Как хранить граф

1. Матрица смежности

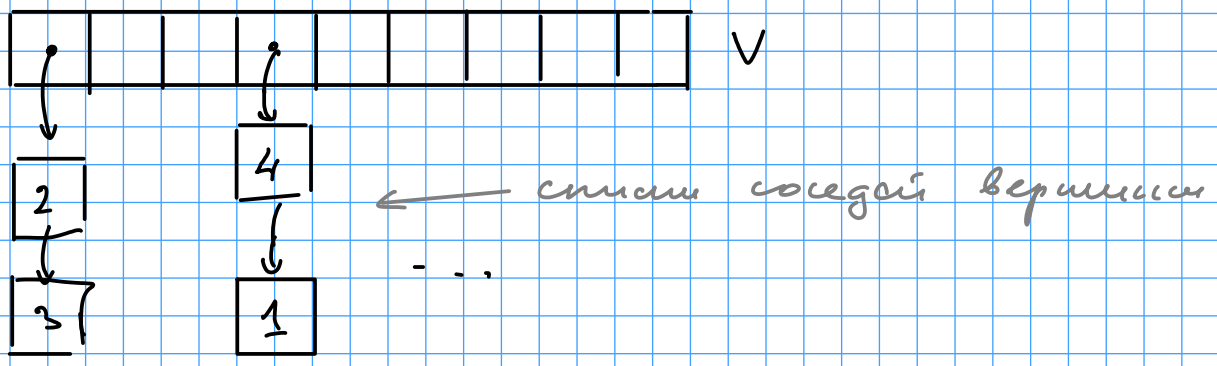
Размер $|V| \times |V|$

\+ Проверить ребро за $O(1)$

\- Большой размер (квадратичный для разр. графов)

\- Обход графа за $O(|V|^2)$

2. Списки смежности



← списки соседей вершины

\+ Размер $O(|V| + |E|)$

\+ Обход $O(|V| + |E|)$

\- Проверка ребра за $O(|V|)$

3. Матрица инцидентности

Таблица $|V| \times |E| \sim O(|V|^3)$

$O(n^{1.5})$ $n = O(|V|^2)$

Поиск в глубину (DFS, depth first search)

```
Explore(v):
  visited[v] = true
  Previsit(v)
  for (v, u) ∈ E:        O(E)
      if not visited[u]:
          Explore(u)
  Postvisit(v)

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

visited[v] - массив булевых значений

Сложность:

$O(V+E)$

списки смежн.

$O(V^2)$

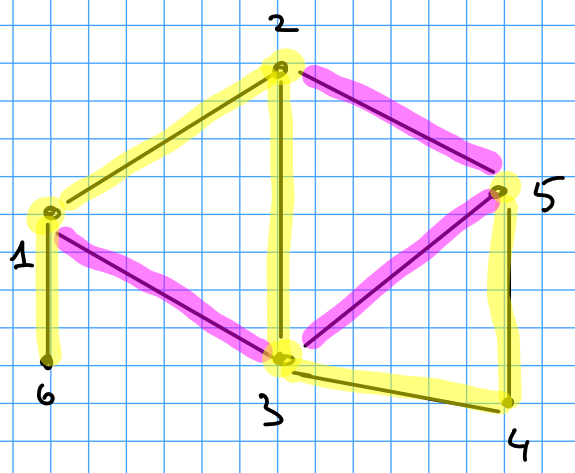
матриц. смежн.

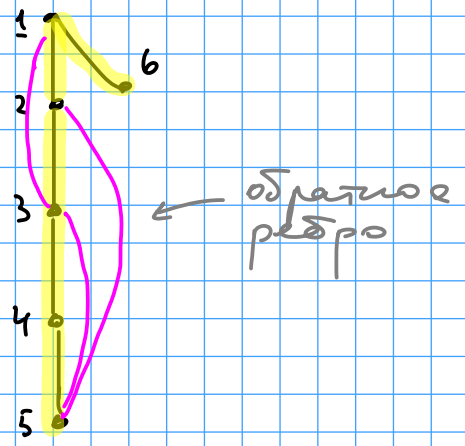
Поиск компонент связности в неор. графах

Заведём массив cc[v] - номер компоненты связности для # вершины.

