3

Затраты на выполнение операций обработки графов в худшем случае				
	Массив ребер	Матрица смежности	Списки смежности	Матрица инцидентности
Память	E	V^2	V+E	V*E
Инициализация пустого объекта	1	V^2	V	V*E
Вставка вершины	-	V^2	1	V*E
Удаление вершины	E	V^2	V+E	V*E
Вставка ребра	1	1	1	V*E
Поиск/удаление ребра	E	1	E	E / V*E
Вершина v изолирована?	E	V	1	E

Обход графа (поиск на графах)

- Под **обходом графов (поиском на графах)** понимается процесс *систематического* просмотра всех ребер или *вершин графа* с целью отыскания ребер или вершин, удовлетворяющих некоторому условию.
- При решении многих задач, использующих графы, необходимы эффективные методы регулярного обхода вершин и ребер графов. К стандартным и наиболее распространенным методам относятся:
  - *поиск в глубину* (Depth First Search, *DFS*);
  - *поиск в ширину* (Breadth First Search, *BFS*).

Поиск в глубину

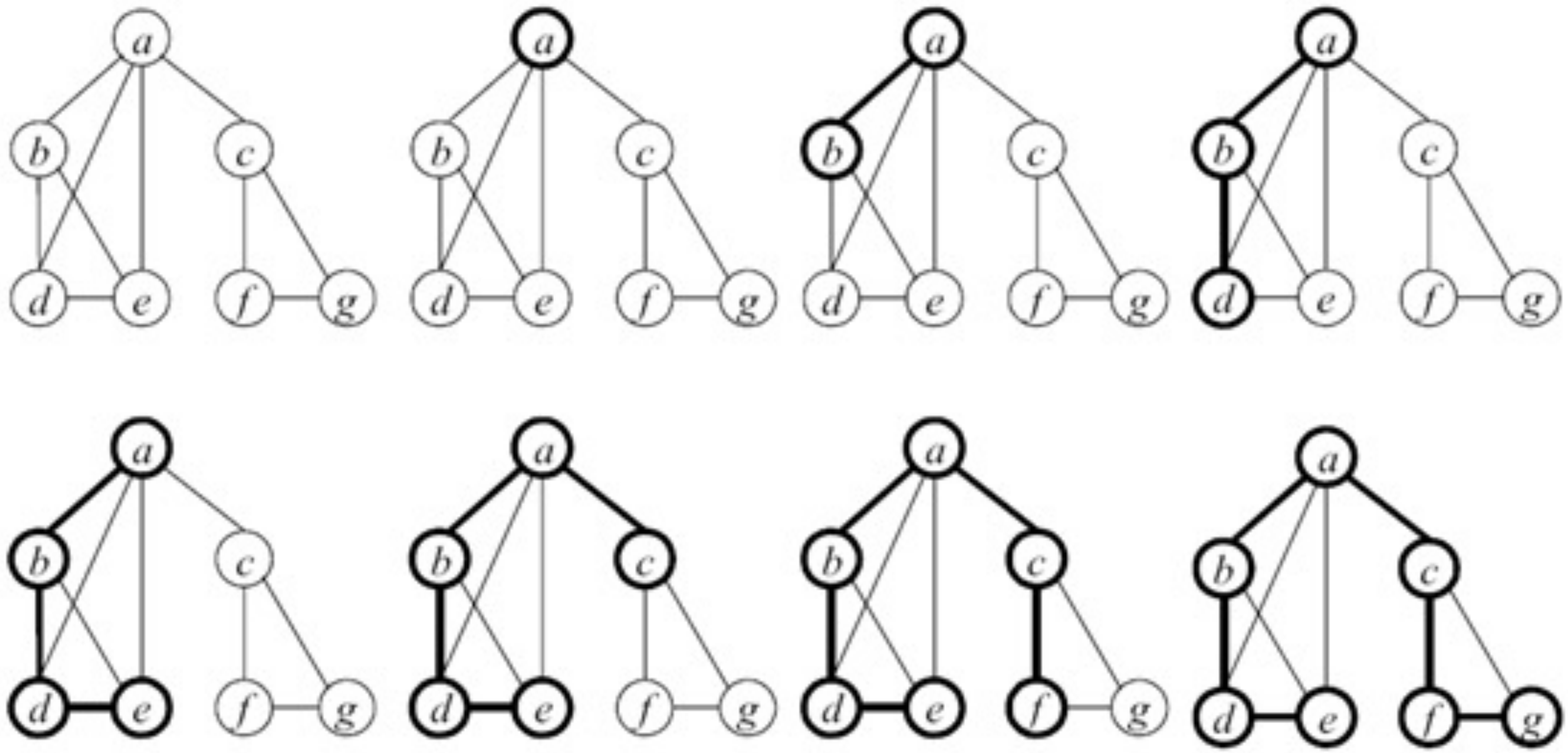
Идея поиска в глубину – когда возможные пути по ребрам, выходящим из вершин, разветвляются, нужно сначала полностью исследовать одну ветку и только потом переходить к другим веткам (если они останутся нерассмотренными).

