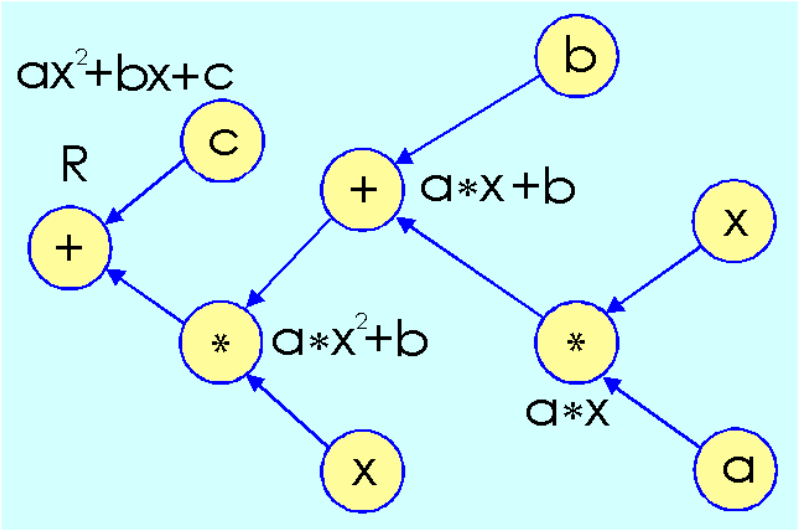
Топологическая сортировка

Topological sort (Topsort)

Задачи

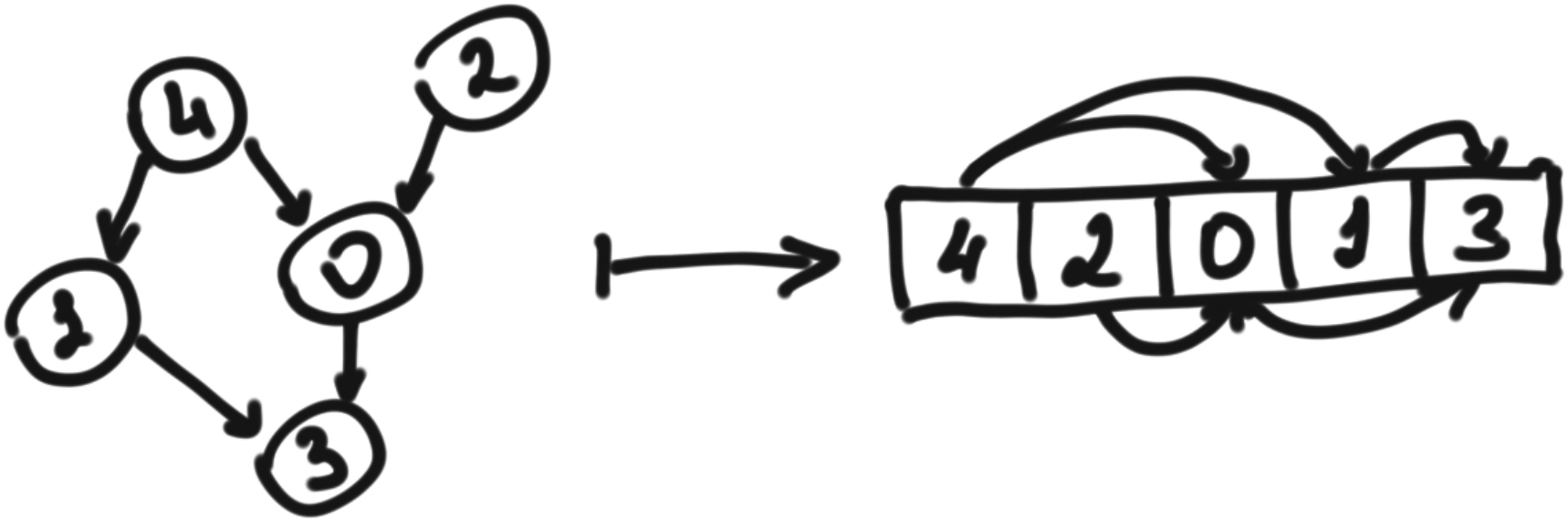
- В компьютерной игре есть квесты *A*, *B*, *C*, ... Причем некоторые квесты открываются только при выполнении каких-то предыдущих. Необходимо задать порядок прохождения игры.
- Вычисление сложной функции состоит из операций *A*, *B*, *C*, ... Причем некоторые операции возможно вычислить только после вычисления каких=то предыдущих. Необходимо задать порядок вычислений



