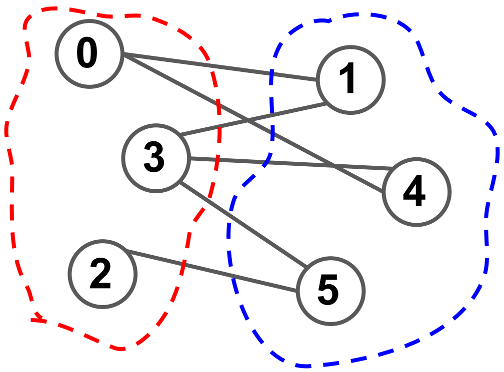
Мини-задача #29 (1 балл)

По заданному неориентированному (и, возможно, несвязному) графу понять, является ли он двудольным.

https://leetcode.com/problems/is-graph-bipartite/



Мини-задача #30 (2 балла)

Задан набор ограничений на элементы: каждый элемент может или нет находится в некоторой группе, и должен быть расположен перед некоторыми другими элементами.

Упорядочить все элементы так, чтобы все ограничения на условия предшествования были выполнены, и при этом элементы в группах стояли рядом друг с другом.

Если это невозможно - вернуть пустой список.

https://leetcode.com/problems/sort-items-by-groups-respecting-dependencies

Мини-задача #31 (2 балла)

Пусть после статического анализа про каждую функцию в программе известно, вызовы каких других функций присутствуют в ее коде. Формат входных данных:

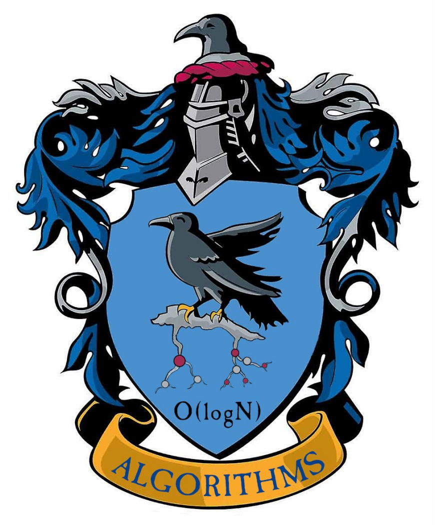
```
foo: bar, baz, qux
bar: baz, foo, bar
qux: qux
```

Вам необходимо найти наибольшую рекурсивную компоненту: множество функций, при исполнении которых можно попасть в любую другую функцию из этого множества.

Кроме того, для каждой функции определить, есть ли в ней рекурсивные (в т.ч. непрямые) вызовы.

Алгоритмы и структуры данных

Графы: применения DFS, Topsort, SCC



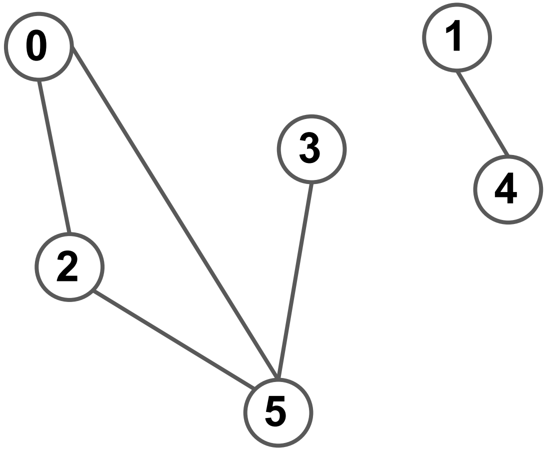
Графы: терминология

Графы: терминология

$G = (V, E)$ — граф

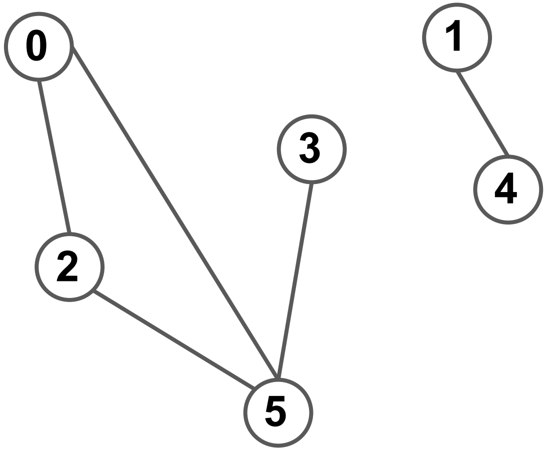
V — множество вершин

$E \subseteq V \times V$ — множество ребер



Графы: терминология

неориентированный



$$G = (V, E)$$ — граф

V — множество вершин

$E \subseteq V \times V$ — множество ребер

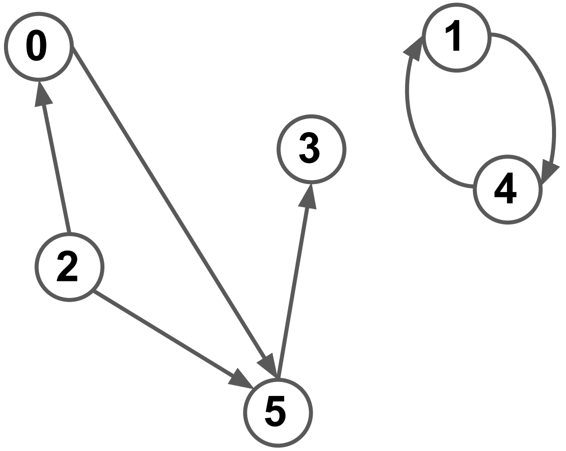
Графы: терминология

$G = (V, E)$ — граф

V — множество вершин

$E \subseteq V \times V$ — множество ребер

ориентированный



Графы: терминология

$G = (V, E)$ — граф

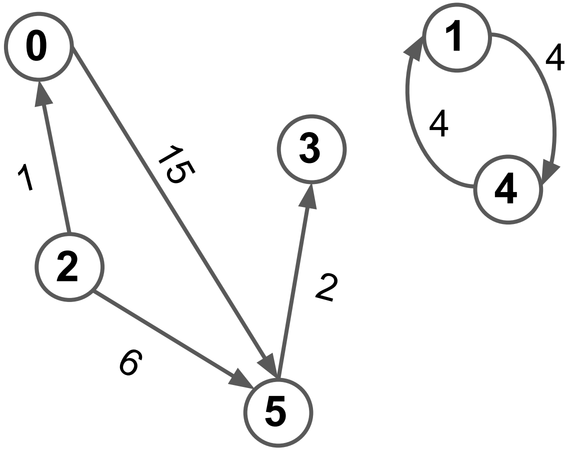
V — множество вершин

$E \subseteq V \times V$ — множество ребер

$f : E \to Weights$

— весовая функция

ориентированный



взвешенный

Графы: терминология

$G = (V, E)$ — граф

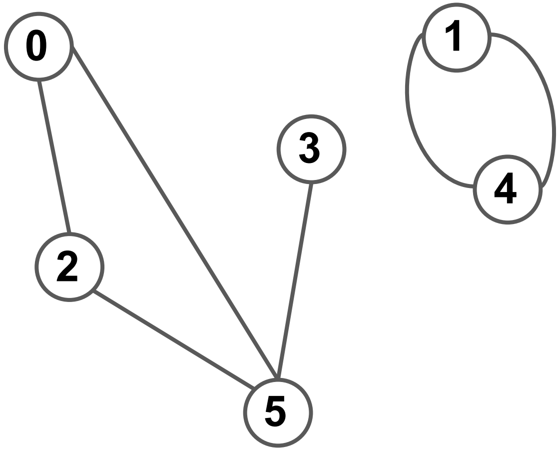
Путь в графе из k в p:

$e_0, e_1, \ldots, e_n \in E$

$e_0 = (k, b_0);$

$e_n = (a_0, p);$

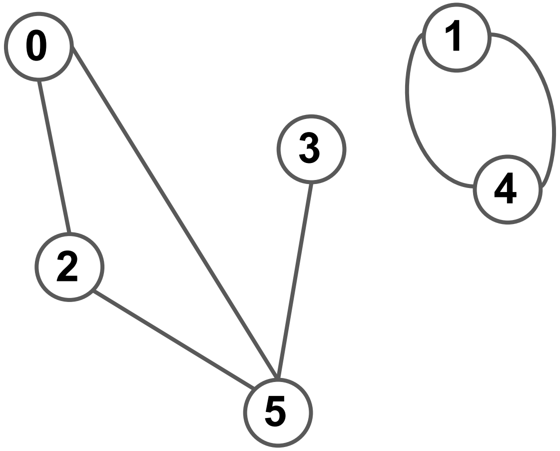
$\forall 0 < i \leq n : e_{i-1} = (a_{i-1}, b_{i-1}), e_i = (a_i, bi) \Rightarrow b_{i-1} = a_i$



Графы: терминология

$G = (V, E)$ − граф

Вершина k достижима из вершины p,
если существует путь из k в p

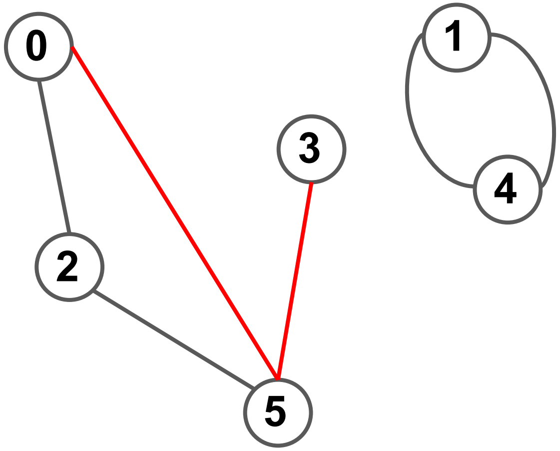


Графы: терминология

$G = (V, E)$ – граф

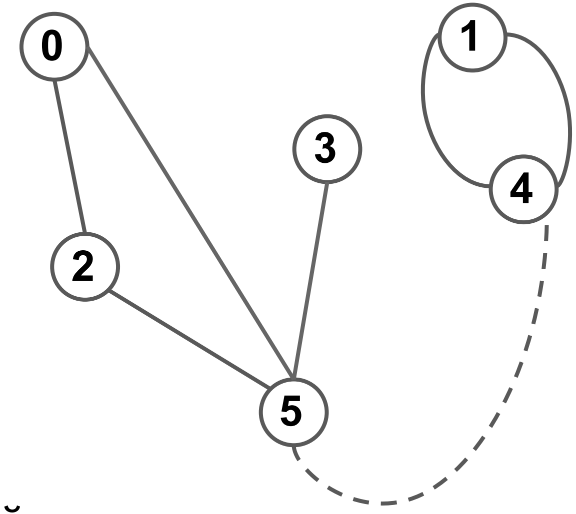
Вершина k достижима из вершины p, если существует путь из k в p

Например, 3 достижима из 0



Графы: терминология

$G = (V, E)$ — граф



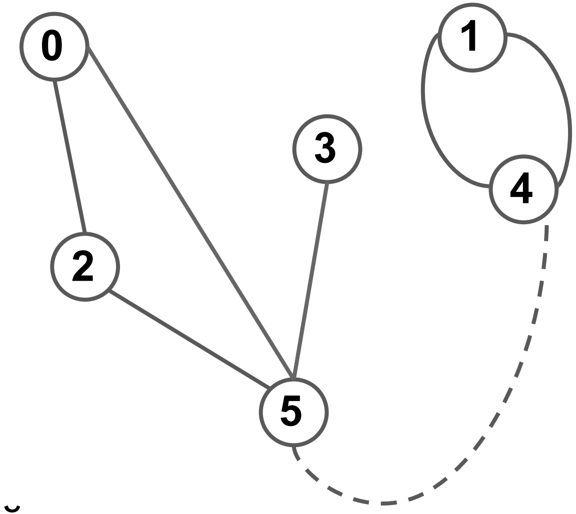
Вершина k достижима из вершины p, если существует путь из k в p

Например, 3 достижима из 0

Граф называется связным, если для каждой пары вершин есть хотя бы один соединяющий их путь

Графы: терминология

$G = (V, E)$ — граф



Вершина k достижима из вершины p,
если существует путь из k в p

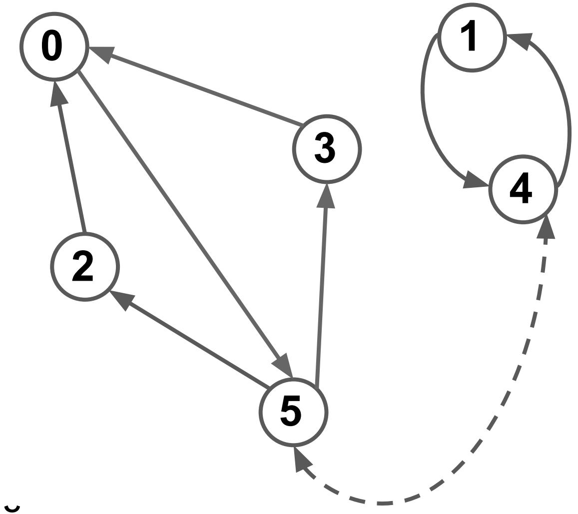
Например, 3 достижима из 0

Граф называется связным, если для каждой пары вершин есть хотя бы один соединяющий их путь

Для ориентированных вводят сильно- и слабо-связные графы

Графы: терминология

$$G = (V, E)$$ — граф



Вершина k достижима из вершины p,
если существует путь из k в p

Например, 3 достижима из 0

Граф называется связным, если для каждой пары вершин есть хотя бы один соединяющий их путь

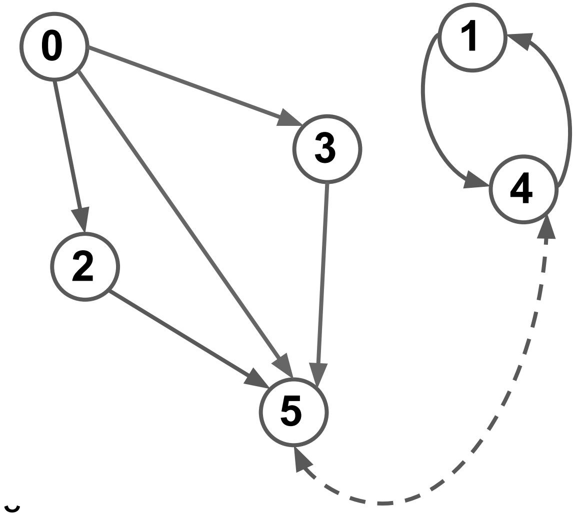
Для ориентированных сильно-связные графы - соответствуют определению выше

Графы: терминология

$G = (V, E)$ – граф

Вершина k достижима из вершины p,
если существует путь из k в p

Например, 3 достижима из 0



Граф называется связным, если для каждой пары вершин есть хотя бы один соединяющий их путь

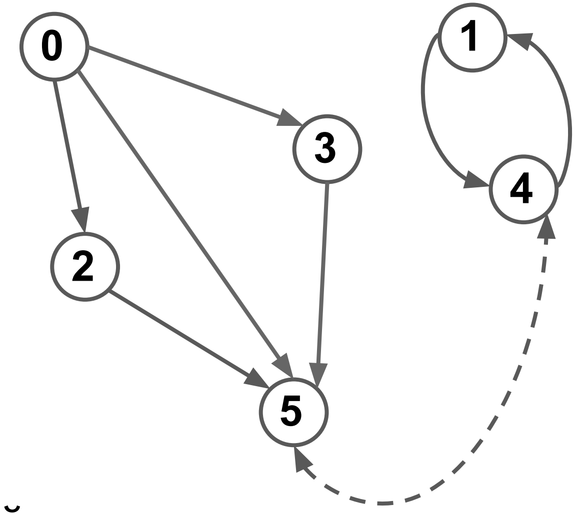
Для ориентированных слабо-связные графы

Графы: терминология

$G = (V, E)$ – граф

Вершина k достижима из вершины p,
если существует путь из k в p

Например, 3 достижима из 0

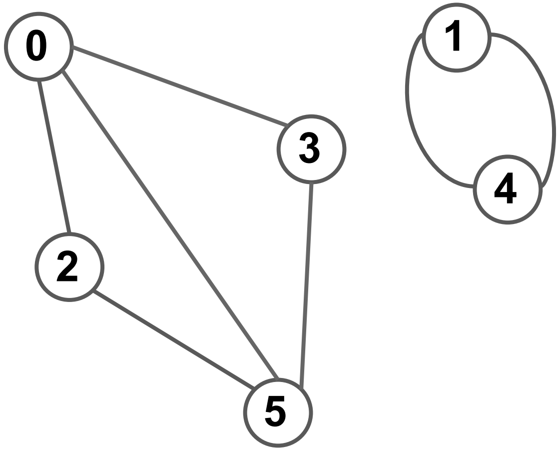


Граф называется связным, если для каждой пары вершин есть хотя бы один соединяющий их путь

Для ориентированных слабо-связные графы - если заменить ребра на неориентированные, получится связный граф.

Графы: терминология

$G = (V, E)$ – граф



Вершина k достижима из вершины p,
если существует путь из k в p

Например, 3 достижима из 0

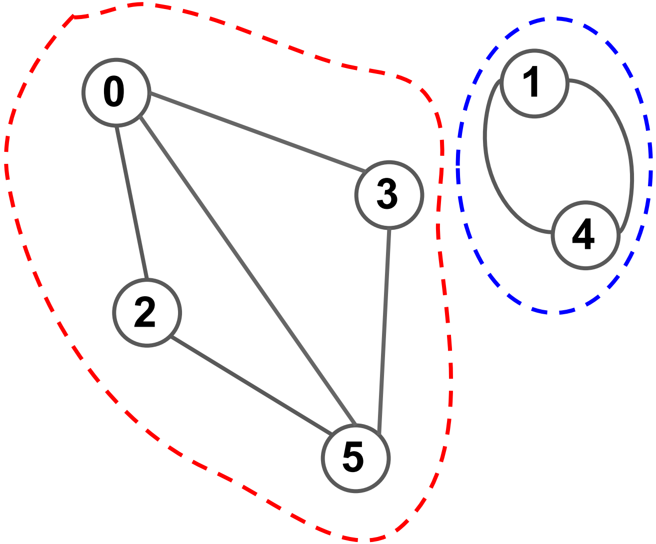
Для несвязных (неориентированных) графов вводят понятие компонент связности.

Графы: терминология

$G = (V, E)$ – граф

Вершина k достижима из вершины p, если существует путь из k в p

Например, 3 достижима из 0



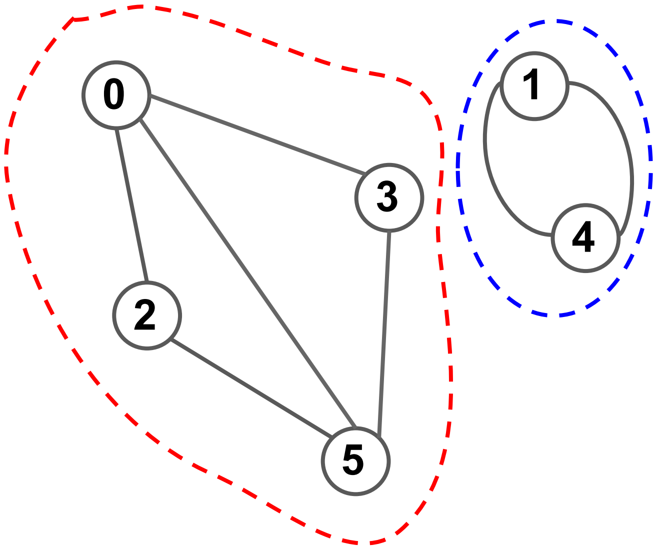
Для несвязных (неориентированных) графов вводят понятие компонент связности.

Графы: терминология

$G = (V, E)$ — граф

Вершина k достижима из вершины p,
если существует путь из k в p

Например, 3 достижима из 0



Для несвязных (неориентированных) графов вводят понятие компонент связности.

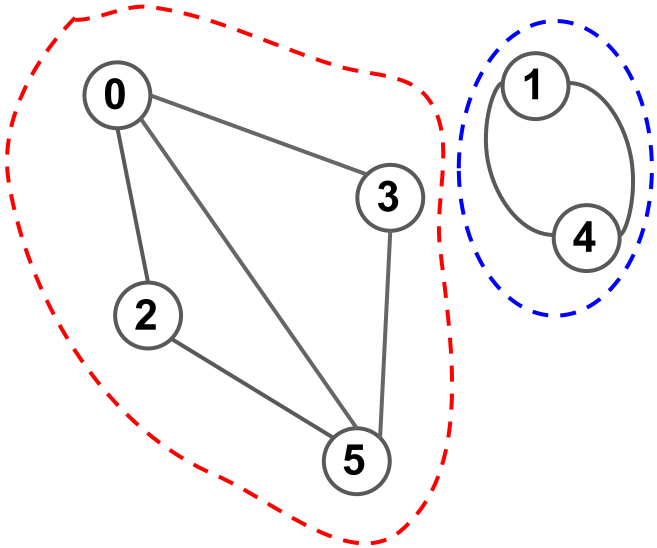
Связные подграфы, между которыми нет ребер.

Графы: терминология

$G = (V, E)$ — граф

Вершина k достижима из вершины p,
если существует путь из k в p

Например, 3 достижима из 0



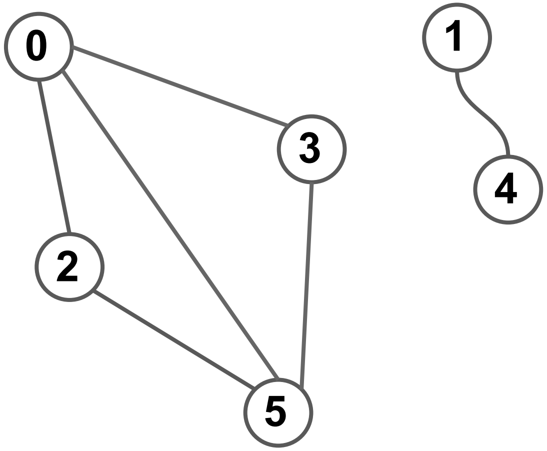
Для ориентированных графов вводят понятие компонент сильной связности (SCC).

Про них поговорим позже.

Графы: терминология

$G = (V, E)$ — граф

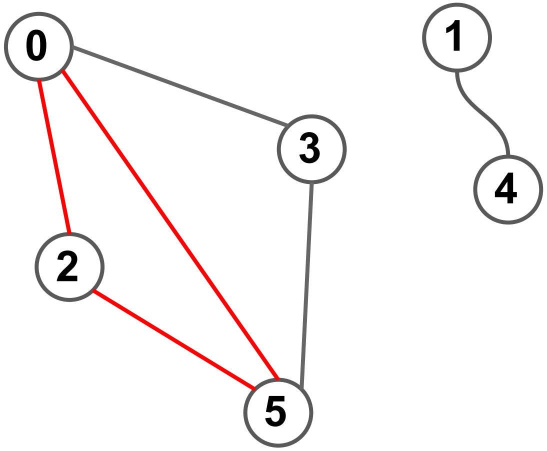
Цикл в графе - путь, который начинается и заканчивается в одной вершине



Графы: терминология

$$G = (V, E)$$ — граф

(Простой) цикл в графе - путь, который начинается и заканчивается в одной вершине (а все ребра при этом различные)



Графы: представление

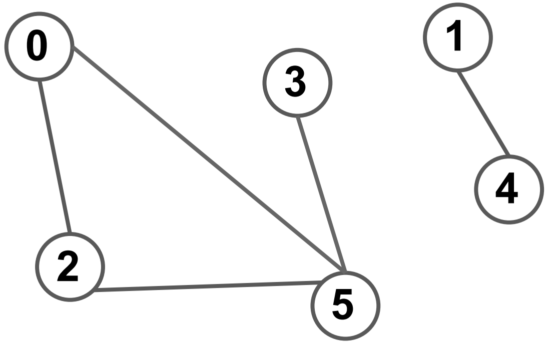
Графы: представление

Матрица смежности

$$|V| = n$$

$$A = \begin{pmatrix} a_{00} & \cdots & a_{0(n-1)} \\ \vdots & \ddots & \vdots \\ a_{(n-1)0} & \cdots & a_{(n-1)(n-1)} \end{pmatrix}$$

$$a_{ij} = 1 \Rightarrow (i, j) \in E$$



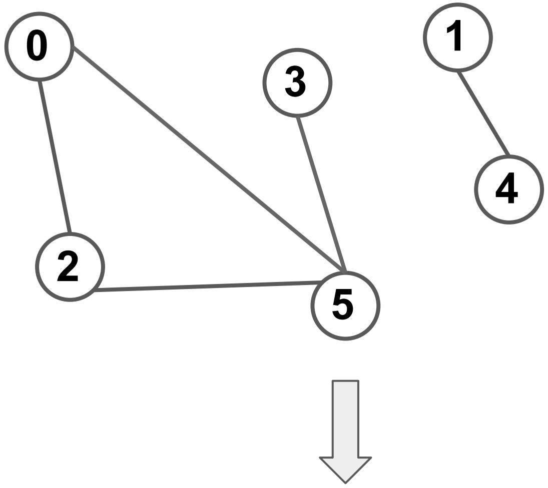
Графы: представление

Матрица смежности

$$|V| = n$$

$$A = \begin{pmatrix} a_{00} & \cdots & a_{0(n-1)} \\ \vdots & \ddots & \vdots \\ a_{(n-1)0} & \cdots & a_{(n-1)(n-1)} \end{pmatrix}$$

$$a_{ij} = 1 \Rightarrow (i, j) \in E$$



$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

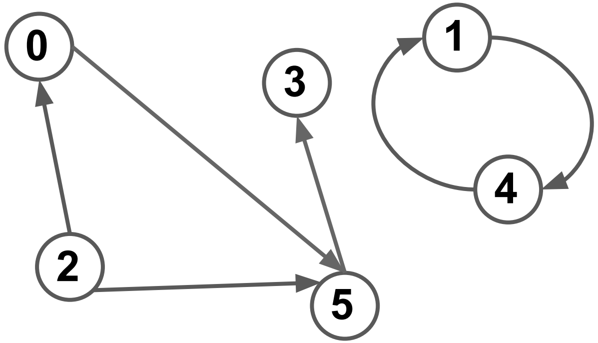
Графы: представление

Матрица смежности

$$|V| = n$$

$$A = \begin{pmatrix} a_{00} & \cdots & a_{0(n-1)} \\ \vdots & \ddots & \vdots \\ a_{(n-1)0} & \cdots & a_{(n-1)(n-1)} \end{pmatrix}$$

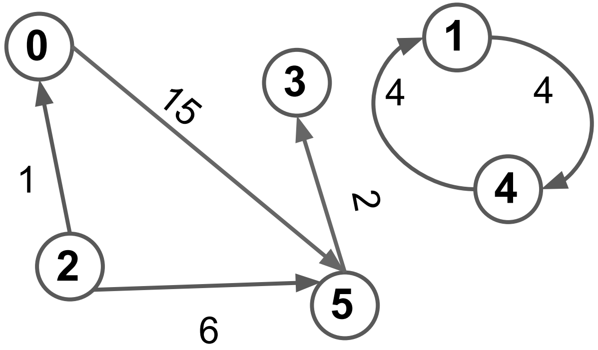
$$a_{ij} = 1 \Rightarrow (i, j) \in E$$



$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Графы: представление

Матрица смежности



$$|V| = n$$

$$A = \begin{pmatrix} a_{00} & \cdots & a_{0(n-1)} \\ \vdots & \ddots & \vdots \\ a_{(n-1)0} & \cdots & a_{(n-1)(n-1)} \end{pmatrix}$$

$$a_{ij} \neq 0 \Rightarrow (i, j) \in E; weight((i, j)) = a_{ij}$$

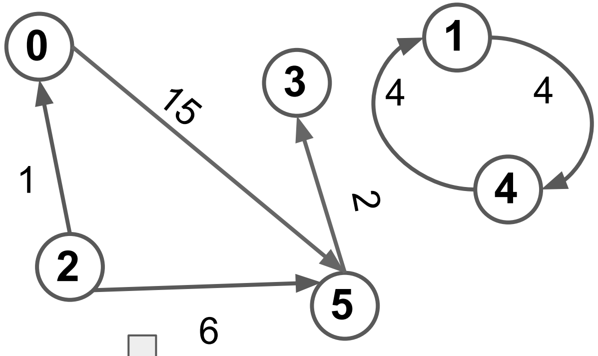
Графы: представление

Матрица смежности

$$|V| = n$$

$$A = \begin{pmatrix} a_{00} & \cdots & a_{0(n-1)} \\ \vdots & \ddots & \vdots \\ a_{(n-1)0} & \cdots & a_{(n-1)(n-1)} \end{pmatrix}$$

$$a_{ij} \neq 0 \Rightarrow (i, j) \in E; weight((i, j)) = a_{ij}$$



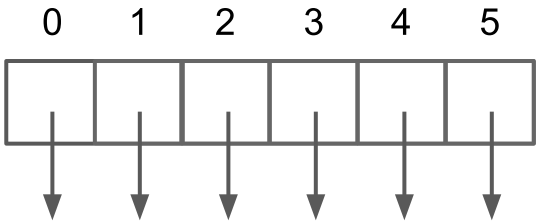
$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 15 & 0 & 6 & 0 & 0 & 0 \end{pmatrix}$$

