Как хранить графы

1. Матрица смежности (≠ матрица инцидентн.)

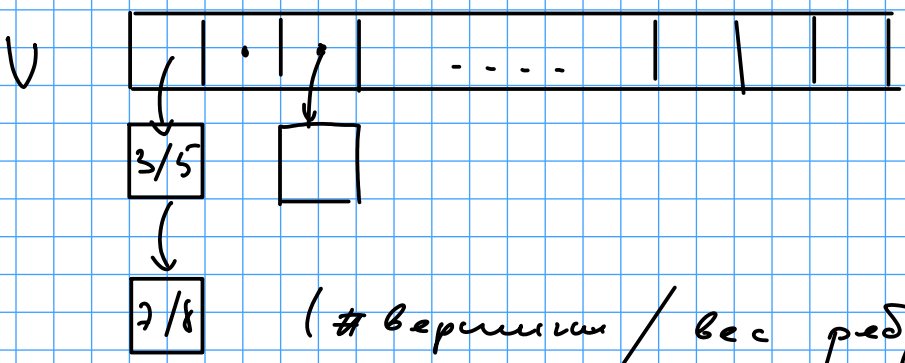
Размер $|V|^2$

⊕ Проверить наличие ребра $O(1)$

⊖ Большой размер

⊖ Обход / перечисление всех рёбер из данной вершины за $O(N^2)$ / $O(|V|)$

2. Списки смежности



(# вершины / вес ребра)

⊕ Размер $O(|V| + |E|)$

⊕ Обход $O(|V| + |E|)$

⊖ Проверка наличия ребра за $O(|V|)$

Поиск в глубину (DFS, depth-first-search)

```
Explore(v):
    visited[v] = true
    previsit(v)
    for (v, u) ∈ E:
        if not visited[u]:
            Explore(u)
    postvisit(v)
```

Сложность:

$O(|V|)$ вызовов Explore

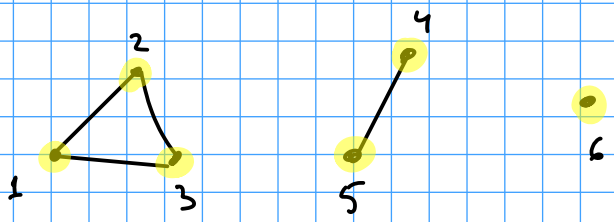
в сумме $O(E)$ операций (если списки смежности)

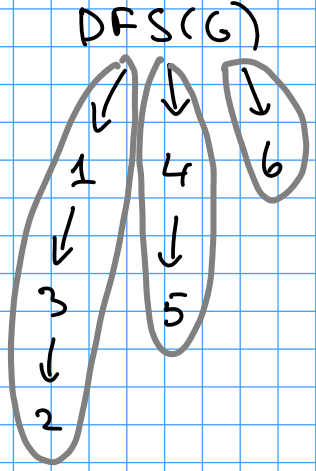
Итого: $O(|V| + |E|)$

```
DFS(G):
    for v = 1 to |V|:
        visited[v] = false
    for v = 1 to |V|:
        if not visited[v]:
            Explore(v)
```

$O(|V|)$

NB: для матрицы смежности в итоге $O(|V| + |V|^2)$





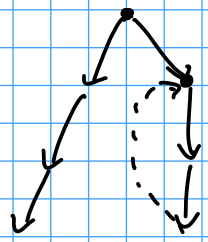
Компоненты связности в неор. графе

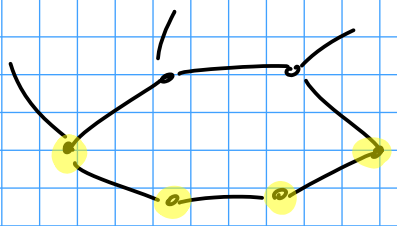
] cc[v] – # компоненты связности компонент – глобальный счётчик.

```
Previsit(v):
    cc[v] = component
```

```
CC(G):
for v = 1 to |V|:
    visited[v] = false
component = 1
for v = 1 to |V|:
    if not visited[v]:
        Explore(v)
        component = component + 1
```

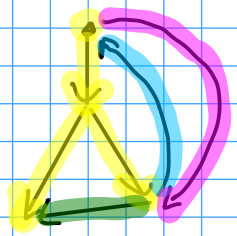
Поиск циклов в неор. графах





Утв: В графе есть цикл $\Leftrightarrow$
В его обходе в глубину есть
„обратное ребро".

Поиск в глубину в ор. графах

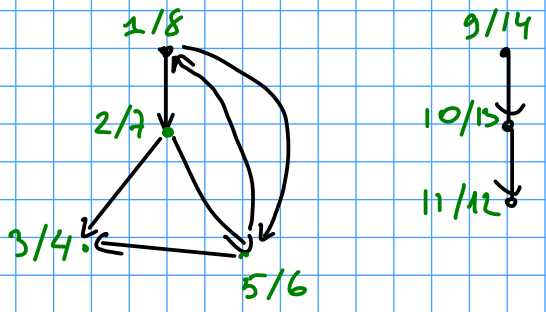


1. ребра дерева поиска
2. перекрестные рёбра
3. обратные рёбра
4. прямые рёбра

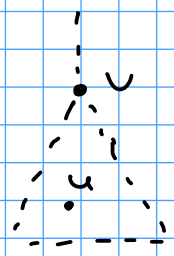
pre[v], post[v] – два массива
time – глобальное время.

previsit(v):
pre[v] = time
time += 1

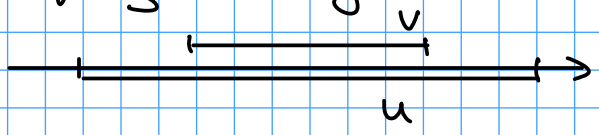
postvisit(v):
post[v] = time
time += 1



Утв: $\forall u, v \in V$ отрезки $[pre[v], post[v]]$ и $[pre[u], post[u]]$ либо один содержит другой, либо не пересекаются.



