ITIVITI

LET IT TRADE

Алгоритмы и структуры данных

Косяков Михаил Сергеевич
к.т.н., доцент кафедры ВТ

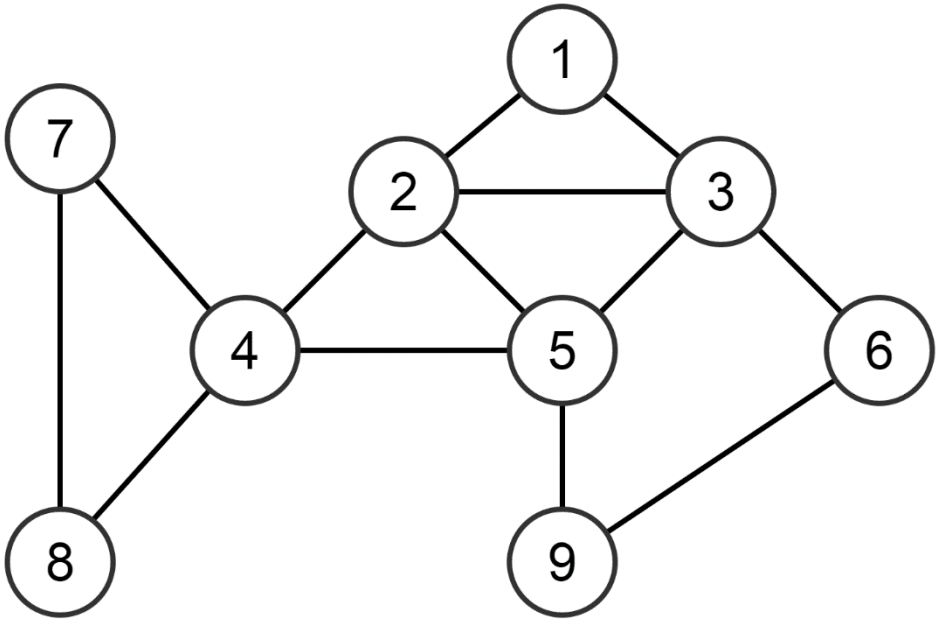
УНИВЕРСИТЕТ ИТМО

Содержание курса

- Введение в теорию алгоритмов
- Алгоритмы сортировок
- Структуры данных
  - Линейные структуры
  - Бинарные деревья поиска
  - Хеши и хеш-функции
- **Алгоритмы на графах**
  - **Обходы графов в ширину и глубину**
  - Минимальные остовные деревья
  - Поиск кратчайших путей в графе

DFS

- Идем вперед пока возможно (если есть непосещённый сосед, идем в него)
- Возвращаемся назад, когда нет возможности идти вперед
- Проверка на двудольность?



DFS: реализация

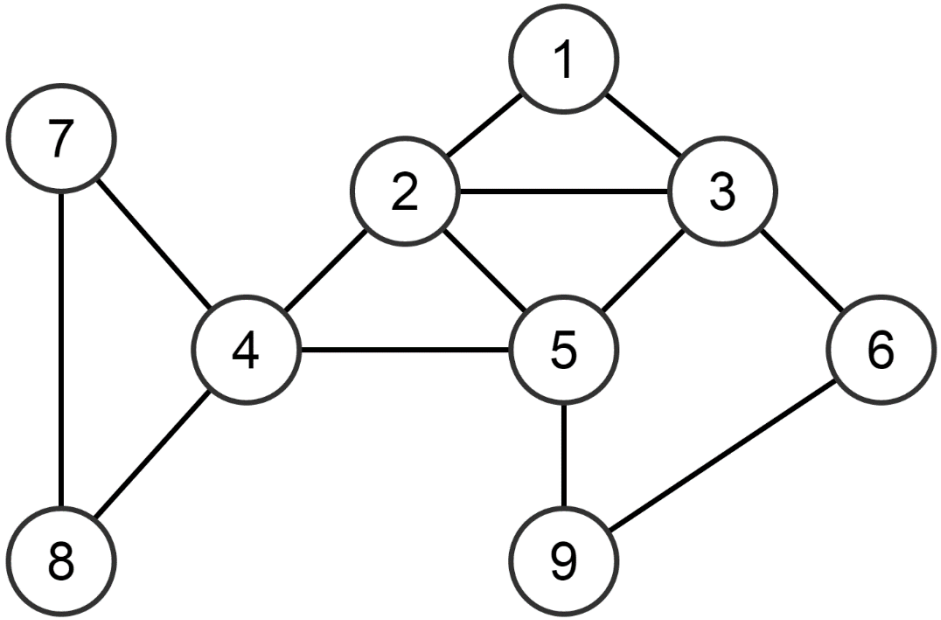
```
01: class Node {
02: public:
03:     Node() : color(WHITE), predecessor(-1) { /* */ }
04:     int color;
05:     int predecessor;
06:     int in, out;
07: };
08:
09: typedef std::vector<int> EdgeVec;
10: typedef std::vector<EdgeVec> Graph;
11: typedef std::vector<Node> VisitedMap;
12:
13: void DFS(const Graph & g, VisitedMap & vmap)
14: {
15:     int t = 0;
16:     for (int i = 0; i < g.size(); ++i)
17:         if (vmap[i].color == WHITE) DFS_visit(g, i, vmap , t);
18: }
```

DFS: реализация

```
01: void DFS_visit(const Graph &g, int s, VisitedMap &vmap, int &t)
02: {
03:     vmap[s].color = GRAY;
04:     cout << "in(" << s << ")" << endl;
05:     vmap[s].in = t++;
06:     for (int i = 0; i < g[s].size(); ++i) {
07:     const int v = g[s][i];
08:     if (vmap[v].color == WHITE) {
09:         vmap[v].predecessor = s;
10:         DFS_visit(g, v, vmap, t);
11:         }
12:     }
13:     cout << "out(" << s << ")" << endl;
14:     vmap[s].out = t++;
15:     vmap[s].color = BLACK;
16: }
```

- Реализация без рекурсии?