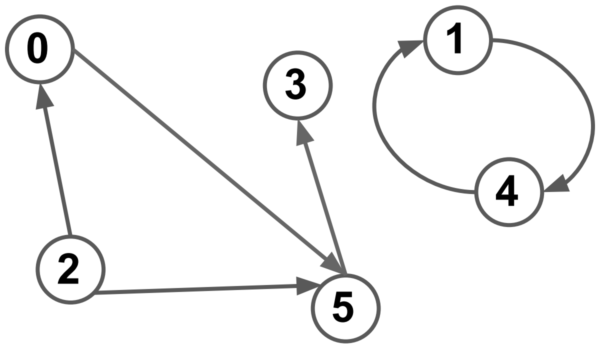
Графы: представление

Списки смежности

Графы: представление

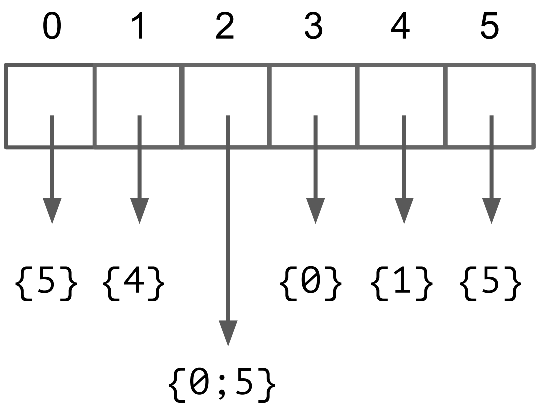
Списки смежности

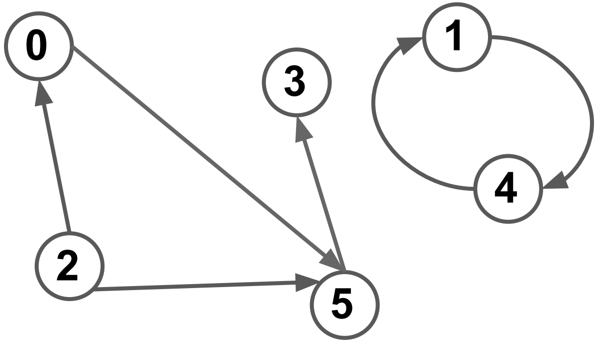




Графы: представление

Списки смежности

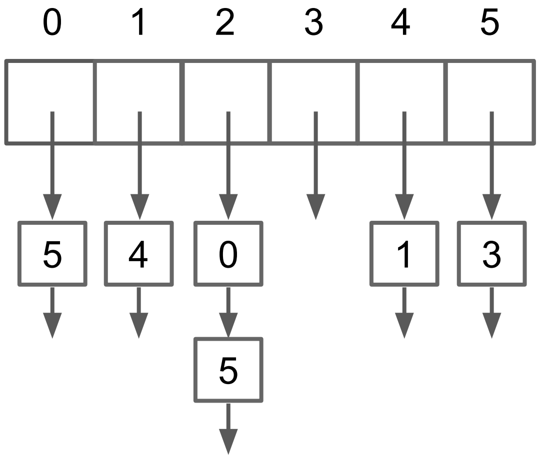


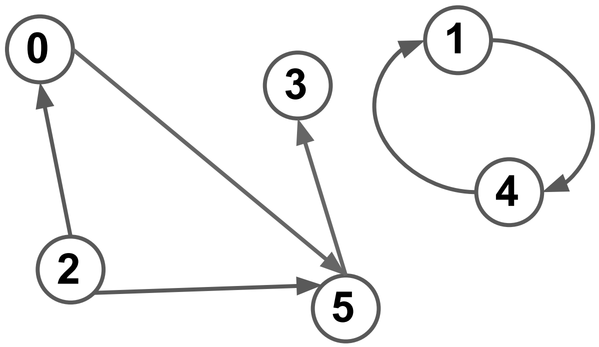


Для каждой вершины коллекция
из смежных вершин

Графы: представление

Списки смежности





Для каждой вершины коллекция
из смежных вершин
(списки, массивы, хеш-мапы)

Графы: представление

Что выбрать списки:
смежности или матрицы смежности?

Графы: представление

Что выбрать списки:
смежности или матрицы смежности?

Зависит от:

1. Алгоритма (списки для Дейкстры, матрицы для Флойда-Уоршелла)

Графы: представление

Что выбрать списки:
смежности или матрицы смежности?

Зависит от:

1. Алгоритма (списки для Дейкстры, матрицы для Флойда-Уоршелла)

2. Требований по памяти

Графы: представление

Что выбрать списки:
смежности или матрицы смежности?

Зависит от:

1. Алгоритма (списки для Дейкстры, матрицы для Флойда-Уоршелла)

2. Требований по памяти

3. Характеристики графа

Графы: алгоритмы

В чем отличие графов от деревьев?

Графы: алгоритмы

В чем отличие графов от деревьев?

1. Они бывают несвязными

2. В них бывают циклы!

Графы: алгоритмы

В чем отличие графов от деревьев?

1. Они бывают несвязными

2. В них бывают циклы!

Два этих момента стоит учитывать в обходах

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать?

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать? Любым обходом - DFS или BFS

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать? Любым обходом - DFS или BFS

```
def check(from, to, visited: bool[]) -> bool:
```

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать? Любым обходом - DFS или BFS

```
def check(from, to, visited: bool[]) -> bool:
    visited[from] = True
```

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать? Любым обходом - DFS или BFS

```
def check(from, to, visited: bool[]) -> bool:
    visited[from] = True
    if from == to: return True
```

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать? Любым обходом - DFS или BFS

```
def check(from, to, visited: bool[]) -> bool:
    visited[from] = True
    if from == to: return True
    for next in adj(from):
        if visited[next]: continue
        if check(next, to, visited): return True
```

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать? Любым обходом - DFS или BFS

```
def check(from, to, visited: bool[]) -> bool:
    visited[from] = True
    if from == to: return True
    for next in adj(from):
        if visited[next]: continue
        if check(next, to, visited): return True
    return False
```

Графы: обходы

Задача: дан (не)ориентированный граф, проверить, достижима ли вершина k из вершины p

Как решать? Любым обходом - DFS или BFS

visited изначально состоит из False

```
def check(from, to, visited: bool[]):
    visited[from] = True
    if from == to: return True
    for next in adj(from):
        if visited[next]: continue
        if check(next, to, visited): return True
    return False
```

защита от зацикливания

Графы: обходы

Задача: дан неориентированный граф, проверить, является ли он связным.

Как решать?

Графы: обходы

Задача: дан неориентированный граф, проверить, является ли он связным.

Как решать?

Запускаем обход от любой вершины =>
Смотрим, что после этого осталось в visited =>
Если остались False, то граф несвязный.

Графы: обходы

Задача: дан неориентированный граф, проверить, является ли он связным.

Как решать?

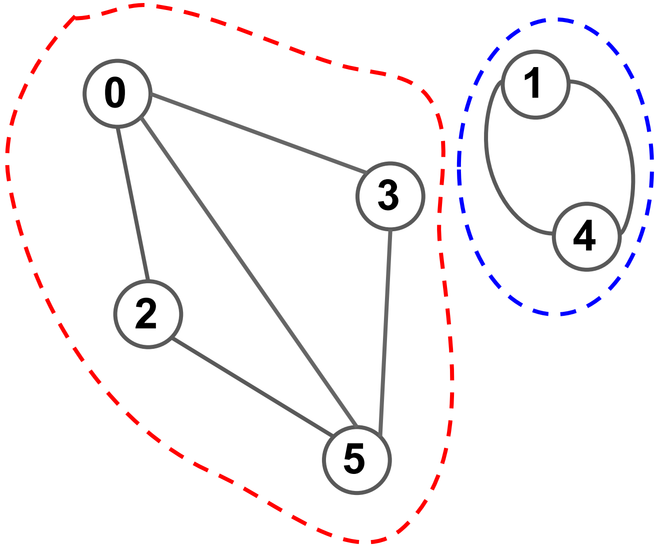
искать при этом уже ничего не нужно, просто размечаем visited

Запускаем обход от любой вершины =>
Смотрим, что после этого осталось в visited =>
Если остались False, то граф несвязный.

Графы: обходы

Задача: дан неориентированный граф. Найти количество компонент связности.

Как решать?



Графы: обходы

Задача: дан неориентированный граф. Найти количество компонент связности.

Как решать?

1) Заводим глобальный счетчик (заносим в него 0)
2) Запускаем обход от любой непосещенной вершины

Графы: обходы

Задача: дан неориентированный граф. Найти количество компонент связности.

Как решать?

1) Заводим глобальный счетчик (заносим в него 0)
2) Запускаем обход от любой непосещенной вершины
3) Во время обхода в visited ставим не True, а значение этого счетчика

Графы: обходы

Задача: дан неориентированный граф. Найти количество компонент связности.

Как решать?

1) Заводим глобальный счетчик (заносим в него 0)
2) Запускаем обход от любой непосещенной вершины
3) Во время обхода в visited ставим не True, а значение этого счетчика
4) На выходе из обхода увеличиваем счетчик
5) Переход на пункт 2)

Графы: обходы

Задача: дан неориентированный граф. Найти количество компонент связности.

Как решать?

Ответ: значение счетчика после работы алгоритма

1) Заводим глобальный счетчик (заносим в него 0)
2) Запускаем обход от любой непосещенной вершины
3) Во время обхода в visited ставим не True, а значение этого счетчика
4) На выходе из обхода увеличиваем счетчик
5) Переход на пункт 2)

Графы: обходы

Задача: дан неориентированный граф. Проверить, есть ли в нем цикл.

Графы: обходы

Задача: дан неориентированный граф. Проверить, есть ли в нем цикл.

```
def check(from, to, visited: bool[]):
    visited[from] = True
    if from == to: return True
    for next in adj(from):
        if visited[next]: continue
        if check(next, to, visited): return True
    return False
```

защита от зацикливания

Графы: обходы

Задача: дан неориентированный граф. Проверить, есть ли в нем цикл.

```
def hasCycle(from, prev, visited: bool[]):
    visited[from] = True
    for next in adj(from):
        if visited[next] and next != prev: return True
        if hasCycle(next, from, visited): return True
    return False
```

Графы: обходы

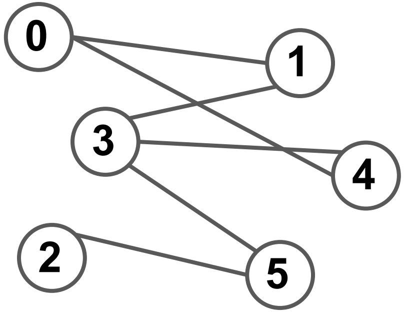
Задача: дан неориентированный граф. Проверить, есть ли в нем цикл.

```
def hasCycle(from, prev, visited: bool[]):
    visited[from] = True
    for next in adj(from):
        if visited[next] and next != prev: return True
        if hasCycle(next, from, visited): return True
    return False
```

Не забываем про другие компоненты связности!

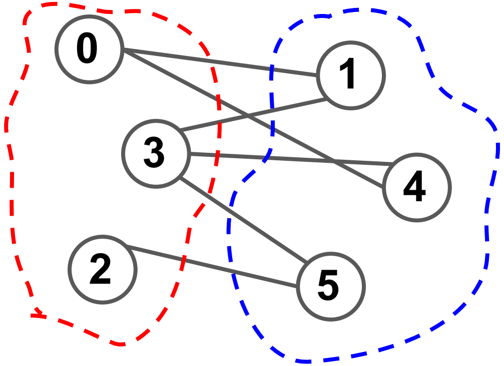
Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.



Графы: обходы

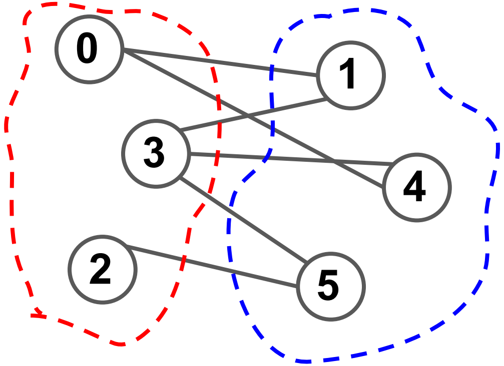
Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.



Двудольный - значит можно разделить все вершины на два множества так, чтобы все ребра графа были только между этими двумя множествами.

Графы: обходы

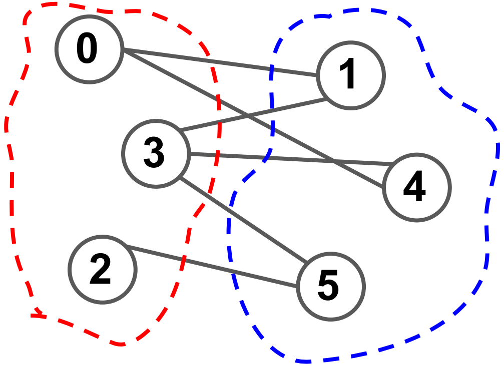
Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.



Решение: снова обход и снова пометки

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

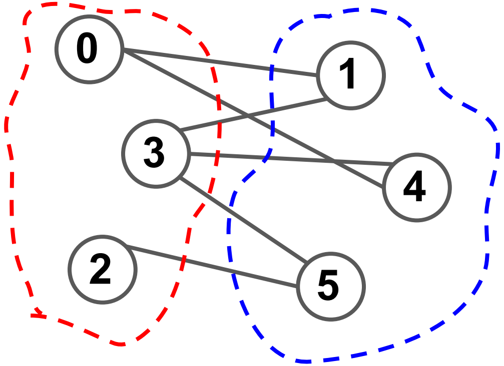


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

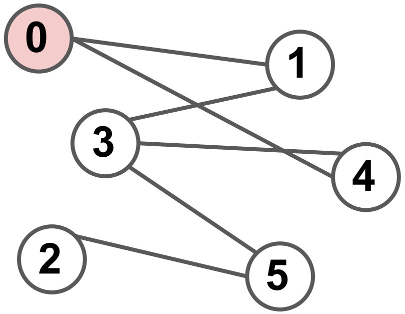


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

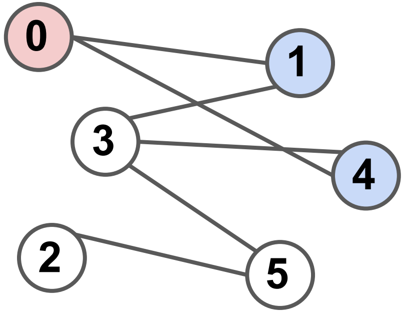


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

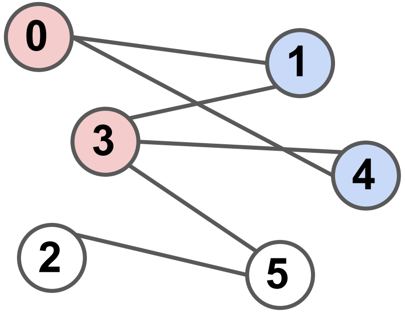


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

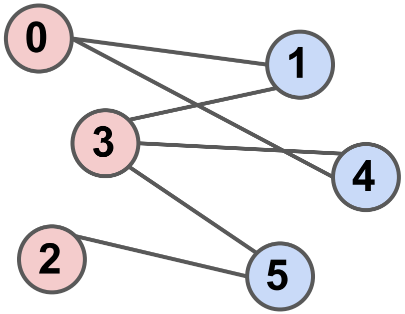


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

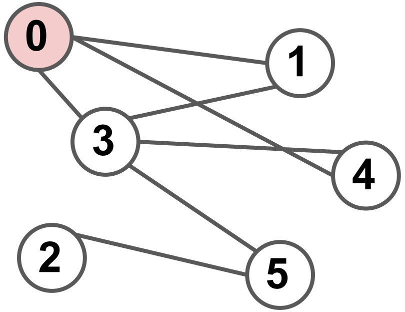


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

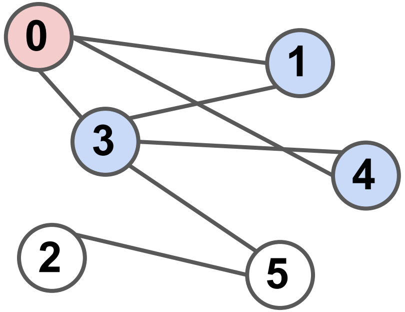


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

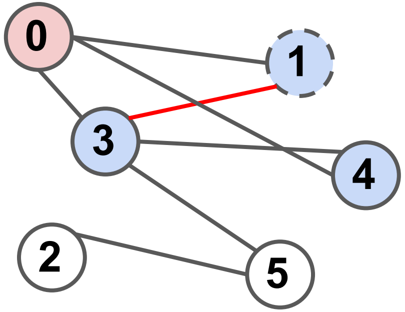


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.

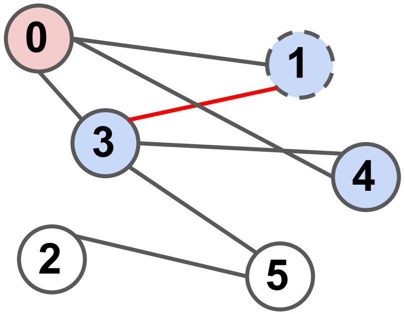


Решение: снова обход и снова пометки

- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Графы: обходы

Задача: дан неориентированный связный граф. Проверить, является ли он двудольным.



Решение: снова обход и снова пометки

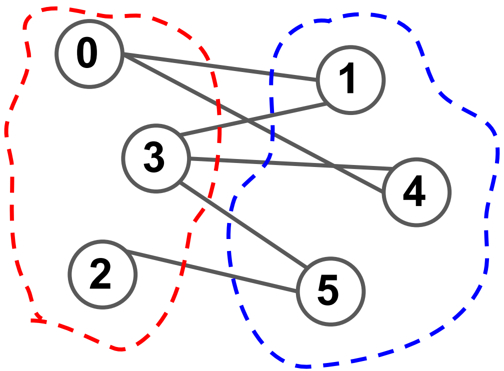
- visited теперь состоит из двух цветов - красный и синий
- в каждом рекурсивном вызове меняем цвет, который поставим в visited
- если увидели соседа своего цвета => не двудольный

Подумайте, что изменится для случая ориентированного графа?

Мини-задача #29 (1 балл)

По заданному неориентированному (и, возможно, несвязному) графу понять, является ли он двудольным.

https://leetcode.com/problems/is-graph-bipartite/

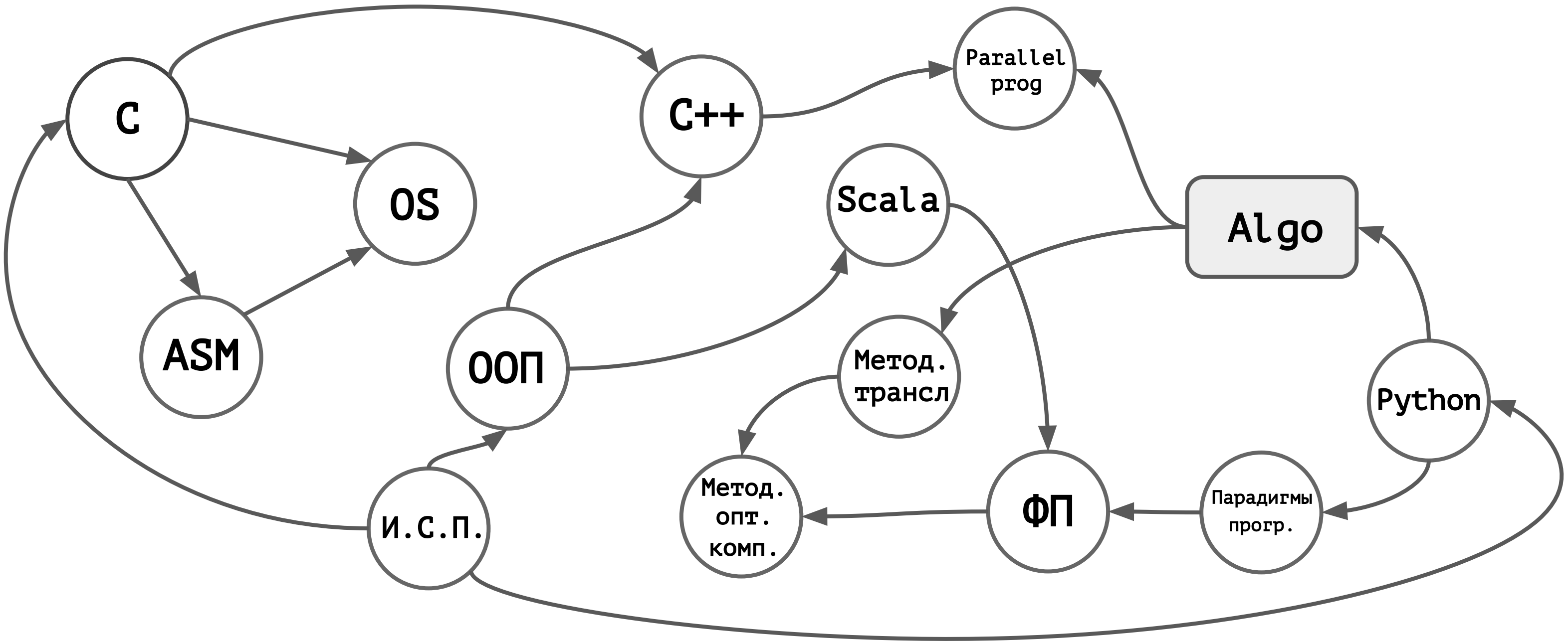


Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.



Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f: V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

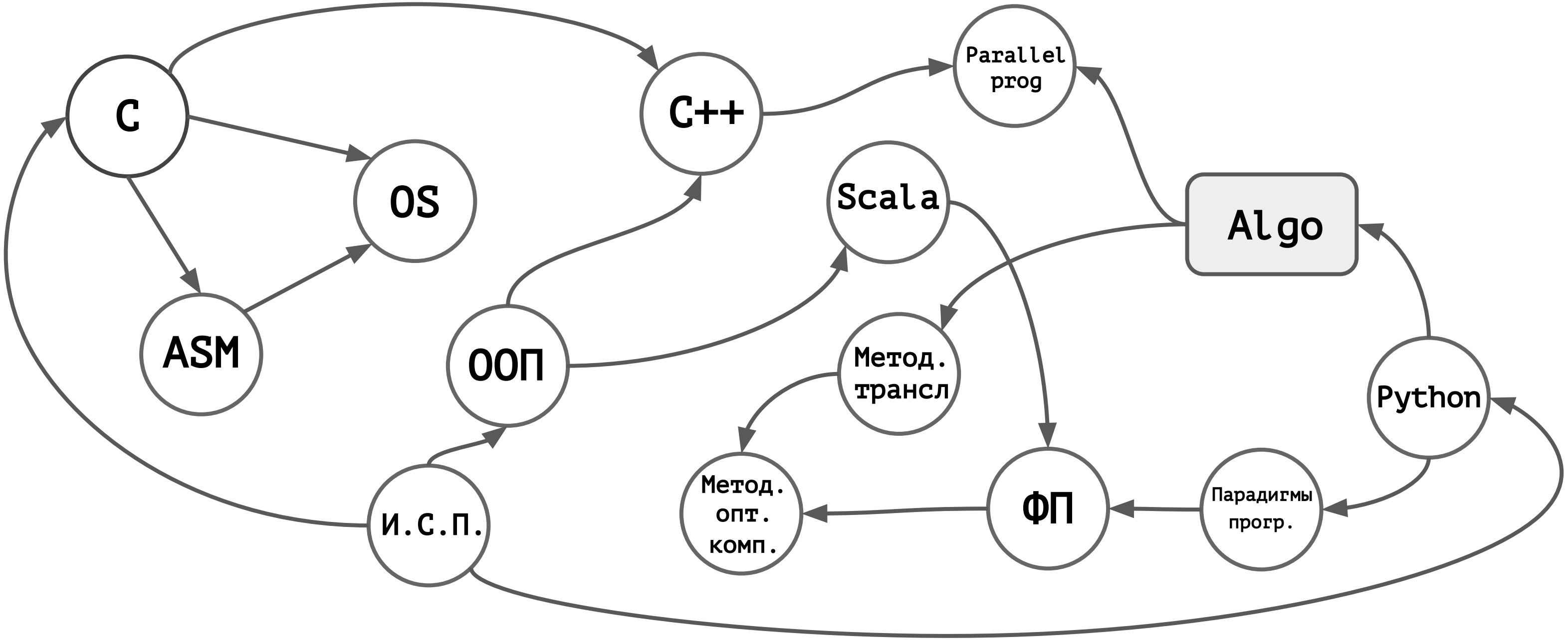
Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

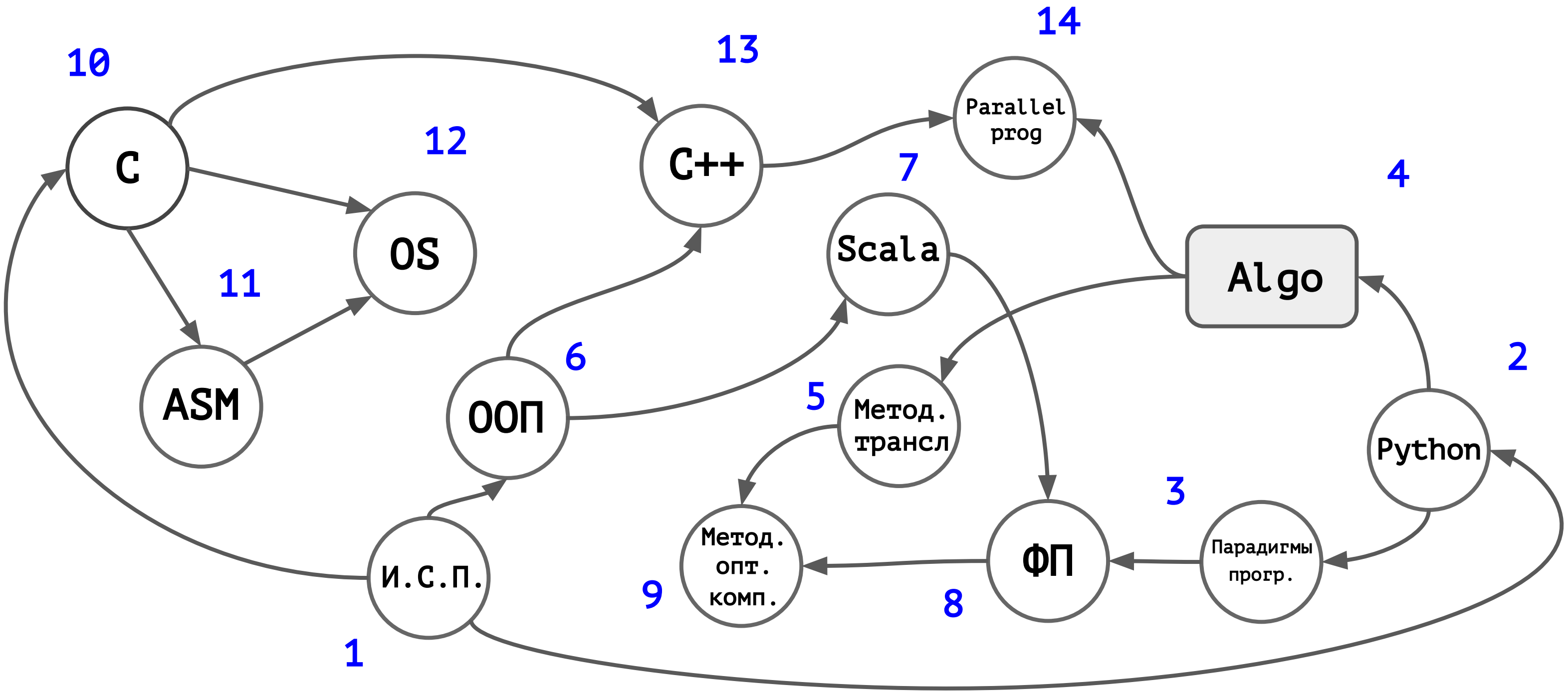
Определение: топологическая сортировка в ор. графе - означивание вершин $f : V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Идея: если научимся получать такое означивание, то сможем упорядочить курсы.

