Метод поиска в глубину

Лекция 9

Поиск в глубину (*Depth-first search, DFS*)

Пусть задан граф $G = (V, E)$.

Алгоритм поиска описывается следующим образом:

> для каждой непройденной вершины необходимо найти все непройденные смежные вершины и повторить поиск для них.

Пусть в начальный момент времени все вершины окрашены в *белый* цвет.

1. Из множества всех *белых* вершин выберем любую вершину: $v1$.
2. Выполним для нее процедуру Поиск($v1$).
3. Перекрасим ее в *черный* цвет.

Повторяем шаги 1-3 до тех пор, пока множество *белых* вершин не пусто.

Процедура Поиск(u)

```
Поиск (u)
{
   цвет [u] ← серый;
   d[u] = time++; // время входа в вершину,
                  // порядковый глубинный номер вершины
   для ∀ v ∈ смежные(u)  выполнить
    {
      если (цвет[v] = белый) то
      {
         отец [v] ← u;
         Поиск(v);
      }
    }
    цвет[u] ← чёрный;
    f [u] ← time++;  // время выхода из вершины
}
```

Процедура DFS(G)

```
DFS(G)
{
  для ∀u ∈ V выполнить
  {
     цвет [u] ← белый;
     отец [u] ← NULL;
  }
  time ← 0;
  для ∀u ∈ V выполнить
     если (цвет [u] = белый) то
          Поиск(u);
}
```

Анализ

Общее число операций при выполнении *DFS*:

$$O(|V|)$$

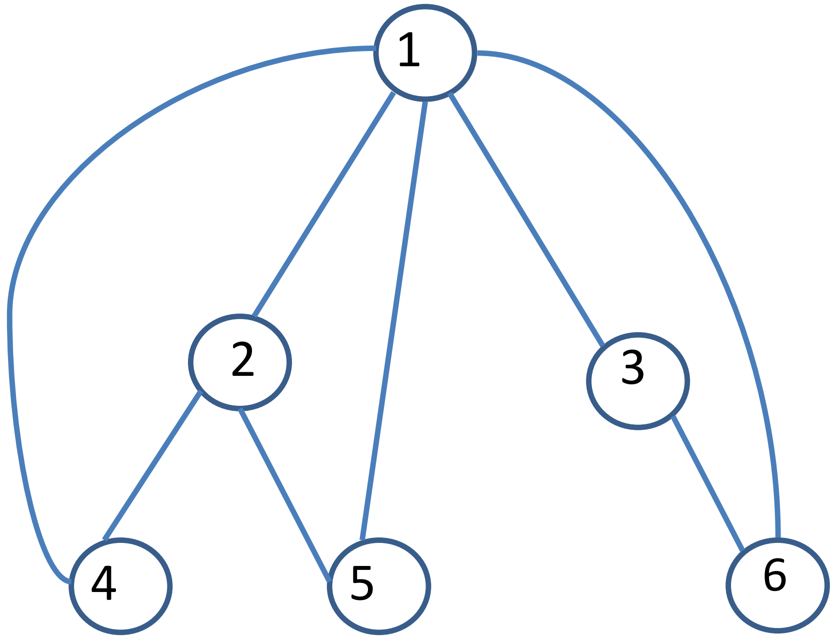
Общее число операций при выполнении *Поиск(u):*

Цикл выполняется |смежные[v]| раз.

$$\sum |\text{смежные}[v]| = O(|E|)$$

Общее число операций: **$O(|V|+|E|)$**

Поиск в глубину в неориентированном графе



Глубинный остовный лес

Поиск в глубину на неориентированном графе *G*= (*V, E*) разбивает ребра, составляющие *E,* на два множества *T* и *B.*

Ребро *(v, w)* помещается в множество *T,* если узел *w* не посещался до того момента, когда мы, рассматривая ребро *(u, w),* оказались в узле *v.* В противном случае ребро (*v*, *w*) помещается в множество *B.*

Ребра из *T* будем называть *древесными*, а из В — *обратными*.

Подграф *(V, T)* представляет собой неориентированный лес, называемый *остовным лесом для G, построенным поиском в глубину,* или, короче, *глубинным остовным лесом для G.*

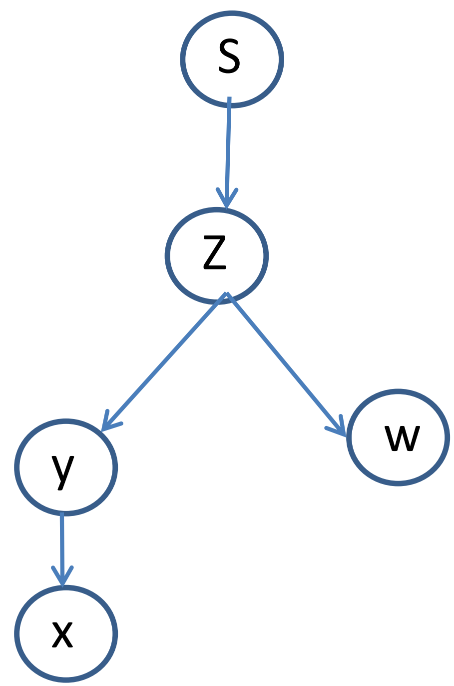
Если этот лес состоит из единственного дерева, *(V, T)* будем называть по аналогии *глубинным остовным деревом.*

Заметим, что если граф связен, то глубинный остовный лес будет деревом.

Узел, с которого начинался поиск, считается корнем соответствующего дерева.