DFS: пример процесса поиска



- Позволяет ли DFS находить кратчайшие расстояния?
- Какова структура дерева?
- Какова структура серых отрезков [*in* (*s*), *out* (*s*)]?

Основные свойства DFS

ITIVITI

LET IT TRADE

- **Свойство 1**: после выполнения алгоритма, вершина v будет отмечена, как посещенная, тогда и только тогда, когда существует путь от s к v
- **Свойство 2:** время выполнения $O(n + m)$

УНИВЕРСИТЕТ ИТМО

Основные свойства DFS

- **Свойство 3 (Теорема о скобках):** для вершин u и v выполняется ровно одно из трех утверждений:
  - серые отрезки $[in(u), out(u)]$ и $[in(v), out(v)]$ не пересекаются и ни u не является потомком v в лесу поиска в глубину, ни v не является потомком u
  - $[in(u), out(u)]$ полностью содержится в $[in(v), out(v)]$, и u является потомком v в дереве поиска в глубину
  - $[in(v), out(v)]$ полностью содержится в $[in(u), out(u)]$, и v является потомком u в дереве поиска в глубину
- Почему? Серые вершины образуют путь из вершины, откуда запустили DFS_visit()

DFS: классификация ребер

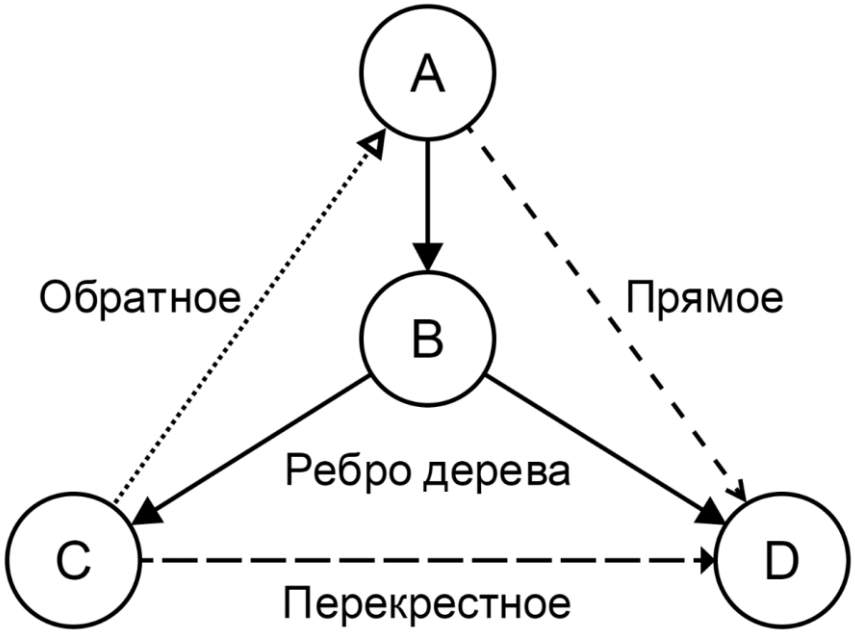
ITIVITI

LET IT TRADE

- Ребро дерева (*tree edge*): в дереве DFS
- Прямое ребро (*forward edge*): ведет к потомку в дереве DFS
- Обратное ребро (*back edge*): ведет к предку в дереве DFS
- Перекрестное ребро (*cross edge*): ведет к вершине, которая ни предок, ни потомок (в одном дереве или в разных деревьях DFS)

УНИВЕРСИТЕТ ИТМО

DFS: классификация ребер



- В каком порядке надо обходить вершины, чтобы получить такое дерево?
- Как определить тип ребра?

DFS: классификация ребер

- При первом исследовании ребра (u, v):
- u – какого цвета?
- v – белая: ребро дерева
- v – серая: обратное ребро
- v – черная: прямое или перекрестное ребро
  - Ребро прямое, если $in(u) < in(v)$
  - Ребро перекрестное, если $in(u) > in(v)$
- В неориентированных графах v может быть черной?
- В неориентированных графах какие бывают ребра?

Ориентированный ациклический граф (DAG)

ITIVITI

- Отношение – множество упорядоченных пар объектов
  - Отношение включения $\subset$ на множестве подмножеств данного множества – частичный порядок
- DAG – неявная модель частичных порядков
  - Достижимость вершин с помощью ненулевого пути задает частичный порядок в DAG без петель
- Частичный порядок предшествования:
  - Генеалогическое дерево
  - Учебный план и зависимости дисциплин
  - Причинно-следственная связь
  - Планирование выполнения задач (scheduling)

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Ориентированный ациклический граф (DAG)

ITIVITI

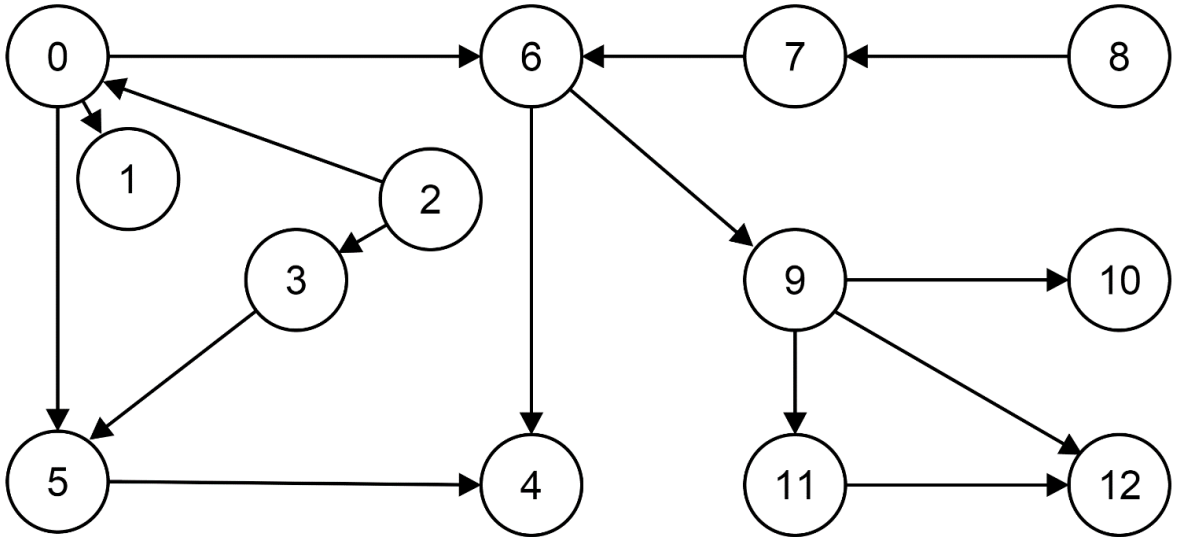
- В каждом DAG есть как минимум один исток (*source* – нет входящих ребер) и один сток (*sink* – нет исходящих ребер)
  - Берем любую вершину и последовательно идем по ребрам
  - Если никогда не остановимся, то есть цикл … или найдем сток
  - Исток – аналогично: идем по ребрам в обратном направлении