```
Previsit(v):
    cc[v] = component
DFS(G):
    component = 1
    for v = 1 to |V|:                (*)
        visited[v] = false
        cc[v] = 0
    for v = 1 to |V|:
        if not visited[v]:
            → Explore(v)
              component += 1
```

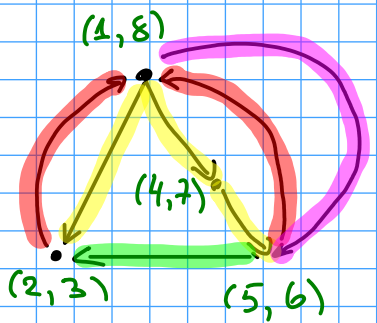
Поиск циклов в неор. графах

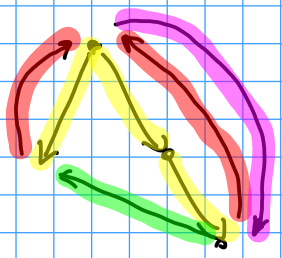


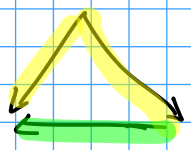


Утв: В ∀ цикле есть обратное ребро
∀ обратное ребро ∈ некоторому циклу

Поиск в глубину в ориентированном графе







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

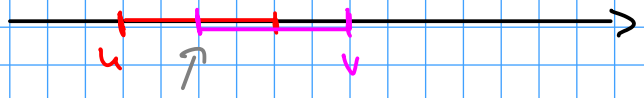
pre[v], post[v], счётчик clock