Свойства поиска в глубину

Времена обнаружения и окончания обработки вершин образуют правильную скобочную структуру.




```
 1  2  3   4  5  6   7   8   9   10
(s (z (y (x  x) y) (w  w)  z)  s)
```

Теорема

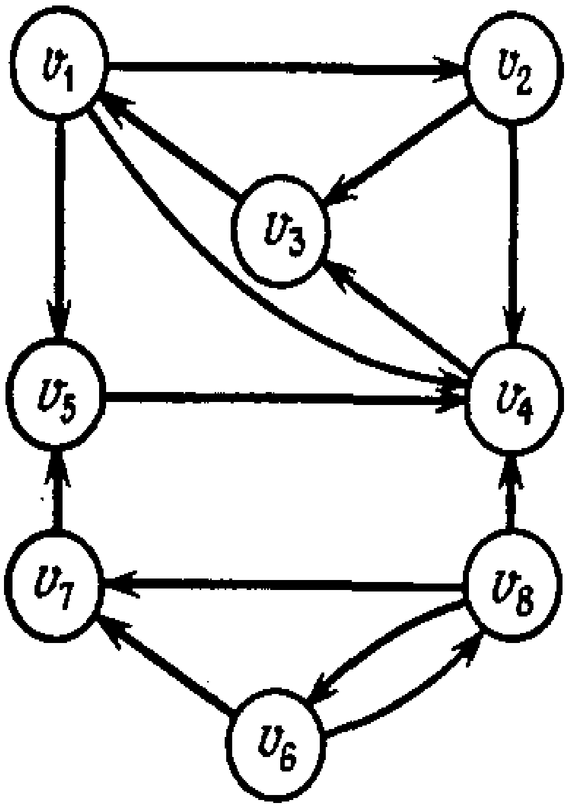
При поиске в глубину в графе $G = (V, E)$ для любых двух вершин u и v выполняется одно из следующих утверждений:

1) Отрезки $[d[u],f[u]]$ и $[d[v],f[v]]$ не пересекаются.

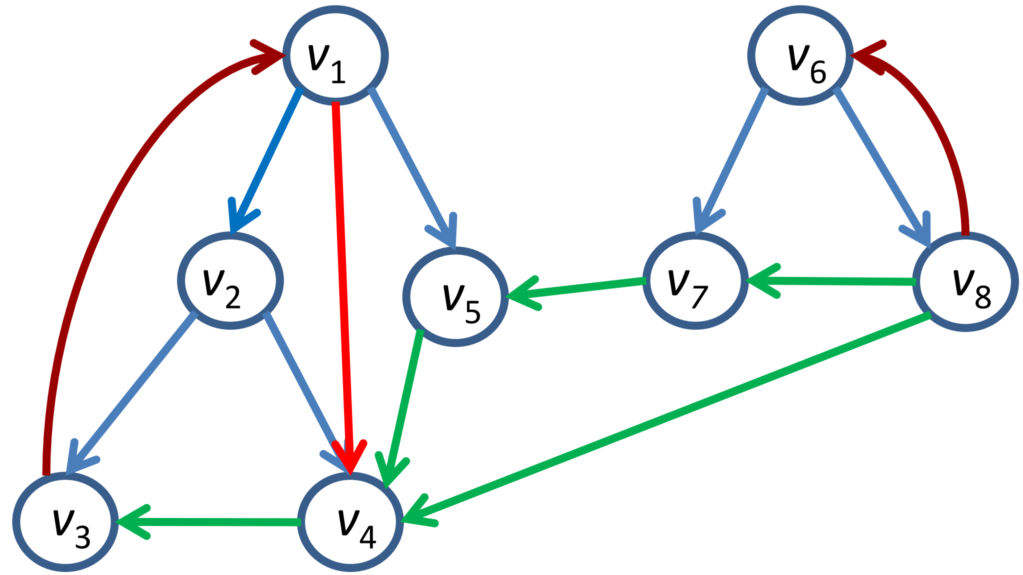
2) Отрезок $[d[u],f[u]]$ целиком содержится внутри отрезка $[d[v],f[v]]$ и u есть потомок v в дереве поиска в глубину.

3) Отрезок $[d[v],f[v]]$ целиком содержится внутри отрезка $[d[u],f[u]]$ и v есть потомок u в дереве поиска в глубину.