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

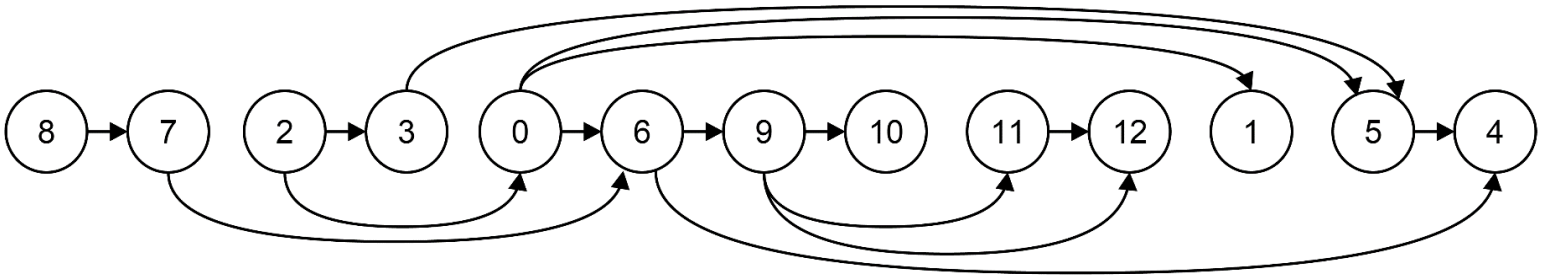
Топологическая сортировка DAG

- Линеаризация частичного порядка (последовательное исполнение задач с сохранением частичного порядка)
- Требуется занумеровать вершины в таком порядке, что ребра будут идти только из вершин с меньшими номерами в вершины с большими номерами



Топологическая сортировка DAG

- Такая перестановка вершин – топологический порядок



- При обходе в топологическом порядке при посещении вершины мы гарантировано посетили всех её предков и посетим всех её потомков после её посещения
- Pre-order: кладем вершину в очередь *перед* рек. вызовами
- Post-order: кладем вершину в очередь *после* рек. вызовов
- Reverse post-order: кладем вершину *в стек после* рекурсивных вызовов

Топологическая сортировка DAG

- Pre-order vs Post-order vs Reverse Post-Order

```
1.  $ cat A/Makefile
2.  build: B C
3.
4.  $ cat B/Makefile
5.  build: D
6.
7.  $ cat C/Makefile
8.  build: D
9.
10. $ cat D/Makefile
11. build:
```

Топологическая сортировка DAG

ITIVITI

- Pre-order vs Post-order vs Reverse Post-Order

```
1.  A$ make
2.  make[1]: Entering directory 'B'
3.  make[2]: Entering directory 'D'
4.  Building lib D
5.  make[2]: Leaving directory 'D'
6.  Building lib B
7.  make[1]: Leaving directory 'B'
8.  make[1]: Entering directory 'C'
9.  make[2]: Entering directory 'D'
10. Building lib D
11. make[2]: Leaving directory 'D'
12. Building lib C
13. make[1]: Leaving directory 'C'
14. Building lib A
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Топологическая сортировка DAG

ITIVITI

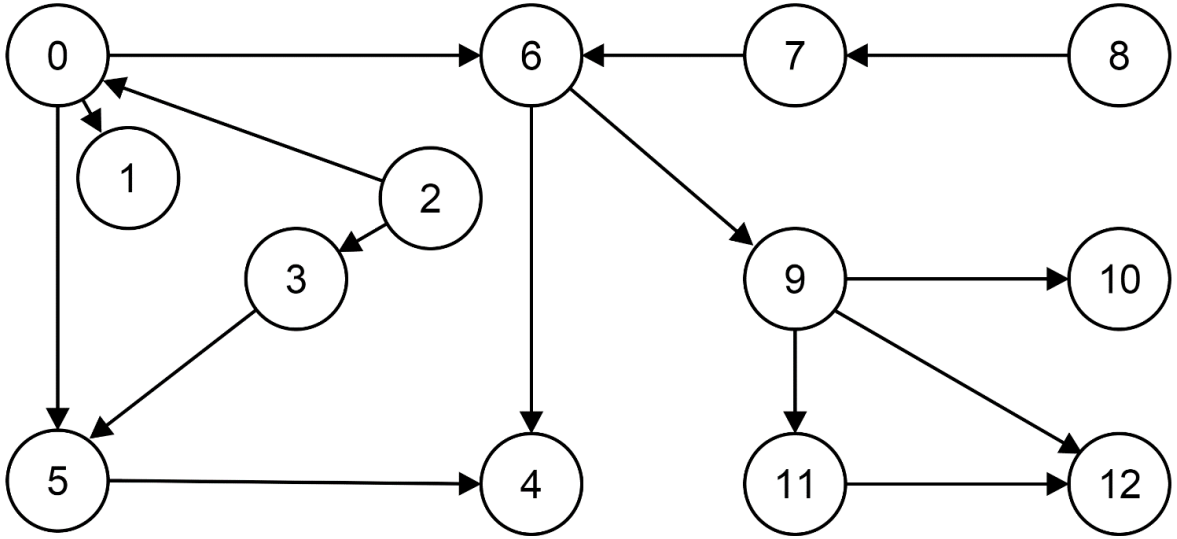
- Pre-order vs Post-order vs Reverse Post-Order