```
Previsit(v):
    pre[v] = clock
    clock += 1
```

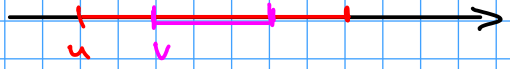
```
Postvisit(v):
    post[v] = clock
    clock += 1
```

Утв: ∀ u,v ∈ V: отрезки [pre[u], post[u]] и [pre[v], post[v]] либо не пересекаются,

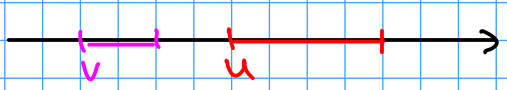
либо один содержит другой.



∀ (u, v) ∈ E

1. 

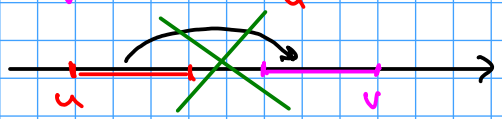
ребро дерева / прямое ребро

2. 

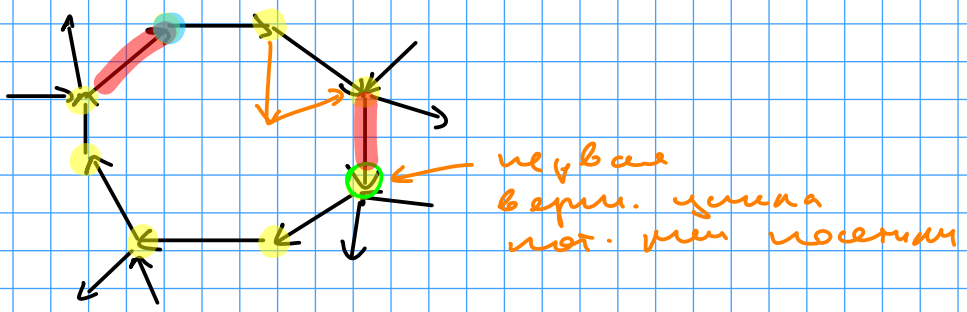
перекрёстное ребро

3. 

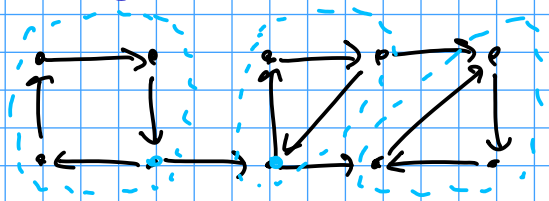
обратное ребро

4. 

Утв: В графе есть обратное ребро ⟺ в графе есть цикл



Выделение компонент сильной связности



≡ C ⊆ V – комп. сильной связности, если
∀ u, v ∈ C: v ⇝ u
u ⇝ v

C – макс. по включению

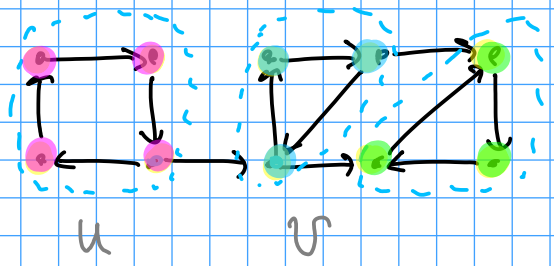
Утв: В любом ациклическом графе есть сток, т.е. вершина исх. степени 0.

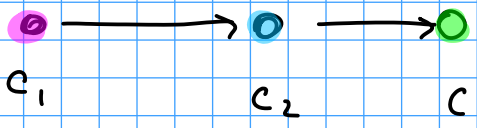
Утв: В любом ациклическом графе есть исток, т.е. вершина вх. степени 0.

Утв: Если G – ациклический, то G^R – ациклич.

Утв: В ациклическом графе $|V|$ комп. сильной связности, т.е. $\forall$ вершина - это комп. сильн. связности.

Замеч: Если G задан списком смежности, то G^R можно построить за $O(V+E)$



H 

= мета граф, граф компонент сильн. связности

Утв: H - ациклический.

Утв: Если (U, V) - ребро $H \Rightarrow$

$\Rightarrow \max\limits_{u \in U} [post[u]] > \max\limits_{v \in V} [post[v]]$

▷ 1. DFS сначала посетил V

Этот вызов Explore не посетит $U \Rightarrow$

post в $V <$ post U (для $\forall$ пары вершин).

2. DFS сначала посетил U (вершину u)

$\Rightarrow$ В этом вызове Explore обойдём V.

$post[u] > post[v]$, $\forall v \in V$ ◁

Следствие:

Вершина с максимальным post лежит в компоненте – истоке.

SCC(G):

DFS(G^R) $O(V+E)$

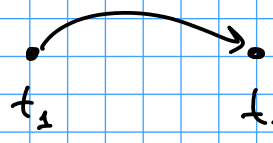
for v in V: // в порядке ↓ post[v]

Explore(v) // помечает аналогично первой графу (*)

первая вершина будет принадлежать компоненте - стоку $O(V+E)$

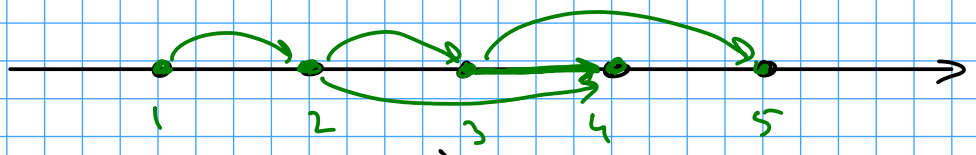
Топологическая сортировка

= для ациклических графов (DAG)



t_1 нужно выполнить до t_2

Нужно расположить вершины DAG на прямой так, что бы рёбра шли слева направо (пронумеровать)



TopSort(G)

DFS(G)

Переупоряд. вершины по убыванию post

$O(V+E)$