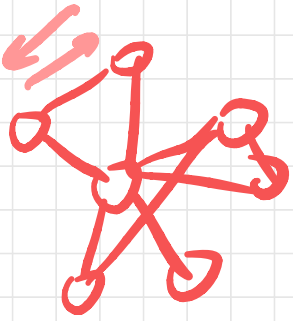
Поиск в глубину
(Depth First Search)

Тремо, 19 век

Графы:



$G = (V, E)$

$V = \{0, 1, \ldots, n-1\}$

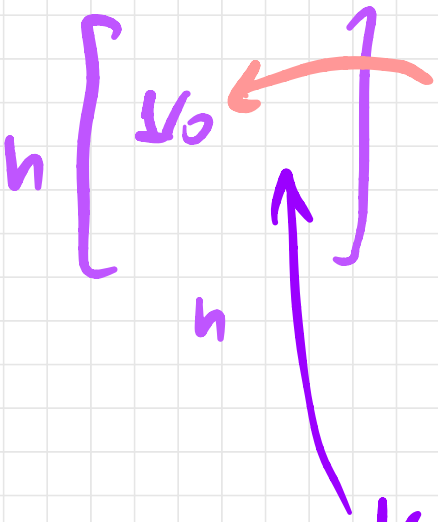
$m = |E|$ E: ориентированные и неориентированные ← 2 ориент гр.

взвешенные и невзвешенные

с кратными рёбрами и без

Способы хранения

1) Матрица смежности



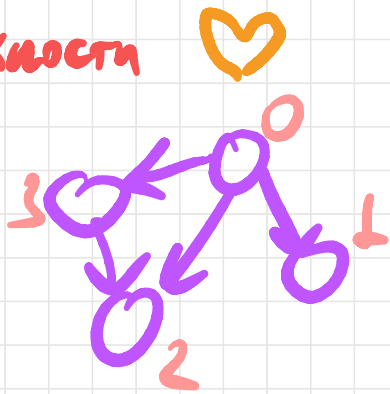
$O(n^2)$ памяти

$m << n^2 \Rightarrow$ не очень удачно

V ребра если оно взвешенное

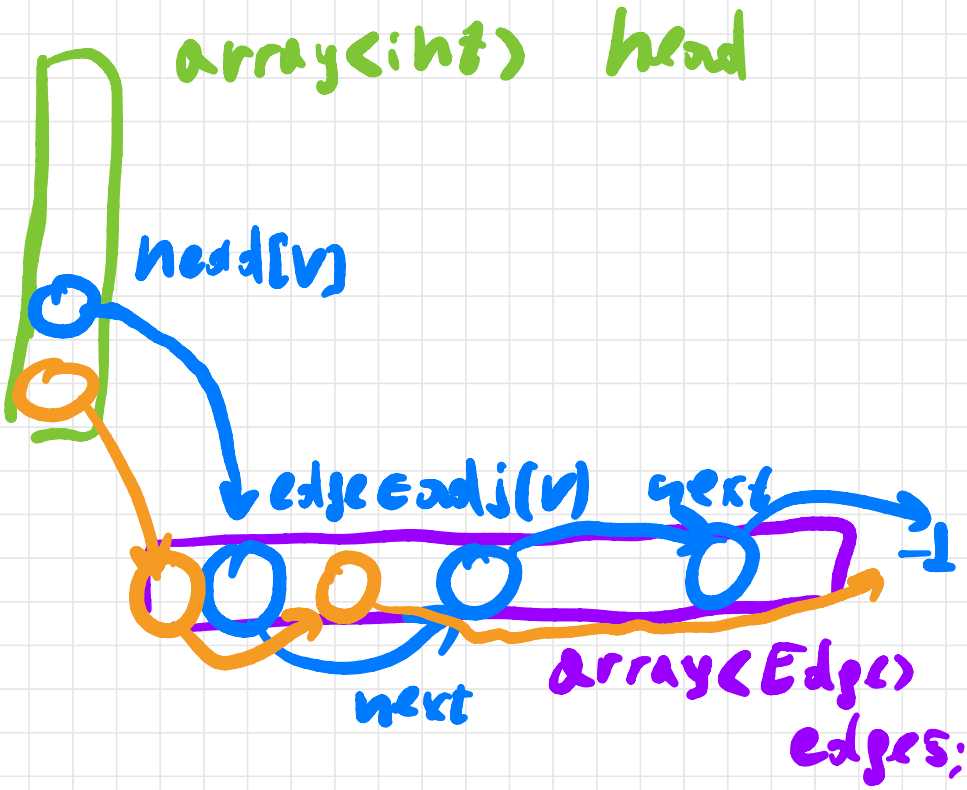
2) Списки смежности

```
adj = [
    [1, 2, 3],
    [],
    [],
    [2]
]
```