Топологическая сортировка

Топологическая сортировка ациклического ориентированного графа (DAG) заключается в поиске перестановки вершин π такой, что
$\forall (x, y) \in E : \pi^{-1}(x) < \pi^{-1}(y)$

TL;DR: расположить вершины в массиве, чтобы ребра шли только слева направо



Топологическая сортировка: план

1. Запустим *DFS*
2. В момент завершения работы над вершиной (перед выходом из *DfsVisit*) положим вершину в начало списка.
3. Так мы ничего не испортим, так как у добавленной вершины либо нет исходящих ребер, либо все ребра ведут в черные вершины (значит они уже в списке).
4. Продолжаем пока не обойдем все вершины
5. Отправляем в контест
6. CE $\rightarrow$ RE $\rightarrow$ WA $\rightarrow$ AC $\rightarrow$ IG $\rightarrow$ AC $\rightarrow$ ... $\rightarrow$ OK $\rightarrow$
7. Profit.

Алгоритм Тарьяна (Tarjan, 1972)

```
def TopSortDfs(G, v):  # возвращает False, если отсортировать нельзя
  colors[v] = GRAY
  for u in G.neighbors(v):
    if colors[u] == GRAY:
      return False
    if colors[u] == WHITE:
      if not TopSortDfs(G, u):
        return False
  colors[v] = BLACK
  top_sorted.push_back(v)
  return True

def TopSort(G):
  colors = [WHITE, ..., WHITE]
  top_sorted = []
  for v in G.V:
    if colors[v] == WHITE:
      if not TopSortDfs(G, v):
        return None
  return reversed(top_sorted)
```

Корректность

Теорема (корректность алгоритма Тарьяна).
Алгоритм $TopSort$ корректно находит топологическую сортировку вершин, либо сообщает о наличии цикла.

Корректность

Теорема (корректность алгоритма Тарьяна).
Алгоритм $TopSort$ корректно находит топологическую сортировку вершин, либо сообщает о наличии цикла.

Доказательство.
Если граф содержит цикл, то по критерию ацикличности $TopSort$ его обнаружит.

Пусть цикла нет. Докажем по индукции, что в построенном массиве нет ребер, идущих слева направо (после *reversed*).

База: изначально массив *top_sorted* пуст и для него утверждение верно.

Переход: Пусть на очередном шаге положили в список вершину v. Если у нее нет соседей, то нет и ребер, ведущих налево. Если соседи есть, то в момент завершения $TopSortedDfs(G, v)$ все соседи были черные (серых нет, так как нет циклов), а значит они уже в списке $\Rightarrow$ все ребра ведут направо. ■

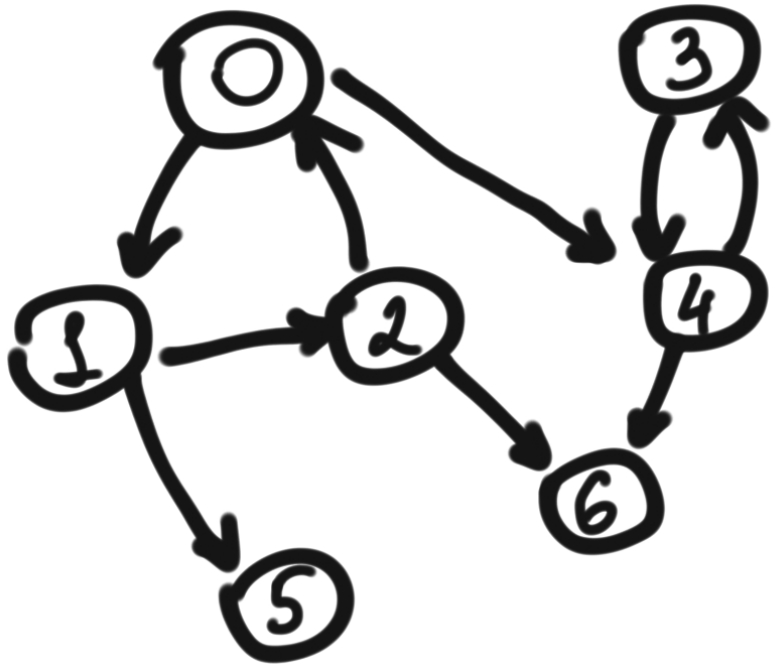
Компоненты сильной связности (КСС)

