Семирар 6

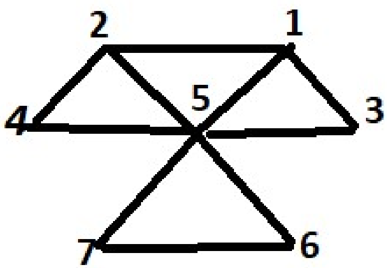
Поиск в глубину. Топологическая сортировка, алгоритм Касарайю

Поиск в глубину

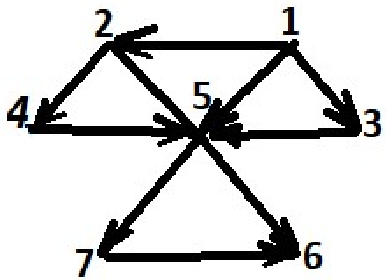
Поиском в глубину (DFS – depth first search) называется один из методов обхода графа G = (V, E), суть которого состоит в том, чтобы идти “вглубь” пока это возможно. В процессе поиска в глубину вершинам графа присваиваются номера, а ребра помечаются. Обход вершин графа происходит согласно принципу: если из текущей вершины есть ребра, ведущие в не пройдённые вершины, то идем туда, иначе возвращаемся назад.

1) Пойти в какую-нибудь смежную вершину.
2) Обойти все, что доступно из этой вершины.
3) Вернуться в начальную вершину.
4) Повторить алгоритм для всех остальных вершин, смежных из начальной.

Пример: неориентированный граф



ориентированный граф



Дерево поиска в глубину (построить)

Итеративная и рекурсивная версии

DFS (ИТЕРАТИВНАЯ ВЕРСИЯ)

Вход: граф $G = (V, E)$, представленный в виде списков смежности, и вершина $s \in V$.

Постусловие: вершина достижима из s тогда и только тогда, когда она помечена как «разведанная».

пометить все вершины как неразведанные

S := стек, инициализированный вершиной s

while S не является пустым **do**

Доделать дома

DFS (РЕКУРСИВНАЯ ВЕРСИЯ)

Вход: граф $G = (V, E)$, представленный в виде списков смежности, и вершина $s \in V$.

Постусловие: вершина достижима из s тогда и только тогда, когда она помечена как «разведанная».

```
// перед внешним вызовом все вершины не разведаны
```

пометить s как разведанную

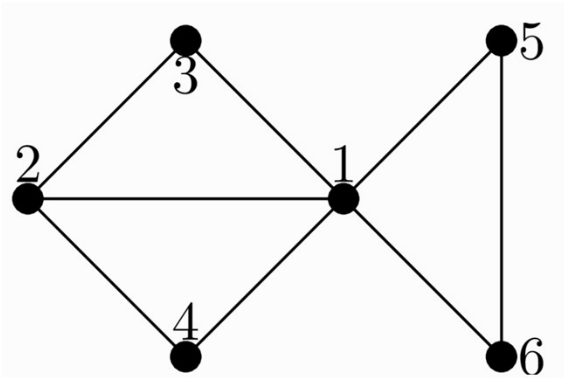
for каждое ребро (s, v) в списке смежности вершины s **do**