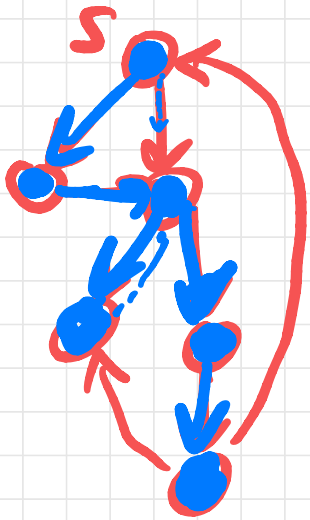
3) Мульти список смежности

```
struct Edge {
    int to;
    int w;
    int next;
};
```



```
for (e = head[v];
     e != -1;
     e = edges[e].next)
print (edges[e].to)
```

Поиск в глубину




```
used = [ false  for _ in range(n)]
```

```
dfs(v):
    used[v] = 1
    for u in adj[v]:
        if not used[u]:
            par[u] = v
            dfs(u)
```

$O(V+E)$, $n+m$

Как можно запускать Dfs

1) Из одной вершины

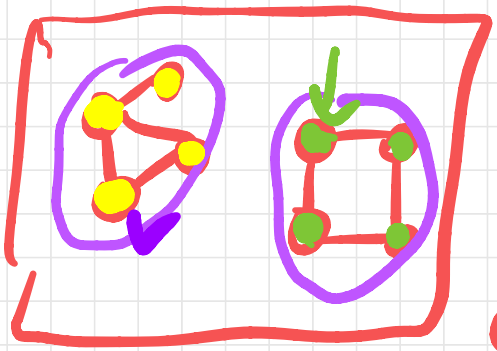
dfs(5)

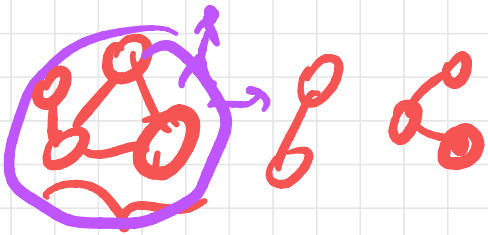
2) по всему графу:

```
for v = 0..n-1
    if !used[v]:
        dfs(v)
```

Применения dfs

1) Компоненты связности (не ор. 2раз)





К. С.

Отн. эк-в по свойству достижимости

```
for v=0..n-1
    if !used[v]:
        dfs(v)
        count+=1
```

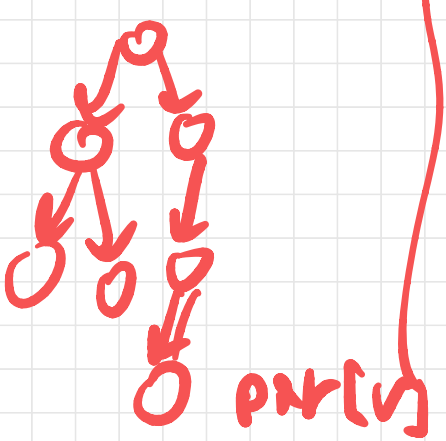
```
for v=0..n-1
    if ! used[v]:
        dfs(v, c)
        c+=1
```

color

```
dfs(v,c)
    color[v]=c
    —11—
```

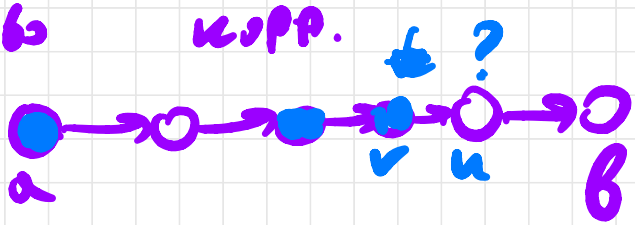
2) Поиск пути

1) dfs(a)
2) if used[b]:

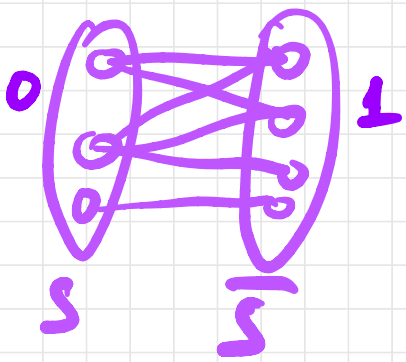


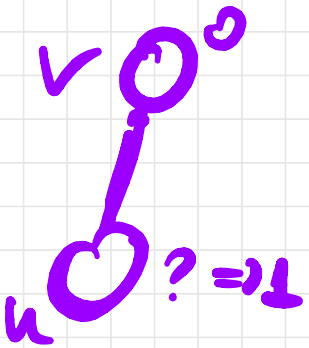

```
dfs(v):
    if v==b:
        посмотреть
        на ответ
        на стек
```

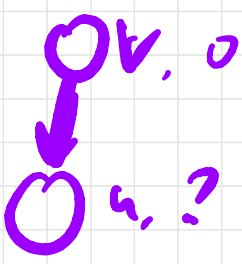
Д-во корр.

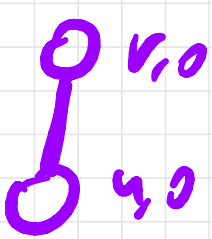


3) Двудольность







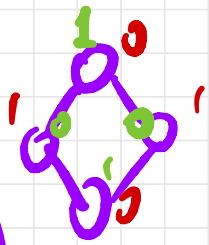


```
color = [-1 for _ in range(n)]

def dfs(v) -> bool:
    for u in adj[v]:
        if color[u] == color[v]:
            return False # There is no coloring.

        if color[u] == -1:
            color[u] = 1 - color[v]
            if not dfs(u):
                return False
    return True

fail = False
for v in range(n):
    if color[v] == -1 and not dfs(v):
        color[v] = 0
        fail = True

if not fail:
    print("Bipartite")
else:
    print("Non bipartite")
```

color[v] yxc jayan

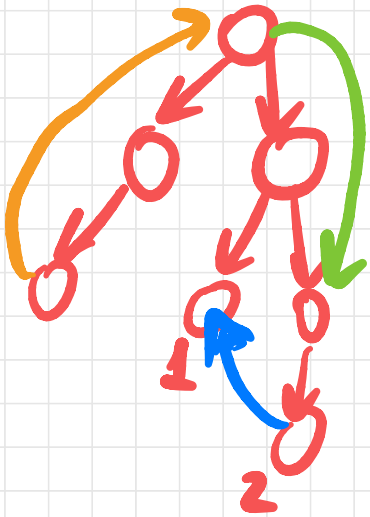
(color[u] ≠ color[v])

pass.




```
fail = false
for v = 0..n-1:
    if color[v] == -1:
        color[v] = 0
        if not dfs(v):
            fail = True
```

Классификация рёбер



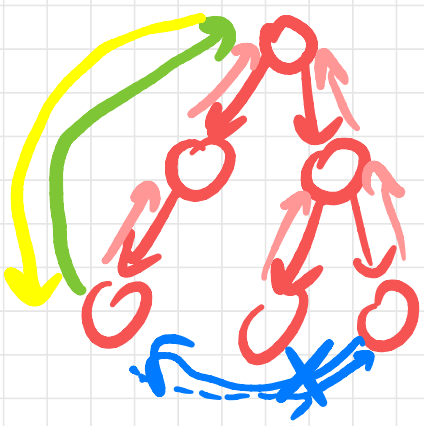
tree

back

forward

cross

Неор. случай



tree

"псевдо"-tree

back

"псевдо"-back

прямое

Cross - ребро

Не существ.