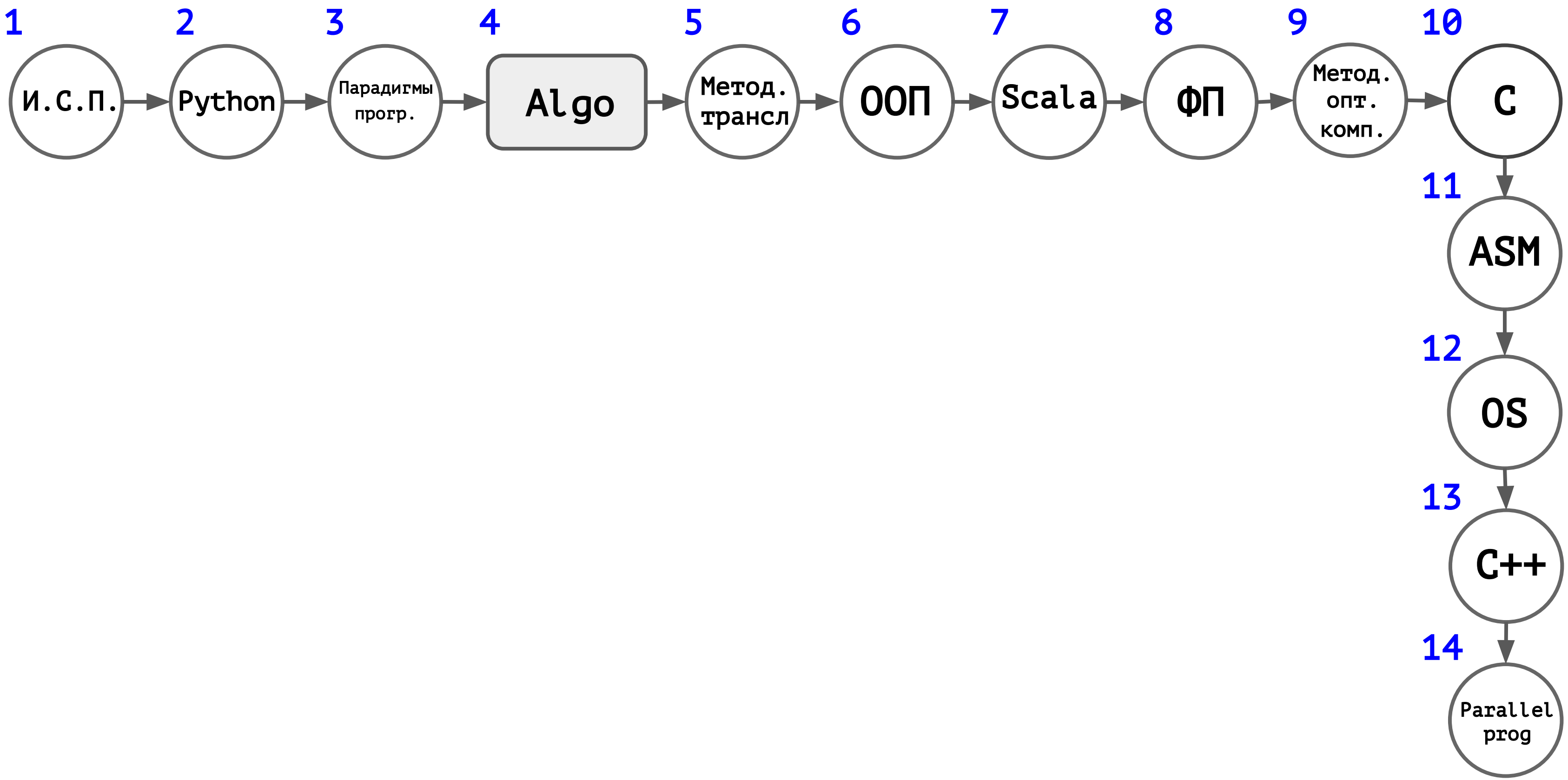
Графы: топологическая сортировка



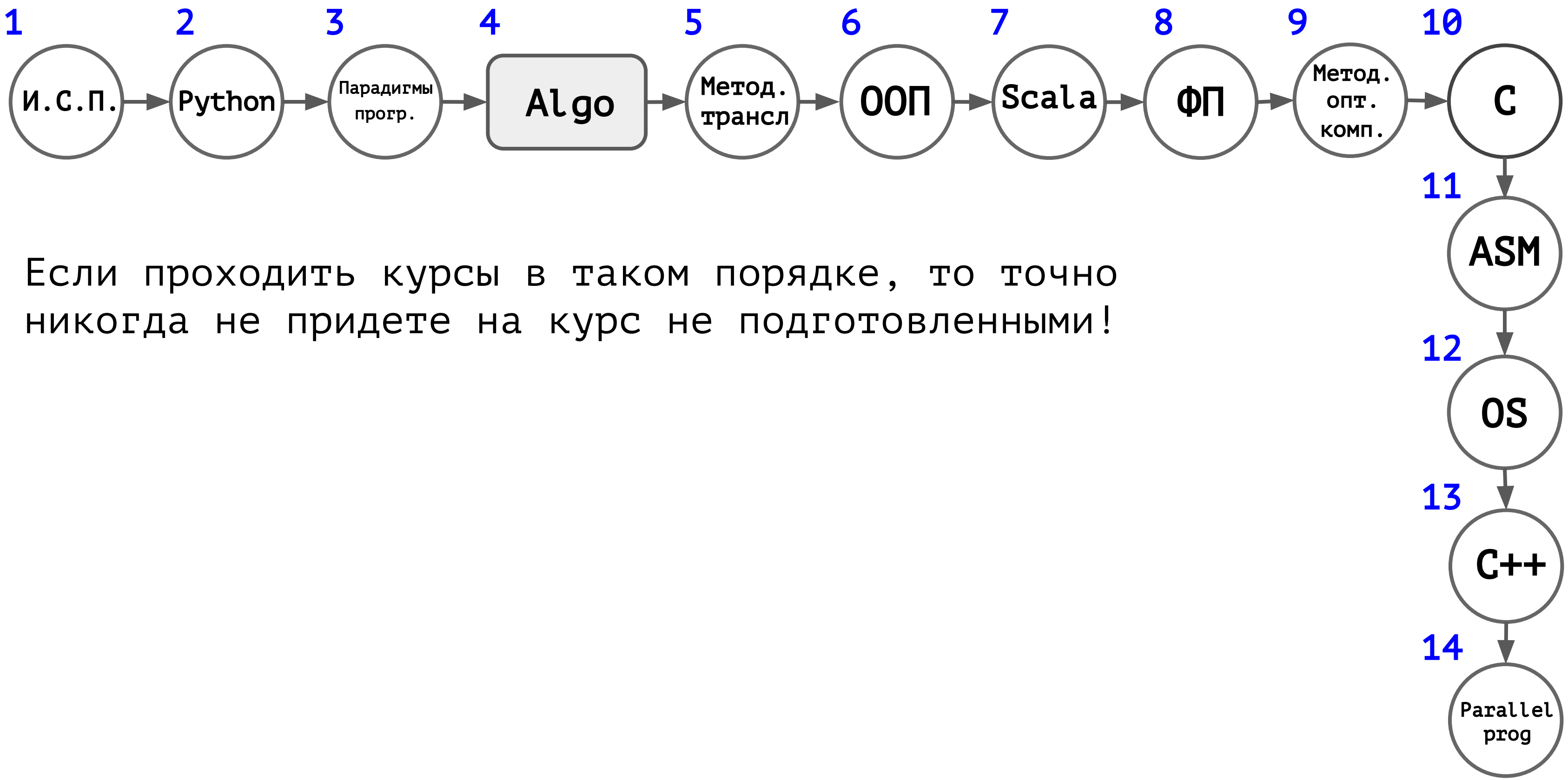
Графы: топологическая сортировка



Графы: топологическая сортировка



Графы: топологическая сортировка



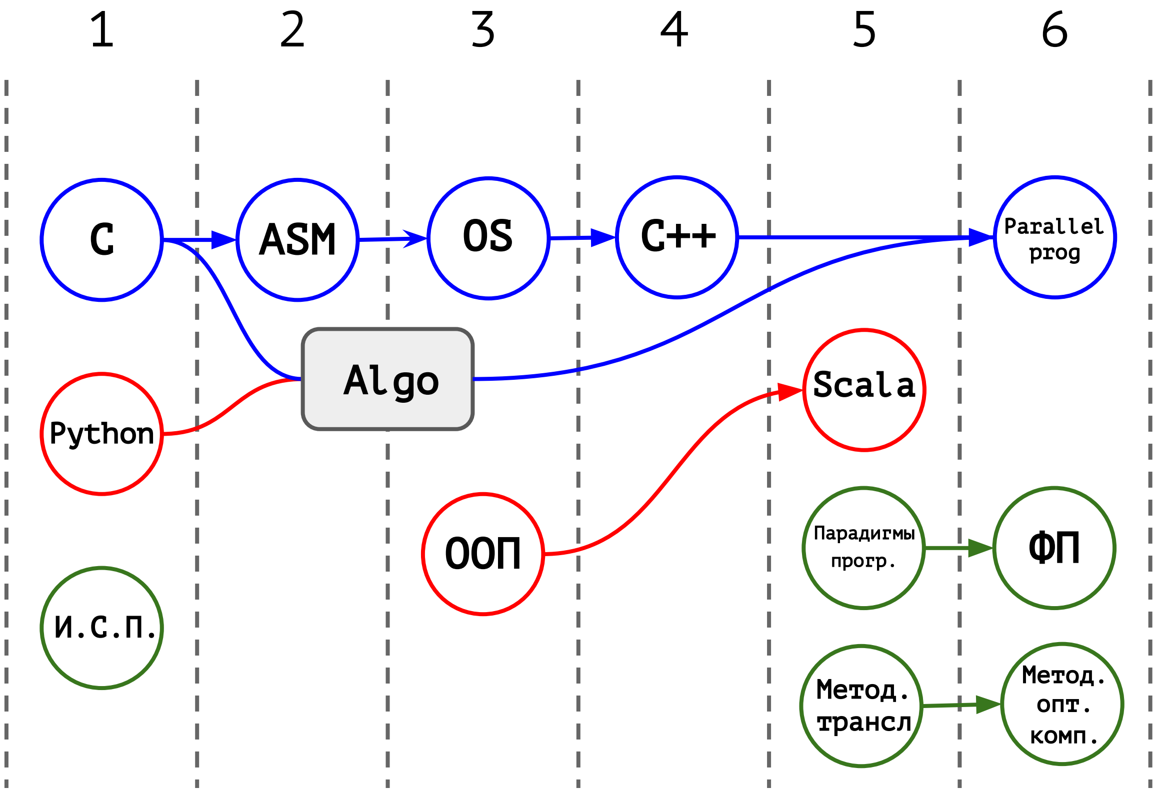
Если проходить курсы в таком порядке, то точно никогда не придете на курс не подготовленными!

Графы: топологическая сортировка

Если проходить курсы в таком порядке, то точно никогда не придете на курс не подготовленными!

Очевидно, это не единственный способ,
но точно один из них.

Графы: топологическая сортировка



Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f: V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Идея: если научимся получать такое означивание, то сможем упорядочить курсы.

Вопрос: в любом ли графе можно построить такой порядок?

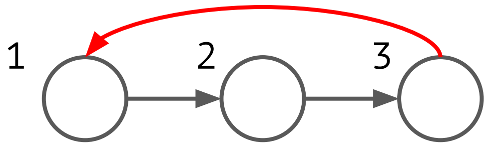
Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f : V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Идея: если научимся получать такое означивание, то сможем упорядочить курсы.

Вопрос: в любом ли графе можно построить такой порядок?
Ответ: нет, любой цикл сломает нумерацию



Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f : V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Идея: если научимся получать такое означивание, то сможем упорядочить курсы.

Вопрос: в любом ли графе можно построить такой порядок?
Ответ: нет, любой цикл сломает нумерацию

Поэтому работает только на ориентированных ациклических графах (**d**irected **a**cyclic **g**raphs).

Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

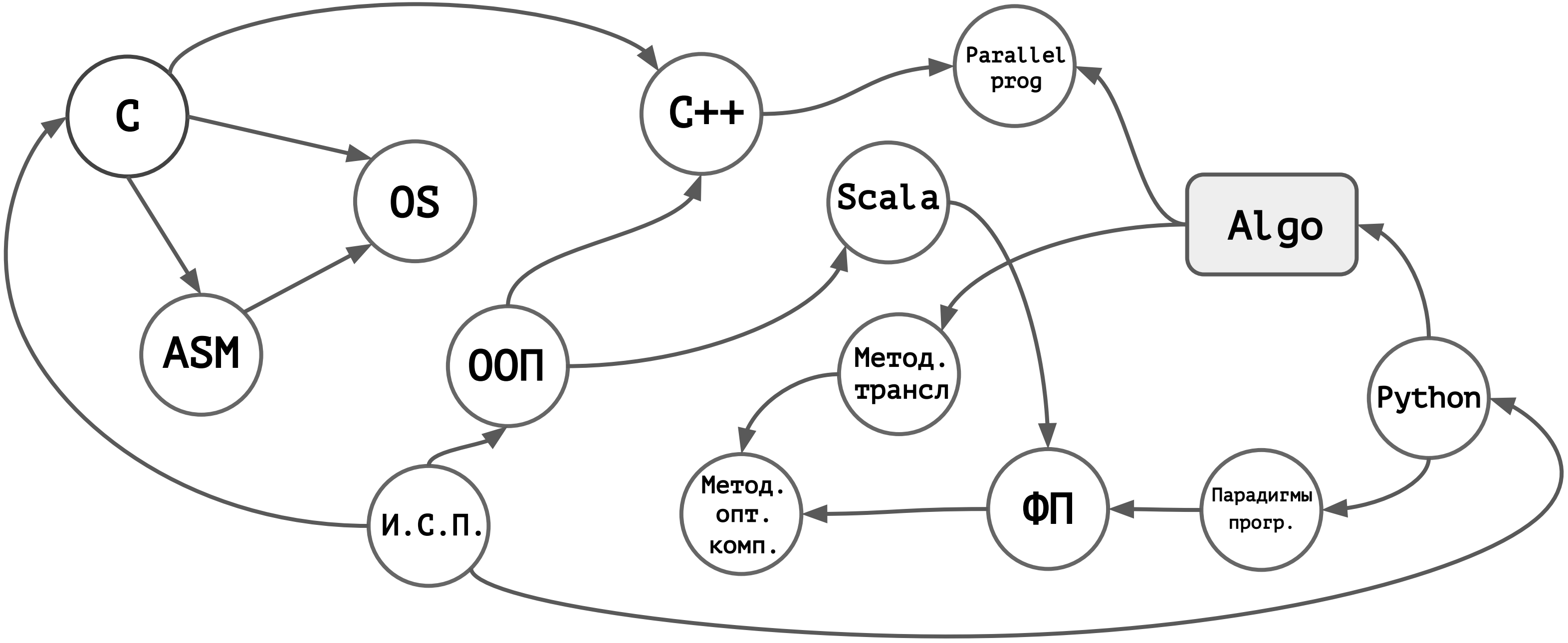
Определение: топологическая сортировка в ор. графе - означивание вершин $f: V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Идея: если научимся получать такое означивание, то сможем упорядочить курсы.

Вопрос: в любом ли графе можно построить такой порядок?
Ответ: нет, любой цикл сломает нумерацию

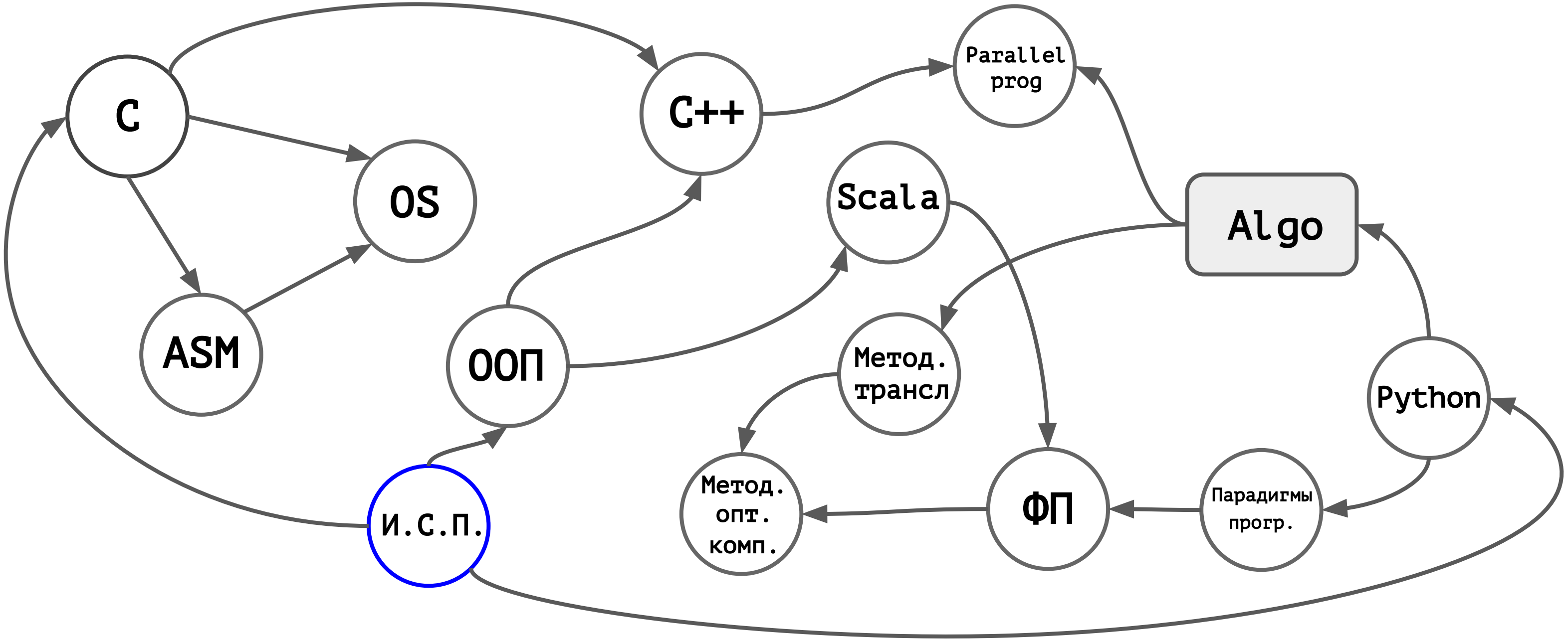
Как искать топологическую сортировку?

Графы: топологическая сортировка



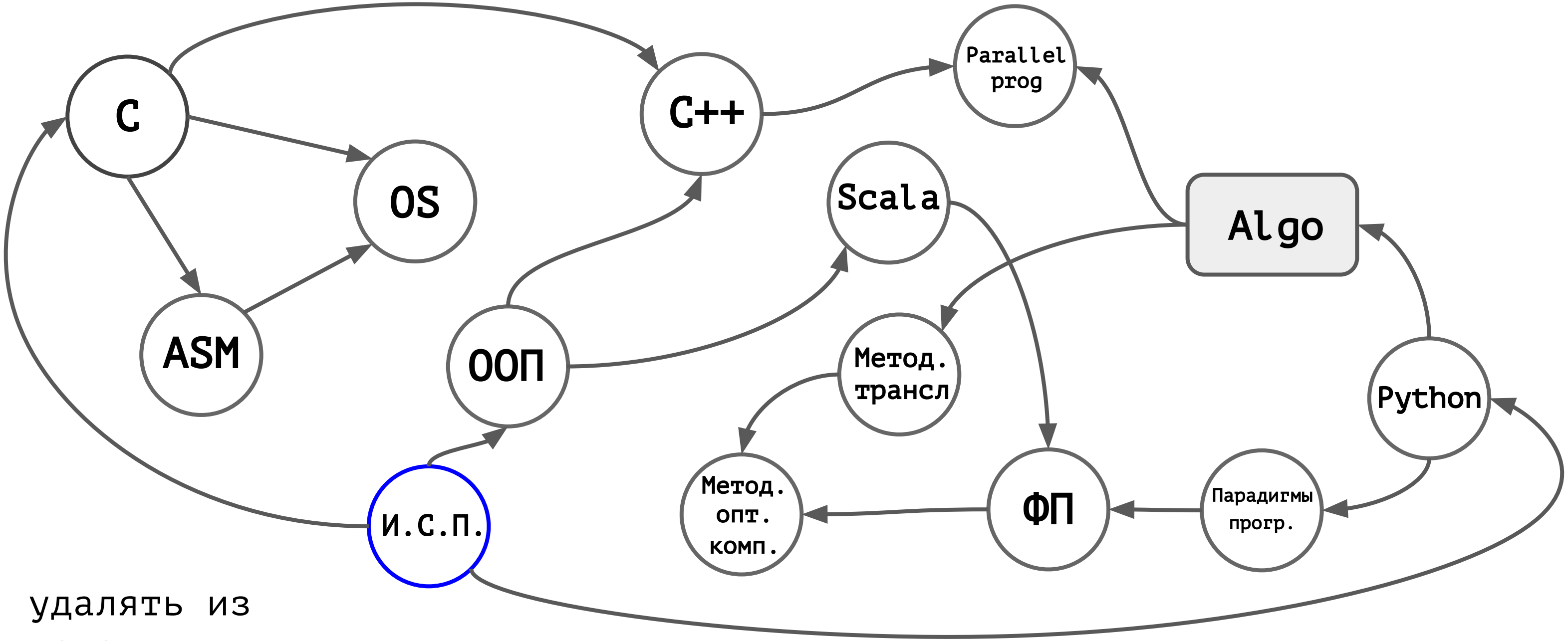
Графы: топологическая сортировка

Исток - вершина, в которую не входят никакие ребра.



Графы: топологическая сортировка

Исток - вершина, в которую не входят никакие ребра.

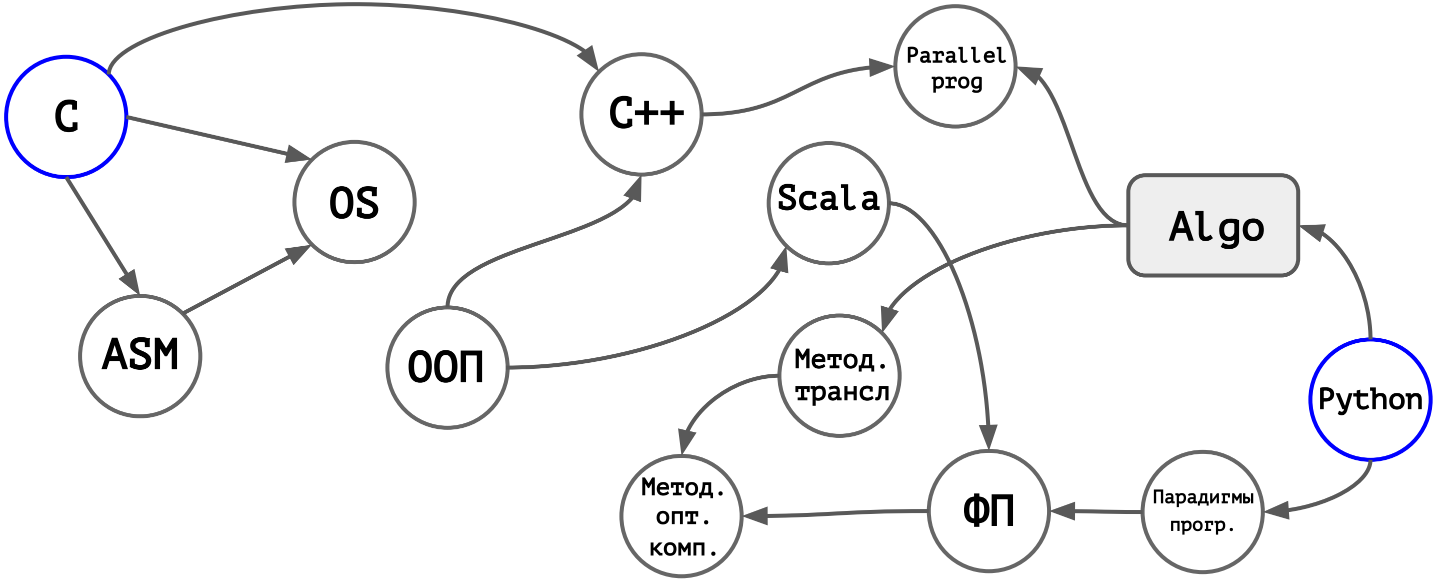


Идея: удалять из графа истоки, добавлять их в ответ

Графы: топологическая сортировка

И.С.П.

Исток - вершина, в которую не входят никакие ребра.



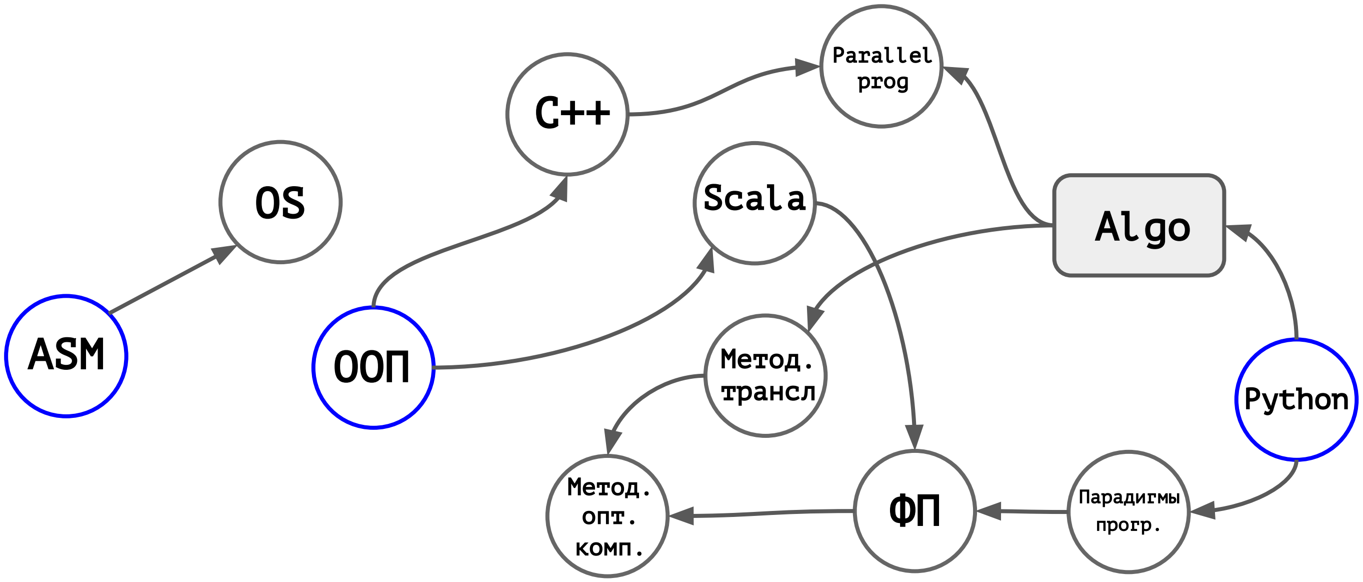
Идея: удалять из графа истоки, добавлять их в ответ

Графы: топологическая сортировка

и.с.п.

С

Исток - вершина, в которую
не входят никакие ребра.