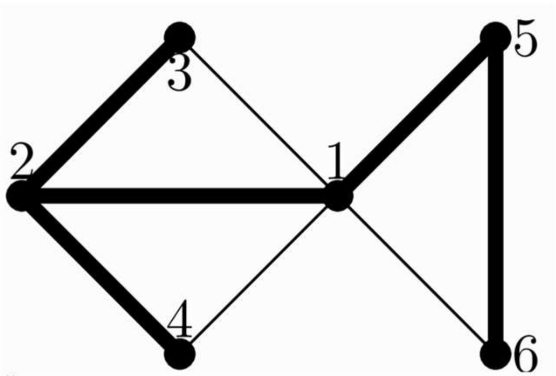
    if v не разведана **then**

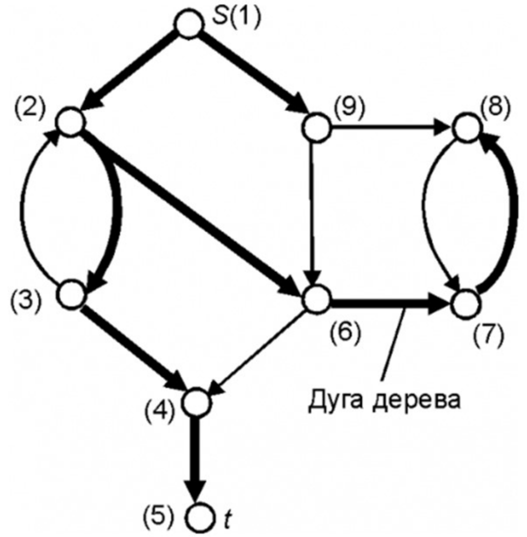
        DFS (G, v)

Примеры

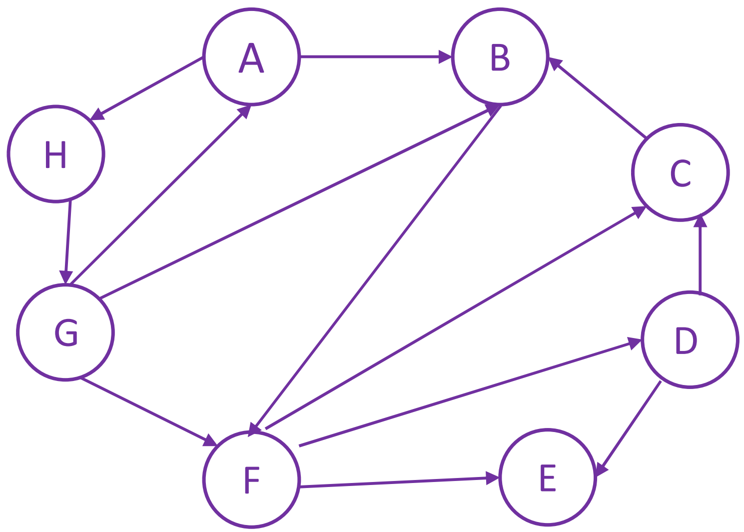
Перебираем вершины в порядке возрастания







Перебирать вершины в алфавитном порядке



Подсчет числа компонент связности в неориентированном графе

Запускаем поиск в глубину из первой вершины. Все вершины, обнаруженные в ходе этого алгоритма, принадлежат одной компоненте связности. Если остались неразведанные вершины, то запускаем поиск в глубину из любой из них. Вновь обнаруженные вершины принадлежат другой компоненте связности. Повторяем этот процесс до тех пор, пока не останется необнаруженных вершин, каждый раз относя вновь обнаруженные вершины к очередной компоненте связности.

```
Dfs(v, numCC);
begin
  cc[v] := numCC; // если v разведана,  cc[v] >0
  for каждого ребра (v, u) do
   if cc[u] = 0 then Dfs(u, numCC) //если  u не разведана
end;
//основная программа
  for {v ∈ V} do cc[v] := 0; //в начале все вершины не разведаны и приписываются компоненте 0
  numCC := 0;
  for i:=1 to n do
    if cc[i] = 0 then begin //найдена очередная компонента связности
    numCC:=numCC+1;
    Dfs(i, numCC)
end;
```

Время?

Поиск цикла в ориентированном графе

Цикл в ориентированном графе можно обнаружить по наличию ребра, ведущего из текущей вершины в вершину, которая в настоящий момент находится в стадии обработки, то есть алгоритм DFS зашел в такую вершину, но еще не вышел из нее. В таком алгоритме DFS будем красить вершины в три цвета. Цветом 0 («белый») будем обозначать еще неразведанные вершины. Цветом 1 («серый») будем обозначать вершины в процессе обработки, а цветом 2 («черный») будем обозначать уже обработанные вершины. Вершина красится в цвет 1 при заходе в эту вершину и в цвет 2 – при выходе.

Цикл в графе существует, если алгоритм DFS обнаруживает ребро, конец

которого покрашен в цвет 1.