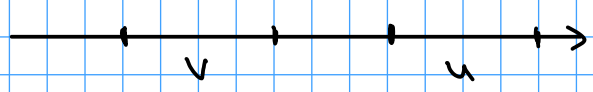
Для отрезков для всех рёбер $(u, v) \in E$:

1. 

ребро дерева поиска / прямое ребро

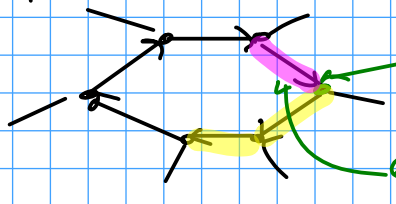
2. 

обратное ребро

3. 

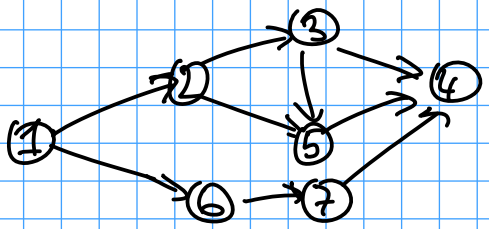
перекрёстное ребро

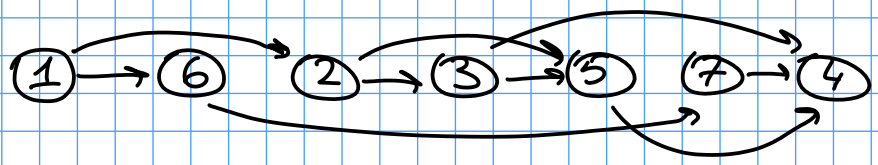
Утв: Если в графе есть обратное ребро $\Leftrightarrow$ в графе есть цикл.



Топологическая сортировка

= Топологическая сортировка DAG — это такой порядок вершин T:
$\forall (u, v) \in E \quad T[u] < T[v]$

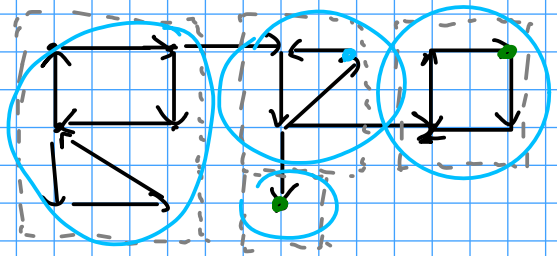




Утв: Топологическая сортировка задаётся порядком времени выхода (в порядке убывания) (post[v])

Выделение компонент сильной связности (Strongly connected components)

≡ $\forall u, v \in SCC : v \rightsquigarrow u, u \rightsquigarrow v$



NB: в DAG $\forall$ вершина — это компонента сильной связности

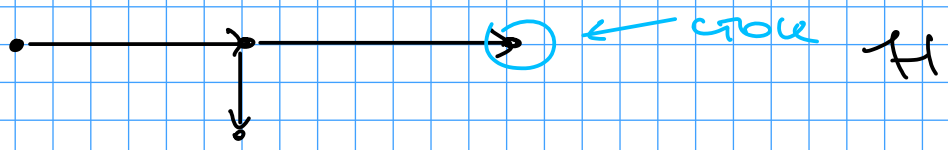
Утв: В $\forall$ ациклическом графе есть сток, т.е. вершина нех. степени 0

Утв В $\forall$ ациклическом графе есть исток, т.е. вершина вх. степени 0.

Утв: Если G - ацикл. $\iff G^R$ - ацикл.

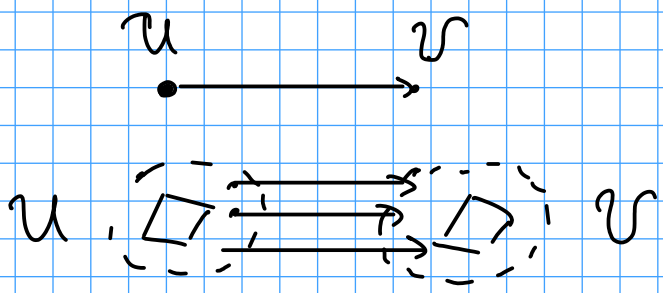
Утв: Если G задан списками смежности, то G^R можно построить за $O(|V| + |E|)$

≡ Метаграф — граф со стянутыми компонентами сильной связности



Утв: Метаграф будет ациклическим.

Утв $\exists (U, V) \in H \Rightarrow$
$\max\limits_{u \in U} [post[u]] > \max\limits_{v \in V} [post[v]]$



1. DFS начал сначала с V
$\Rightarrow$ сначала ~~к~~ все вершины V, а уже потом U

2. DFS начал сначала в U, посетил V по ребру из U $\Rightarrow$ время выхода для вершины – начала этого ребра > $\forall$ времени выхода из V

Следствие: max время выхода лежит в компоненте – истоке.

SCC(G):
DFS(G^R)
CC'(G) // в порядке убывания post[v]

начнем с вершины из компоненты – стока

$O(|V| + |E|)$