Алгоритм поиска в глубину

- **Шаг 1.** Всем вершинам графа присваивается *значение* не посещенная. Выбирается первая *вершина* и помечается как посещенная.
- **Шаг 2.** Для последней помеченной как посещенная вершины выбирается смежная *вершина*, являющаяся первой помеченной как не посещенная, и ей присваивается *значение* посещенная. Если таких вершин нет, то берется предыдущая помеченная *вершина*.
- **Шаг 3.** Повторить шаг 2 до тех пор, пока все вершины не будут помечены как посещенные.

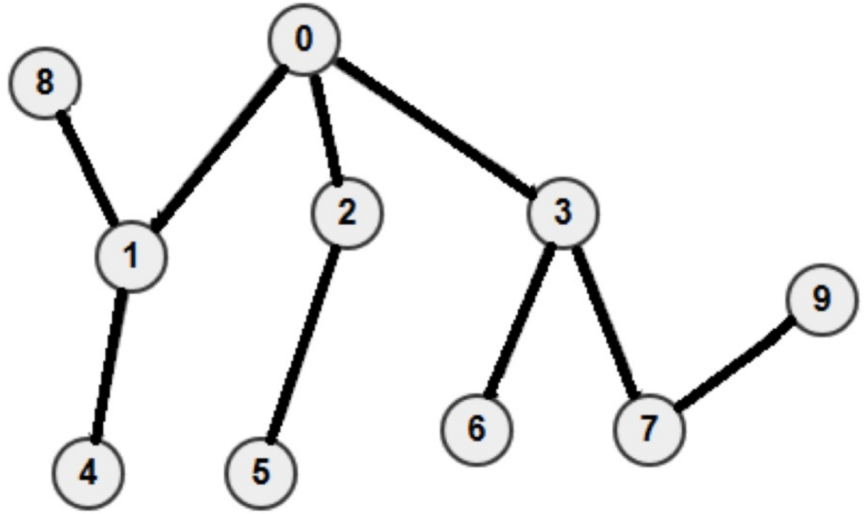
https://www.intuit.ru/studies/courses/648/504/lecture/11474?page=1

Пример



Рекурсивная реализация

Неориентированный граф G



Функция обхода графа в глубину DFS

```
void dfs(int v)
{
      used[v]=1;
       for (auto i : g[v])
            if(!used[i])
                  dfs(i);
}
```

Вызов из основного текста программы

```
dfs (0)
```

Программа обойдет вершины графа в следующем порядке: 0, 1, 4, 8, 2, 5, 3, 6, 7, 9.

Оценка сложности алгоритма включает в себя следующие операции:

- просмотр всех |V| вершин, для каждой из которых *v* просматриваются ее соседи;

- просмотр всех соседей вершины *v*. При этом алгоритм проходит по ребру {v, u}. Причем, каждое такое ребро {v, u} просматривается дважды: для вершины *u* и для вершины *v*.

Итоговая сложность алгоритма dfs, таким образом $O(|V| + |E|)$.

Не рекурсивная реализация

- Все вершины белые
- Выбираем $v \in V$
- Помечаем v черным и помещаем в некоторый контейнер T (стек)
- Пока T не пусто:
  - Извлекаем v из T
  - Производим некоторые действия над v (зависит от задачи)
  - $\forall v' \in \Gamma(v)$:
    - Если v' - белая, то помечаем v' черным и помещаем в T