Поиск в глубину в ориентированном графе



a



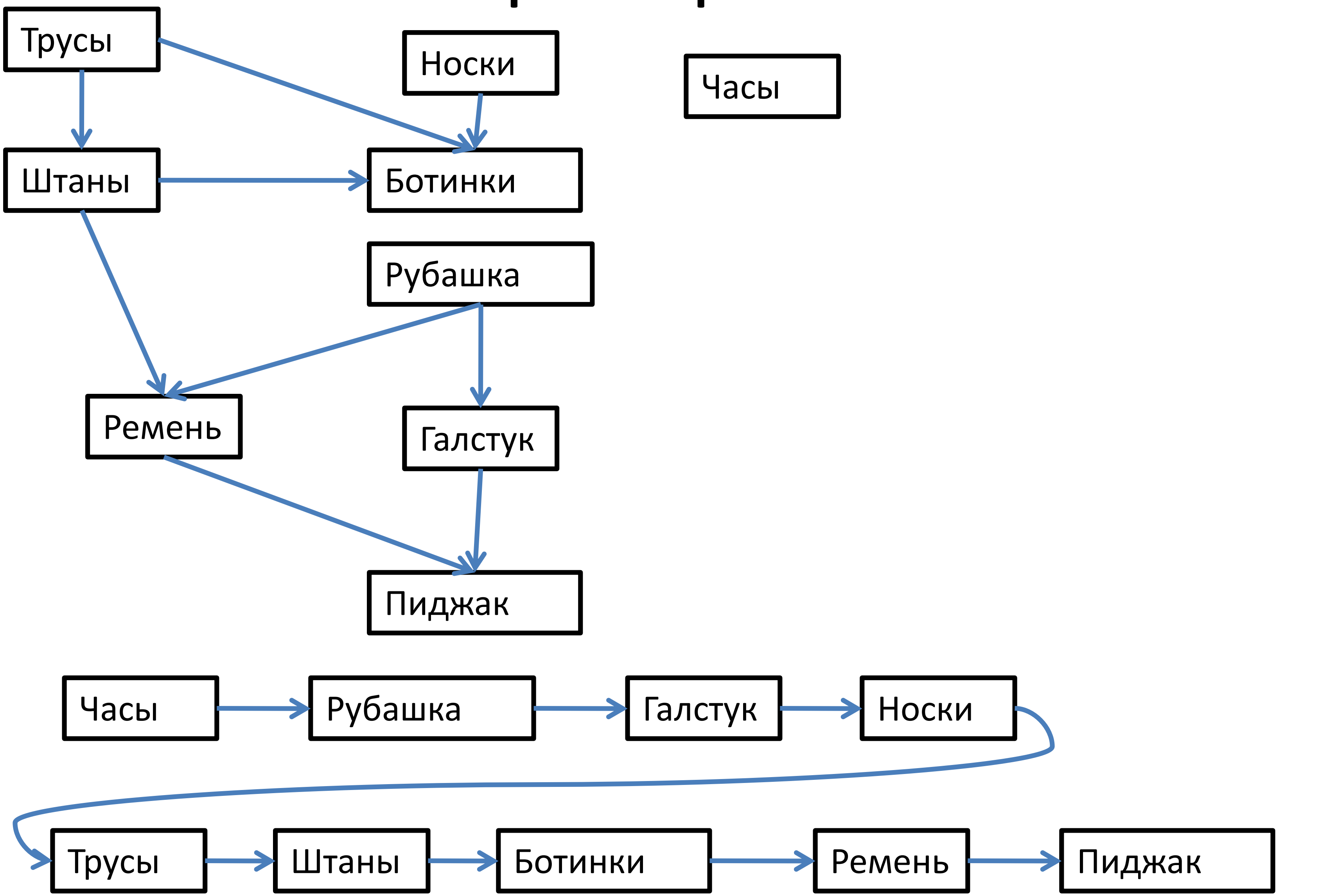
Поиск в глубину в ориентированном графе G разбивает множество его ребер на четыре класса.

1) *Древесные ребра*, идущие к новым узлам в процессе поиска.
2) *Прямые ребра*, идущие от предков к подлинным потомкам, но не являющиеся древесными ребрами.
3) *Обратные ребра*, идущие от потомков к предкам (возможно, из узла в себя).
4) *Поперечные ребра*, соединяющие узлы, которые не являются ни предками, ни потомками друг друга.

Решение задачи топологической сортировки методом поиска в глубину

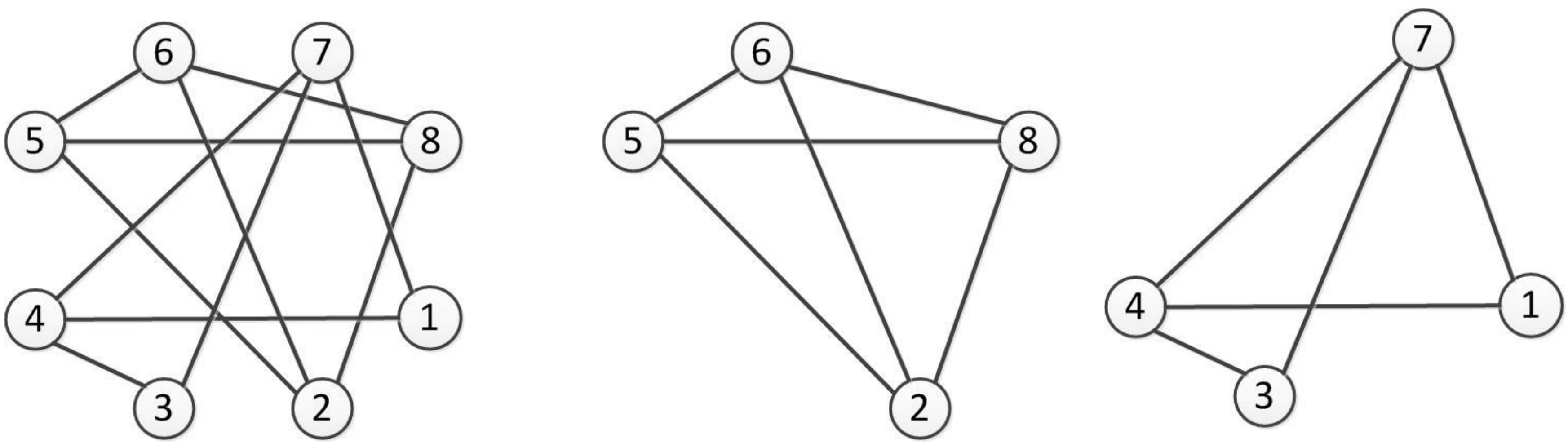
```
Топологическая_сортировка (u)
{
   цвет [u] ← серый;
   для ∀ v ∈ смежные(u)   выполнить
   {
      если (цвет[v] = белый) то
      {
         Топологическая_сортировка(v);
      }
    }
    цвет[u] ← чёрный;
    Поместить u в начало списка;
}
```

Пример


Трусы
Носки
Часы
Штаны
Ботинки
Рубашка
Ремень
Галстук
Пиджак
Часы
Рубашка
Галстук
Носки
Трусы
Штаны
Ботинки
Ремень
Пиджак

Поиск компонент связности в неориентированном графе

Компонента связности графа – это такое множество вершин неориентированного графа, что для любых двух вершин из этого множества существует путь из одной в другую, и не существует пути из вершины этого множества в вершину не из этого множества.



