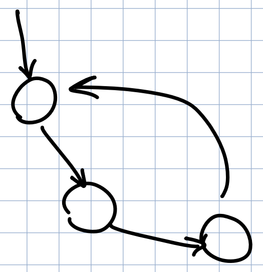
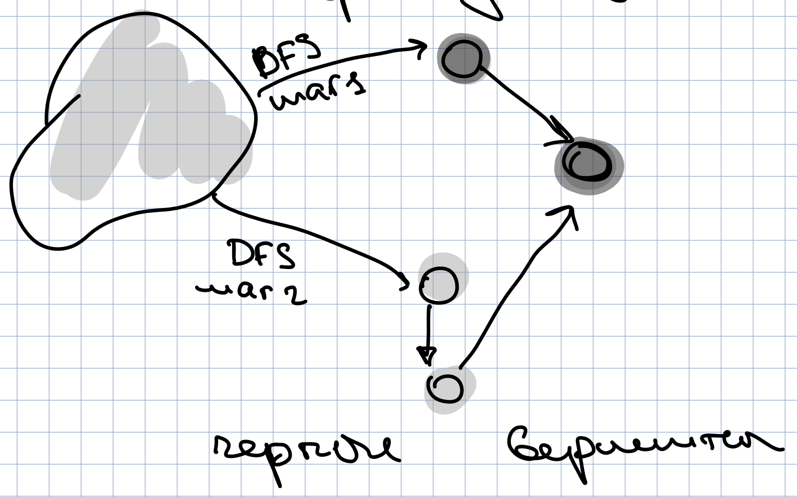
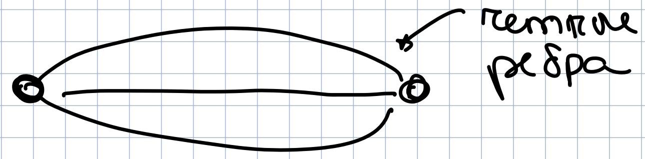
I Нахождение цикла

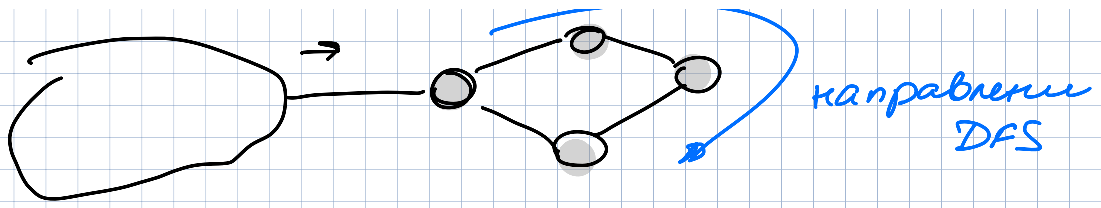
1. В ориентированном графе
   - идем DFS
   - если встречаем серую вершину ⇒ цикл.

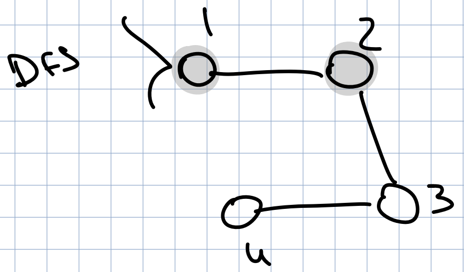




2. Неориентированный граф (без четных ребер)
   - идем DFS
   - встречаем серую вершину ⇓ есть цикл



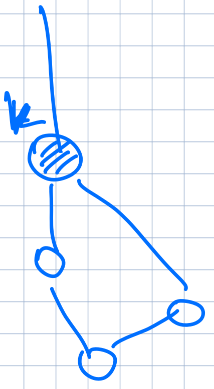




(!) отслеждаваме родителя

```
DFS (G, s, parent)
  ↳ if visited(s) = True
        ↳ "есть цикл"
          return s;

    visited(s) = True
    ch = G.GetChildren(s)
    for i=0... ch.size():
    }   if ch[i] != parent:
            k = DFS(G, ch[i], s)
            if k != -1:
                print(k)
                if k = s:
                    ↳ exit()
```

-1 - показалено корен



}   return $\boxed{k}$ — окончание цикла

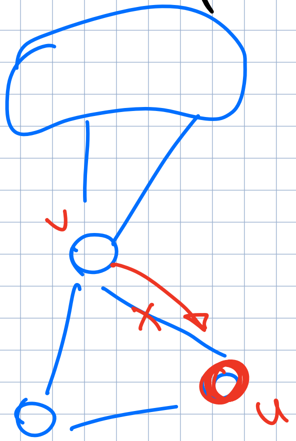
Асимптотика: $T(N, M) = \underline{O}(N+M)$

$\boxed{II}$ ① Эйлеров граф

Эйлеров цикл - цикл, проходящий по всем рёбрам единожды.

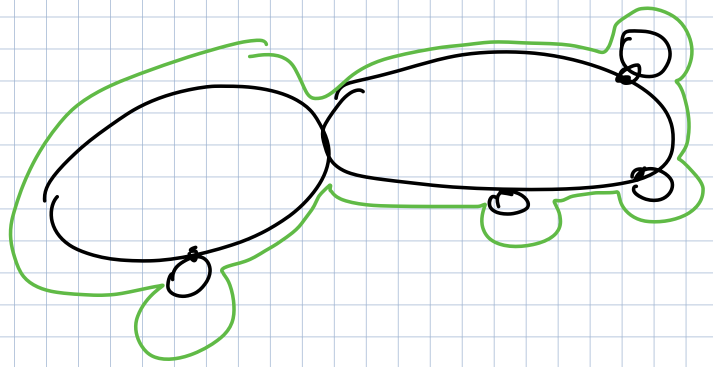
Критерий: степень всех вершин четна.