Идея: удалять из
графа истоки,
добавлять их в
ответ

Графы: топологическая сортировка

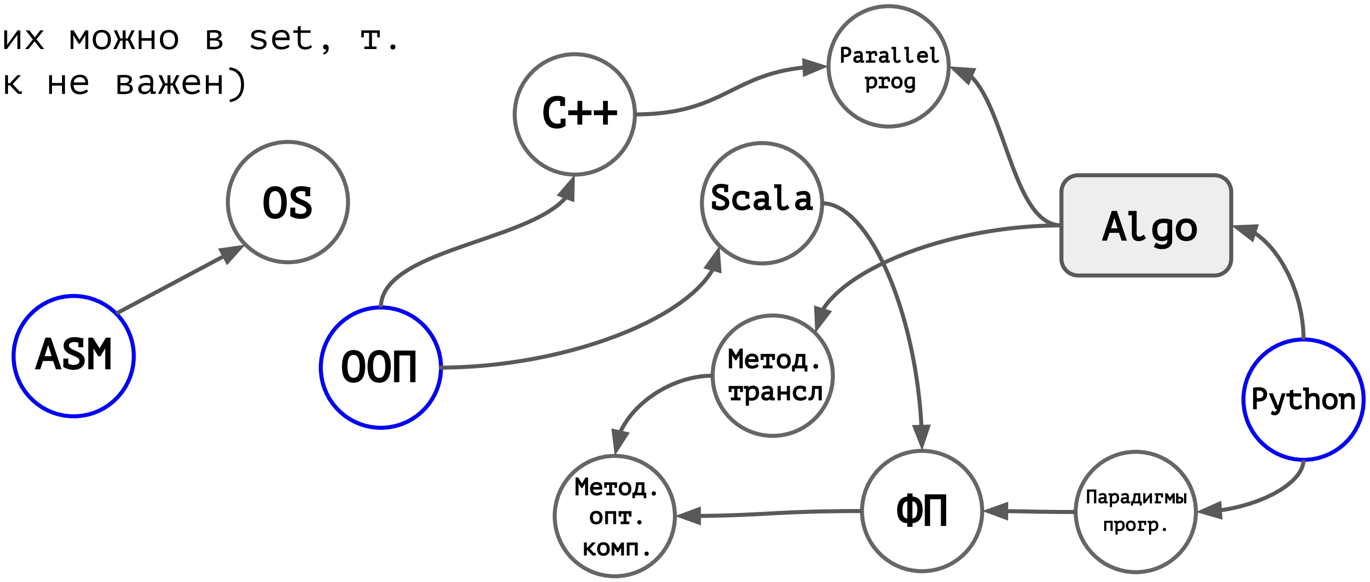
и.с.п.

С

На каждой итерации нужно обновлять множество истоков

(хранить их можно в set, т. к. порядок не важен)

Исток - вершина, в которую не входят никакие ребра.



Идея: удалять из графа истоки, добавлять их в ответ

Графы: топологическая сортировка

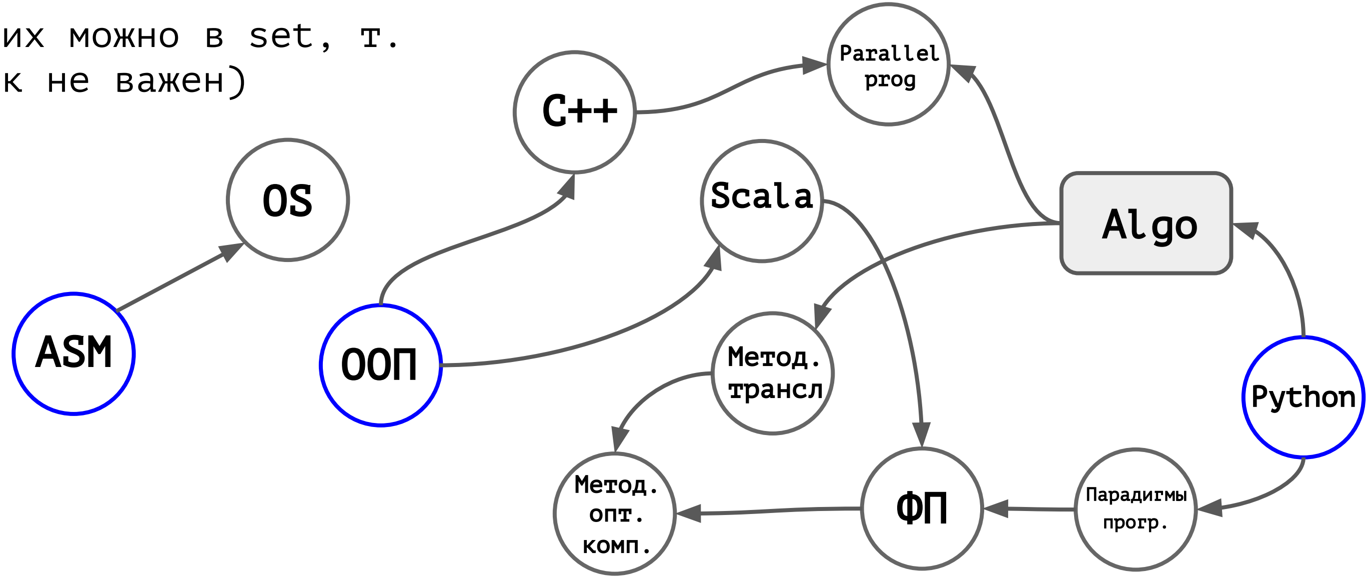
И.С.П.

С

На каждой итерации нужно обновлять множество истоков

(хранить их можно в set, т. к. порядок не важен)

Исток - вершина, в которую не входят никакие ребра.



Идея: удалять из графа истоки, добавлять их в ответ

Алгоритм Кана (1962)

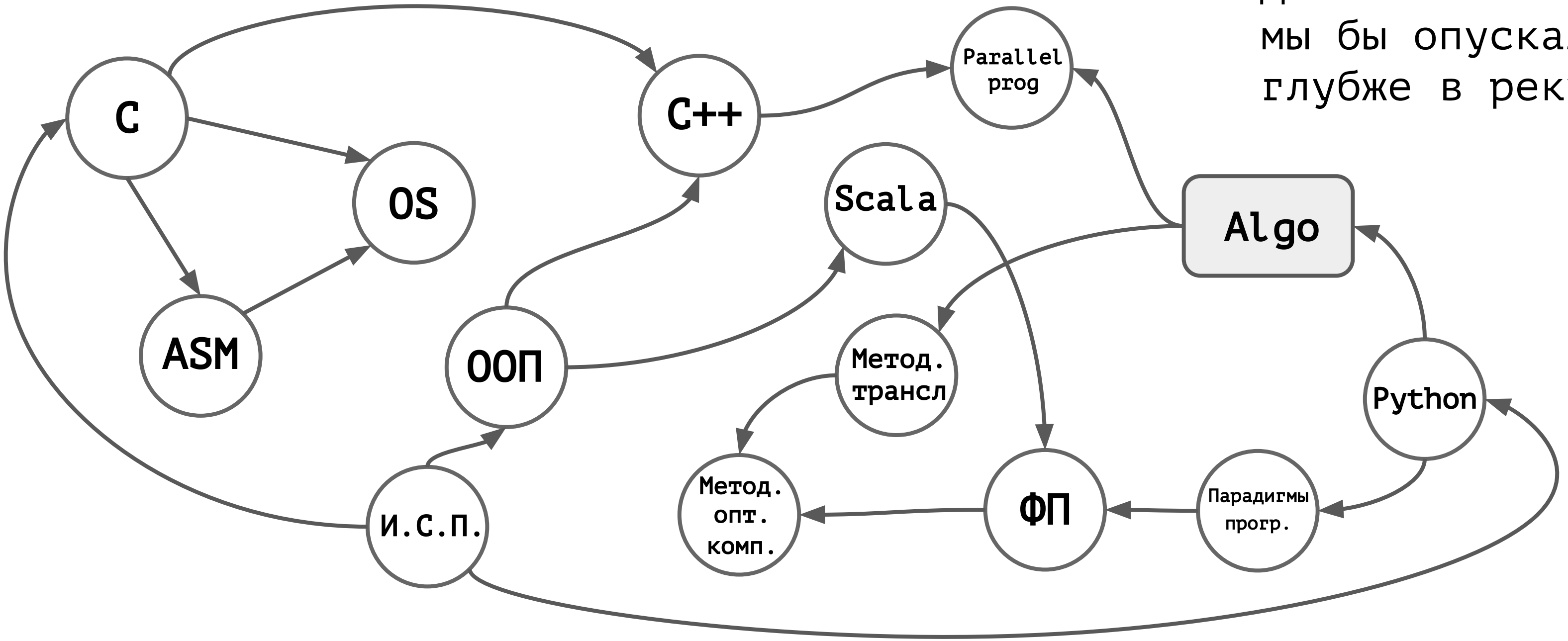
Сложность при правильной реализации O(|V| + |E|)

Но можно и проще!

Графы: топологическая сортировка

Как бы здесь сработал DFS?

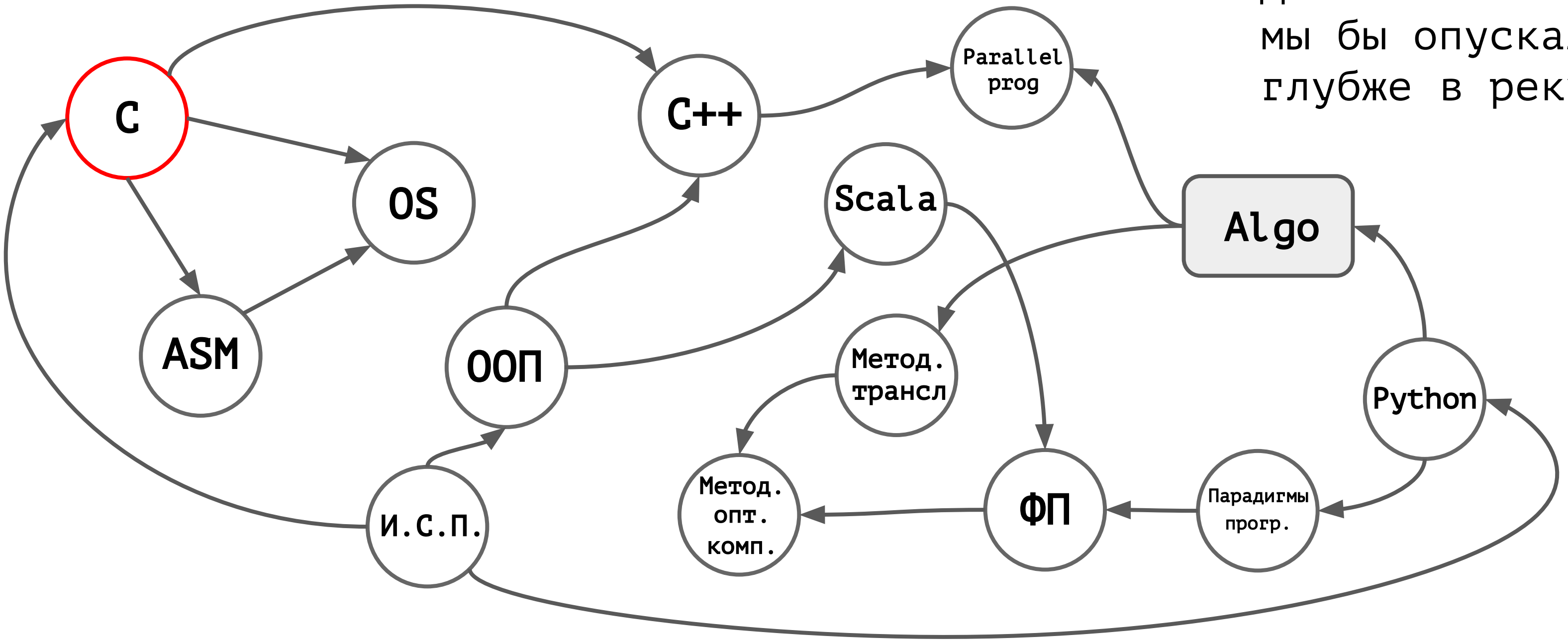
До какого момента мы бы опускались глубже в рекурсию?



Графы: топологическая сортировка

Как бы здесь сработал DFS?

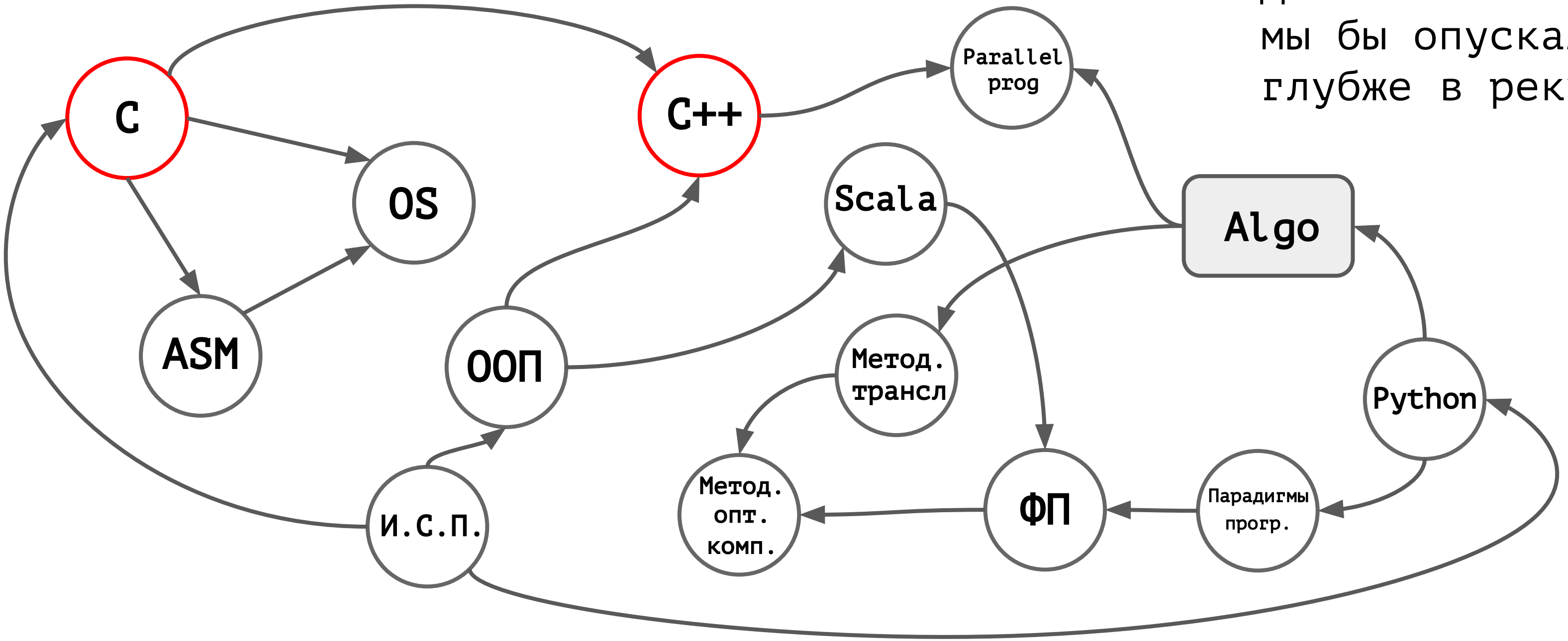
До какого момента мы бы опускались глубже в рекурсию?



Графы: топологическая сортировка

Как бы здесь сработал DFS?

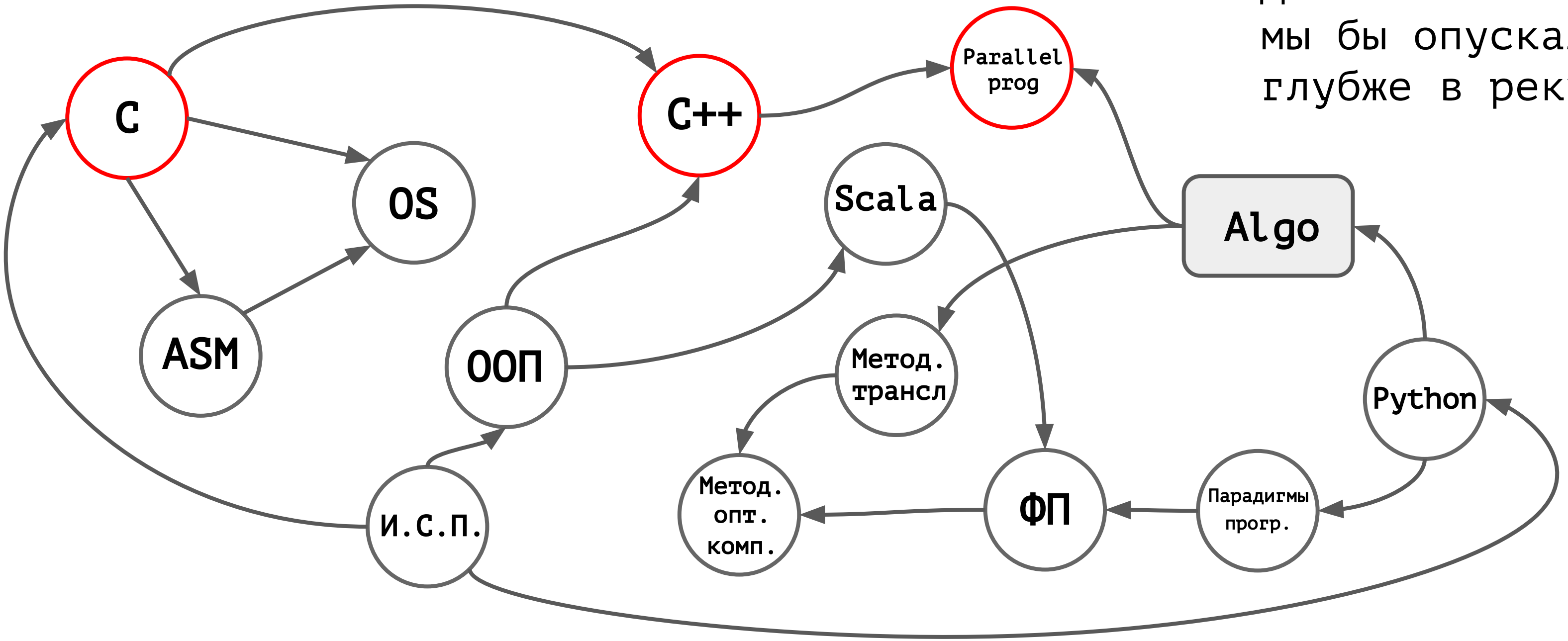
До какого момента мы бы опускались глубже в рекурсию?



Графы: топологическая сортировка

Как бы здесь сработал DFS?

До какого момента мы бы опускались глубже в рекурсию?

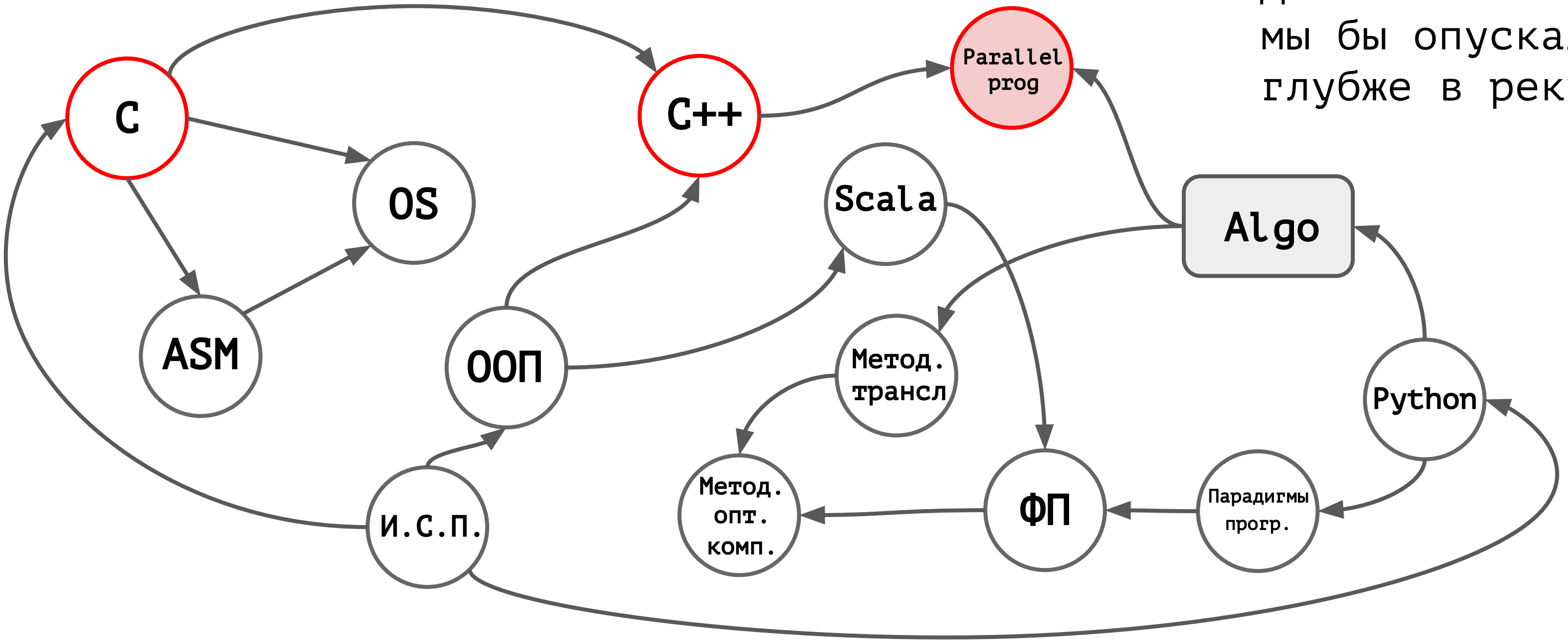


Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов (до стока).

Как бы здесь сработал DFS?

До какого момента мы бы опускались глубже в рекурсию?

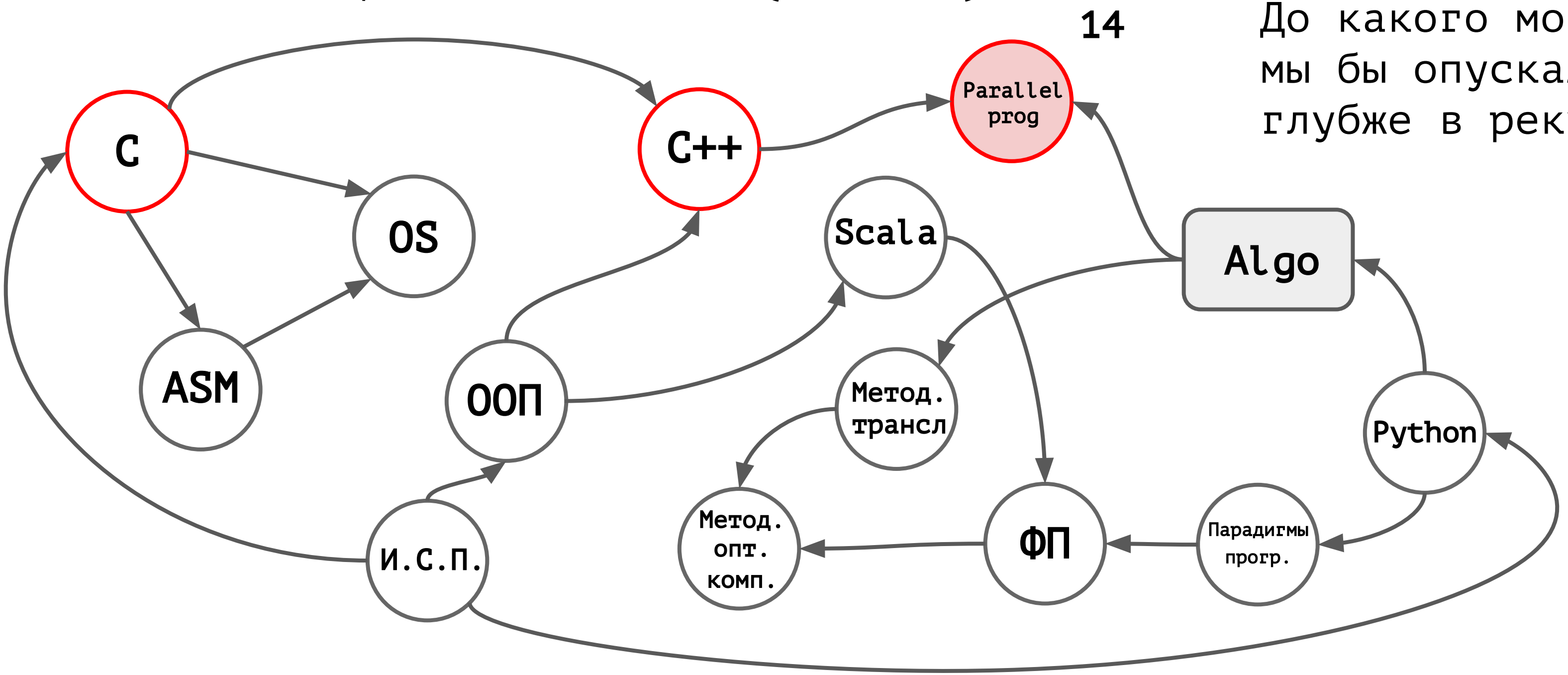


Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов (до стока).

Как бы здесь
сработал DFS?

До какого момента
мы бы опускались
глубже в рекурсию?



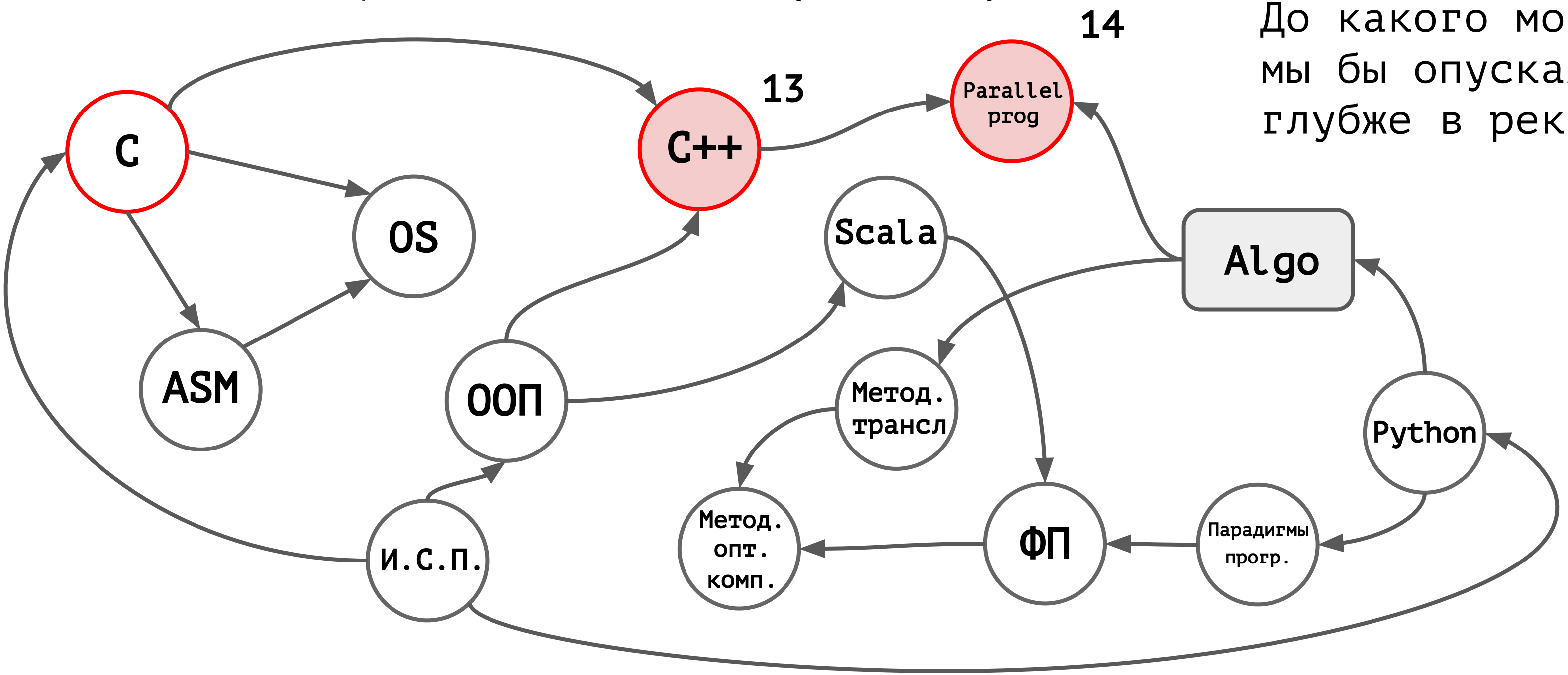
Давайте же в этот момент зафиксируем, что
"Parallel prog" должна быть в конце нашего ответа

Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов* (до стока).

Как бы здесь сработал DFS?

До какого момента мы бы опускались глубже в рекурсию?