Реализация поиска компонент связности в графе

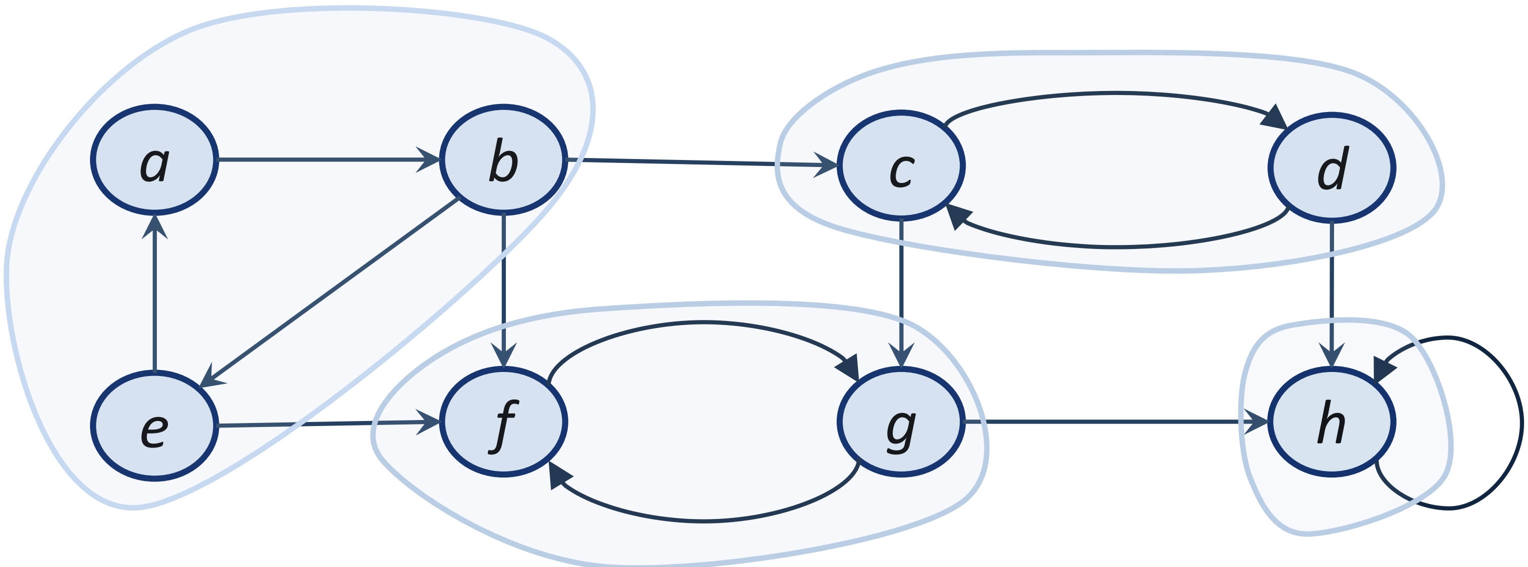
```
Поиск (u, n)
{
  цвет [u] ← серый;
   C[u] ← n;          // номер компоненты связности
   для ∀ v ∈ смежные(u) выполнить
   {
      если (цвет[v] = белый) то
         Поиск(v, n);
   }
   цвет[u] ← чёрный;
}
DFS(G)
{
   для ∀u ∈ V выполнить
       цвет [u] ← белый;
  nk ← 0;
  для ∀u ∈ V выполнить
     если (цвет [u] = белый) то
     {
       nk ++;
       Поиск(u, nk);
     }
}
```

Разложение ориентированного графа на компоненты сильной связности

Компоненты сильной связности (КСС)

Сильно связной компонентой графа $G = (V, E)$ называется максимальное множество вершин $C \subseteq V$, такое что для каждой пары вершин u и v из C справедливо, что как u достижимо из v, так и v достижимо из u.

Пример



Сильно связные компоненты графа

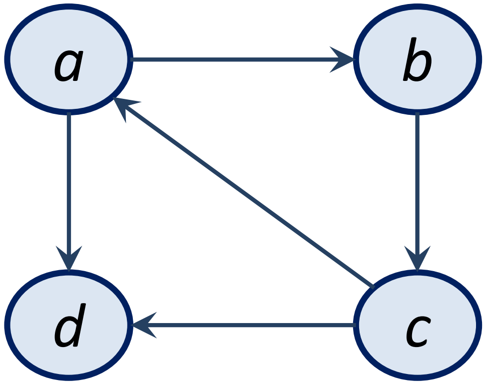
Алгоритм поиска КСС

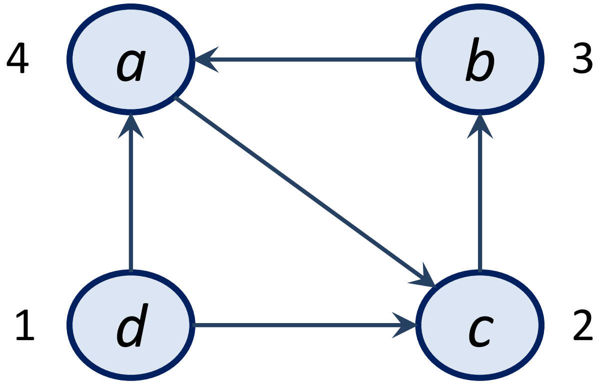
Пусть $G = (V, E)$ – ориентированный граф,
$G^T = (V, E^T)$, где $E^T = \{(u, v) : (v, u) \in E\}$ – транспонированный граф G.