Применение

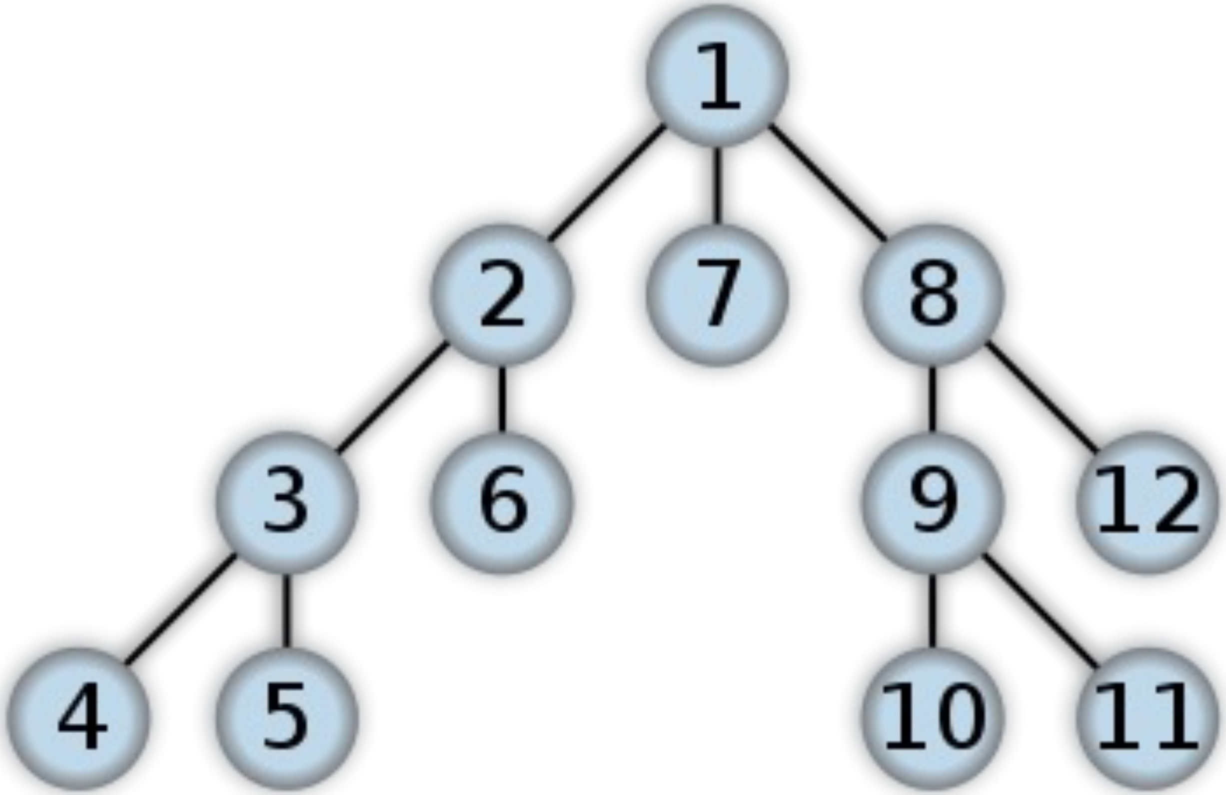
Поиск в глубину — хороший инструмент для исследования топологических свойств графов. Например:

- В качестве подпрограммы в алгоритмах поиска одно- и двусвязных компонент.
- В топологической сортировке.
- Для поиска точек сочленения, мостов.
- В различных расчётах на графах. Например, как часть алгоритма Диница поиска максимального потока.