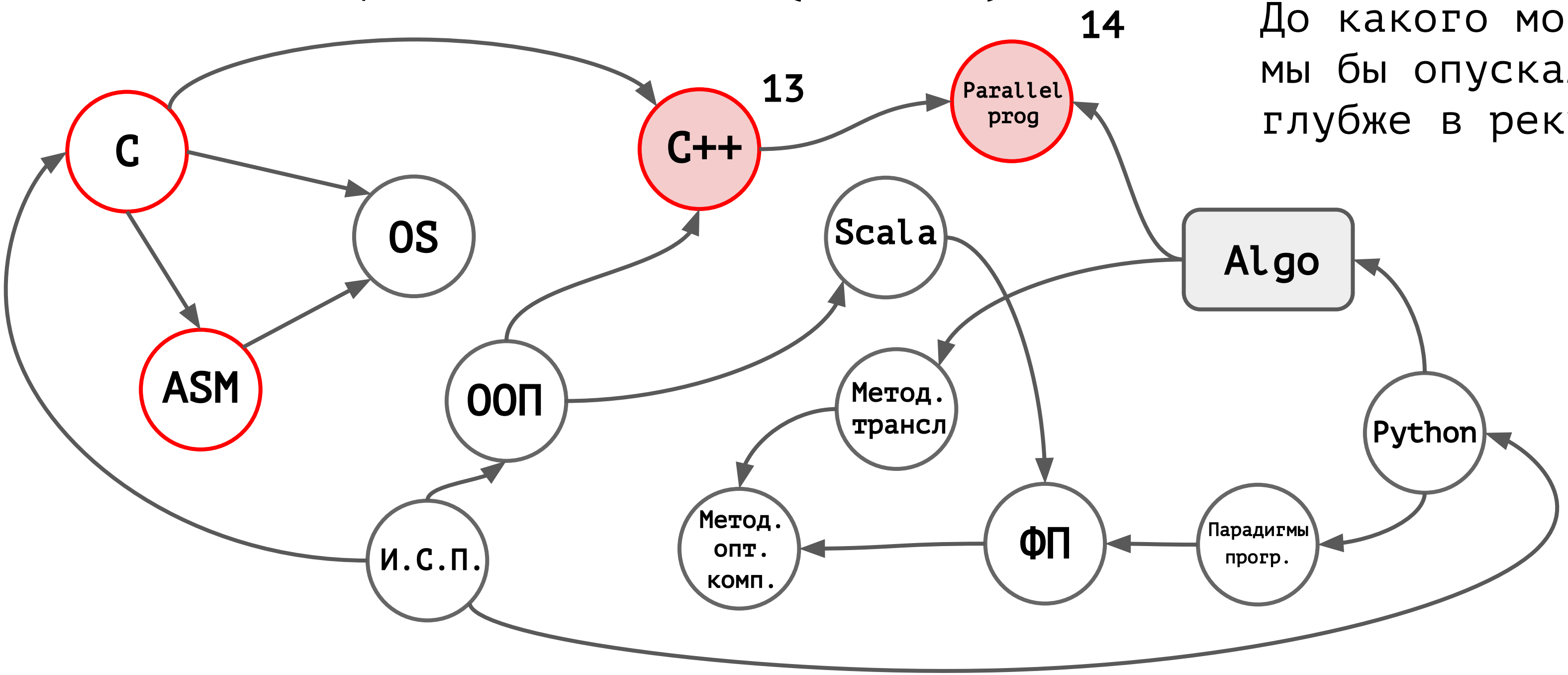
Давайте же в этот момент зафиксируем, что "C++" должна быть в конце нашего ответа

Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов* (до стока).

Как бы здесь сработал DFS?

До какого момента мы бы опускались глубже в рекурсию?



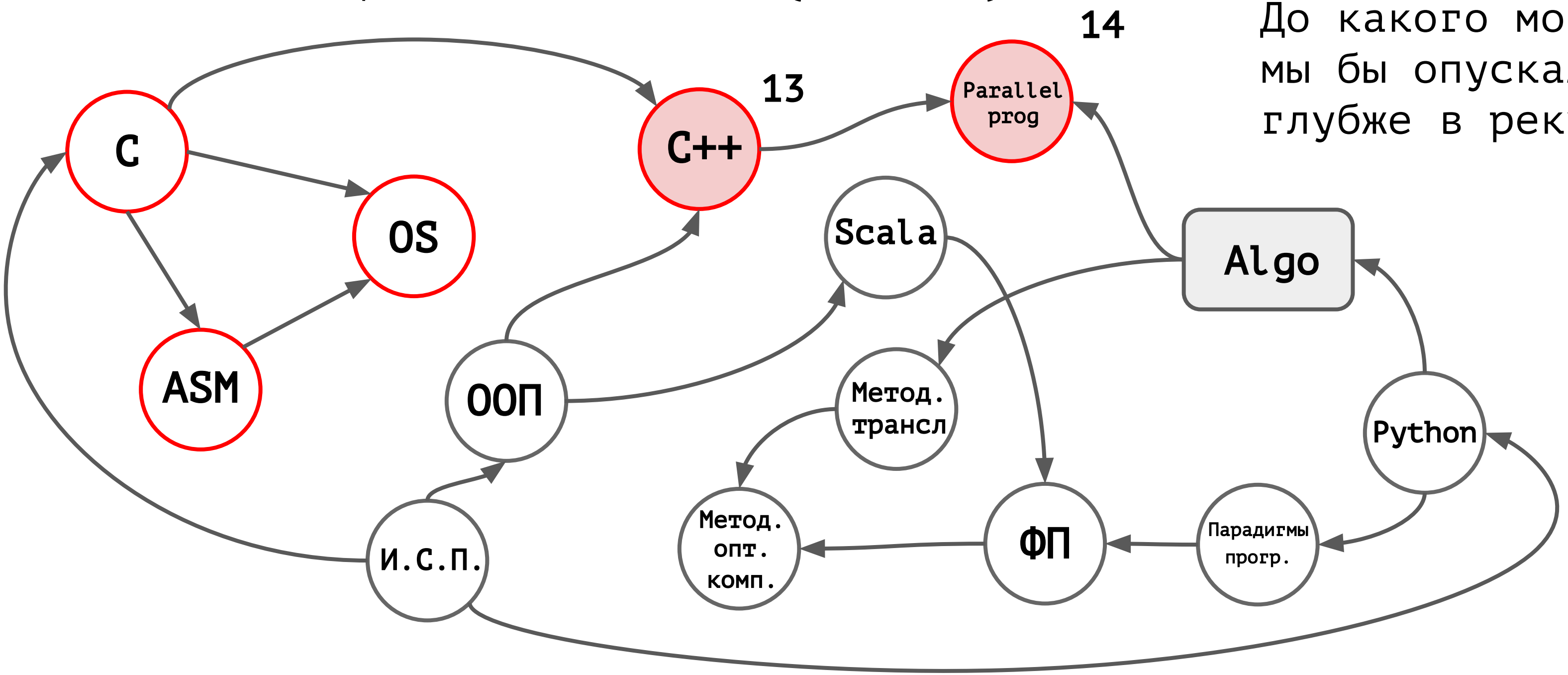
Давайте же в этот момент зафиксируем, что "С++" должна быть в конце нашего ответа

Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов* (до стока).

Как бы здесь сработал DFS?

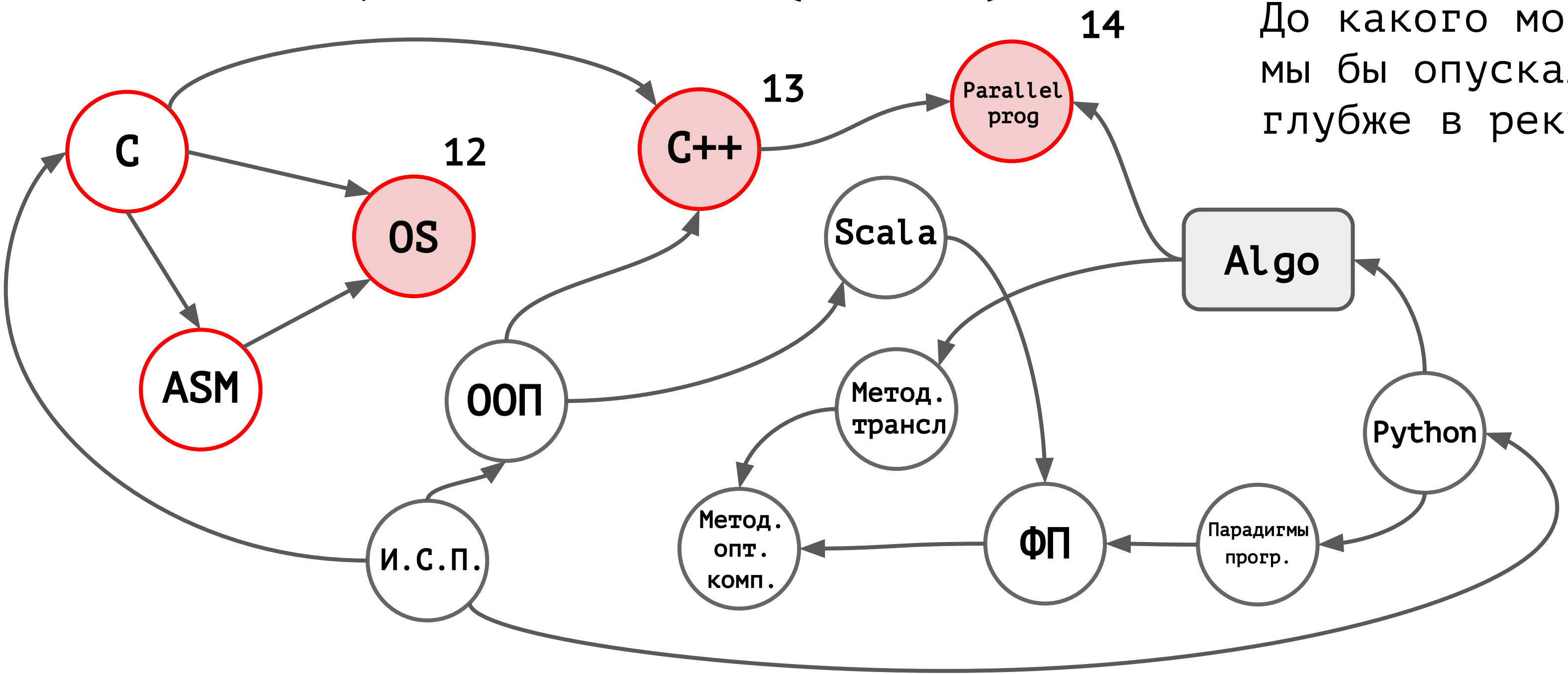
До какого момента мы бы опускались глубже в рекурсию?



Давайте же в этот момент зафиксируем, что "C++" должна быть в конце нашего ответа

Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов* (до стока).



Как бы здесь сработал DFS?

До какого момента мы бы опускались глубже в рекурсию?

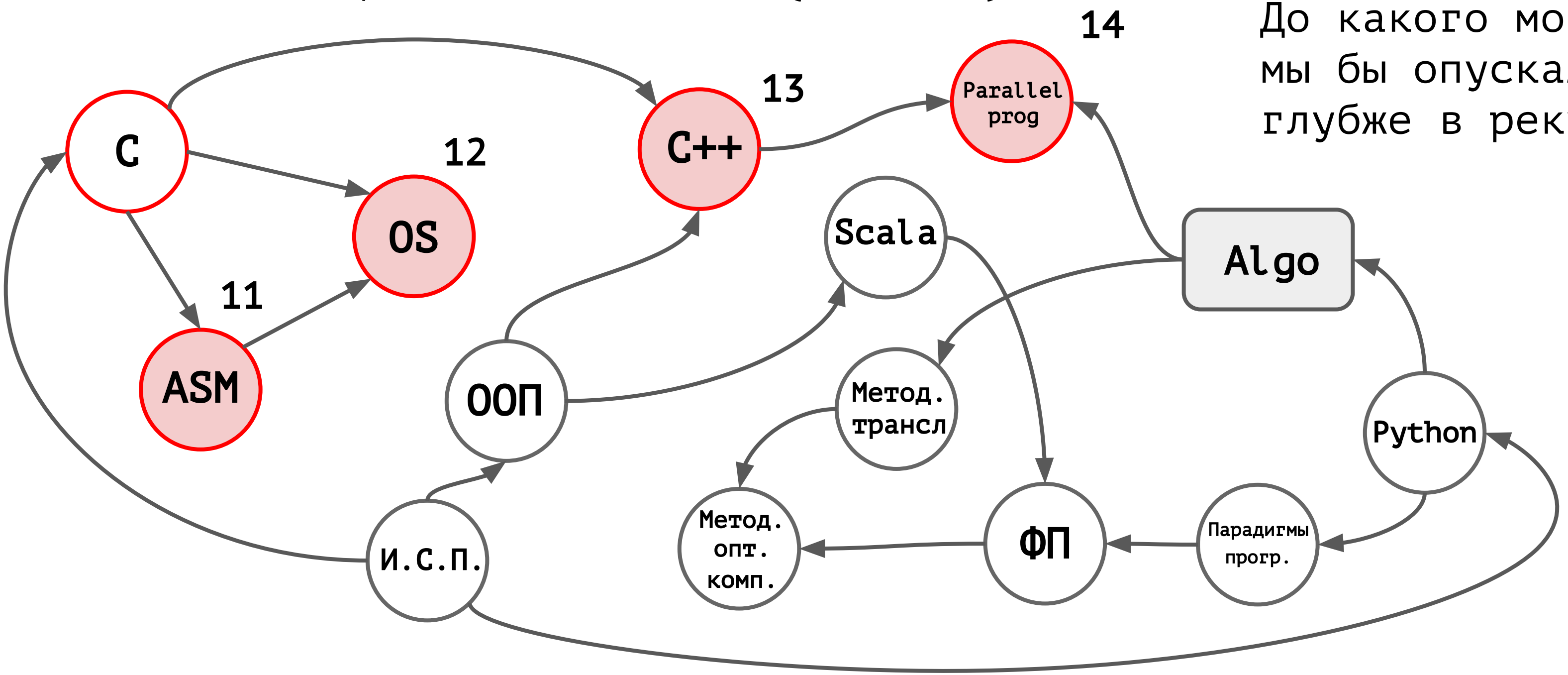
Давайте же в этот момент зафиксируем, что "OS" должна быть в конце нашего ответа

Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов* (до стока).

Как бы здесь сработал DFS?

До какого момента мы бы опускались глубже в рекурсию?



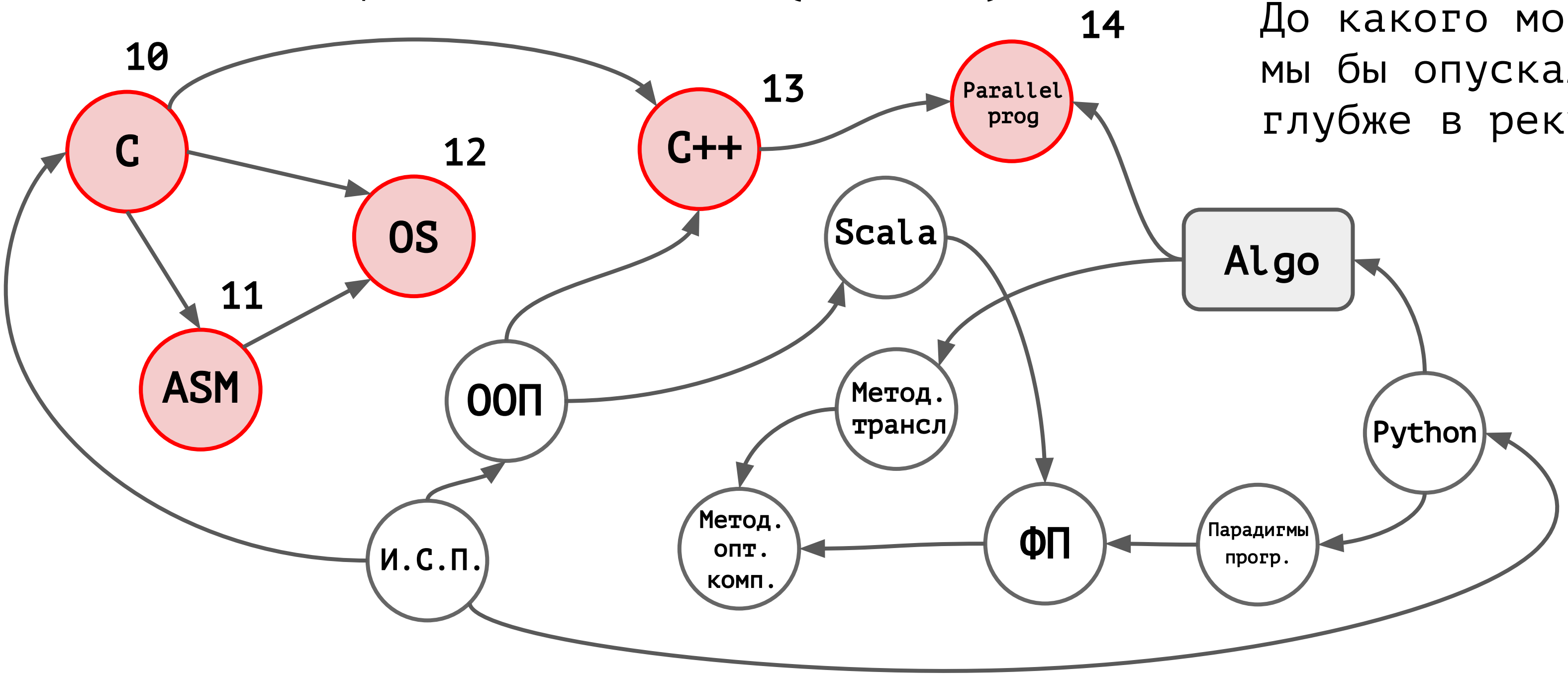
Давайте же в этот момент зафиксируем, что "ASM" должна быть в конце нашего ответа

Графы: топологическая сортировка

Вернулись впервые ровно в тот момент,
когда дошли до вершины без выходов* (до стока).

Как бы здесь сработал DFS?

До какого момента мы бы опускались глубже в рекурсию?



Давайте же в этот момент зафиксируем, что "C" должна быть в конце нашего ответа

Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f : V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Решение:

1. Заводим глобальный счетчик, изначально он равен |V|

Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f: V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Решение:

1. Заводим глобальный счетчик, изначально он равен |V|
2. Запускаем DFS. Каждый раз на выходе из рекурсии нумеруем текущую вершину значением глобального счетчика и уменьшаем его на 1

Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f : V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Решение:

1. Заводим глобальный счетчик, изначально он равен |V|
2. Запускаем DFS. Каждый раз на выходе из рекурсии нумеруем текущую вершину значением глобального счетчика и уменьшаем его на 1
3. Перезапускаем DFS с еще не помеченной вершины

Графы: топологическая сортировка

Задача: пусть задан некий порядок на курсах в университете (одни курсы являются предусловиями других). Найти порядок, в котором корректно проходить данные курсы во время обучения.

Определение: топологическая сортировка в ор. графе - означивание вершин $f: V \to int$, такая что: $(u, v) \in E \Rightarrow f(u) < f(v)$

Решение: Тарьян, 1976, Сложность: O(|V| + |E|), как у обычного DFS.

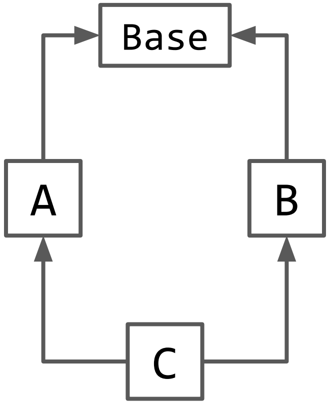
1. Заводим глобальный счетчик, изначально он равен |V|
2. Запускаем DFS. Каждый раз на выходе из рекурсии нумеруем текущую вершину значением глобального счетчика и уменьшаем его на 1
3. Перезапускаем DFS с еще не помеченной вершины

Графы: топологическая сортировка

Кроме упорядочивания задач, топологическая сортировка активно используется в компиляторах и реализациях языков программирования:

1. Для генерации инструкций в корректном порядке

2. Для реализации (множественного) наследования



Мини-задача #30 (2 балла)

Задан набор ограничений на элементы: каждый элемент может или нет находится в некоторой группе, и должен быть расположен перед некоторыми другими элементами.

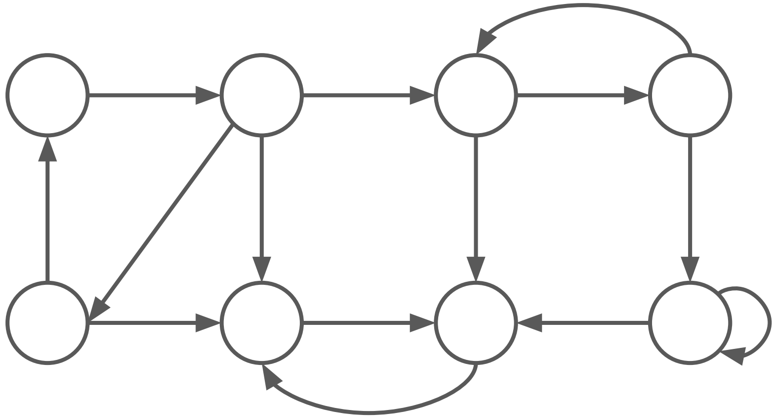
Упорядочить все элементы так, чтобы все ограничения на условия предшествования были выполнены, и при этом элементы в группах стояли рядом друг с другом.

Если это невозможно - вернуть пустой список.

https://leetcode.com/problems/sort-items-by-groups-respecting-dependencies

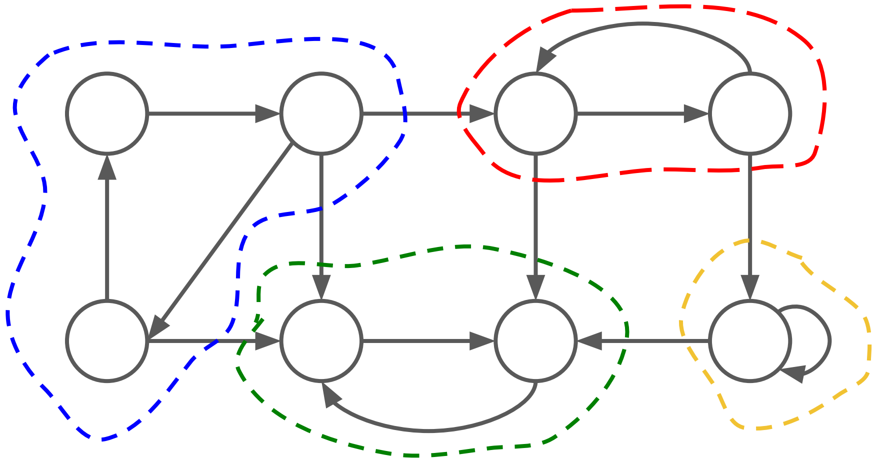
Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.



Графы: поиск компонент сильной связности

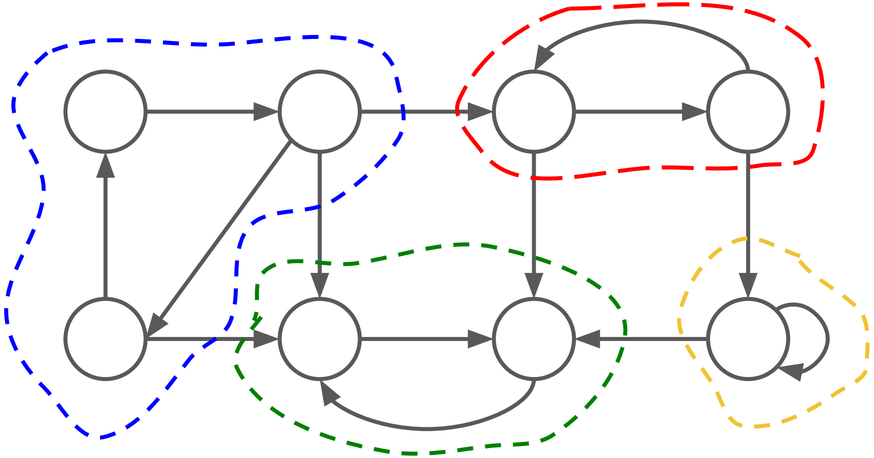
Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.



Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

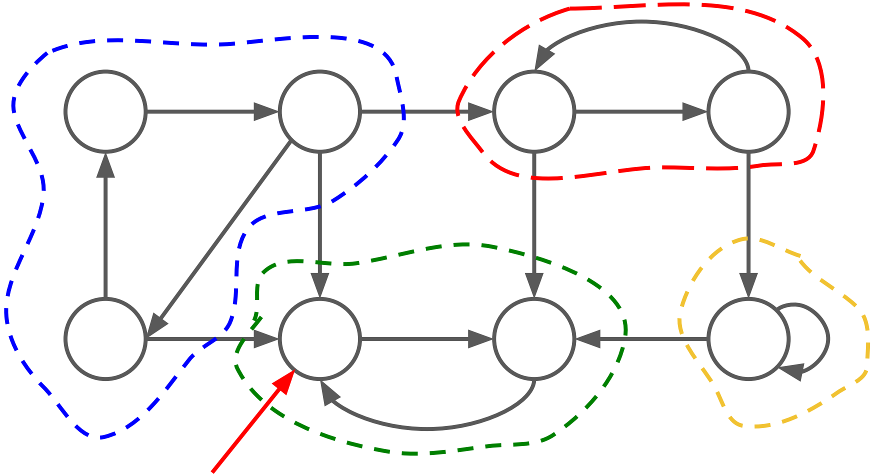
Идея: давайте запускать DFS и смотреть, что получается.



Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

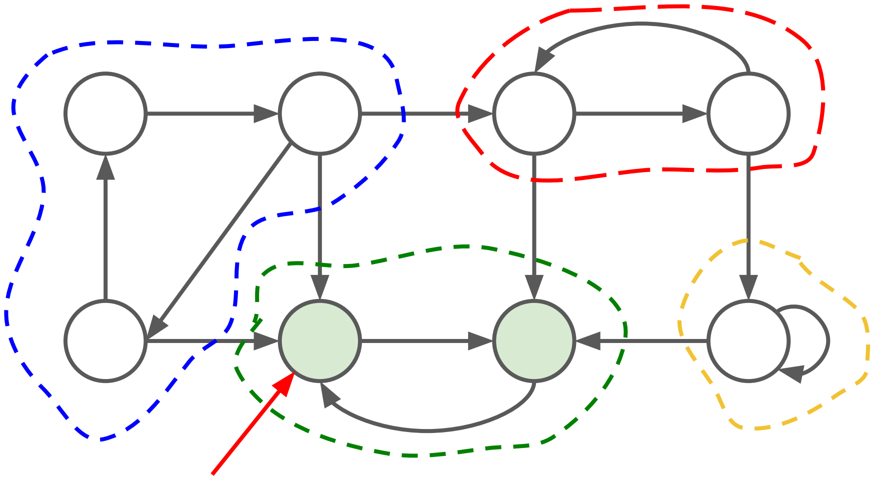


Если запускать отсюда, то получим одну SCC!

Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

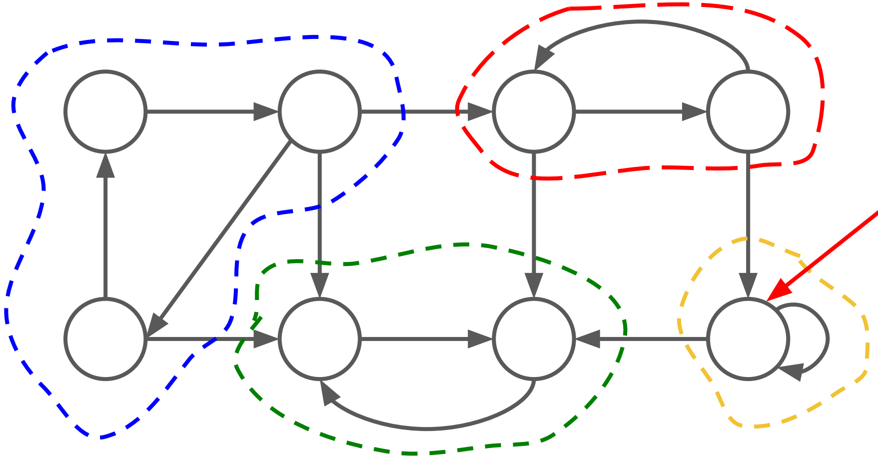


Если запускать отсюда, то получим одну SCC!

Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

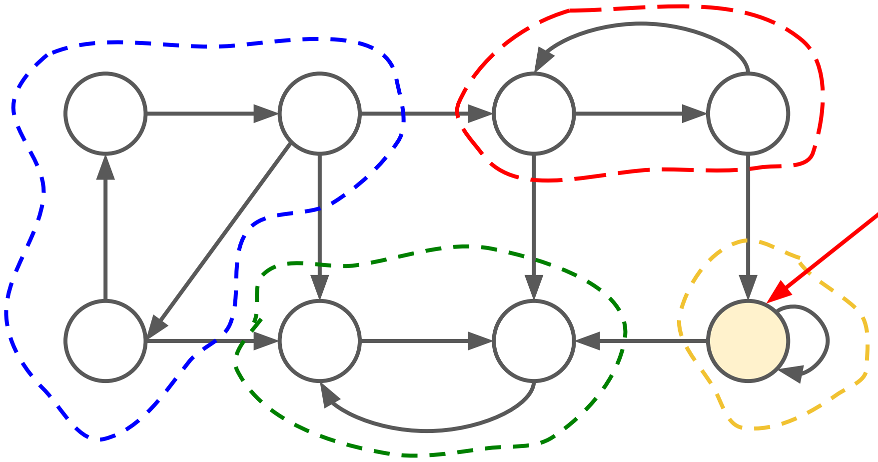


А если отсюда, то разметим 2 SCC

Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

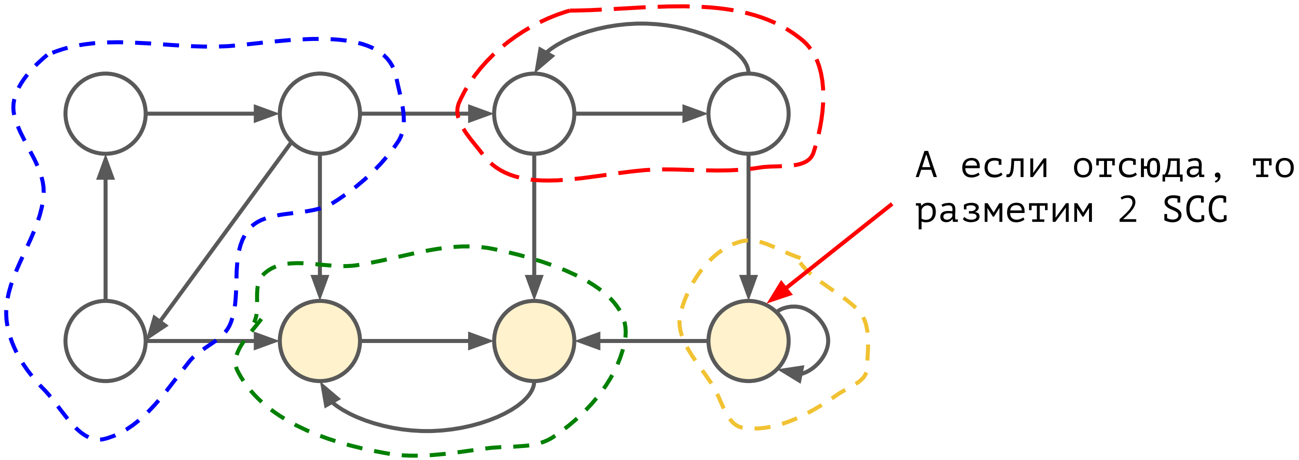


А если отсюда, то разметим 2 SCC

Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

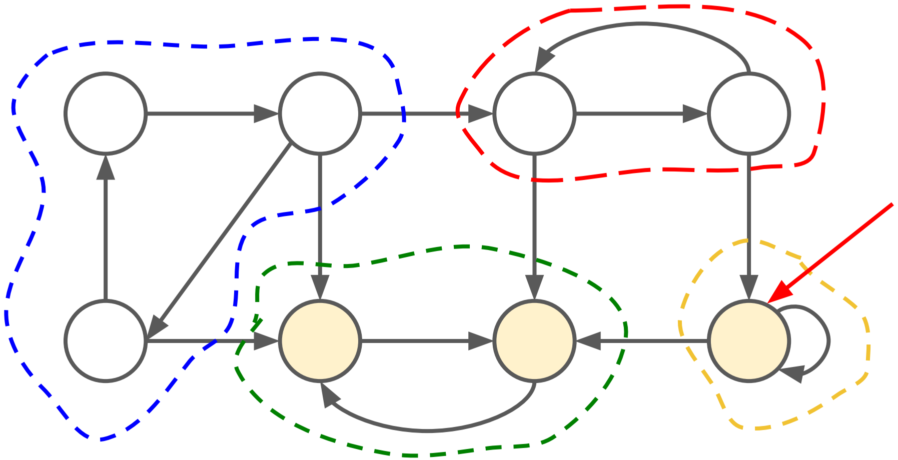
Идея: давайте запускать DFS и смотреть, что получается.



Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.



А если отсюда, то разметим 2 SCC

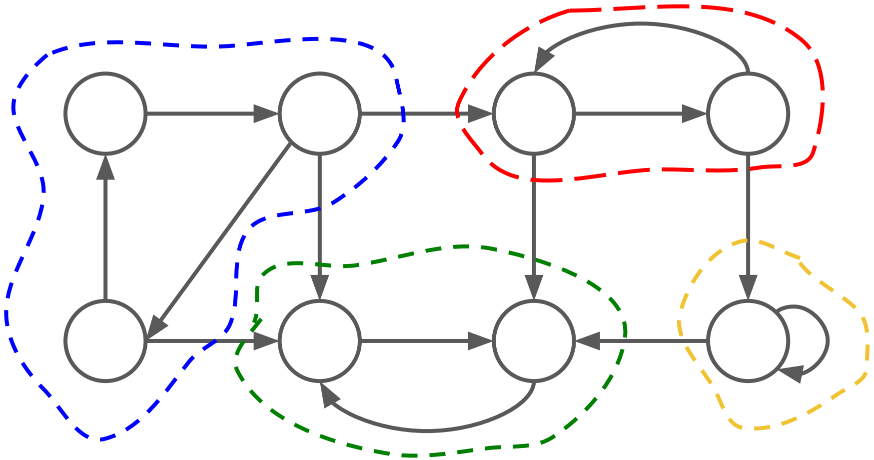
Но по крайней мере мы точно разметим и изначальную SCC

Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

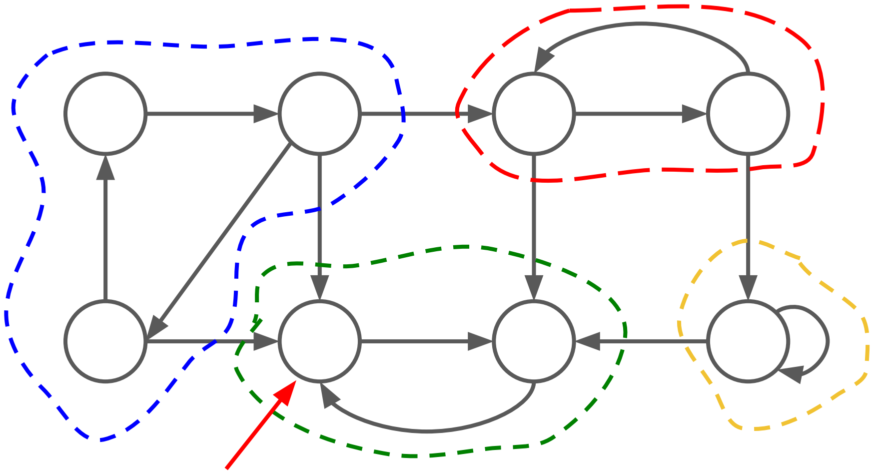


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

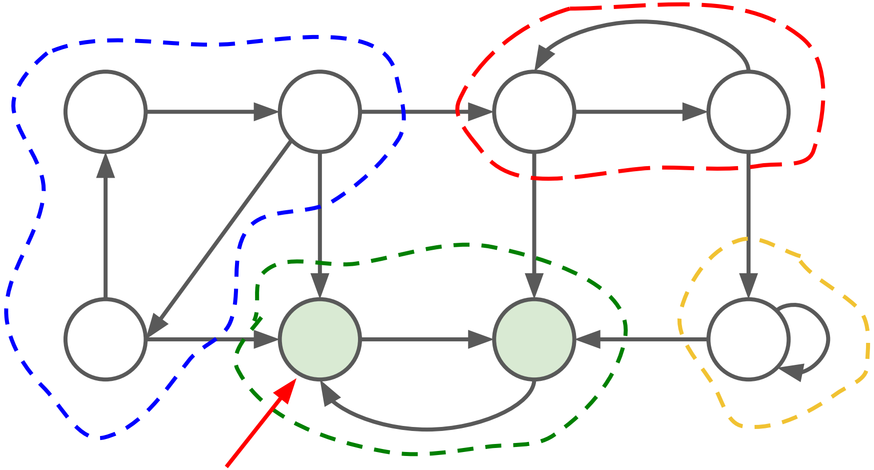


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

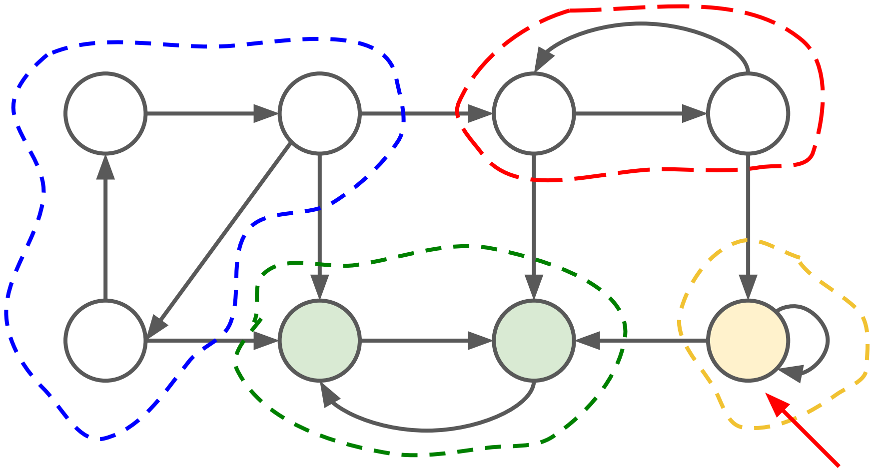


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

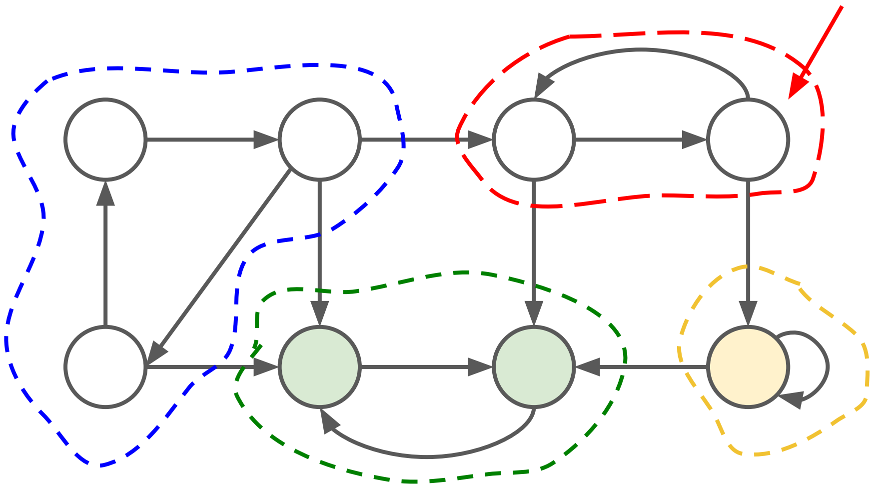


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

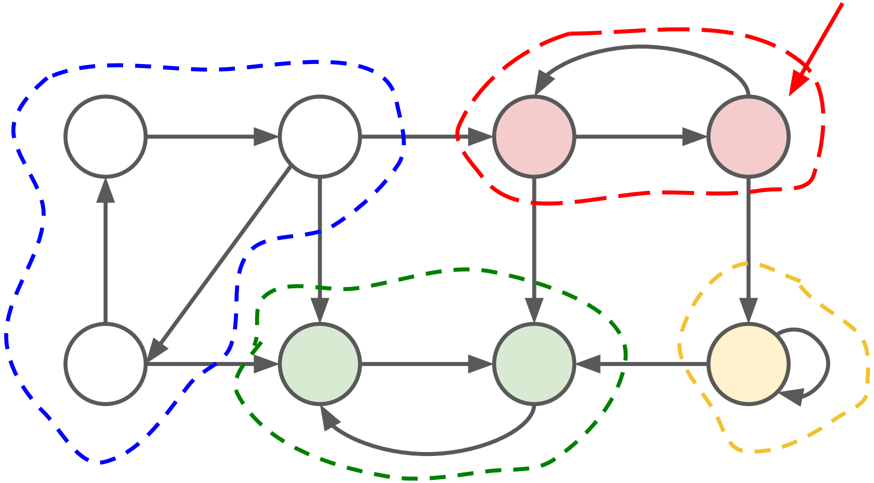


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

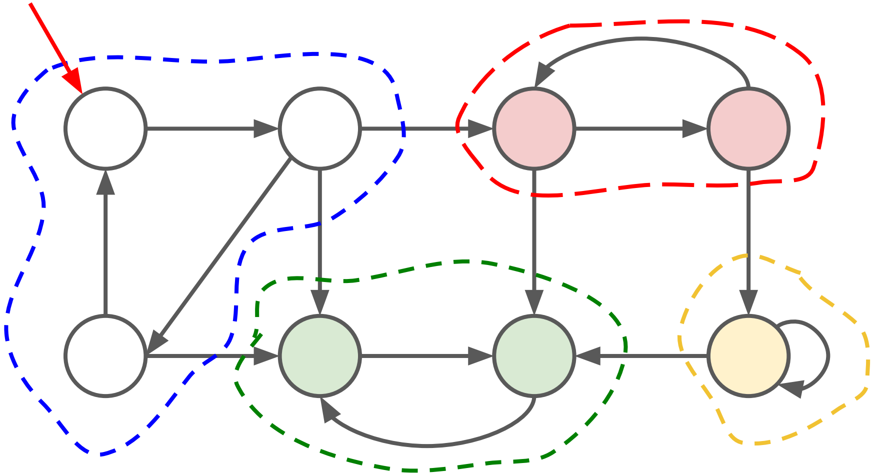


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

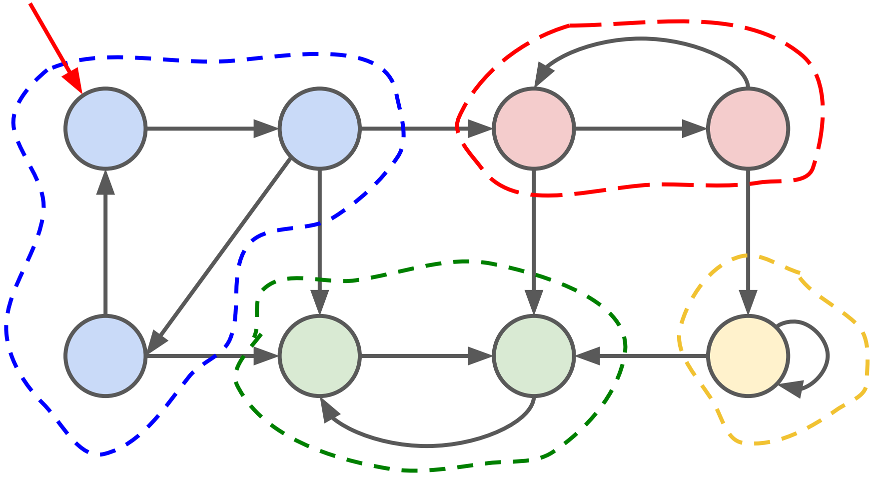


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)

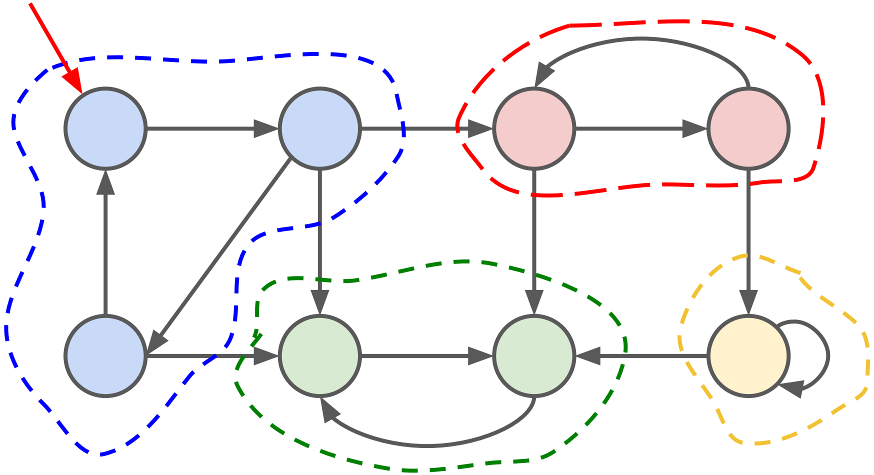


Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)



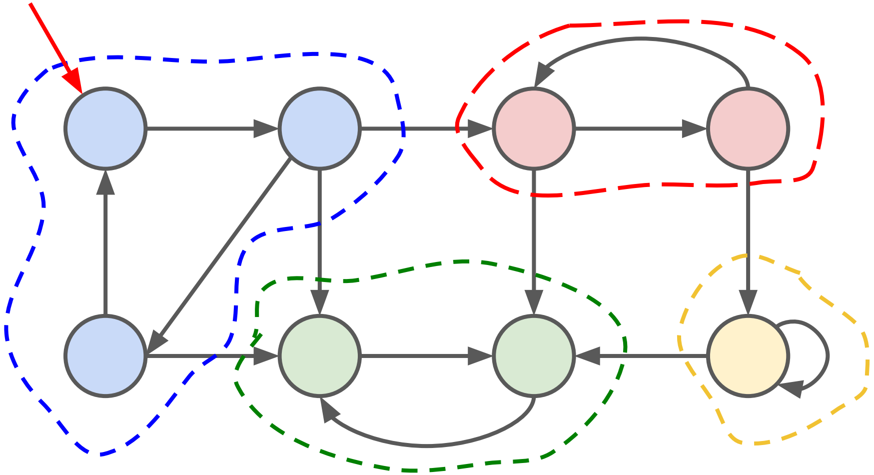
Каждый запуск DFS-а давал нам новую SCC!

Графы: поиск компонент сильной связности

Задача: пусть дан ориентированный граф G. Найти все компоненты сильной связности этого графа.

Идея: давайте запускать DFS и смотреть, что получается.

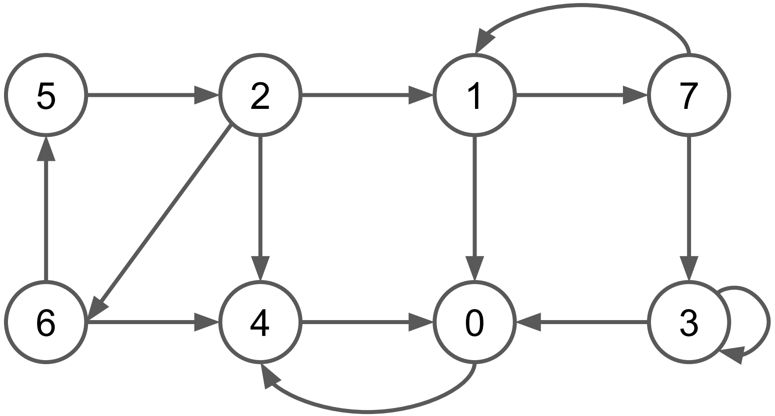
А что если, запускать DFS в правильном порядке (и останавливаться, если что-то уже размечено)



Каждый запуск DFS-а давал нам новую SCC!

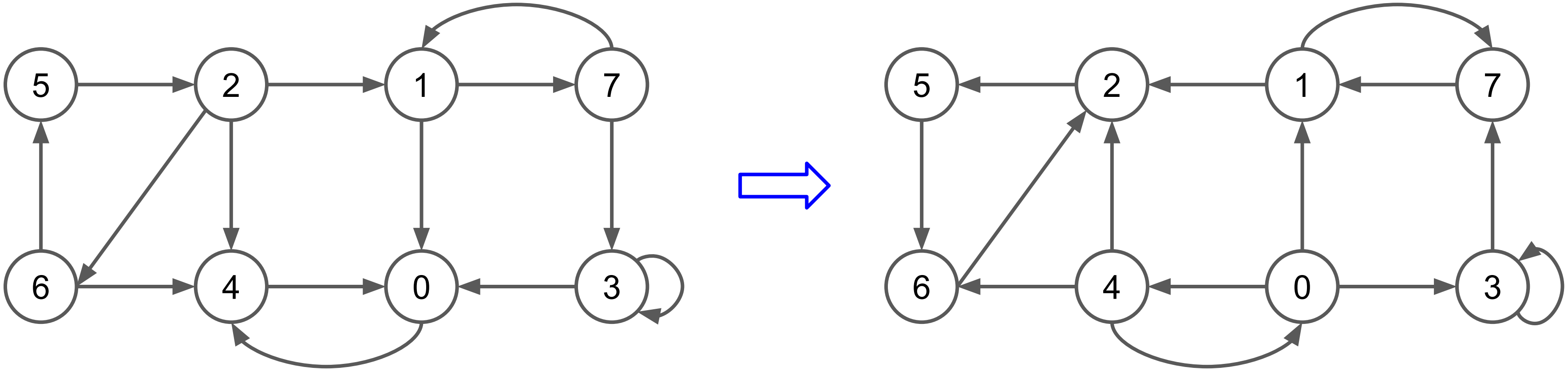
Но при этом нужно знать какой-то магический порядок запуска