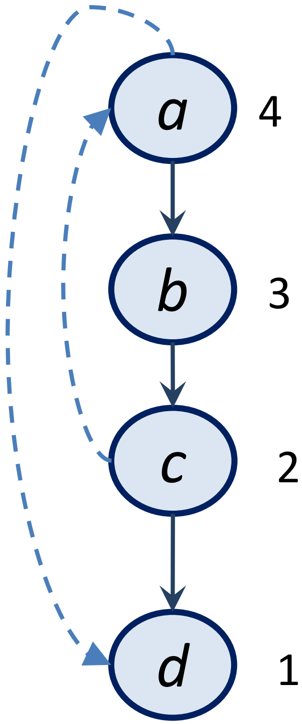
Strongly_Connected_Components(*G*)

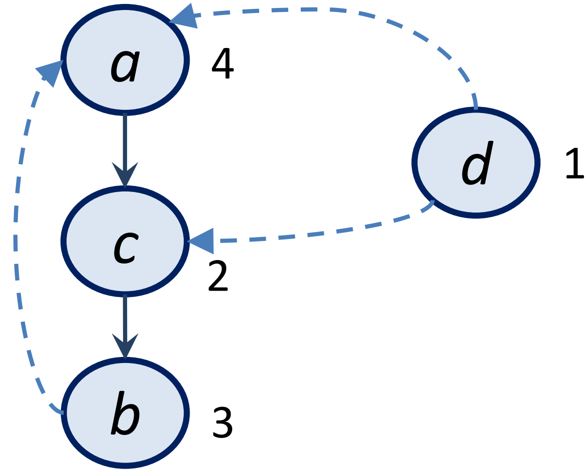
1. Вызов *DFS*(*G*) для вычисления времен завершения $f[u]$ для каждой вершины.
2 Построение G^T.
3. Вызов *DFS*(G^T), но в главном цикле процедуры *DFS*, вершины рассматриваются в порядке убывания значений $f[u]$, вычисленных в строке 1.
4. Деревья глубинного остовного леса, полученного в строке 3, представляют собой сильно связные компоненты.

Пример работы алгоритма









Двусвязность

Определения

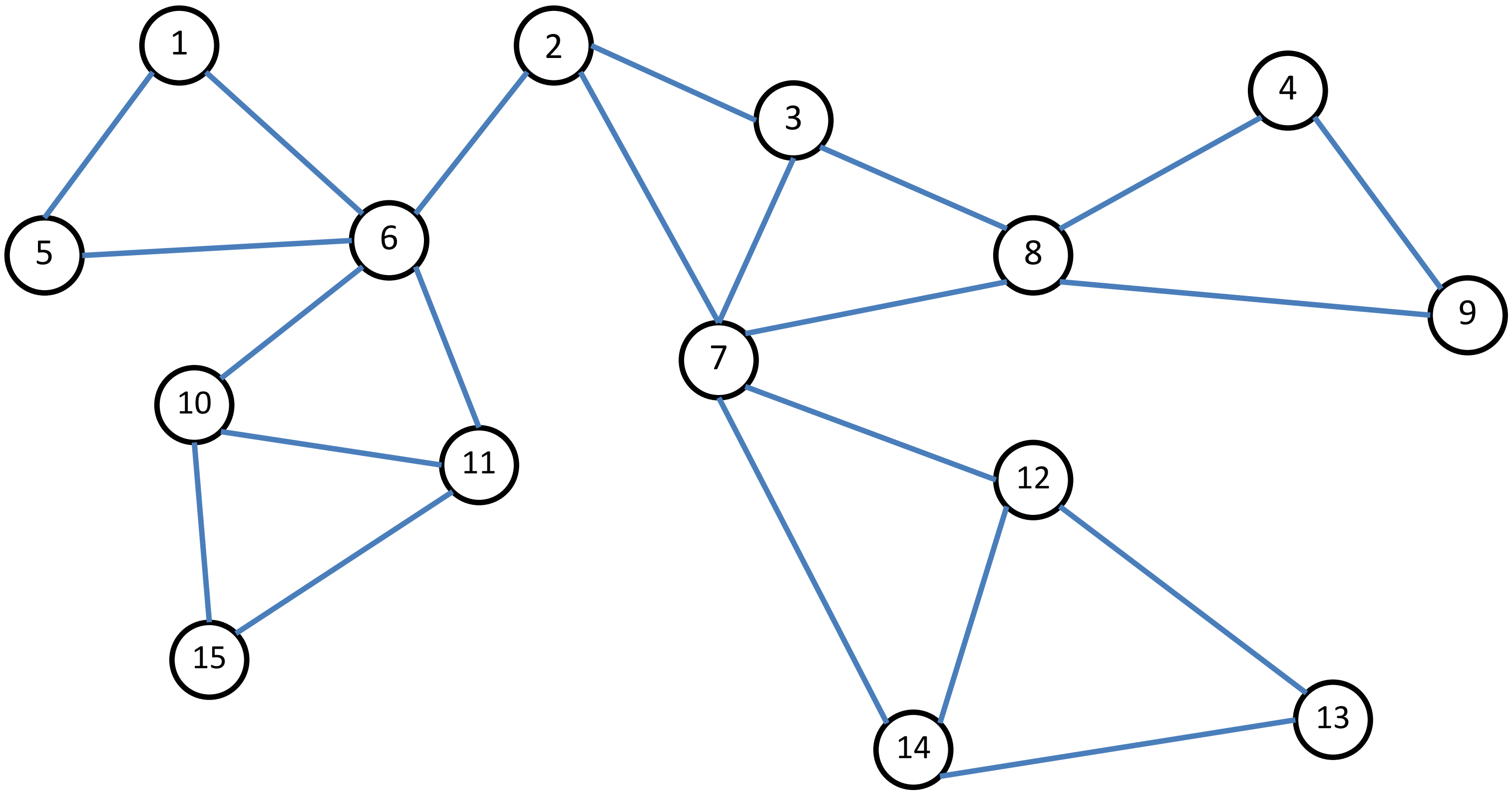
Пусть $G = (V, E)$ — связный неориентированный граф. Узел a называют *точкой сочленения* графа G, если существуют такие узлы v и w, что v, w и a различны и всякий путь между v и w содержит узел a.

Иначе говоря, a — точка сочленения графа G, если удаление узла a расщепляет G не менее чем на две части.

Граф G называется *двусвязным*, если для любой тройки различных узлов v, w, a существует путь между v и w, не содержащий a.