Поиск цикла

1) в неор графе.

```python
def find_cycle(v, par):
    used[v] = 1

    for u in adj[v]:
        if u == par:
            # копия древесного ребра в другую сторону
            continue

        if used[u]:
            return True # нашли обратное ребро
        elif find_cycle(u, v):
            # Идём по древесному ребро,
            # нашли в рекурсии цикл
            return True
```

return false

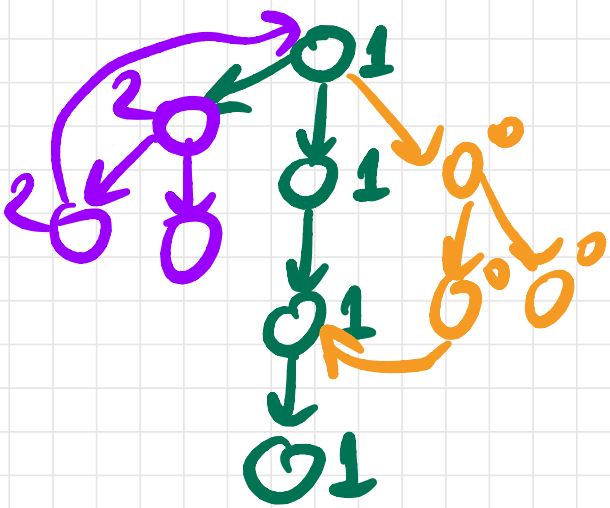
Найдём обратное ребро, ведущее вниз

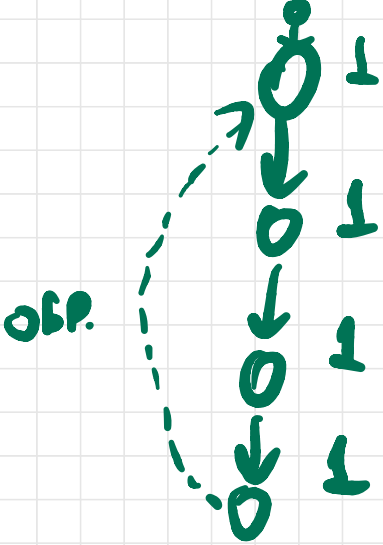
Ориент. граф

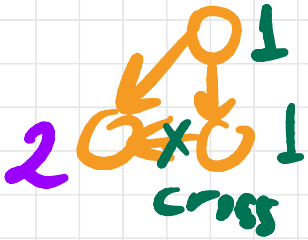
3 цвета в DFS

```
used = [0 for v in range(n)]

def dfs(v):
    used[v] = 1

    for u in adj(v):
        if not used[u]:
            dfs(u)

    used[v] = 2
```

new

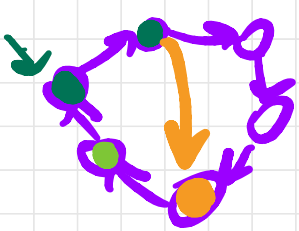




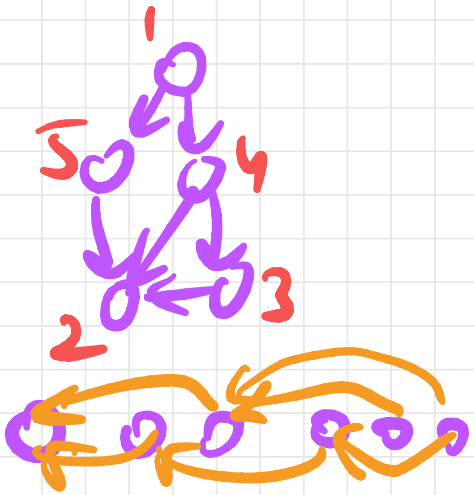


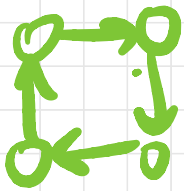

```python
# ориентированный граф

used = [False for _ in range(n)]:

def find_cycle(v):
    used[v] = 1

    for u in adj[v]:
        if not used[u] and find_cycle(u):
            return True

        if used[u] == 1: # Back edge
            return True

        if used[u] == 2:
            # Forward or Cross edge
            # Их можно отличить дополнительными проверками,
            # но в данном случае нам оба из них не интересны.
            pass

    used[v] = 2
    return False
```

tree



Топологическая сорт.