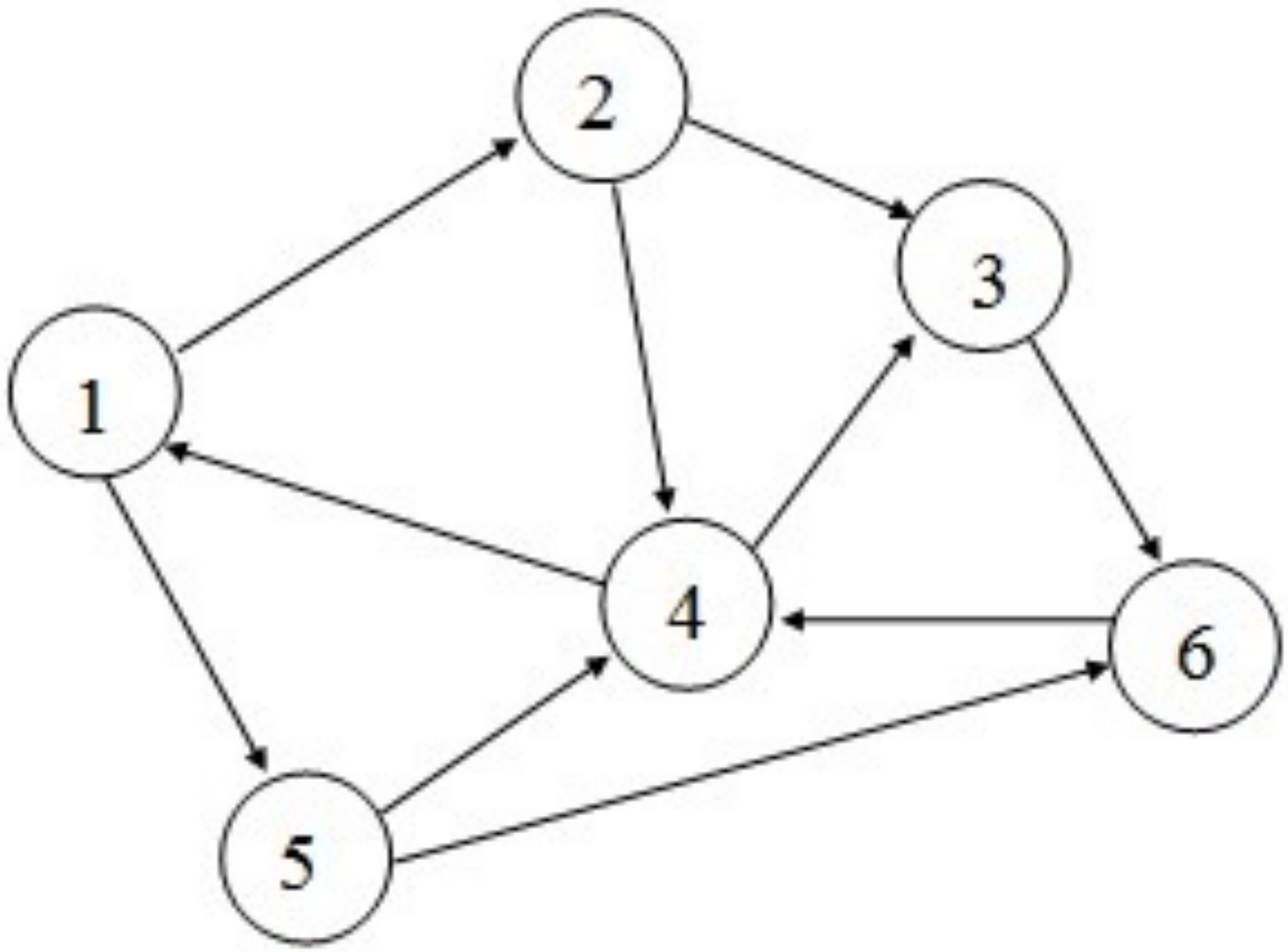
Реализации

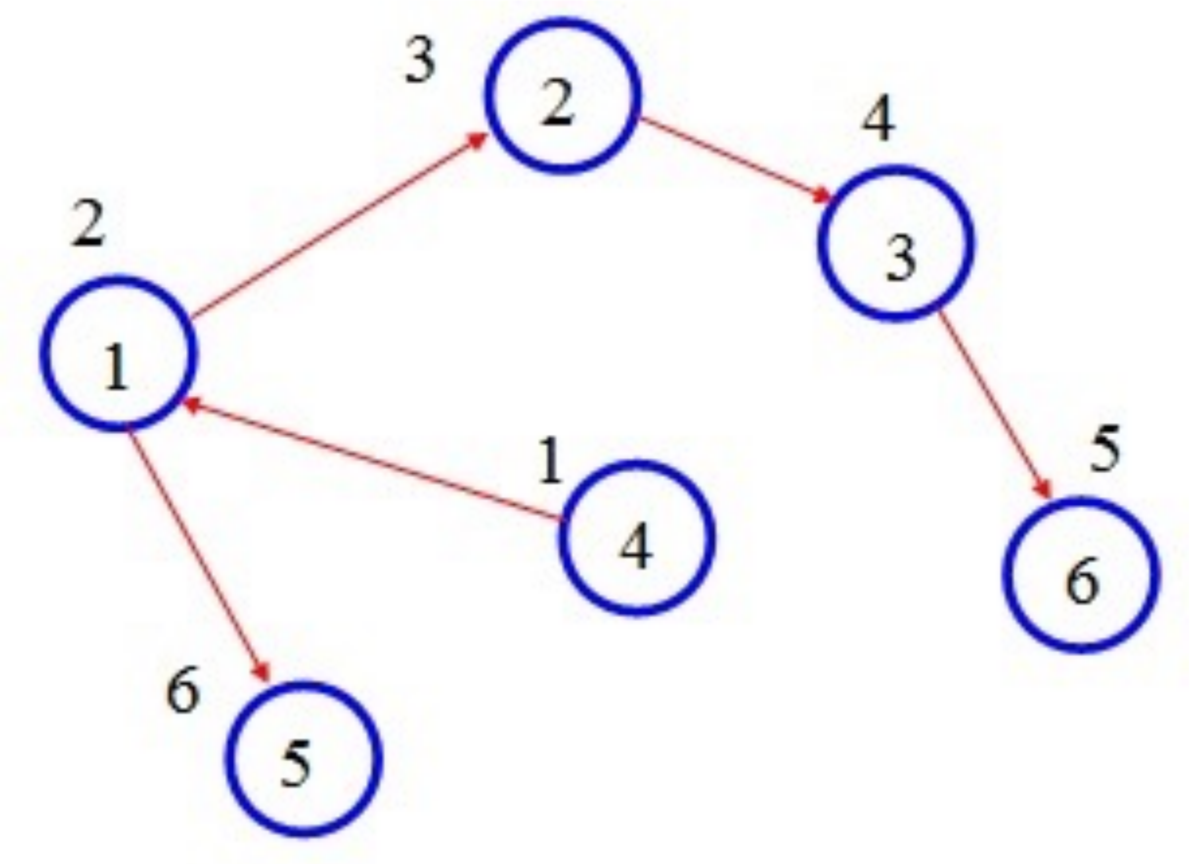
- https://ru.wikibooks.org/wiki/Реализации_алгоритмов/Поиск_в_глубину https://ru.wikipedia.org/wiki/Поиск_в_глубину