Таким образом, неориентированный связный граф двусвязен тогда и только тогда, когда в нем нет точек сочленения.

Точки сочленения. Пример



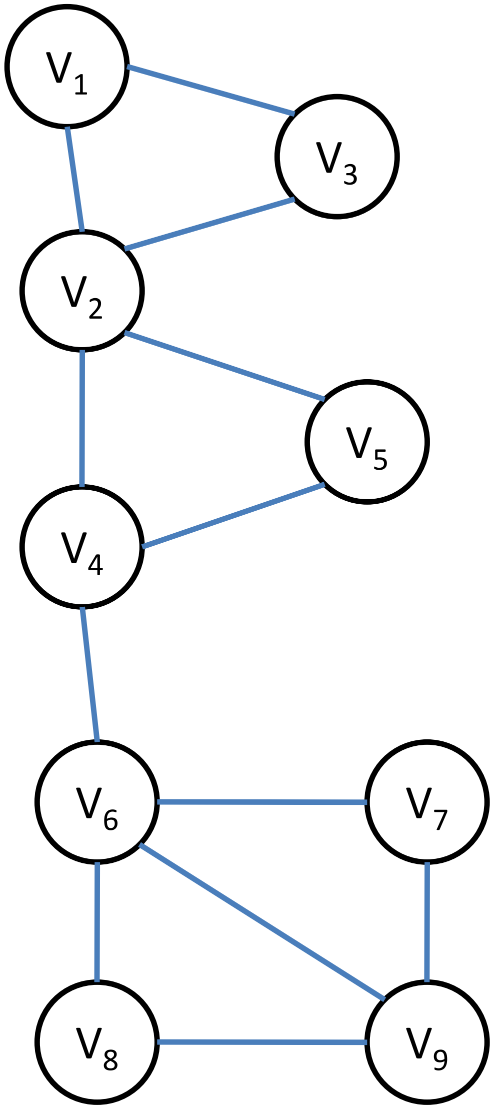
Двусвязные компоненты

На множестве ребер графа G можно задать естественное отношение, полагая, что для ребер e_1 и e_2 выполняется это отношение, если $e_1 = e_2$ или они лежат на некотором цикле.

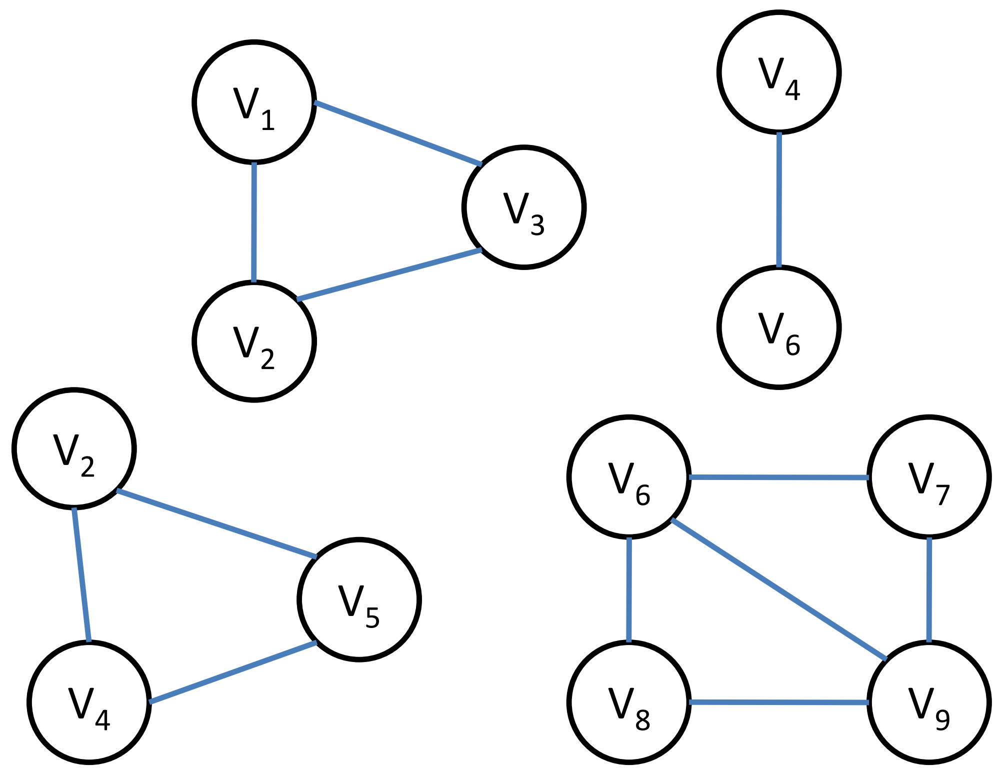
Легко показать, что это отношение является отношением эквивалентности (R называется *отношением эквивалентности* на множестве S, если R *рефлексивно* (aRa для всех $a \in S$), *симметрично* (из aRb следует bRa для всех $a, b \in S$) и *транзитивно* (из aRb и bRc следует aRc)), разбивающим множество ребер графа G на такие классы эквивалентности $E_1, E_2, \ldots, E_k$, что два различных ребра принадлежат одному и тому же классу тогда и только тогда, когда они лежат на общем цикле.

Для $1 \leq i \leq k$ обозначим через V_i множество узлов, лежащих на ребрах из E_i . Каждый граф $G_i = (V_i, E_i)$ называется *двусвязной компонентой* графа G.

Двусвязные компоненты. Пример



$$E_1 = \{ (v_1, v_2), (v_1, v_3), (v_2, v_3)\},$$
$$E_2 = \{ (v_2, v_4), (v_2, v_5), (v_4, v_5)\},$$
$$E_3 = \{ (v_4, v_6)\},$$
$$E_4 = \{ (v_6, v_7), (v_6, v_8), (v_6, v_9), (v_7, v_9), (v_8, v_9)\}$$



Лемма **1.**

Пусть $G_i=(V_i, E_i)$ *для* $1 \le i \le k$ — двусвязные компоненты связного неориентированного графа $G = (V, E)$.