```
1.  A$ make
2.  make[1]: Entering directory 'D'
3.  Building lib D
4.  make[1]: Leaving directory 'D'
5.  make[1]: Entering directory 'B'
6.  Building lib B
7.  make[1]: Leaving directory 'B'
8.  make[1]: Entering directory 'C'
9.  Building lib C
10. make[1]: Leaving directory 'C'
11. Building lib A
```

LET IT TRADE

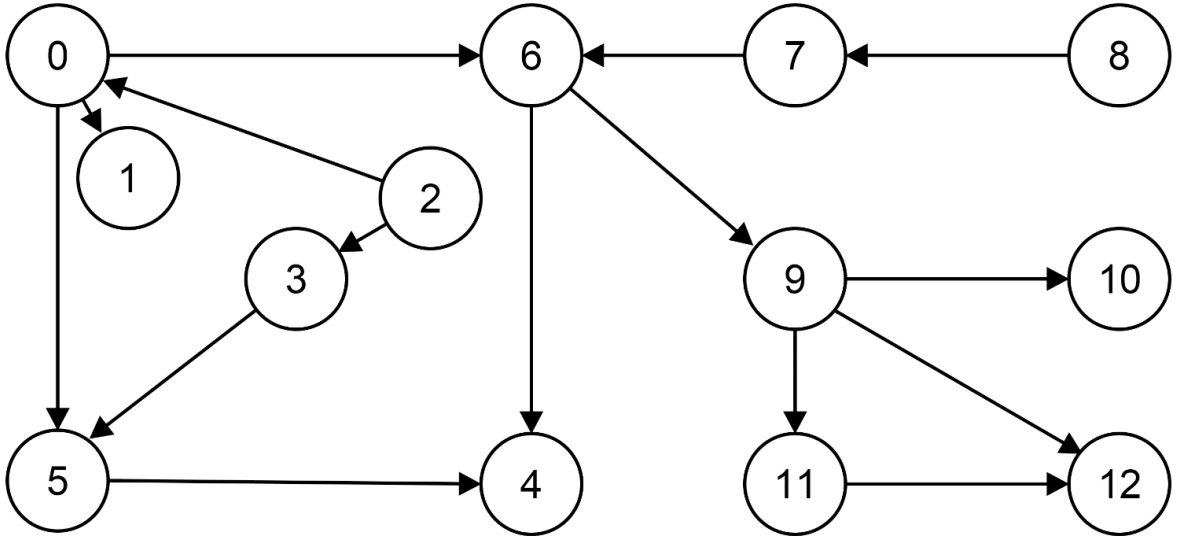
УНИВЕРСИТЕТ ИТМО

Топологическая сортировка DAG



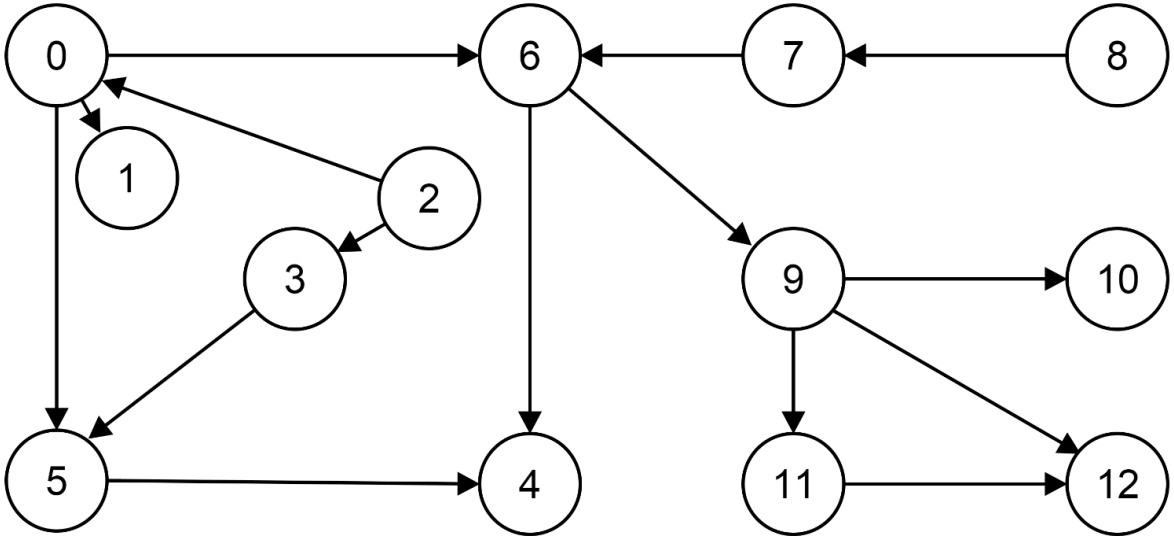
- Когда граф может иметь топологический порядок?
  - Если есть цикл, то топологического порядка нет
  - Если нет цикла, есть ли топологический порядок?

Топологическая сортировка DAG



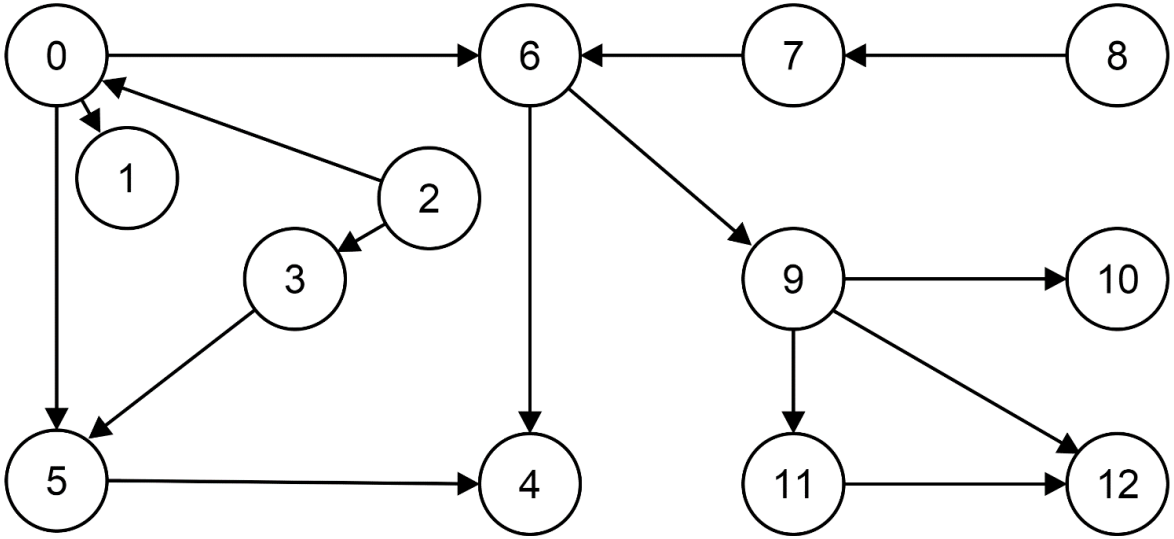
- Какая вершина кандидат на первое место в топологическом порядке?

Топологическая сортировка DAG



- Если удалить исток с его ребрами появятся ли циклы?

Топологическая сортировка DAG



- Если удалить исток с его ребрами появятся ли циклы?
- Почему это работает?
  - С удалением истока удаляются ребра, которые идут в еще непронумерованные вершины (получат большие номера)

Топологическая сортировка с помощью DFS

ITIVITI

- Если в DAG запустить DFS из вершины, из которой достижимы все остальные вершины, то для произвольного ребра (u, v): $out(v) < out(u)$
  - v не может быть серой, т.к. иначе (u, v) – обратное ребро
  - если v – белая, то v становится потомком u, и $out(v) < out(u)$
  - если v – черная, значит $out(v)$ уже установлено, а $out(u)$ – еще нет, поэтому $out(v) < out(u)$
- Топологическая сортировка – в порядке убывания значений $out(s)$
- Какое будет время выполнения?

LET IT TRADE

Топологическая сортировка с помощью DFS