Графы: алгоритм Косараджу



Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

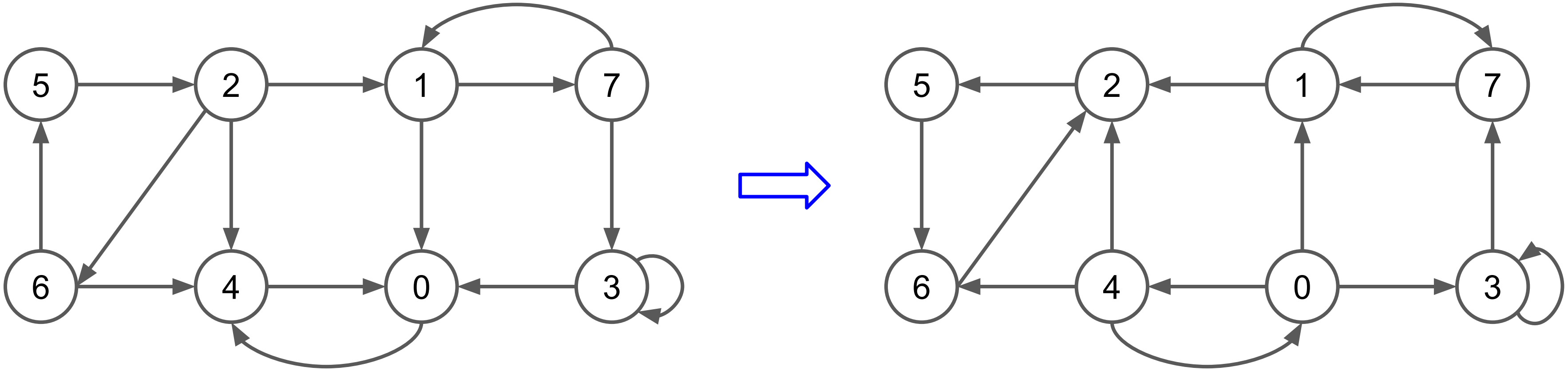


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

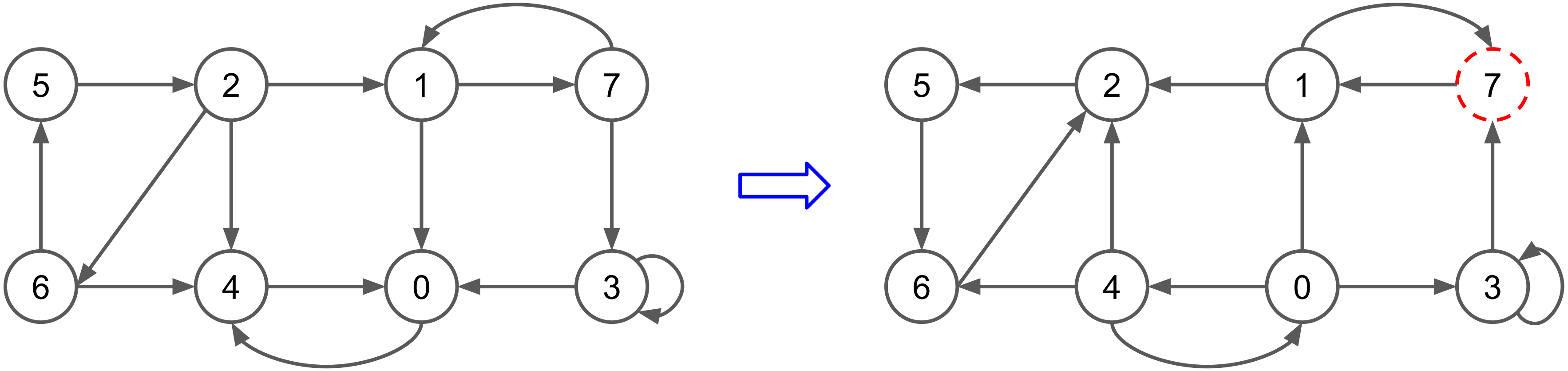


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

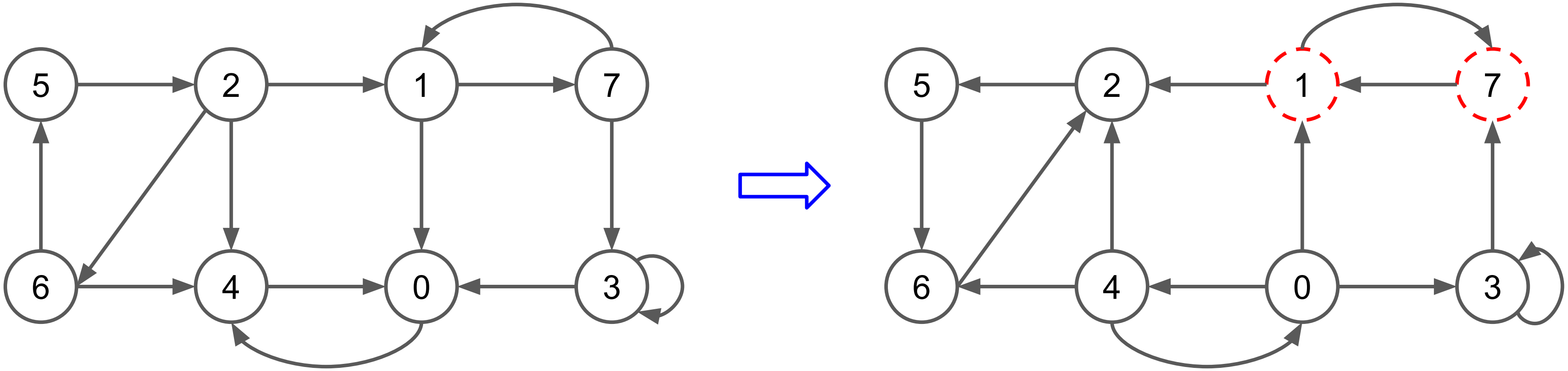


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

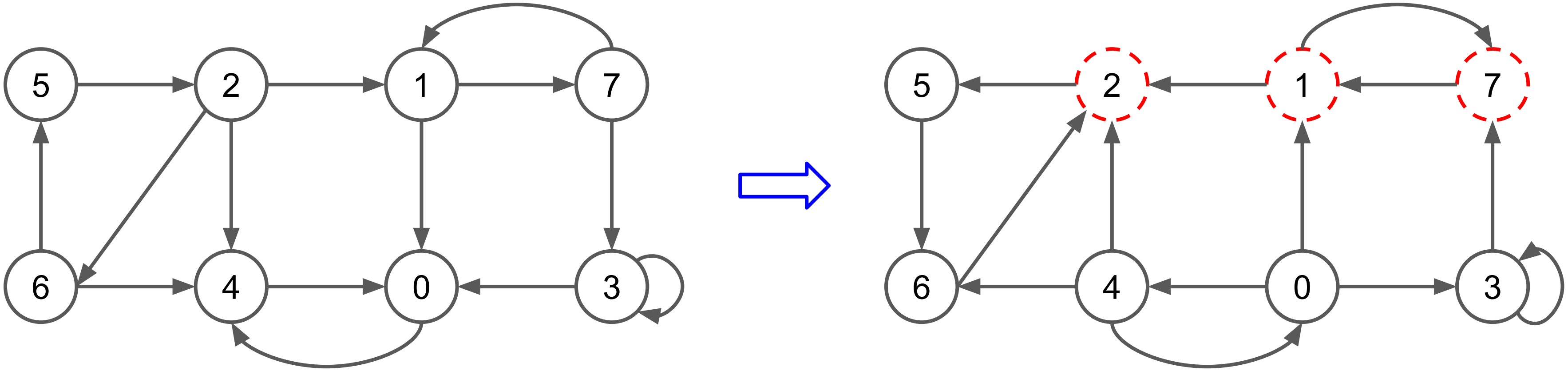


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

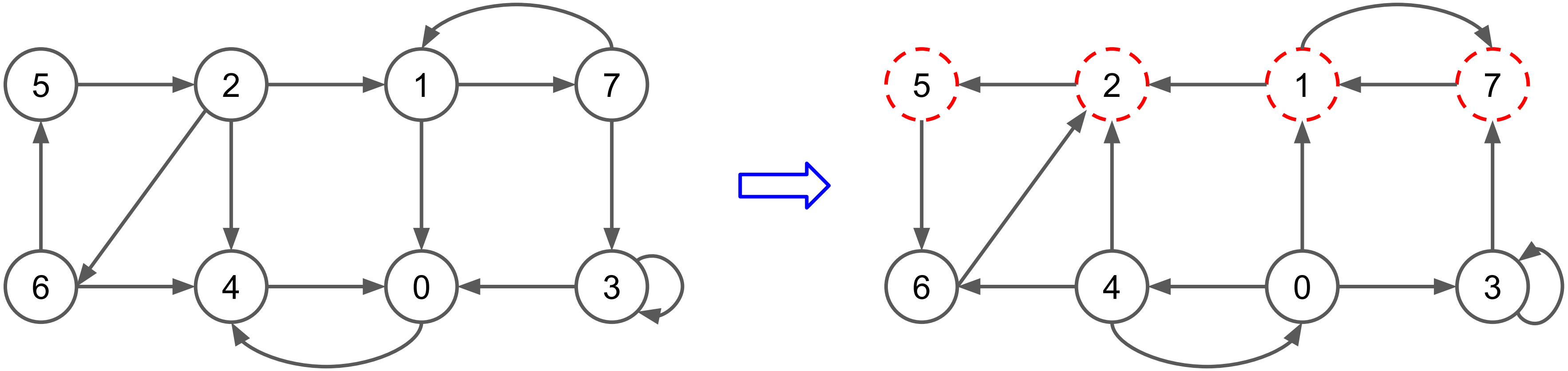


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

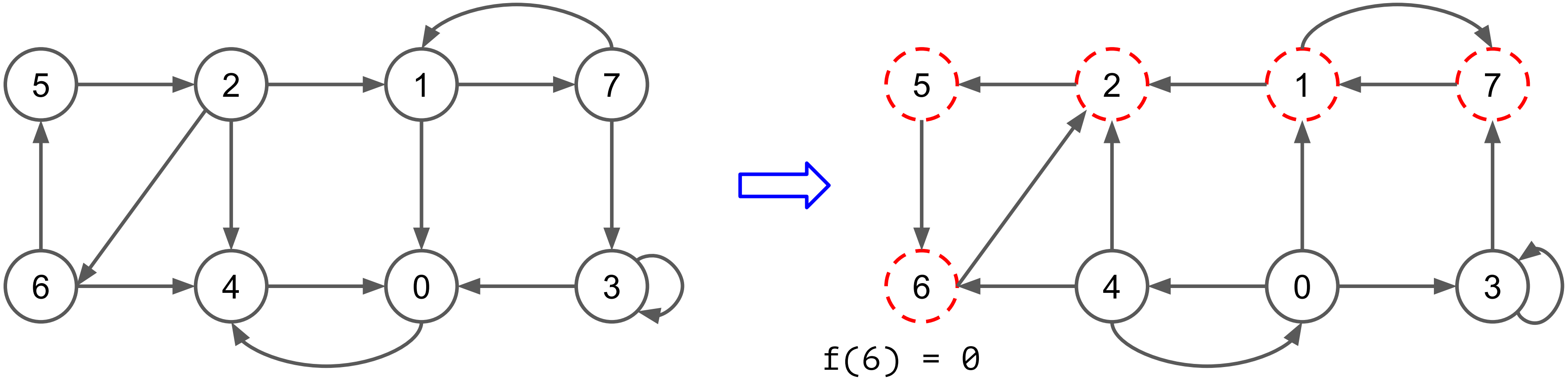


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

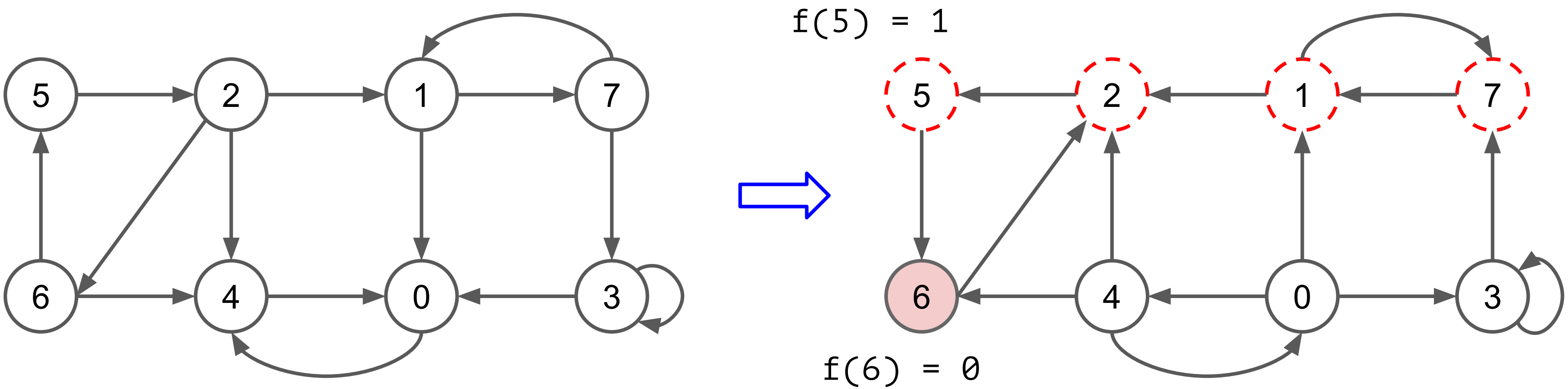


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

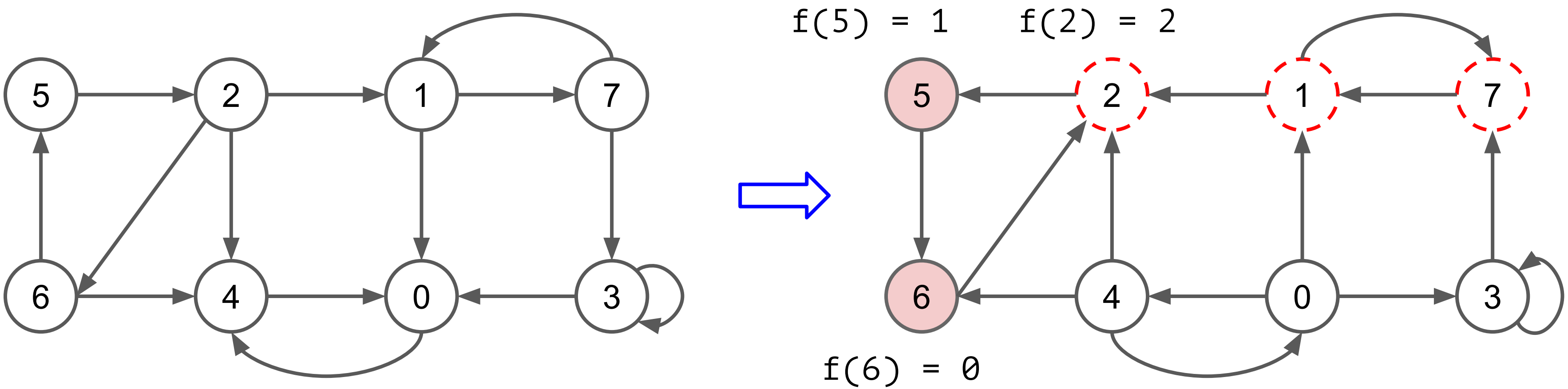


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

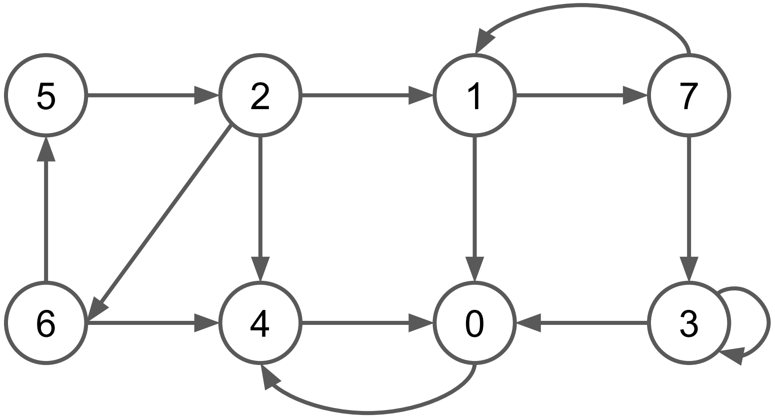


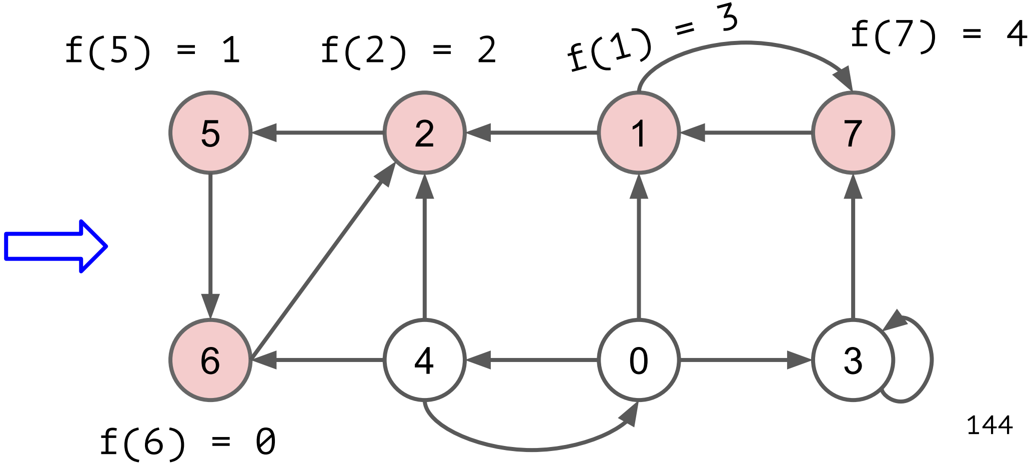
Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.



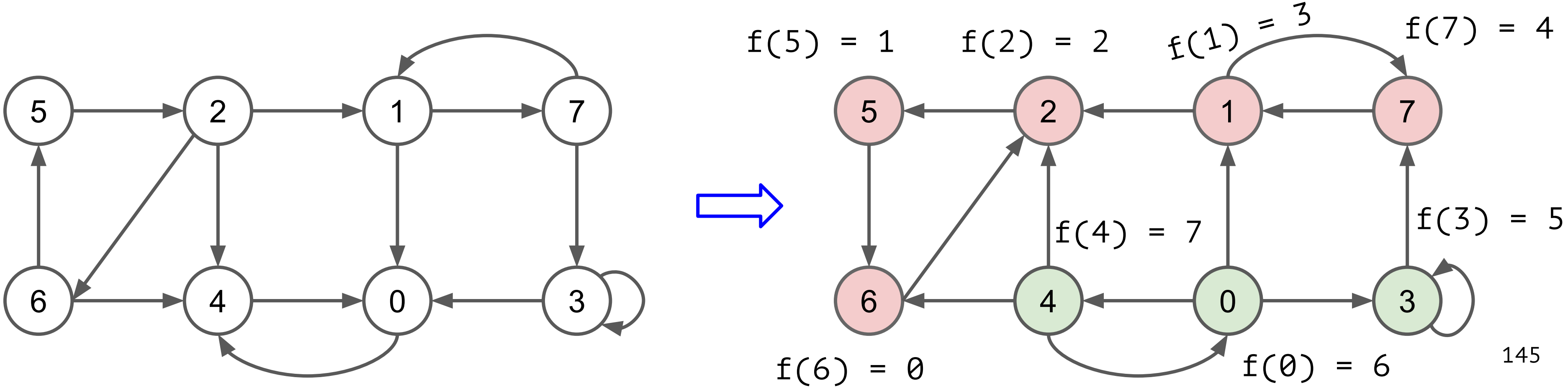


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

   Для определенности начнем с наибольшей вершины.

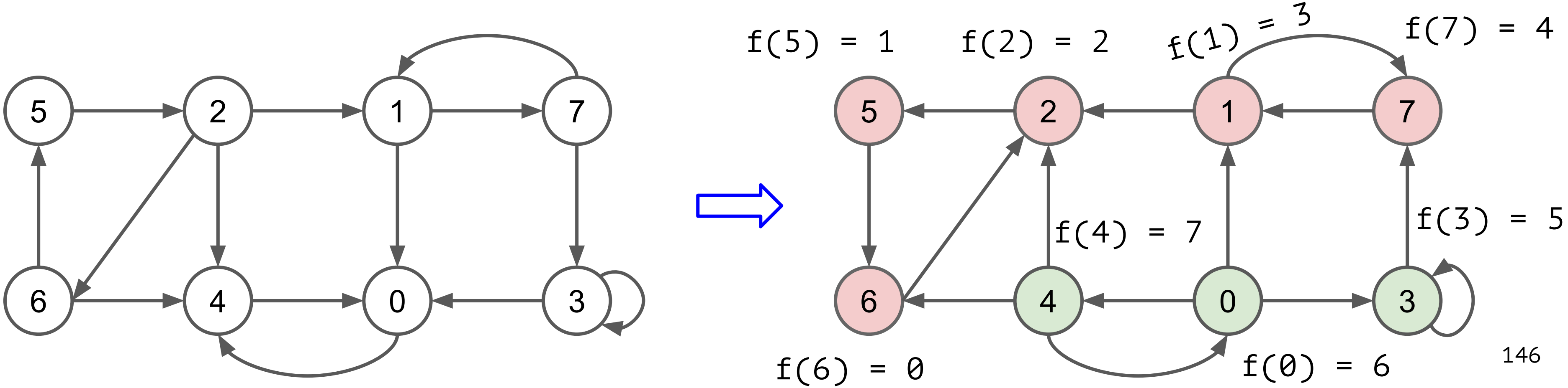


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f

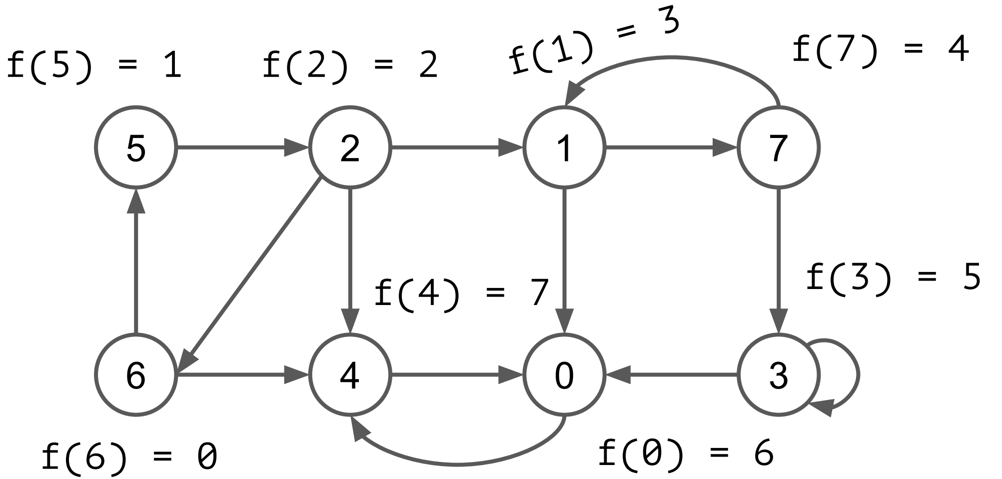


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f

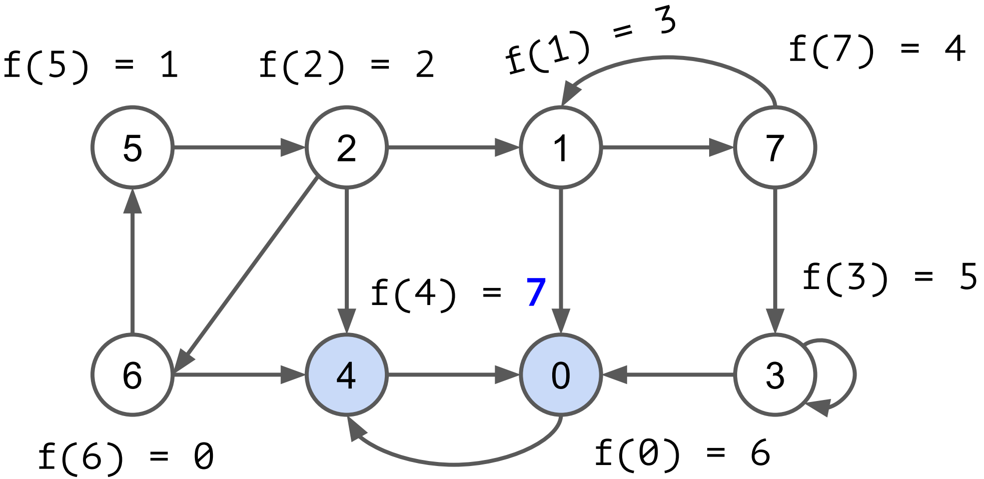


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f

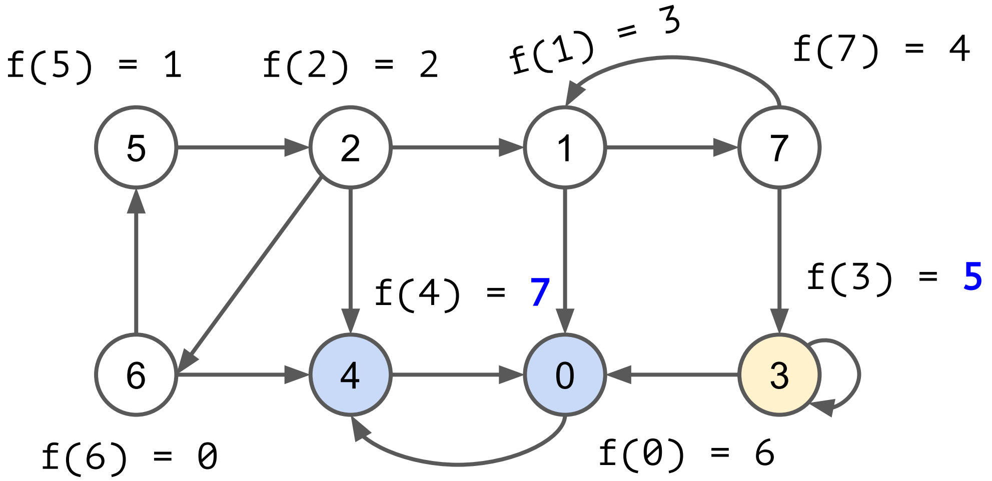


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f

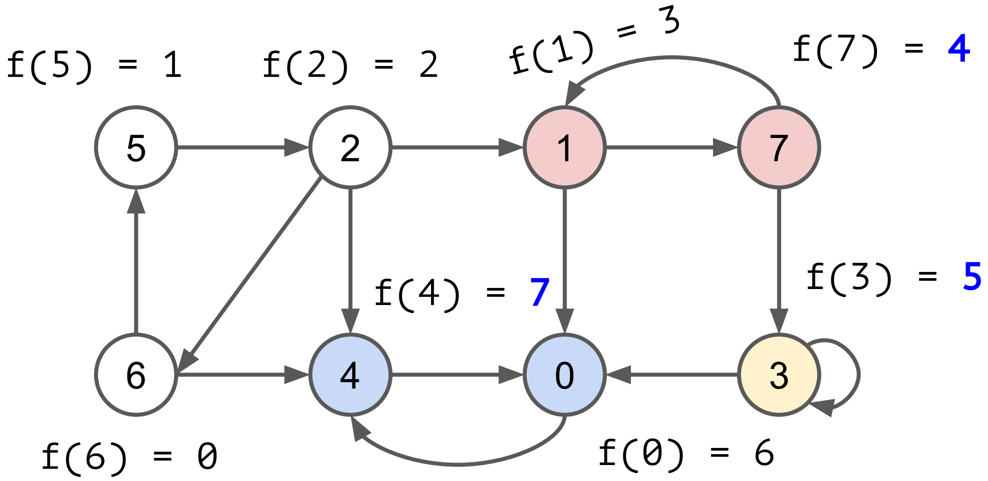


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f

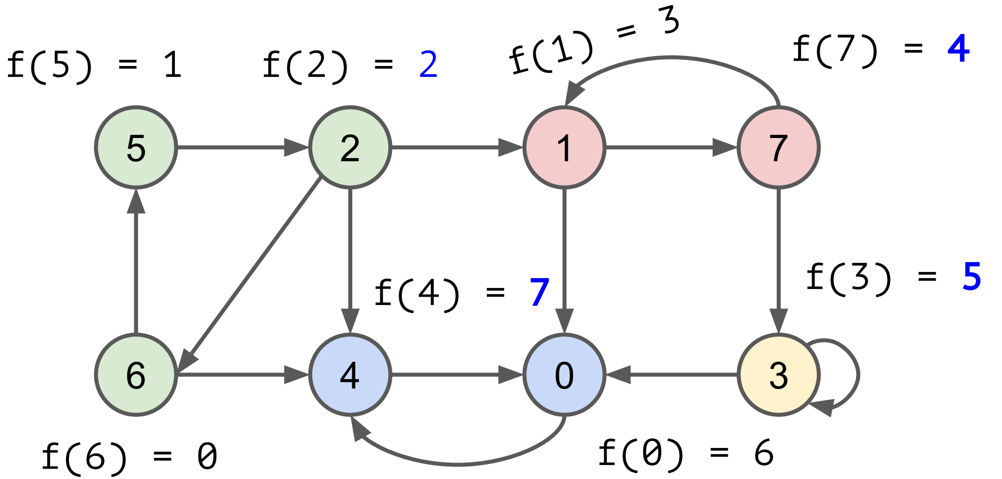


Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f



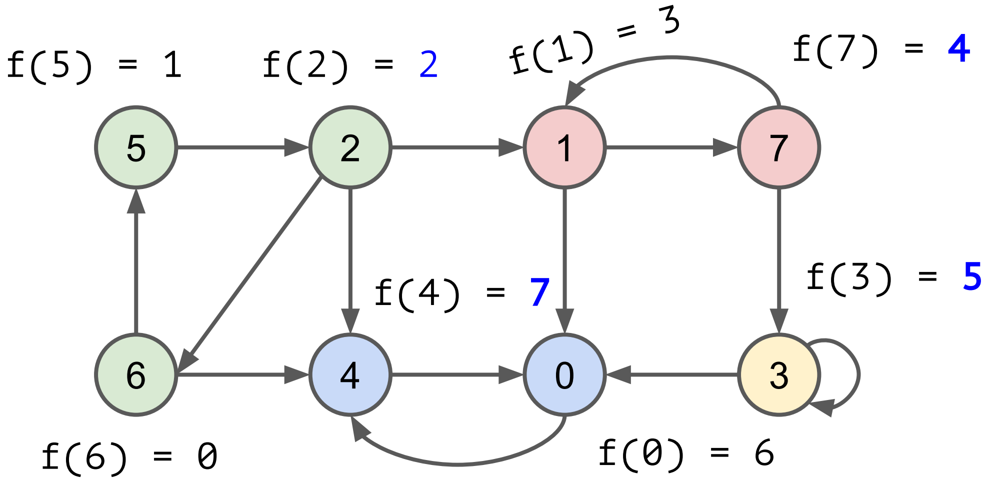
Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f

Если это работает, то за сколько?



Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

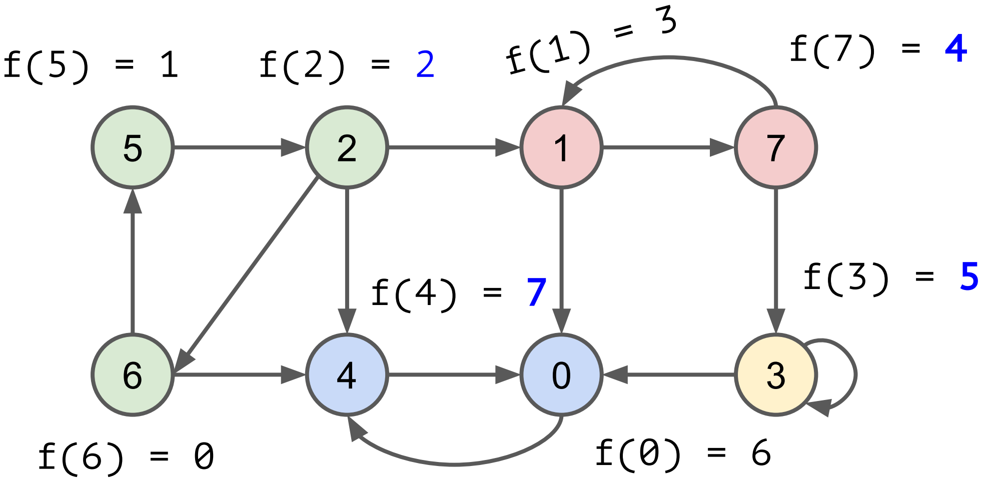
2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f

Если это работает, то за сколько?

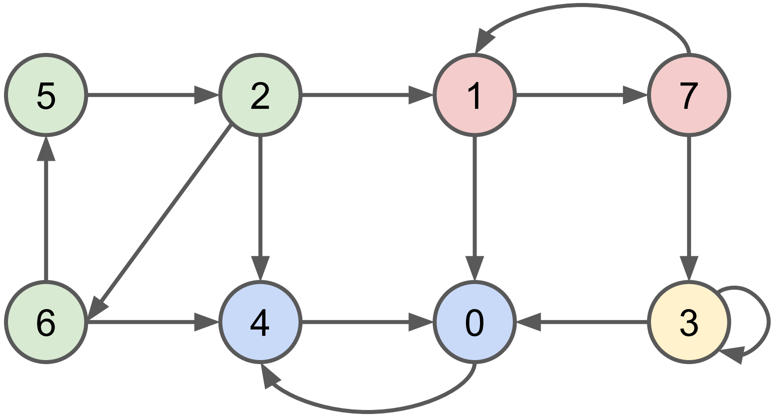
$$O(|V| + |E|)$$

Это ведь два DFS-а



Алгоритм Косараджу: корректность

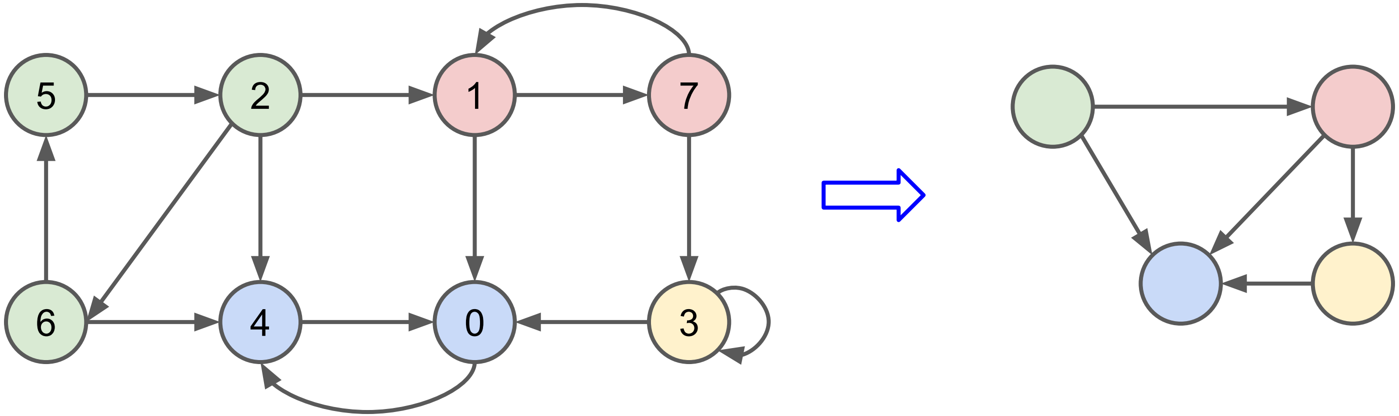
Несколько соображений для доказательства корректности.