Тогда

1) граф G_i двусвязен для каждого i, $1 \le i \le k$;

2) для всех $i \neq j$ пересечение $V_i \cap V_j$ содержит не более одного узла;

3) a — точка сочленения графа G тогда и только тогда, когда $a \in V_i \cap V_j$ для некоторых $i \neq j$.

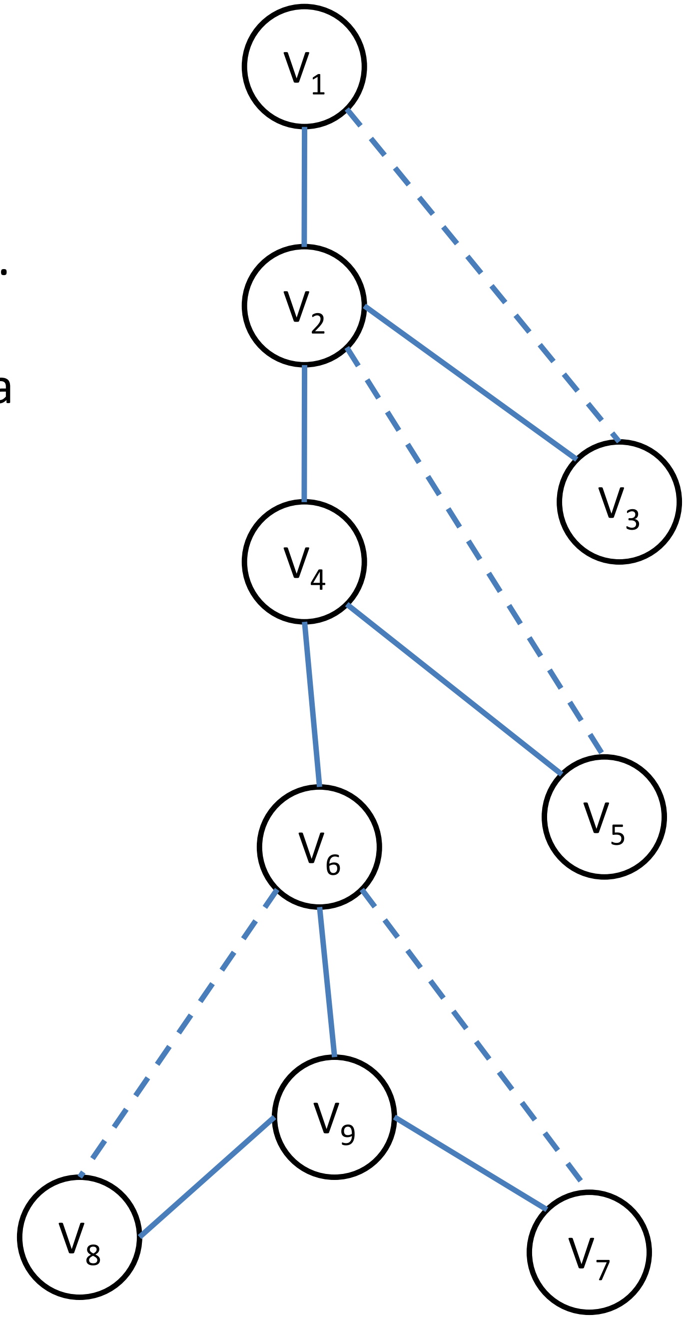
Лемма 2.

Пусть G = (V, E) — связный неориентированный граф,

S = (V, T) —глубинное остовное дерево для него.

Узел a является точкой сочленения графа G тогда и только тогда, когда выполнено одно из условий:

1. a — корень и a имеет более одного сына;
2. a — не корень и для некоторого его сына s нет обратных ребер между потомками узла s (в том числе самим s) и подлинными предками узла a.



Нахождение двусвязных компонент и точек сочленения методом поиска в глубину

1. Для всех вершин v вычисляются числа $dfnumber[v]$. Они фиксируют последовательность обхода вершин глубинного остовного дерева в прямом порядке.
2. Для каждой вершины v вычисляется число

$$low[v] = \min \begin{cases} dfnumber[v]; \\ dfnumber[z],\ (v, z) \text{ – обратное ребро;} \\ low[x],\ x \text{ – потомок } v. \end{cases}$$

3. Точки сочленения определяются следующим образом:
   - корень остовного дерева будет точкой сочленения тогда и только тогда, когда он имеет двух и более сыновей;
   - вершина v, отличная от корня, будет точкой сочленения тогда и только тогда, когда имеет такого сына w, что

$$low[w] \geq dfnumber\,[v].$$

Алгоритм нахождения двусвязных компонент и точек сочленения

Вход. Связный неориентированный граф G= *(V, E).*

Выход. Список ребер каждой двусвязной компоненты графа *G*.