```
01: void DFS(const Graph &g, int s, vector<int> &visited, \
02: vector<int> &in_order)
03: {
04:     visited[s] = true;
05:     for (int i = 0; i < g[s].size(); ++i) {
06:         const int v = g[s][i];
07:         if (!visited[v]) DFS(g, v, visited, in_order);
08:     }
09:     in_order.push_back(s);
10: }
11:
12: void topological_sort(const Graph &g, int s, vector<int> &in_order)
13: {
14:     vector<int> visited(g.size());
15:     for (int i = 0; i < g.size(); ++i)
16:         if (!visited[i]) DFS(g, i, visited, in_order);
17:     reverse(in_order.begin(), in_order.end());
18: }
```

Топологическая сортировка с помощью DFS

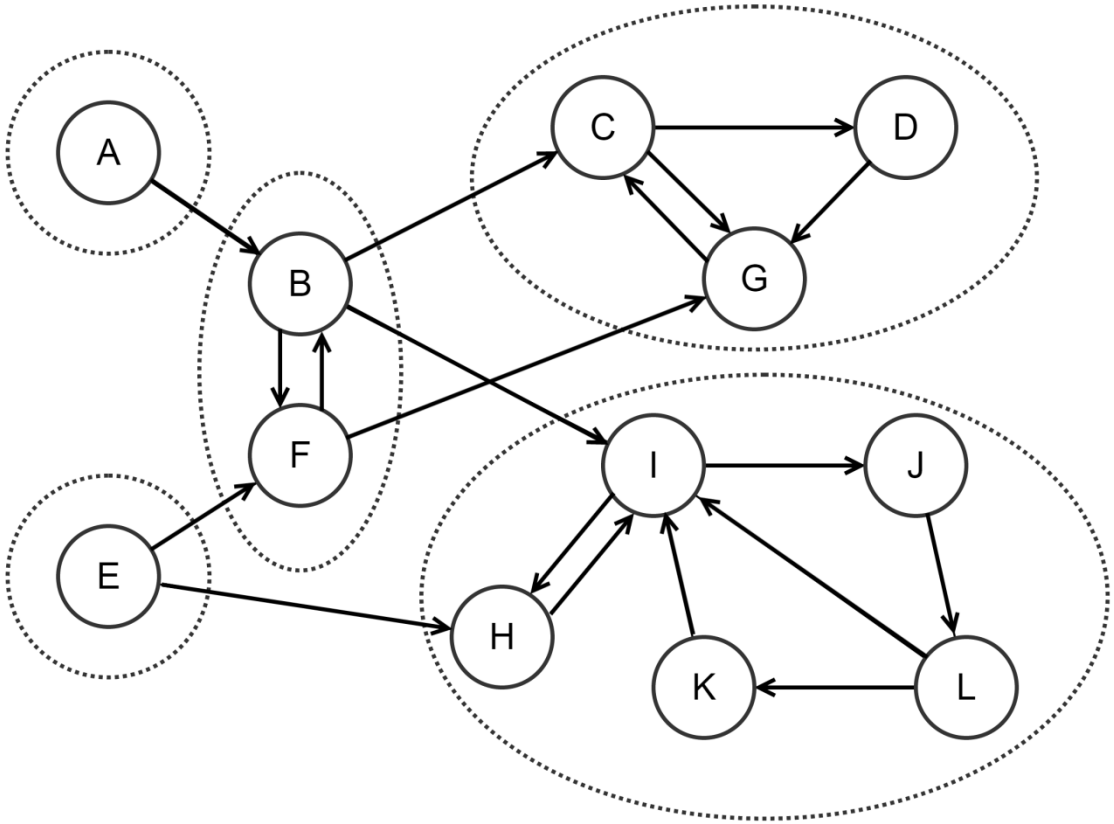
ITIVITI

- Как обнаружить есть ли цикл?
- Если ребро (u, v) обратное (при исследовании u вершина v – серая) то цикл есть
- Если есть цикл, то мы его найдем?