Strongly Connected Components (SCC)

Сильная связность

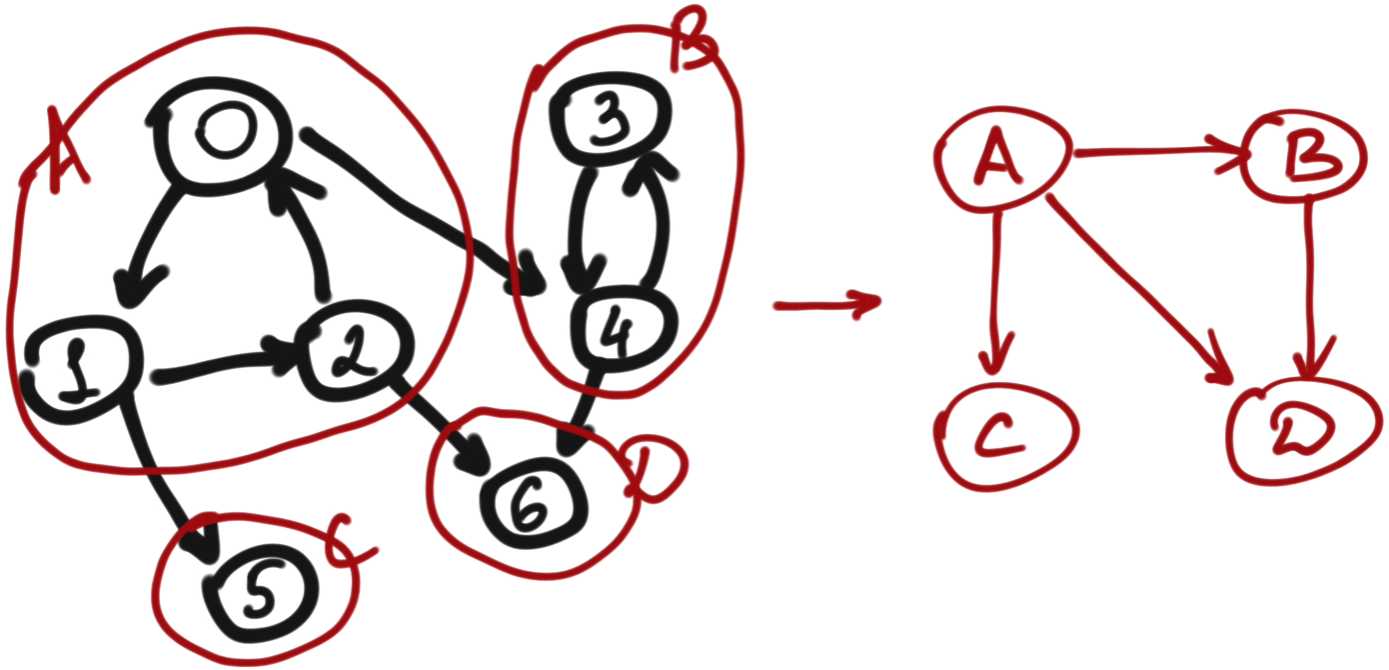
Ориентированный граф называется *сильно связным*, если из любой вершины существует путь до любой другой.

Компонента сильной связности - максимальный по включению сильно связный подграф (или класс эквивалентности по отношению взаимной достижимости).