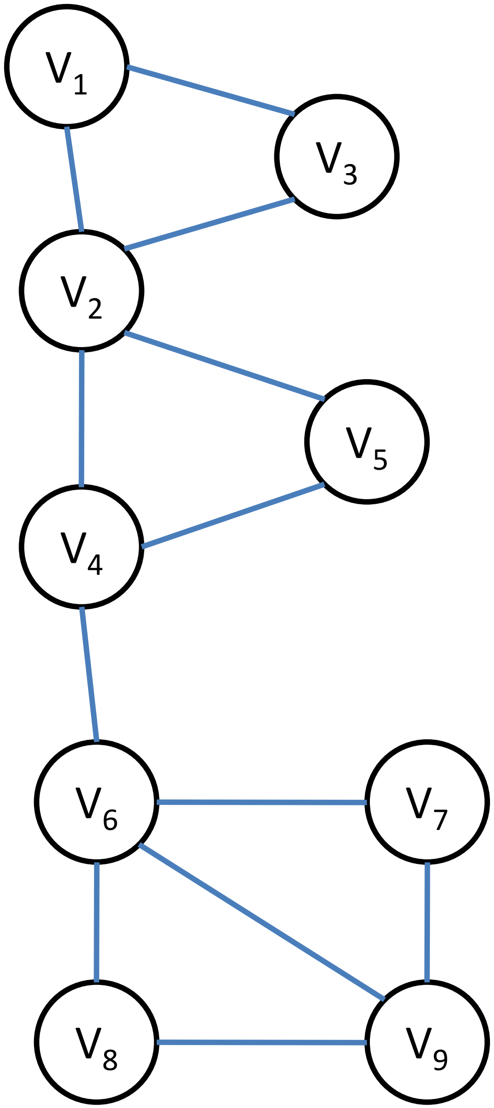
Метод.

Вначале полагаем $T=\varnothing$ и СЧЕТ=1. Помечаем каждый узел в *V* как "белый", выбираем произвольный узел v_0 в *V,* отец[v_0] = 0 и вызываем Поиск_дк(v_0), чтобы построить глубинное остовное дерево S=(*V,T*) и вычислить *low*[v] для каждого *v* из *V.*

Процедура Поиск_дк(v)

```
Поиск_дк(v)
{    цвет [v] ← серый;    dfnumber[v] ← СЧЕТ;   СЧЕТ ← СЧЕТ+1;
     low[v] ← dfnumber[v] ;
     для ∀ w ∈ смежные(v)  выполнить
     {        если (цвет[w] = белый) то
              {     поместить  (v, w) в СТЕК;    добавить (v, w) к T;     отец [w] ← v;
                    Поиск_дк (w);
                    если low[w] ≥ dfnumber[v]  то
                    {       обнаружена двусвязная компонента:
                            вытолкнуть из СТЕКа все ребра вплоть до ребра (v, w) ;
                    }
                    low[v] ←  min ( low[v], low[w]);
              }
              иначе
                    если (w ≠ отец[v]) то
                    {       если (dfnumber[w] < dfnumber[v] ) то
                            {    поместить  (v, w) в СТЕК;
                                 low[v] ← min ( low[v], dfnumber[w] ) }
                    }
     }
      цвет[v] ← чёрный;
}
```

Пример



low[1]= 1

low[2]= 2 1

low[3]= 3 2

low[4]= 4

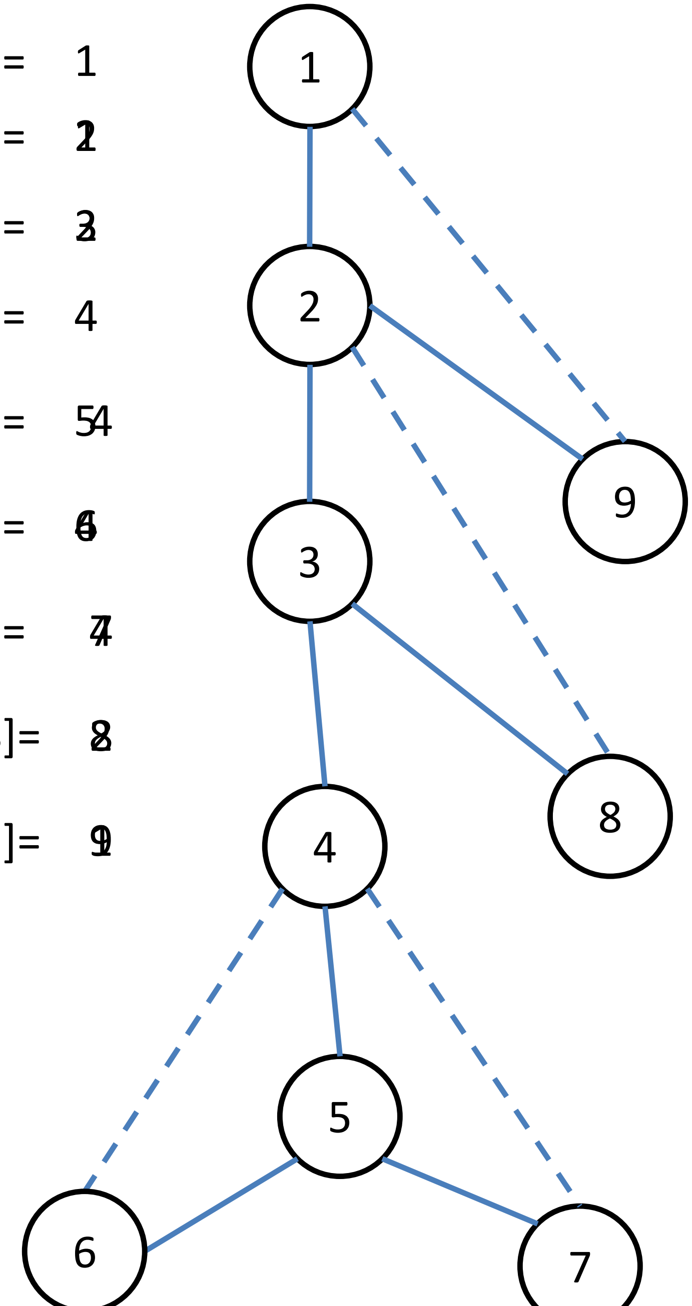
low[5]= 5 4

low[6]= 6 4

low[7]= 7 4

low[8]= 8 2

low[9]= 9 1



Стек

(V_7, V_6)
(V_9, V_7)
(V_8, V_6)
(V_9, V_8)
(V_6, V_9)
(V_4, V_6)
(V_2, V_4)
(V_1, V_2)

Теорема

Алгоритм правильно находит двусвязные компоненты графа G с e ребрами и тратит на это время $O(e)$.