  - Рассмотрим вершину цикла u, которая первой станет черной; пусть v – следующая за u на цикле
  - v не может быть черной (по условию)
  - v не может быть белой (иначе u не почернеет)
  - если v – серая, значит найдем цикл
- Как найти сам цикл?

Декомпозиция орграфа

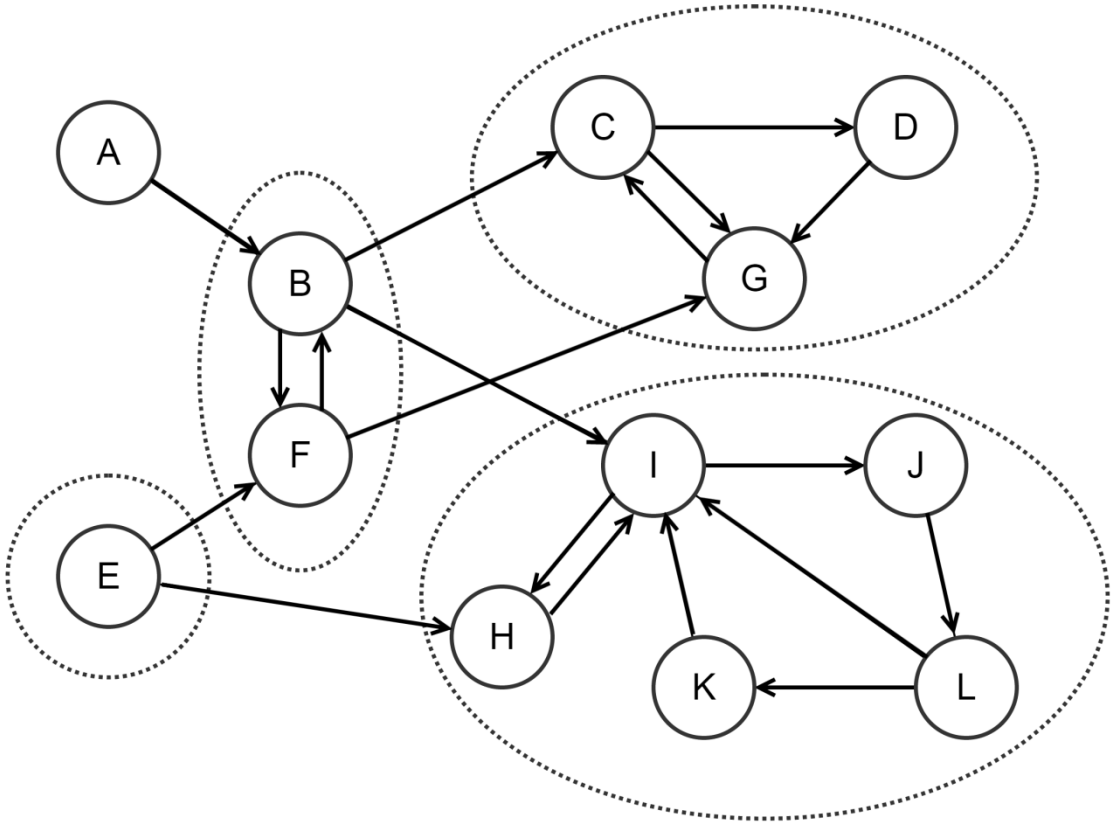
- Post-order для графа с циклами?
- В неориентированных графах использовали отношение эквивалентности «связность» - *v* достижима из *u*
- В орграфах, «связность» не является отношением эквивалентности: *v* достижима из *u* ≠> *u* достижима из *v*
- Сильно связные компоненты: путь из *u* в *v* и обратный путь из *v* в *u*
- Сильно связные компоненты (*strongly connected components – SCC*) задают классы эквивалентности
- Выявление взаимозависимых вершин, например, для организации программных модулей

Сильно связные компоненты

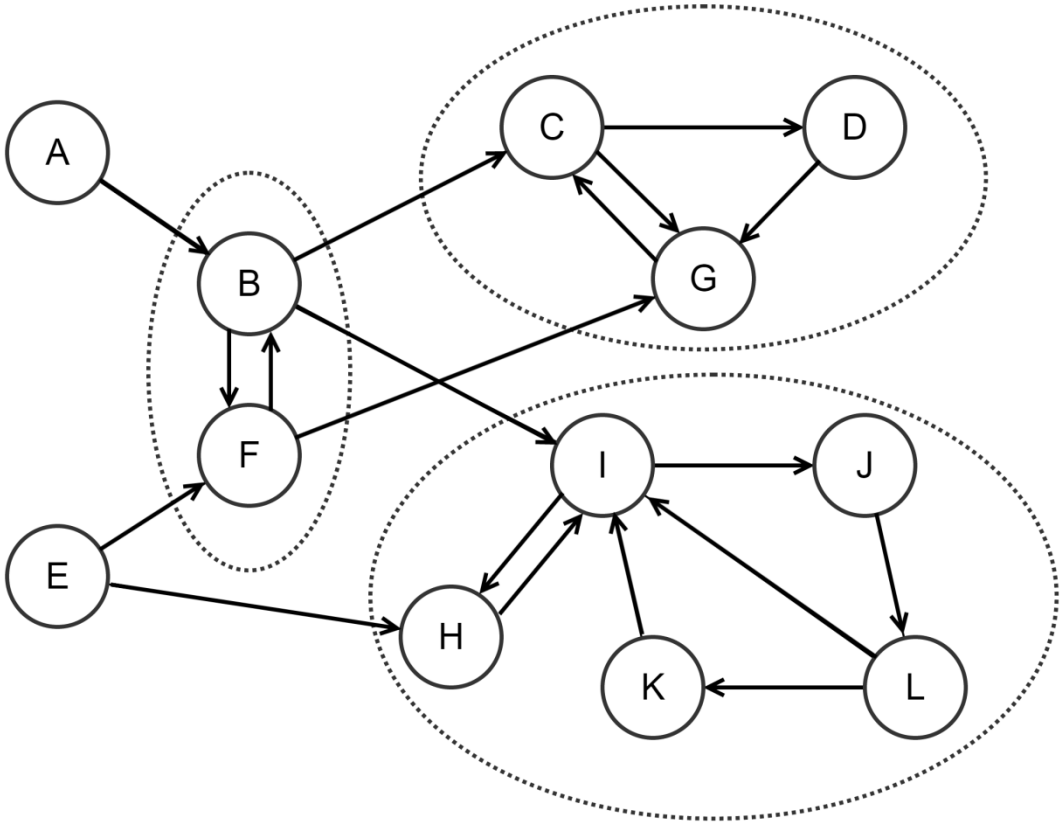