Проверка, является ли граф деревом

Деревом называется связный граф без циклов, лесом называется произвольный, т.е. не обязательно связный, граф без циклов. Как проверить, является ли граф деревом или лесом? Надо проверить, что циклы в графе отсутствуют.

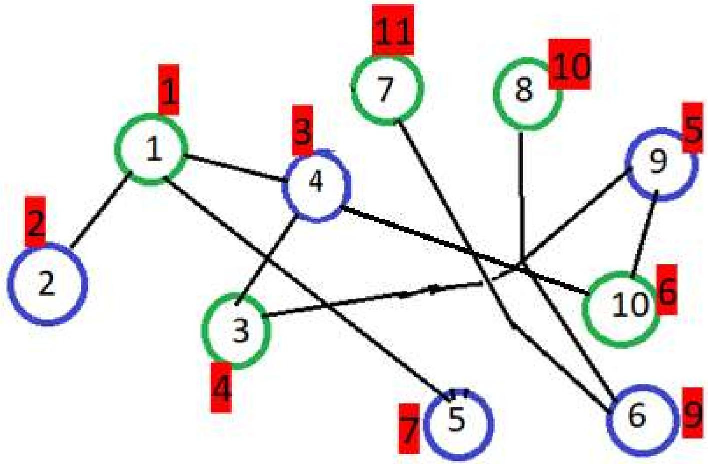
Если нам надо проверить, является ли граф деревом, то запустимся из первой вершины. Если хоть раз вернёмся в вершину, где мы уже побывали, то граф точно не дерево. Иначе в конце проверим, верно ли, что мы побывали во всех вершинах. Если да, то граф связен, а отсутствие циклов мы уже проверили — ок. Иначе не дерево.

Если нам надо проверить, является ли граф лесом, то все аналогично поиску всех компонент связности, закончив поиск в глубину из первой вершины, запускаемся из первой ещё не неразведанной и т.д. — такой же цикл, как и при поиске компонент связности.

Проверка графа на двудольность

Начинаем покраску с произвольной вершины, которую красим в произвольный цвет. При прохождении по каждому ребру красим следующую вершину в противоположный цвет. Если при переборе соседних вершин мы нашли вершину, уже покрашенную в тот же цвет, что и текущая, то в графе существует нечётный цикл, а значит, он не является двудольным.

Так как граф является двудольным тогда и только тогда, когда все циклы четны, определить двудольность можно за один проход в глубину**.** На каждом шаге обхода в глубину помечаем вершину. Допустим, мы пошли в первую вершину — помечаем её как 1. Затем просматриваем все смежные вершины, и если не помечена вершина, то на ней ставим пометку 2 и рекурсивно переходим в нее. Если же она помечена и на ней стоит та же пометка, что и у той, из которой шли (в нашем случае 1), значит граф не двудольный.



Топологическая сортировка

События – вершины.

A – проснуться

B – одеться

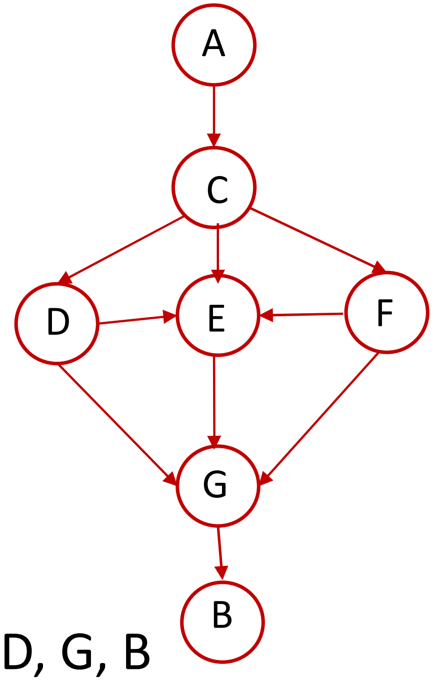
C –встать с кровати

D – принять душ

E – почистить зубы

F – сделать зарядку

G - позавтракать



A, C, E, F, D, G, B или A, C, F, E, D, G, B

Топологическое упорядочение

Дан граф G(V, E). $\forall v \in V$ определим $f(v)$: $\forall$ребра $(v, w) \in E$ $f(v) < f(w)$

Утв. Топологическую упорядочение в орграфе можно проводить только если в графе нет циклов (ациклический граф)

Проведем серию поисков в глубину. Самой дальней вершине (перед тупиком) назначим самый большой номер, «сматывая веревку» назад, разведанным вершинам будем присваивать номера, уменьшая их на 1.