Алгоритм поиска: → проверим критерий.

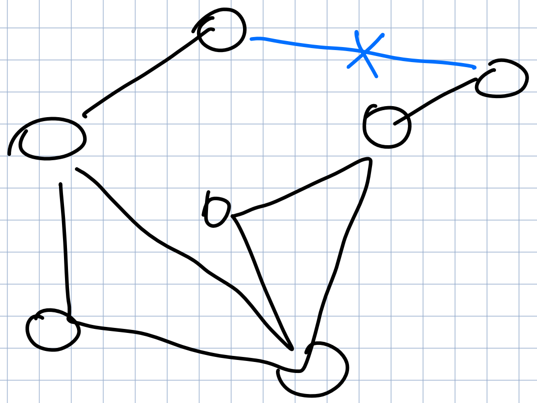


функция euler(G, v)

```
↳ while ( G. countChildren(v) != 0):
  {
      u = G.getChildren(v)[0]    ← берем рандомного ребенка
      G.RemoveEdge([v,u])
      euler(G, u)
  }
  print(v)
```



② Эйлеров путь



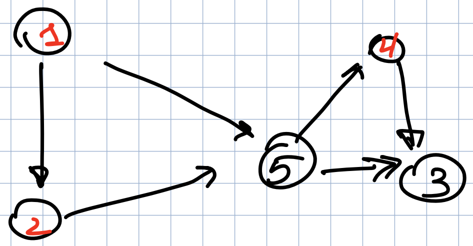
<u>Критерий</u> (полуэйлеровости): все вершины, кроме не более двух имеют чётную степень.

Путь начинается и заканчивается в нечётной вершине

II Топологическая сортировка (ориентированный граф)

Задача: Орграф. "все ребра должны идти из более ранних вершин в более поздние"

$$\begin{tikzcd}
1 \arrow[r, bend left] \arrow[rr, bend left=40] & 2 \arrow[r] & 5 \arrow[r] \arrow[rr, bend right] & 4 \arrow[r] & 3
\end{tikzcd}$$

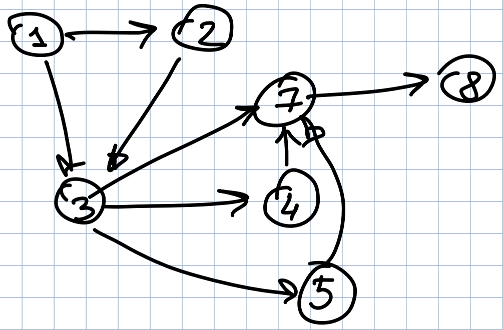


Если есть цикл: граф нельзя отсортировать топологически.

Алгоритм (идея): берем tout в порядке убывания
↑ покраска в черный цвет.

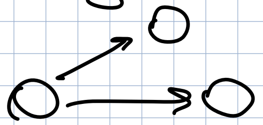
Реализация:
```
std::vector<int> top_sort ;

функция      dfs ( G, v)
             {
                visited [v] = True
                children = G. GetChildren ( v )
                for  i = 0 ... children.size()
                     if  visited [ children[i] ] = False
                         ↳ DFS ( G, children[i] )

                top_sort . push_back ( v )
             }

функция      topological_ sort () {
             visited [ False ...  False ];
             for  i = 0 ... G. size() :
                ↳  if  visited [i] = False
                           ↳ DFS ( G, i )
}            reverse ( top_sort )
```
Все компоненты связности

Пример:



IV Компонента сильной связности. (для орграфов)

Слабая связность

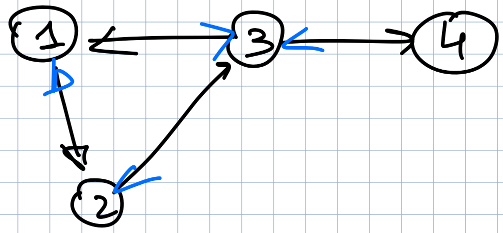


- убрали ориентированность ребер
- посмотрели на связность графа (неорграфа)

Сильная связность
- из любой вершины можно добраться в любую.

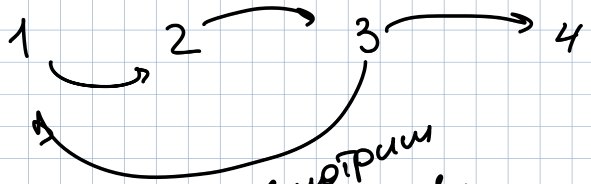
Алгоритм поиска компонент сильной связности (Косарайю)

Идея:
- выполним „топологическую сортировку"
- инвертируем ребра



инвертируем рёбра

порядок эл-в



• в том же порядке эл-в (из п1) запускаем DFS.

↳ все вершины из этого DFS из одной комп. сильной связности.

Реализация:

аналог топ сорта

```
DFS1 (G, v)
{
        —//—
        t.push_back(v)
}
```

↙ в серое вершина не идем.

покраска
компонент
связности $\curvearrowright$

```
DFS2 (G, v, k)        components = [-1; -1; ... -1]
{
    components [v] = k
    ch = G. Get Children ( v)
    for  i = 0 ... ch.size()
        if  components[ch[i]] == -1:
            DFS2 (G, ch[i], k)
}
```

Поиск компонент сильн. связности:

1. получаем vector t — "топсорт"

2. Строим граф с инв. ребрами.

```
    k = 0
for  i = 0, ...  t.size() :
{
    if  components[i] == -1 :
    {
        DFS2 (G, i, k)
```

```
          k++;
     }
 }
print (k)
```

кол-во компонент.