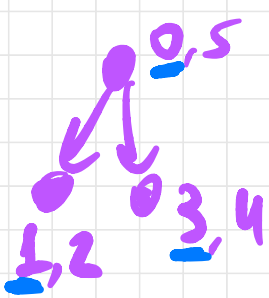




2 3 4 5 1

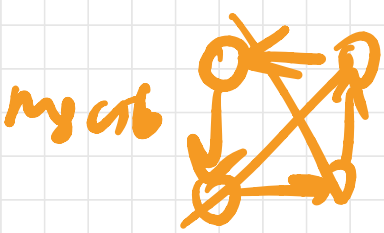
tin, tout
время входа и выхода

```
dfs(v):
    used[v] = 1
    tin[v] = t, t+=1
    for u in adj[v]:
        if ! used[u]:
            dfs(u)
    order.push(v)
    tout[v] = t, t+=1
```



1) order это вершины в порядке сортировки по tout

2) Order зад. топ. сорт, если он вообще сущ.

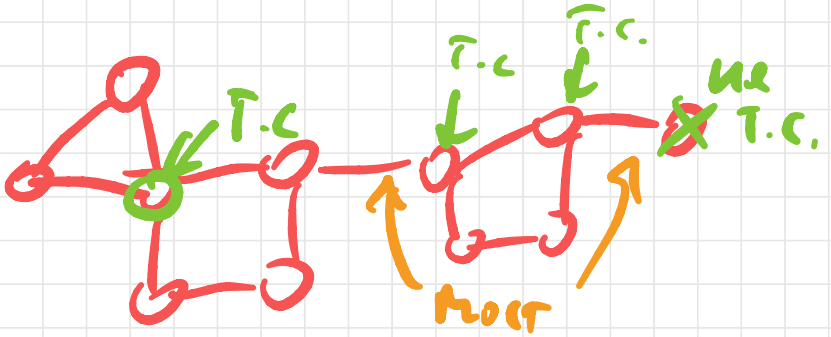


Lm:
tout[a] > tout[б], если в графе нет цикла.

Д-во:
Случай 1: Зашли раньше в а
Случай 2: Зашли раньше в б.

Lm => Тэн. Спрт корректна (если нет цикла)

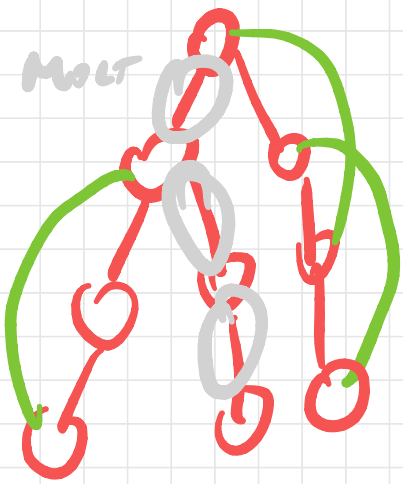
Мэйти и Точка Сочленения



Def:
Мост, если $c(G-e) > c(G)$

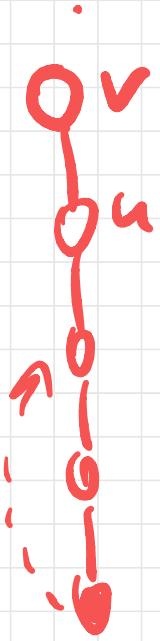
Def:
Т.С., если $c(G-v) > c(G)$

Мосты



обр. ребро никогда не мост

прям. ребро есть мостом, когда оно не покрыто обр.



$depth[v] < depth[u]$

$tin[v] < tin[u]$.

или tin, или $depth$

up[v] = самая неглубокая вершина, дост. таким образом

up[v] = глубина такой вершины

up[b] ≤ depth[a]

⇓

не мост, иначе мост.

```
dfs(v, par=-1):          # depth[v] уже вычислен
    used[v] = 1
    up[v] = depth[v]
    for u in adj[v]:     # (v,u)
        if u == par:
            continue
        if not used[u]:
            depth[u] = depth[v] + 1
            dfs(u, v)
            up[v] = min(up[v], up[u])
            if up[u] > depth[v]:
                # (v,u) - мост
        else:
            up[v] = min(up[v], depth[u])
```