Пример



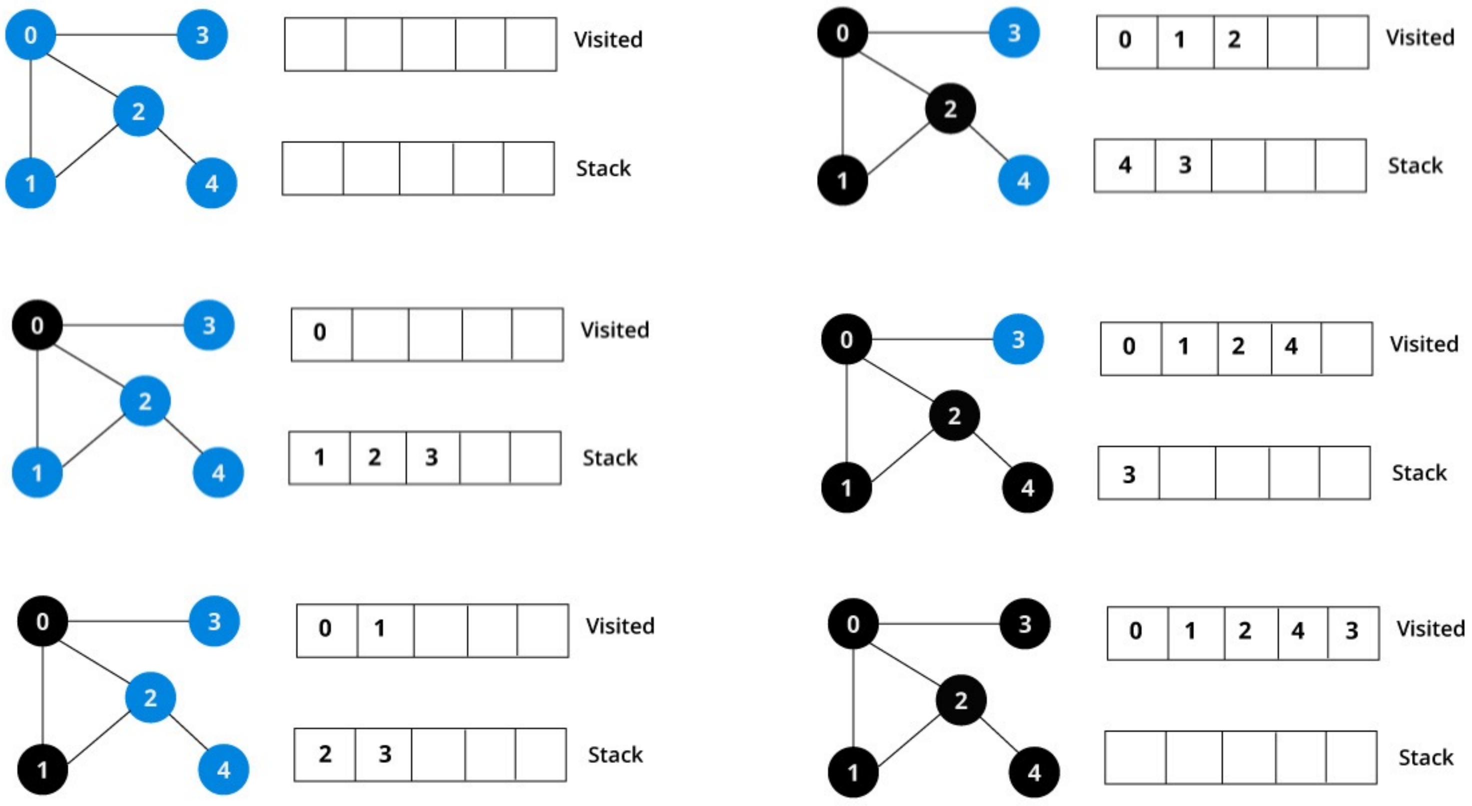
Пример




3
2
4
2
3
1
1
4
5
6
6
5

https://evileg.com/ru/post/494/

Пример работы не рекурсивной реализации



Поиск в ширину

Идея *поиска в ширину* заключается в том, что сначала исследуются все вершины, смежные с начальной вершиной (*вершина* с которой начинается обход). Эти вершины находятся на расстоянии 1 от начальной. Затем исследуются все вершины на расстоянии 2 от начальной, затем все на расстоянии 3 и т.д. Обратим внимание, что при этом для каждой вершины сразу находятся *длина* кратчайшего маршрута от начальной вершины.