- Что будет, если мы сожмем каждый SCC в одну вершину?

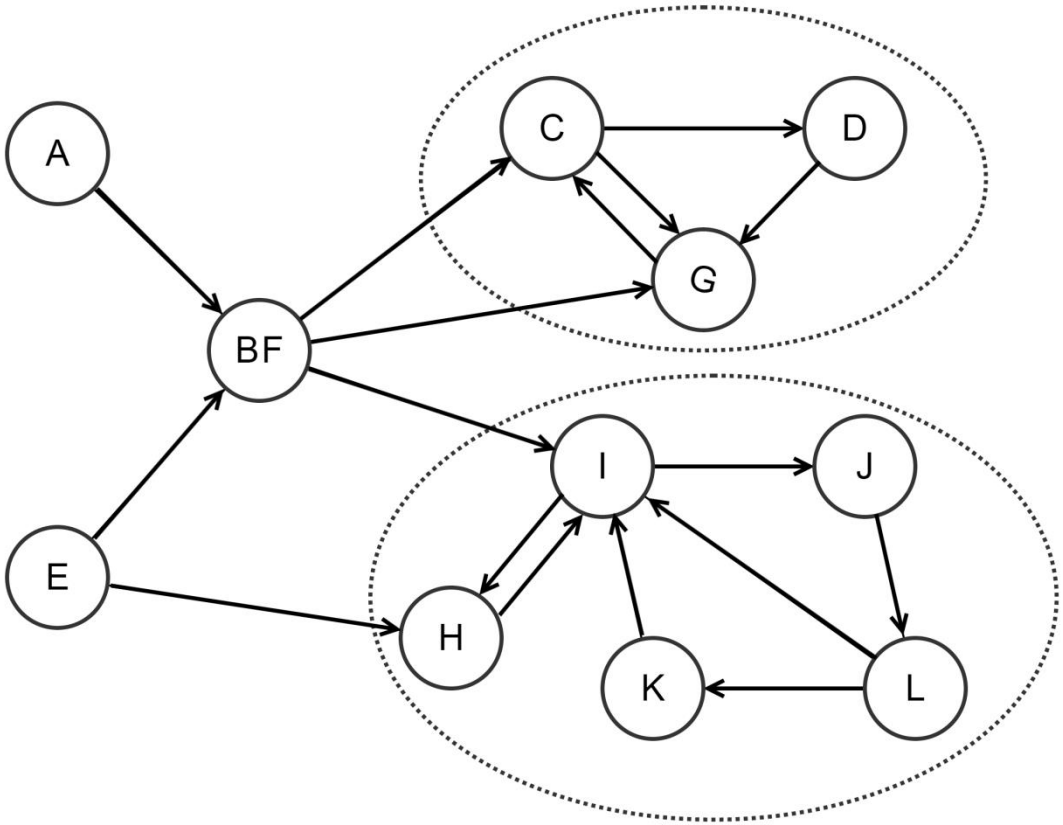
Сильно связные компоненты



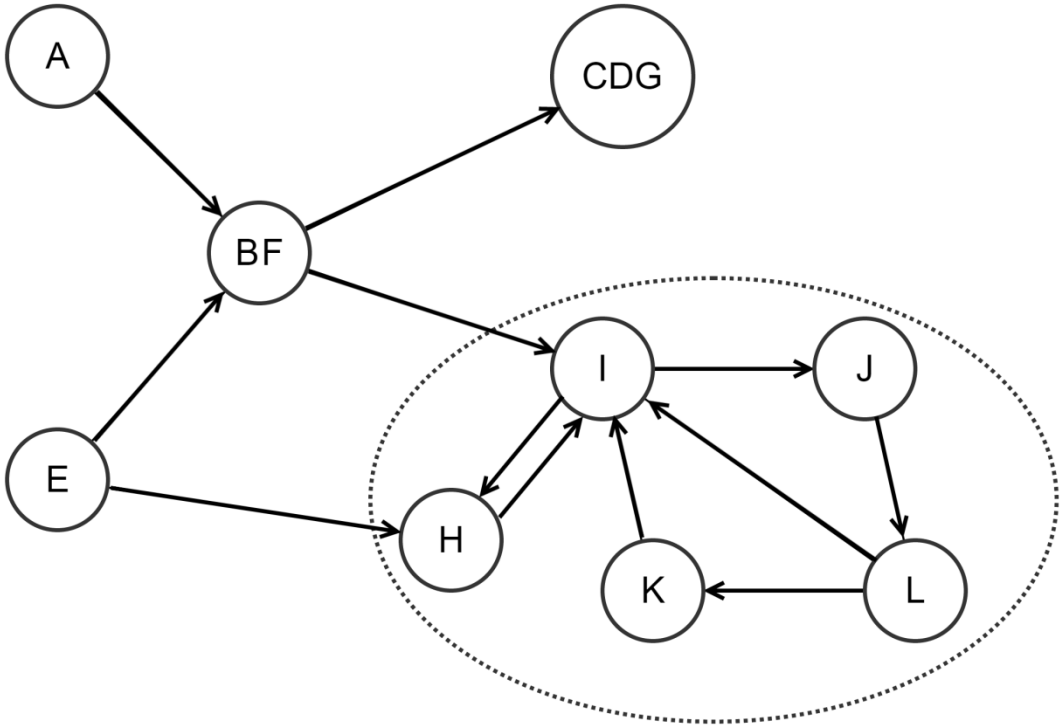
Сильно связные компоненты



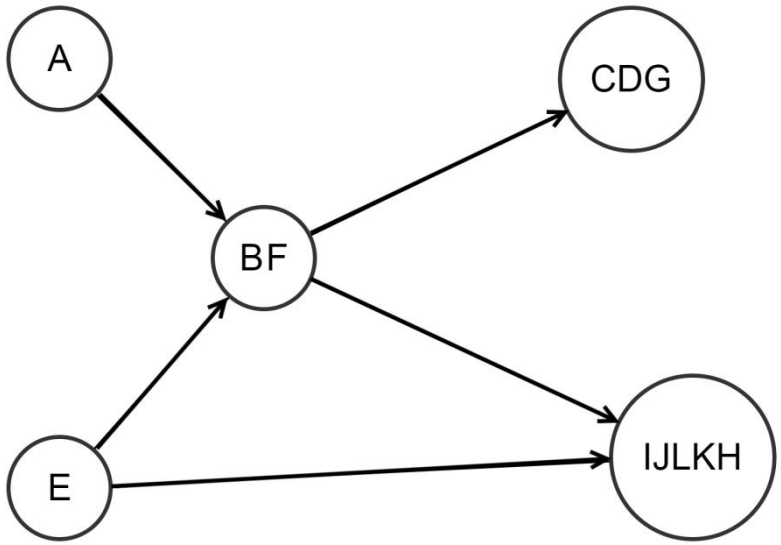
Сильно связные компоненты



Сильно связные компоненты

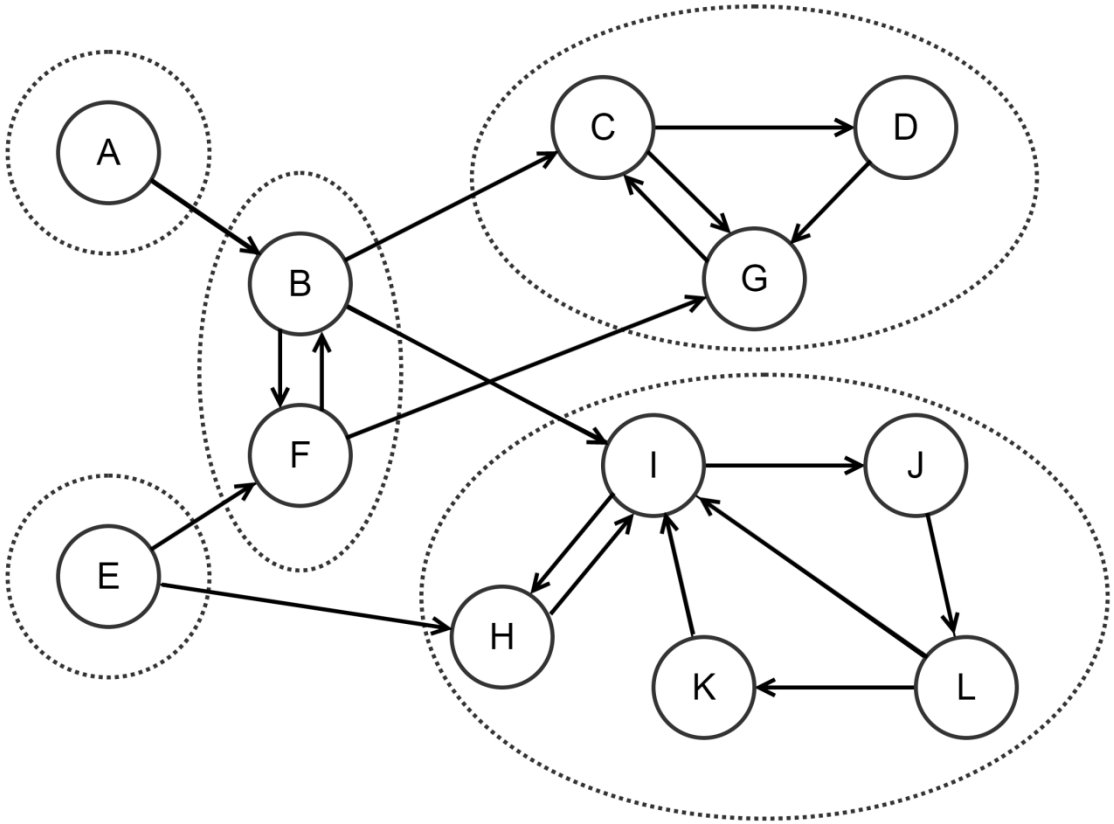


Сильно связные компоненты



- Мы получим DAG !
  Каждый орграф – DAG своих SCC (SCC-исток и SCC-сток)
- Как нам найти все SCC? Ответ: DFS

Сильно связные компоненты



- Из какого SCC надо начать поиск DFS?
- Что дальше?

Сильно связные компоненты

- Вспоминаем: если (u, v) – ребро в DAG, то $out(u) > out(v)$
- **Теорема:** если C и C' – два SCC и существует ребро из C в C', тогда $\max[out(u)]$ для $u \in C$ больше, чем $\max[out(u)]$ для $u \in C'$
  - Если DFS вначале посещает вершину из C' …
  - Если DFS вначале посещает вершину из C …
- **Следствие:** вершина u с наибольшим значением $out(u)$ принадлежит SCC-истоку

Сильно связные компоненты

ITIVITI

LET IT TRADE

- Мы знаем как найти SCC-исток, но нужен SCC-сток…
- Обращаем ребра графа: получаем G^R
- Если запустить DFS в графе G^R вершина u с наибольшим значением $out(u)$ будет принадлежать SCC-стоку в графе G

УНИВЕРСИТЕТ ИТМО

Сильно связные компоненты

- Алгоритм:
- Запускаем DFS в G^R, сохраняем $f(u) = out(u)$; цель – вычисляем порядок запуска DFS для второго прогона
- Запускаем DFS в G, обрабатывая вершины в порядке убывания сохраненных $f(u)$; цель – определяем SCC один за другим
- Помечаем посещаемые вершины значением $f(u)$ вершины, из которой запустили DFS; SCC – вершины с одинаковыми отметками $f(u)$
- Как построить DAG компонентов?
- Пример !!!

ITIVITI

LET IT TRADE

Спасибо за внимание!

УНИВЕРСИТЕТ ИТМО