Алгоритм TOPOSORT

TOPOSORT

Вход: ориентированный ациклический граф $G = (V, E)$, представленный в виде списков смежности.

Постусловие: значения f вершин образуют топологическую упорядоченность графа G.

```
пометить все вершины как неразведанные
curLabel := |V|              // отслеживает упорядочивание
for каждая v ∈ V do
    if v не разведана then    // в предыдущем DFS
        DFS-Topo (G, v)
```

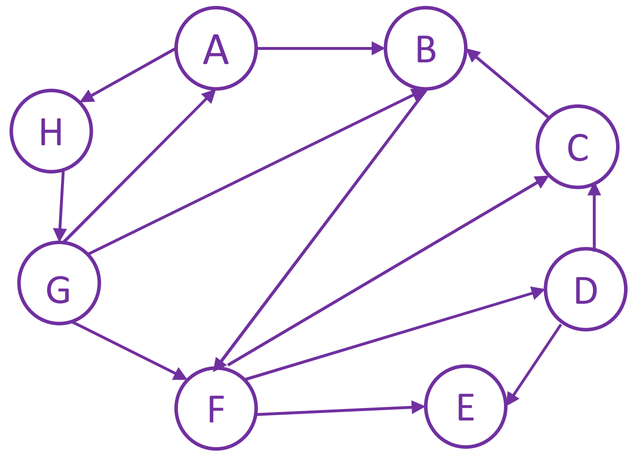
DFS-TOPO

Вход: граф $G=(V, E)$, представленный в виде списков смежности, и вершина $s \in V$.

Постусловие: каждая вершина, достижимая из s, помечается как «разведанная» и имеет присвоенное ей значение f.

```
пометить s как разведанную
for каждое ребро (s, v) в исходящем списке смежности s do
    if v не разведана then
        DFS-Topo (G, v)
f(s) := curLabel              // позиция s в упорядочении
curLabel := curLabel − 1      // двигаться справа налево
```

Пример 1: применить алгоритм TOPOSORT

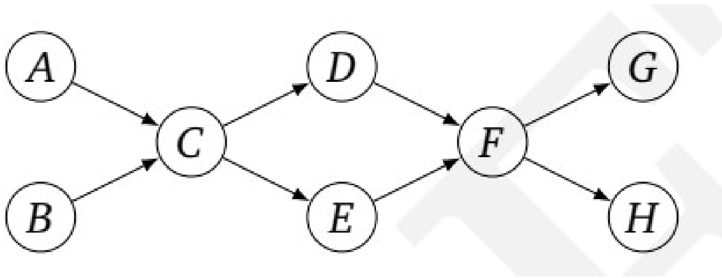


Каждый раз в качестве следующей выбирайте первую в алфавитном порядке вершину.

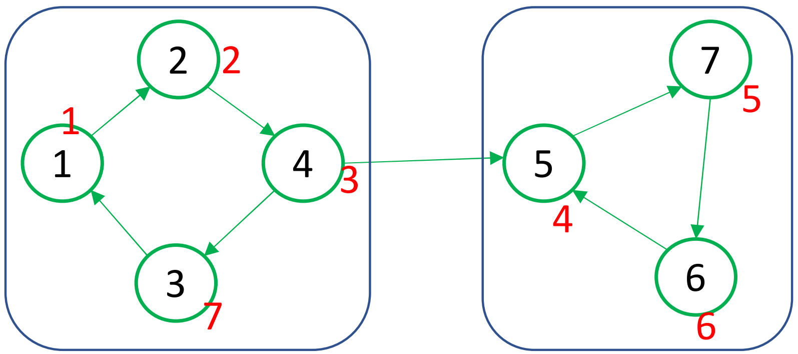
$$f(C) = 8, f(E) = 7, f(D) = 6, f(F) = 5,$$
$$f(B) = 4, f(G) = 3, f(H) = 2, f(A) = 1$$

Пример 2: применить алгоритм TOPOSORT.