Не рекурсивная реализация

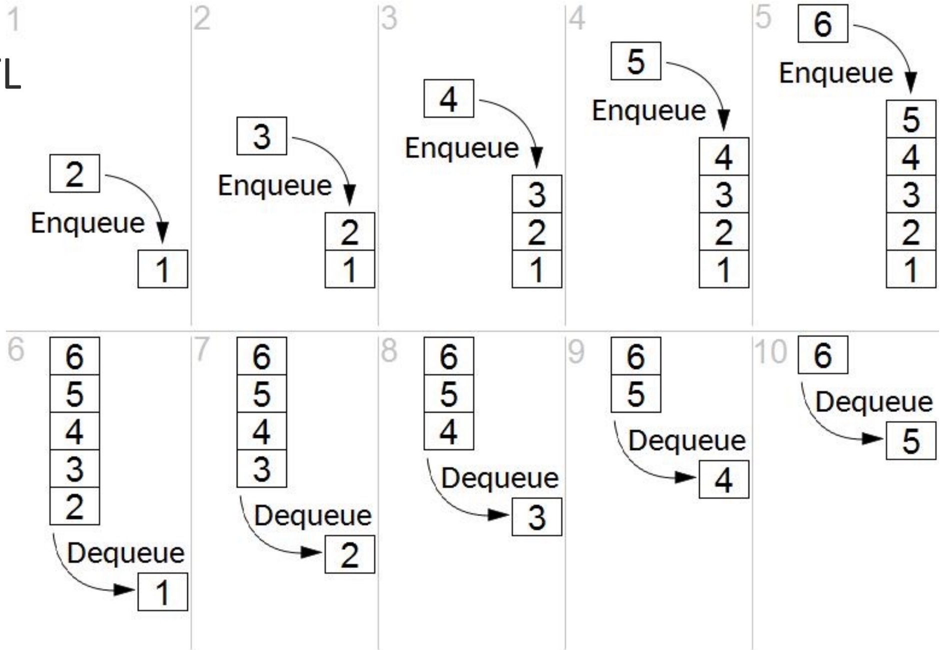
- Все вершины белые
- Выбираем $v \in V$
- Помечаем v черным и помещаем в некоторый контейнер T (очередь)
- Пока T не пусто:
    - Извлекаем v из T
    - Производим некоторые действия над v (зависит от задачи)
    - $\forall v' \in \Gamma(v)$:
        - Если v' - белая, то помечаем v' черным и помещаем в T

http://www.8bytes.net/2017/10/24/struktura-dannyh-ochered-fifo/

Queue

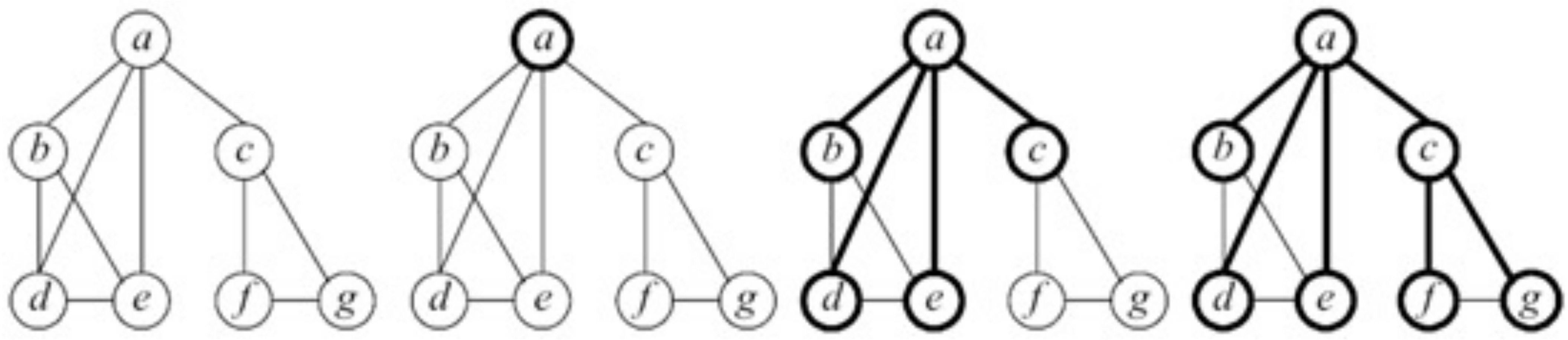
- **Очередью** (англ. – queue) называется структура данных, из которой удаляется первым тот элемент, который был первым в очередь добавлен. То есть очередь в программировании соответствует «бытовому» понятию очереди. Очередь также называют структурой типа FIFO (first in, first out — первым пришел, первым ушел).

В C++ уже есть готовый STL контейнер — queue.