Конденсация графа

Конденсацией ориентированного графа $G = (V, E)$ называется граф $SCC(G) = (V', E')$ такой, что V' состоит из компонент сильной связности графа G, а ребро $(A, B) \in E'$, если $\exists a \in A, b \in B : (a, b) \in E$.



Конденсация графа

Утверждение. В $SCC(G)$ нет циклов.

Конденсация графа

Утверждение. *В $SCC(G)$ нет циклов.*

Доказательство.

Допустим есть цикл. Это означает, что внутри этого цикла из любой вершины можно добраться до любой другой. А это, в свою очередь означает, что из любой вершины исходного графа, принадлежащей компоненте цикла, можно добраться до любой другой вершины этого цикла.

То есть цикл образует бОльшую компоненту сильной связности. Противоречие с определением компонент сильной связности и конденсации. ■

Конденсация графа

Лемма (в которой мы узнаем, что будет, если забыть проверять ацикличность в алгоритме топологической сортировки).
Пусть A и B компоненты сильной связности графа G и в $SCC(G)$ есть ребро из A в B. Запустим алгоритм топологической сортировки на графе G, но без проверки на ацикличность. Тогда найдется вершина $a \in A$, которая будет располагаться левее любой вершины из B.

Конденсация графа

Лемма 1 (в которой мы узнаем, что будет, если забыть проверять ацикличность в алгоритме топологической сортировки).
Пусть A и B компоненты сильной связности графа G и в $SCC(G)$ есть ребро A в B. Запустим алгоритм топологической сортировки на графе G, но без проверки на ацикличность. Тогда найдется вершина $a \in A$, которая будет располагаться левее любой вершины из B.

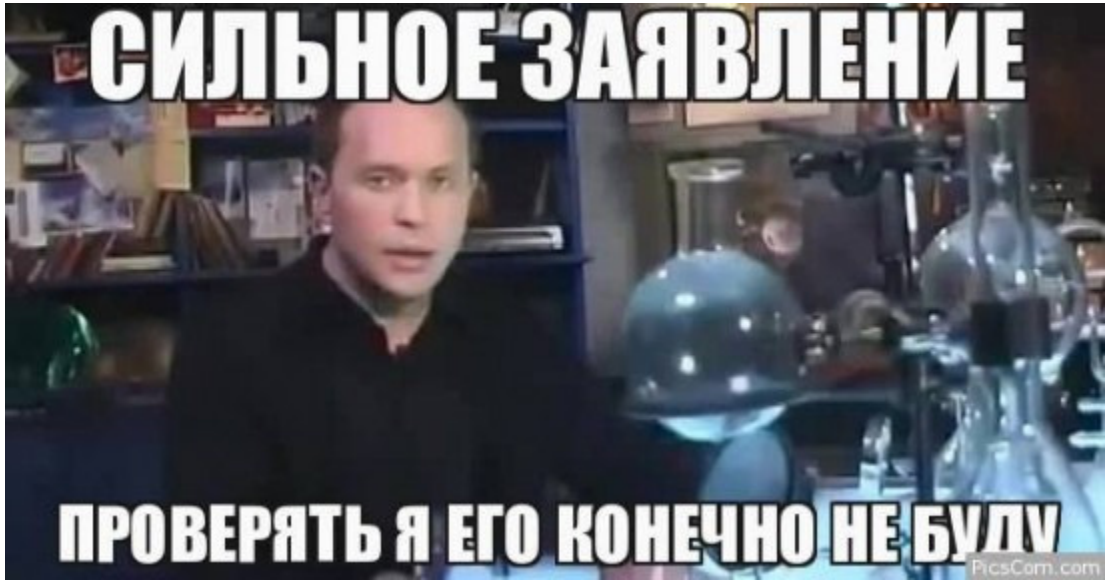
Доказательство. **1 случай)** Пусть алгоритм $TopSort$ среди вершин $A \cup B$ первой встретил вершину $a \in A$. По лемме о белых путях будут посещены все вершины из $A \cup B$, только затем a станет черной (попадет в список). Доказали.