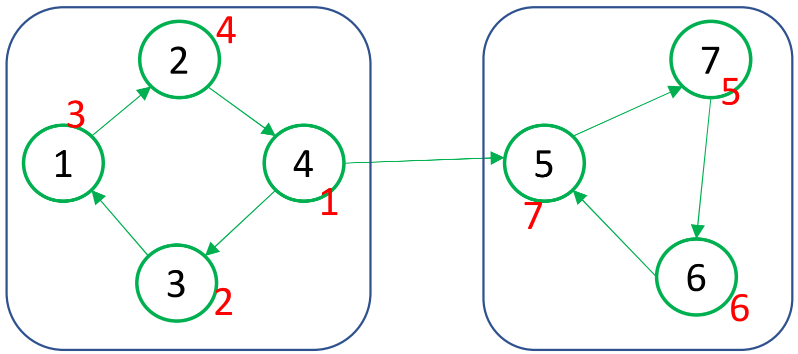
Задание, аналогичное примеру 1.



Нахождение компонент сильной связности с помощью топологической сортировки.



Вершины перебираются в возрастающем порядке



Вершины перебираются в убывающем порядке

Вершины в первой позиции принадлежит
а) Стоковой компоненте.
б) Истоковой компоненте.
в) Может быть в любой компоненте.

Метаграф

Вершины метаграфа можно топологически упорядочить, т.к. он ациклический. Если найти сток-вершину, то ее можно отсечь. В оставшемся графе также найти сто-вершину и т.д., пока не дойдет до первой метавершины – истока. Как найти сток-вершину? – При любом перечислении вершин вершина в первой позиции окажется в метавершине-истоке, т.к. справедлива

<u>Теорема.</u> Пусть G – орграф, вершины которого произвольно упорядочены. $\forall v$ определена позиция $f(v)$ с помощью TOPOSORT. Пусть S_1 и S_2 - компоненты сильной связности, (v, w) - ребро, причем $v \in S_1$, $w \in S_2$.

Тогда $$\min_{x \in S_1} f(x) < \min_{y \in S_2} f(y)$$

Если развернуть граф в обратную сторону и запустить TOPOSORT, то истоковая метавершина будет стоковой для исходного графа. Далее следует запустить поиск в глубину еще раз, при этом отсекая получающиеся стоковые метавершины. Будем нумеровать их в обратном порядке (стоковая #1,…, исток #N)

Алгоритм Косарайю

KOSARAJU

Вход: ориентированный граф $G = (V, E)$, представленный в виде списков смежности, с $V = \{1, 2, 3, ..., n\}$.

Постусловие: для каждой $v, w \in V$, $scc(v) = scc(w)$ тогда и только тогда, когда v, w находятся в одной и той же сильной связной компоненте графа G.

$G^{rev} := G$, в котором все ребра развернуты в обратную сторону

пометить все вершины G^{rev} как неразведанные

// первый проход поиска в глубину

// (вычисляет позиции f(v), волшебную упорядоченность)

TopoSort (G^{rev})

// второй проход поиска в глубину

// (находит сильно связные компоненты

// в обратном топологическом порядке)

пометить все вершины G как неразведанные

$numSCC := 0$ // глобальная переменная

for каждая $v \in V$, в порядке возрастания $f(v)$ **do**

  if v не разведана **then**

    $numSCC := numSCC + 1$

    // назначить scc-значения

    DFS-SCC (G, v)

DFS-SCC

Вход: ориентированный граф $G = (V, E)$, представленный в виде списков смежности, и вершина $s \in V$.

Постусловие: каждая вершина, достижимая из s, помечается как «разведанная» и имеет присвоенное ей значение scc.

пометить s как разведанную

$\boxed{scc(s) := numSCC}$ // приведенная выше глобальная переменная

for каждое ребро (s, v) в исходящем списке смежности s **do**

  if v не разведана **then**

    DFS-SCC (G, v)

Время работы?

Пример отыскания компонент сильной связности