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

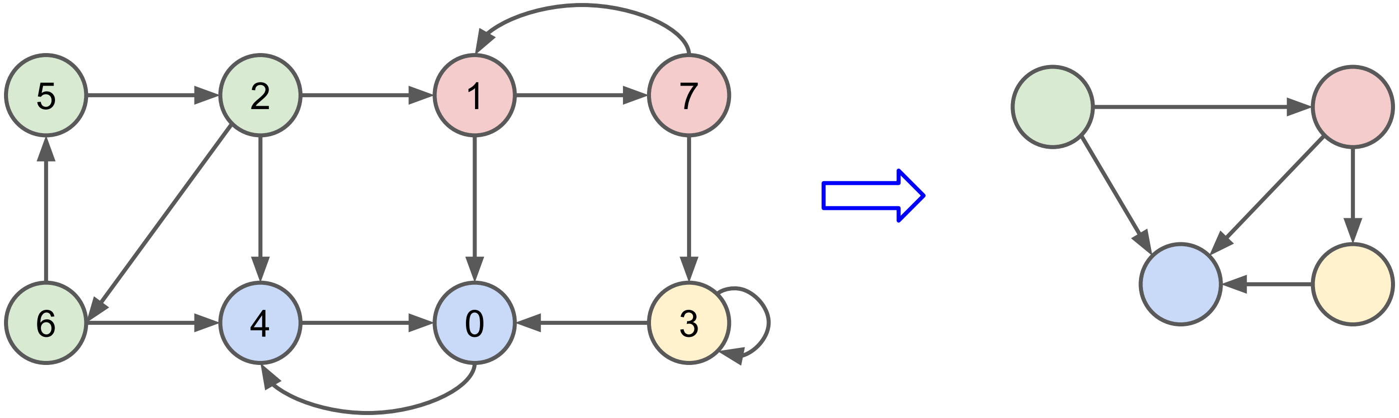
1. SCC графа образуют граф



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

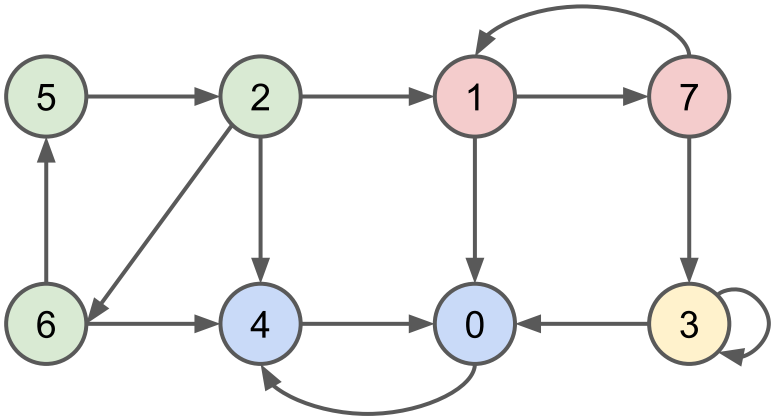
1. SCC графа образуют граф, и в нем нет циклов!



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

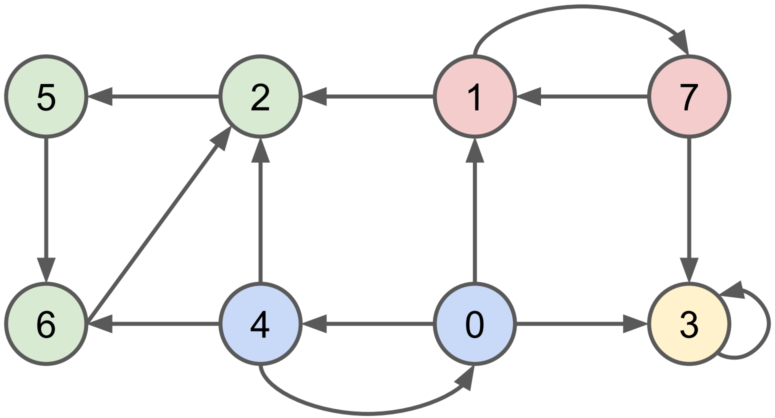
1. SCC графа образуют граф, и в нем нет циклов!
2. SCC в инвертированном графе совпадают с SCC в обычном



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

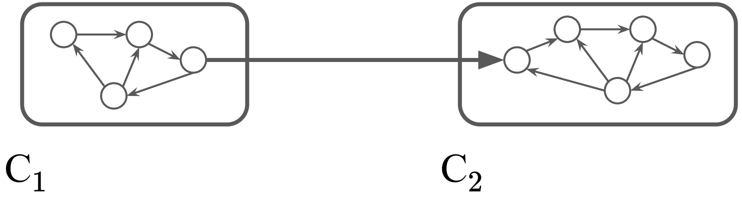
1. SCC графа образуют граф, и в нем нет циклов!
2. SCC в инвертированном графе совпадают с SCC в обычном



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

1. SCC графа образуют граф, и в нем нет циклов!
2. SCC в инвертированном графе совпадают с SCC в обычном
3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

1. SCC графа образуют граф, и в нем нет циклов!
2. SCC в инвертированном графе совпадают с SCC в обычном
3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
   Тогда верно, что: $\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$

C_1 C_2

Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

1. SCC графа образуют граф, и в нем нет циклов!

2. SCC в инвертированном графе совпадают с SCC в обычном

3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
   Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

4. Если лемма верна, то и весь алгоритм корректен.

Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

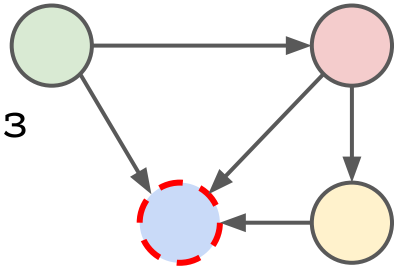
1. SCC графа образуют граф, и в нем нет циклов!

2. SCC в инвертированном графе совпадают с SCC в обычном

3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
   Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

4. Если лемма верна, то и весь алгоритм корректен.

   Действительно: взяв максимальный по f элемент мы получим сток в графе из SCC.



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

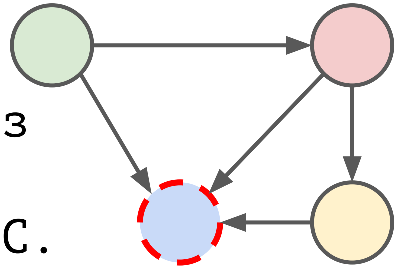
1. SCC графа образуют граф, и в нем нет циклов!

2. SCC в инвертированном графе совпадают с SCC в обычном

3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
   Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

4. Если лемма верна, то и весь алгоритм корректен.

   Действительно: взяв максимальный
   по f элемент мы получим сток в графе из
   SCC.
   Поэтому DFS разметит только вершины SCC.



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

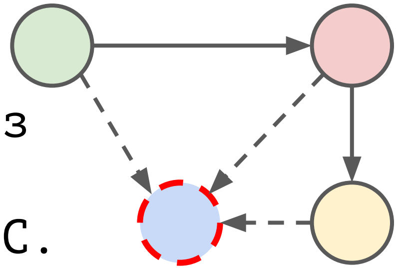
1. SCC графа образуют граф, и в нем нет циклов!

2. SCC в инвертированном графе совпадают с SCC в обычном

3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
   Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

4. Если лемма верна, то и весь алгоритм корректен.

   Действительно: взяв максимальный
   по f элемент мы получим сток в графе из
   SCC.
   Поэтому DFS разметит только вершины SCC.



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

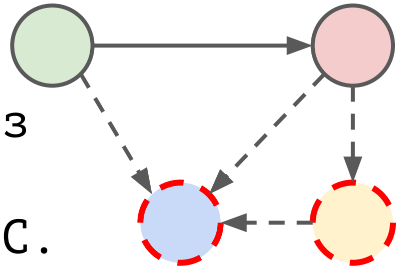
1. SCC графа образуют граф, и в нем нет циклов!

2. SCC в инвертированном графе совпадают с SCC в обычном

3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
   Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

4. Если лемма верна, то и весь алгоритм корректен.

   Действительно: взяв максимальный по f элемент мы получим сток в графе из SCC.
   Поэтому DFS разметит только вершины SCC.



Алгоритм Косараджу: корректность

Несколько соображений для доказательства корректности.

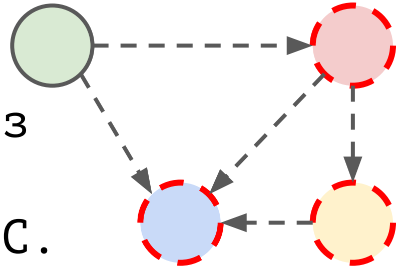
1. SCC графа образуют граф, и в нем нет циклов!

2. SCC в инвертированном графе совпадают с SCC в обычном

3. Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
   Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

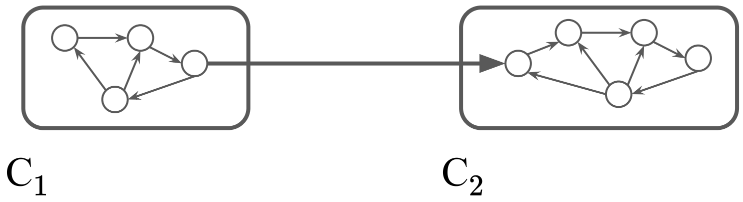
4. Если лемма верна, то и весь алгоритм корректен.

   Действительно: взяв максимальный по f элемент мы получим сток в графе из SCC.
   Поэтому DFS разметит только вершины SCC.



Алгоритм Косараджу: корректность

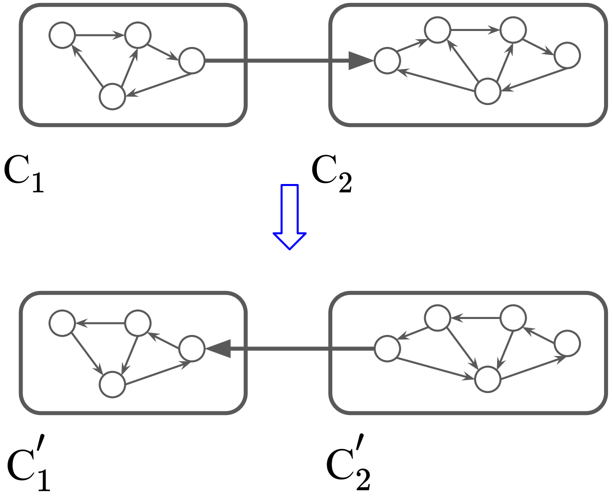
Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



Алгоритм Косараджу: корректность

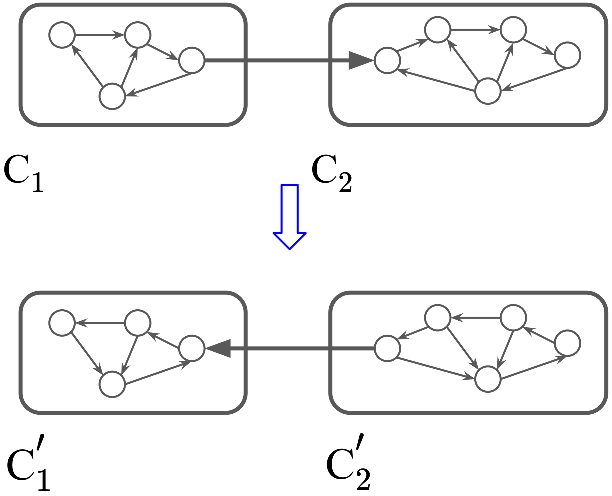
Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами



Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$

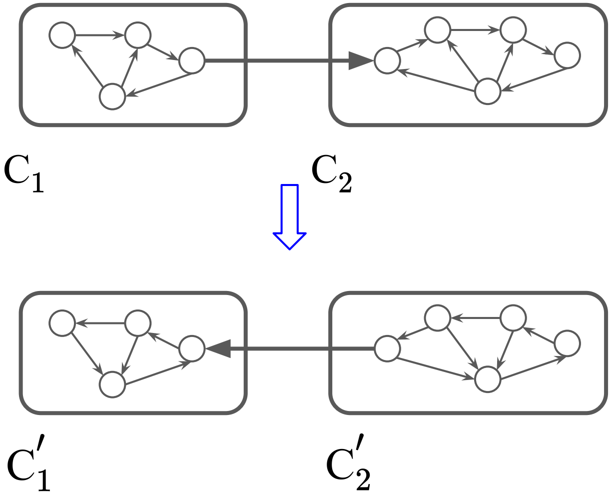


Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

Заметим, что они при этом остаются SCC

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



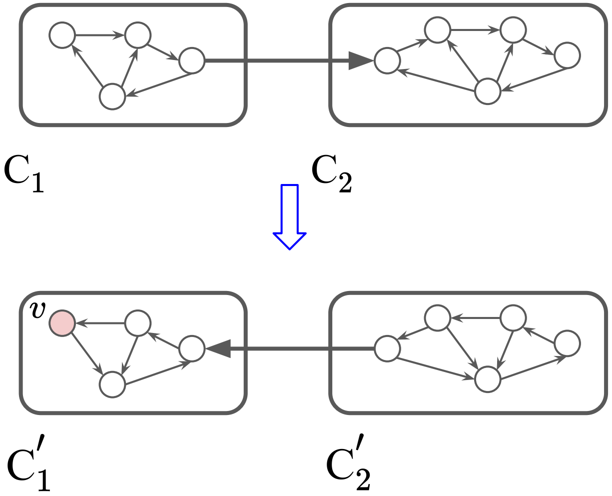
Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

Заметим, что они при этом остаются SCC

Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

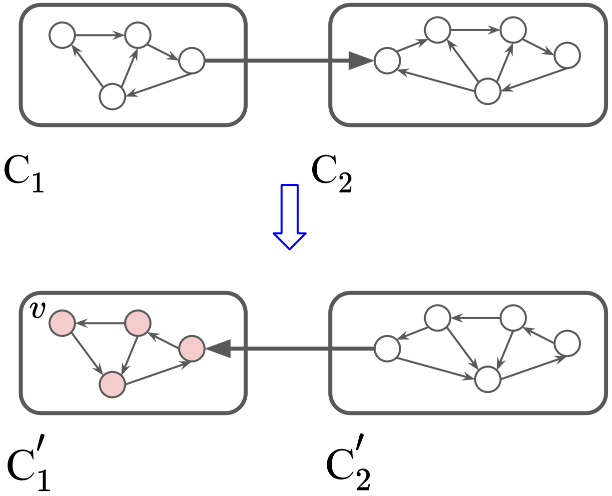
Заметим, что они при этом остаются SCC

Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

1) Пусть $v \in C_1' \Rightarrow$

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

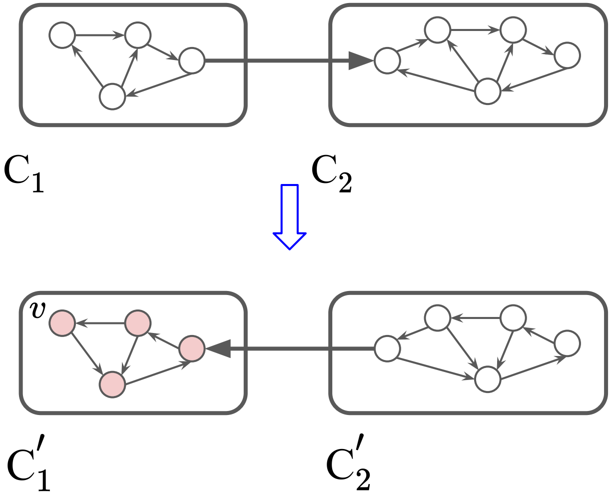
Заметим, что они при этом остаются SCC

Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

1) Пусть $v \in C_1' \Rightarrow$ тогда в C_2' мы придем только после того, как прошли всех в C_1', т.к. в графе из SCC циклов нет.

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$



Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

Заметим, что они при этом остаются SCC

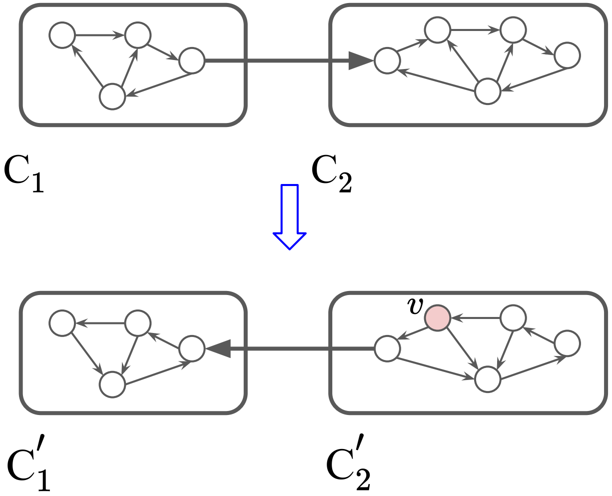
Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

1) Пусть $v \in C_1' \Rightarrow$ тогда в C_2' мы придем только после того, как прошли всех в C_1', т.к. в графе из SCC циклов нет.

   Следовательно для этого случая верно $\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

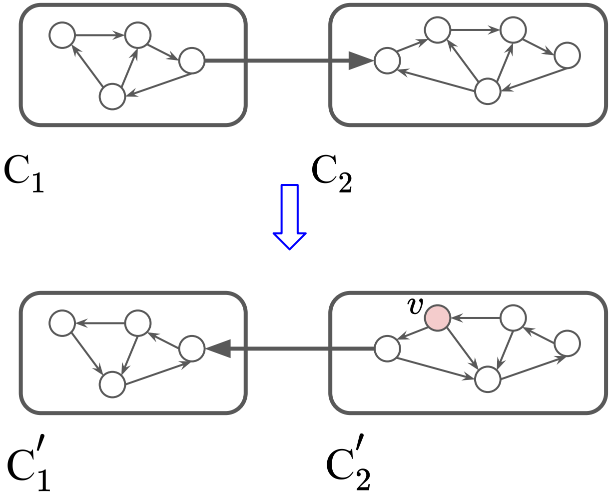
Заметим, что они при этом остаются SCC

Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

1) Пусть $v \in C_1' \Rightarrow$ тогда в C_2' мы придем только после того, как прошли всех в C_1', т.к. в графе из SCC циклов нет.
2) Пусть $v \in C_2' \Rightarrow$

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

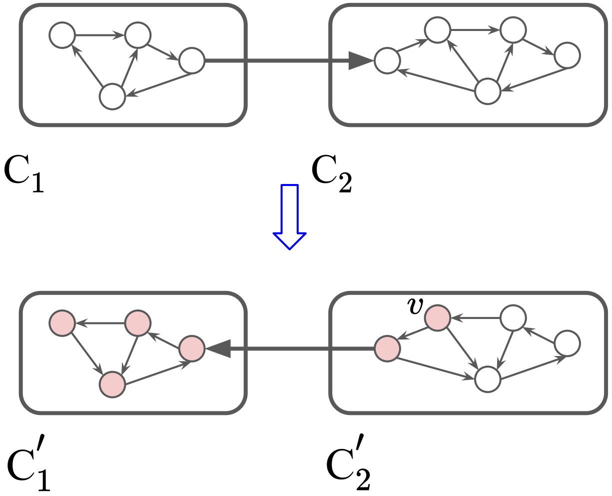
Заметим, что они при этом остаются SCC

Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

1) Пусть $v \in C_1' \Rightarrow$ тогда в C_2' мы придем только после того, как прошли всех в C_1', т.к. в графе из SCC циклов нет.
2) Пусть $v \in C_2' \Rightarrow$ тогда DFS закончится не раньше, чем обойдет все вершины из C_1'!

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

Заметим, что они при этом остаются SCC

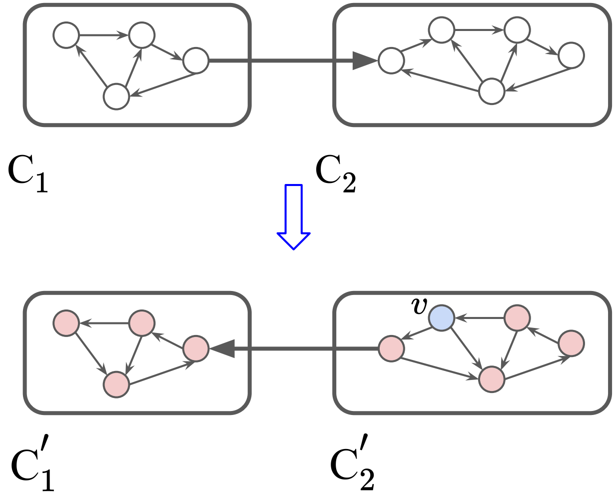
Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

1) Пусть $v \in C_1' \Rightarrow$ тогда в C_2' мы придем только после того, как прошли всех в C_1', т.к. в графе из SCC циклов нет.
2) Пусть $v \in C_2' \Rightarrow$ тогда DFS закончится не раньше, чем обойдет все вершины из C_1'!

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$

Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами



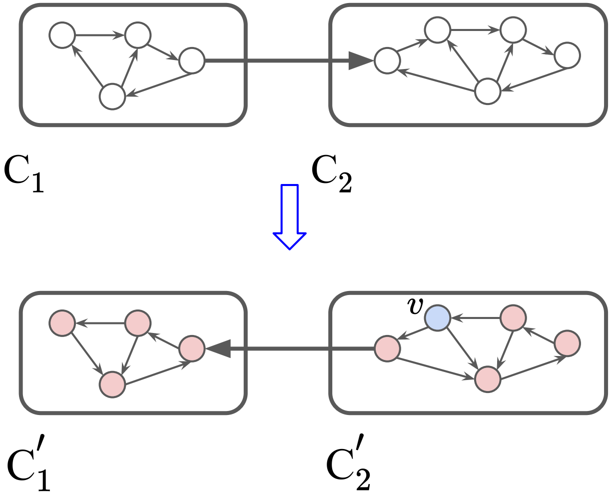
Заметим, что они при этом остаются SCC

Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

1) Пусть $v \in C_1' \Rightarrow$ тогда в C_2' мы придем только после того, как прошли всех в C_1', т.к. в графе из SCC циклов нет.
2) Пусть $v \in C_2' \Rightarrow$ тогда DFS закончится не раньше, чем обойдет все вершины из C_1'!

Алгоритм Косараджу: корректность

Лемма: пусть есть две смежные SCC в графе: C_1 и C_2
Тогда верно, что: $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$



Доказательство: рассмотрим C_1 и C_2 с инвертированными ребрами

Заметим, что они при этом остаются SCC

Пусть v - вершина из, $C_1' \cup C_2'$ в которую впервые пришел DFS по G^{rev}

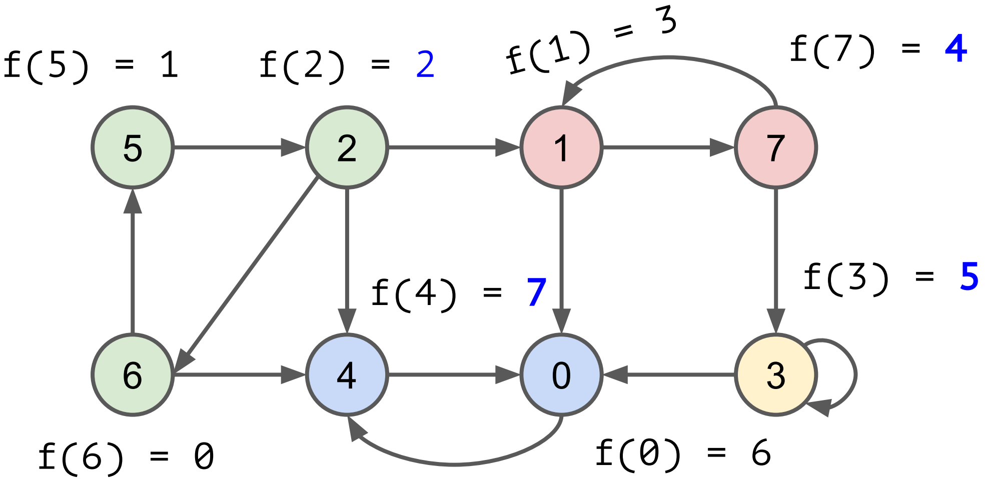
1) Пусть $v \in C_1' \Rightarrow$ тогда в C_2' мы придем только после того, как прошли всех в C_1', т.к. в графе из SCC циклов нет.
2) Пусть $v \in C_2' \Rightarrow$ тогда DFS закончится не раньше, чем обойдет все вершины из C_1'! Значит и в этом случае $\max\limits_{v \in C_1} f(v) < \max\limits_{v \in C_2} f(v)$ $\square$

Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f



Если это работает, то за сколько?

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

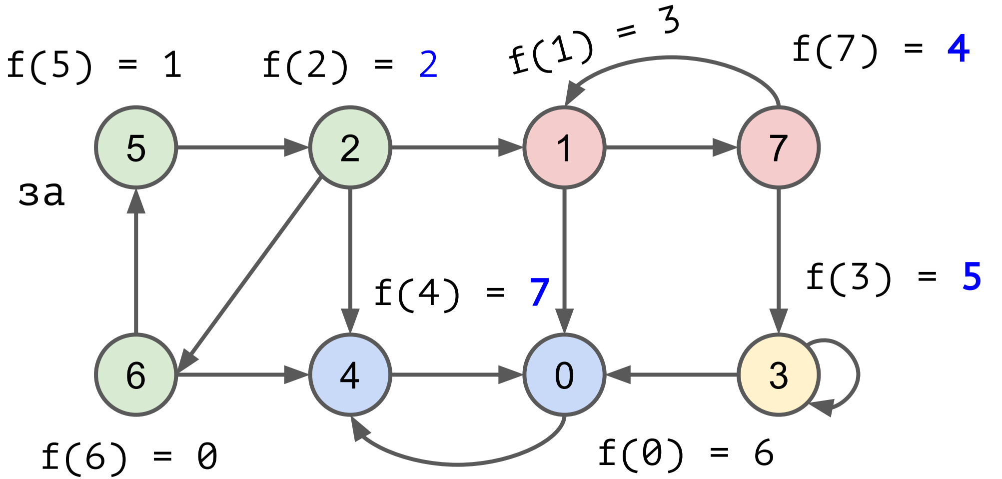
Это ведь два DFS-а

Графы: алгоритм Косараджу

1. По графу G построим граф G^{rev}, где ребра обращены

2. На этом графе G^{rev} запускаем DFS (с циклом, который посетит в результате все вершины). Для каждой вершины запоминаем f(v) - время окончания DFS для нее.

3. В исходном графе запускаем DFS в порядке убывания f



Работает корректно за

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

Это ведь два DFS-а

Мини-задача #31 (2 балла)

Пусть после статического анализа про каждую функцию в программе известно, вызовы каких других функций присутствуют в ее коде. Формат входных данных:

```
foo: bar, baz, qux
bar: baz, foo, bar
qux: qux
```

Вам необходимо найти наибольшую рекурсивную компоненту: множество функций, при исполнении которых можно попасть в любую другую функцию из этого множества.

Кроме того, для каждой функции определить, есть ли в ней рекурсивные (в т.ч. непрямые) вызовы.

Takeaways

- DFS - это рабочая лошадка для огромного количества алгоритмов на графах
- Зачастую его используют, как часть более сложного алгоритма: запускают в разных конфигурациях и комбинациях

Takeaways

- DFS - это рабочая лошадка для огромного количества алгоритмов на графах

- Зачастую его используют, как часть более сложного алгоритма: запускают в разных конфигурациях и комбинациях

- DFS - не всегда рекурсия, итерация + стек даст вам аналогичный эффект

Убедитесь, что вы вынесли с этого курса

Убедитесь, что вы вынесли с этого курса

- Понятие сложности алгоритма, виды сложности и способы ее оценки;

Убедитесь, что вы вынесли с этого курса

- Понятие сложности алгоритма, виды сложности и способы ее оценки;

- Divide-and-conquer алгоритмы и master method;

Убедитесь, что вы вынесли с этого курса

- Понятие сложности алгоритма, виды сложности и способы ее оценки;
- Divide-and-conquer алгоритмы и master method;
- Рандомизированные алгоритмы, зачем они нужны и почему они работают;

Убедитесь, что вы вынесли с этого курса

- Понятие сложности алгоритма, виды сложности и способы ее оценки;
- Divide-and-conquer алгоритмы и master method;
- Рандомизированные алгоритмы, зачем они нужны и почему они работают;
- Отличие абстрактного типа данных от структуры данных;

Убедитесь, что вы вынесли с этого курса

- Понятие сложности алгоритма, виды сложности и способы ее оценки;
- Divide-and-conquer алгоритмы и master method;
- Рандомизированные алгоритмы, зачем они нужны и почему они работают;
- Отличие абстрактного типа данных от структуры данных;
- Выбор подходящей структуры данных и трейдофы при нем.

Алгоритмы и структуры данных продолжатся
(в третьем семестре)