2 случай) Первой среди $A \cup B$ была обнаружена $b \in B$. Так как в конденсации нет циклов, то по лемме о белых путях будут посещены все вершины из B, а вершины из A посещены не будут. Следовательно, все вершины из A будут располагаться левее вершин B в массиве *top_sorted*. ■

Конденсация графа

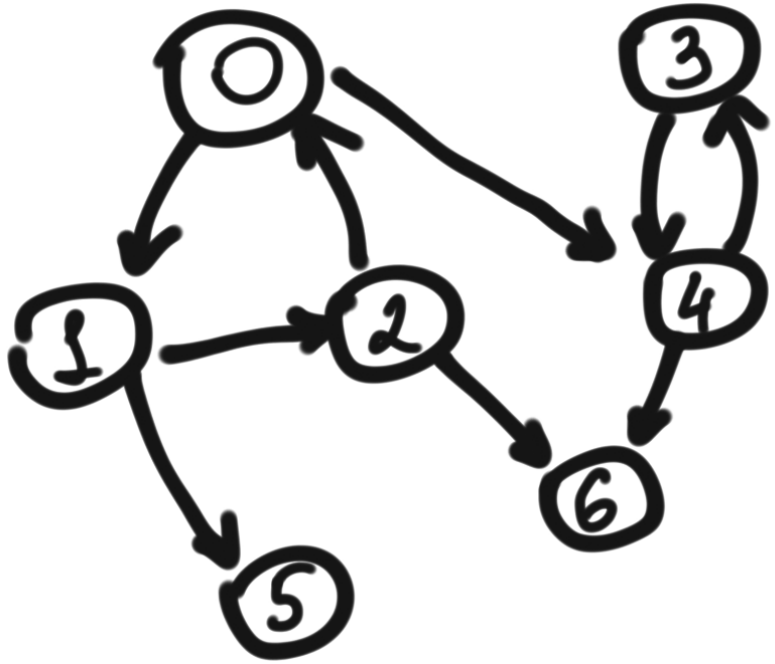
Транспонированием графа $G = (V, E)$ называется граф $G^T = (V, E^T)$, где $E^T = \{(u, v) | (v, u) \in E\}$

Лемма 2. G^T имеет те же компоненты сильной связности, что и G.



Алгоритм поиска SSC: идея

1. Запустим $TopSort$ без проверки на ацикличность.
2. Транспонируем граф.
3. Запустим DFS в порядке, полученном на шаге 1.
4. Найденные на шаге 3 компоненты - компоненты сильной связности.



Алгоритм Косарайю (Kosaraju, 1981)

```
def DFS(G, order):
  colors = [WHITE, ..., WHITE]
  components = []
  for v in order:
    if colors[v] == WHITE:
      components.emplace_back()  # добавляем новую компоненту
      DfsVisit(G, v, components[-1])  # заполняем компоненту
  return components

def SCC(G):
  order = TopSortNoCycles(G)
  components = DFS(G.T, order)
  return components
```

Алгоритм состоит из 2х *DFS* + транспонирования графа

$O(V + E)$ для списков смежности, $O(V^2)$ для матрицы смежности

Корректность

Теорема (о корректности алгоритма Косарайю).

Алгоритм Косарайю корректно находит все компоненты сильной связности.

Корректность

Теорема (о корректности алгоритма Косарайю).
Алгоритм Косарайю корректно находит все компоненты сильной связности.

Доказательство.

После первого шага $TopSortNoCycles$ по **лемме 1** отсортировали компоненты сильной связности топологически, а после шага 2 (транспонирование) по **лемме 2** остались прежние компоненты сильной связности.

Рассмотрим 3 шаг. По **лемме 1** в начале $order$ лежит вершина, которая принадлежит компоненте, в которую после транспонирования, возможно, входят ребра из других компонент, но не исходит ни одного. Значит по лемме о белых путях обойдем только вершины из этой компоненты.

Аналогично, у SCC следующей вершины в $order$ есть исходящие ребра только в найденные SCC, а в другие - нет. Следовательно посетим только одну SCC ■