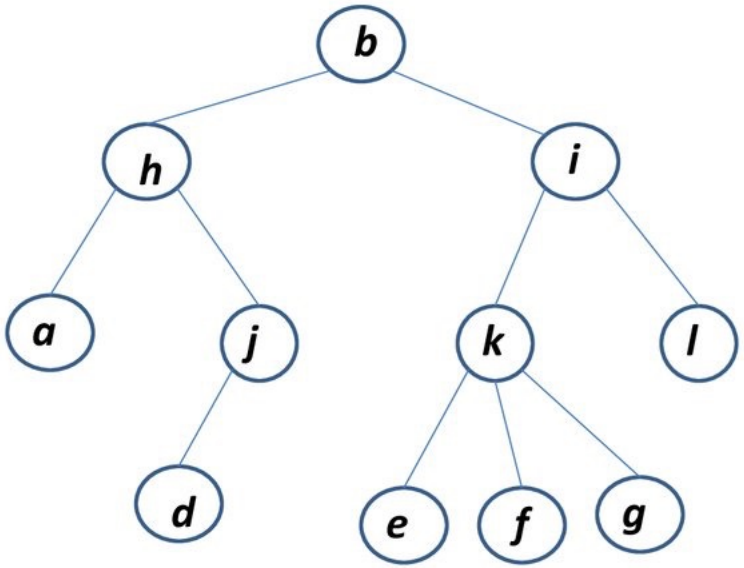


https://www.intuit.ru/studies/courses/648/504/lecture/11474?page=1

Пример

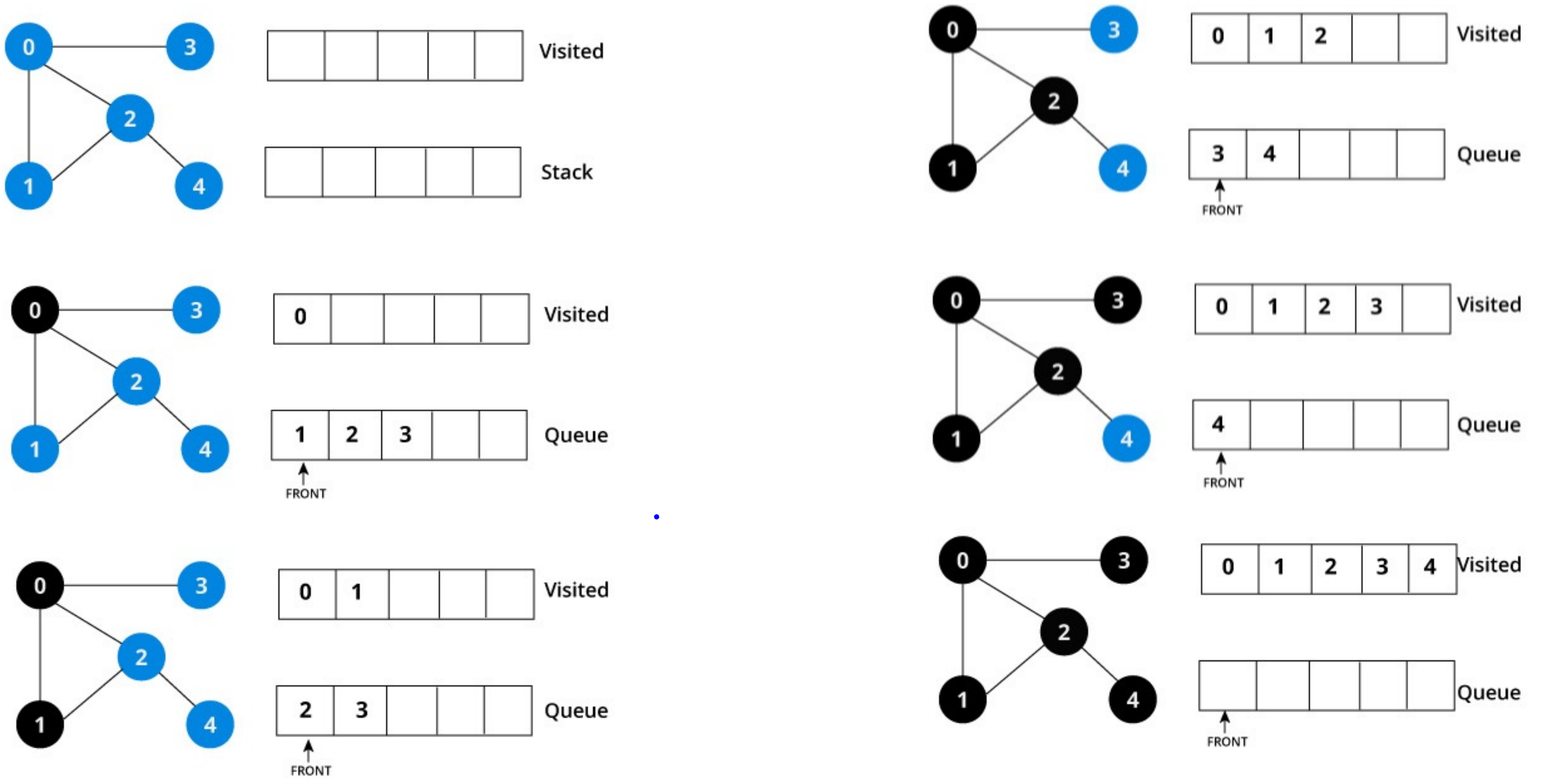


Пример



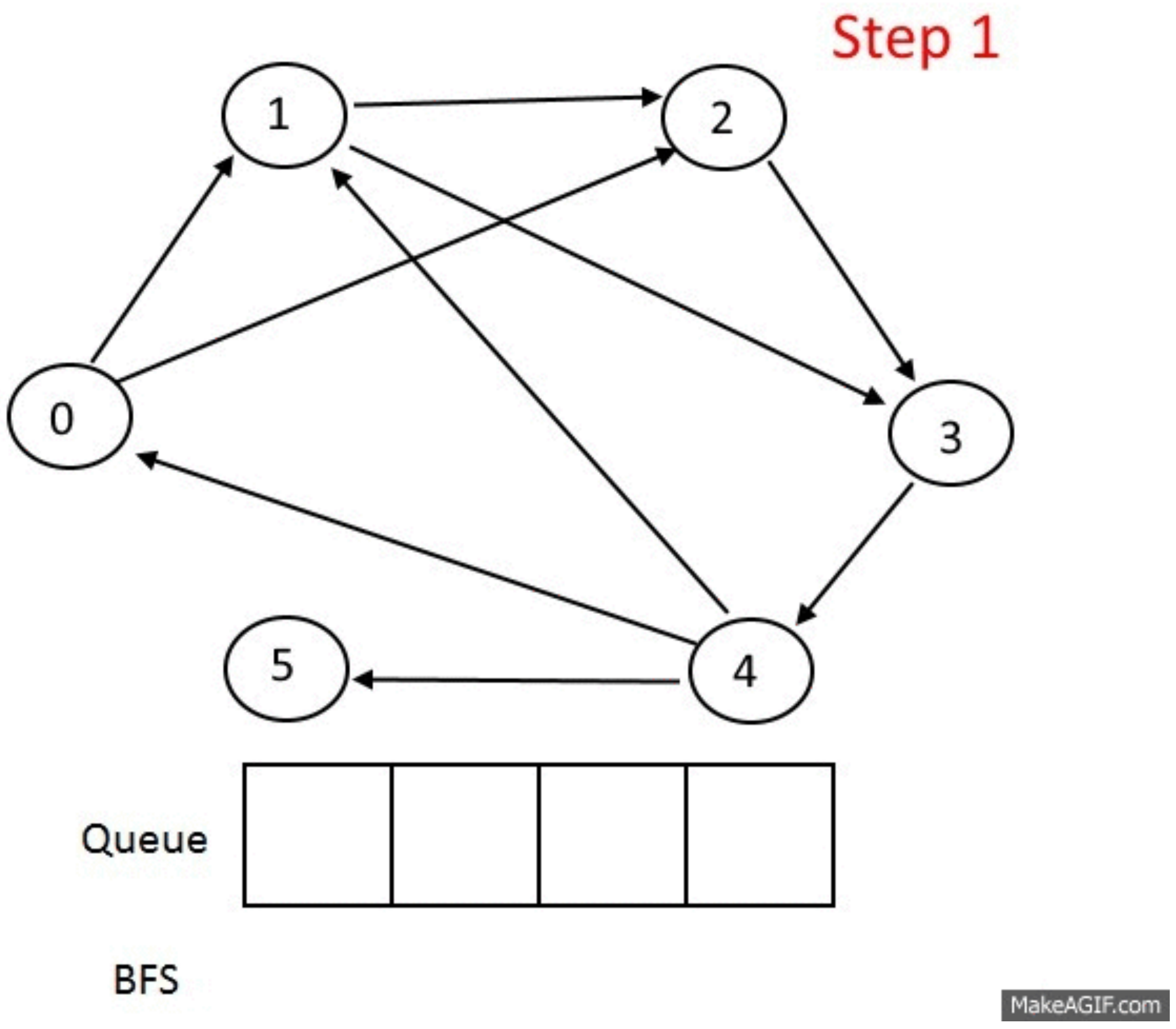
b	*h*	*i*	*a*	*j*	*k*	*l*	*d*	*e*	*f*	*g*

Пример


0
3
2
1
4
Visited
Stack
0
3
2
1
4
0
Visited
1 2 3
Queue
FRONT
0
3
2
1
4
0 1
Visited
2 3
Queue
FRONT
0
3
2
1
4
0 1 2
Visited
3 4
Queue
FRONT
0
3
2
1
4
0 1 2 3
Visited
4
Queue
FRONT
0
3
2
1
4
0 1 2 3 4
Visited
Queue
FRONT

https://neerc.ifmo.ru/wiki/index.php?title=Обход_в_ширину

Пример



Применение

Поиск в ширину может применяться для решения задач, связанных с теорией графов:

- Волновой алгоритм поиска пути в лабиринте
- Волновая трассировка печатных плат
- Поиск компонент связности в графе
- Поиск кратчайшего пути между двумя узлами невзвешенного графа
- Поиск в пространстве состояний: нахождение решения задачи с наименьшим числом ходов, если каждое состояние системы можно представить вершиной графа, а переходы из одного состояния в другое — рёбрами графа
- Нахождение кратчайшего цикла в ориентированном невзвешенном графе
- Нахождение всех вершин и рёбер, лежащих на каком-либо кратчайшем пути между двумя вершинами
- Поиск увеличивающего пути в алгоритме Форда-Фалкерсона (алгоритм Эдмондса-Карпа)

https://repl.it/@MariiaGordienko/DFSBFS#main.cpp

https://visualgo.net/en/dfsbfs