Алгоритмы и структуры данных

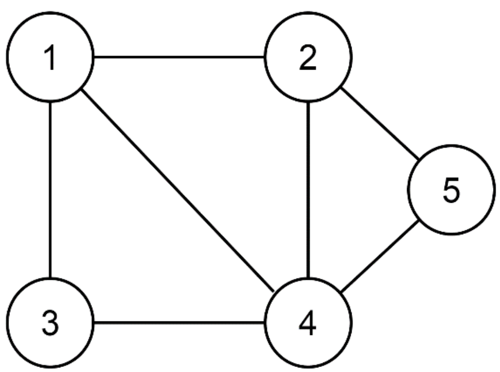
Косяков Михаил Сергеевич
к.т.н., доцент кафедры ВТ

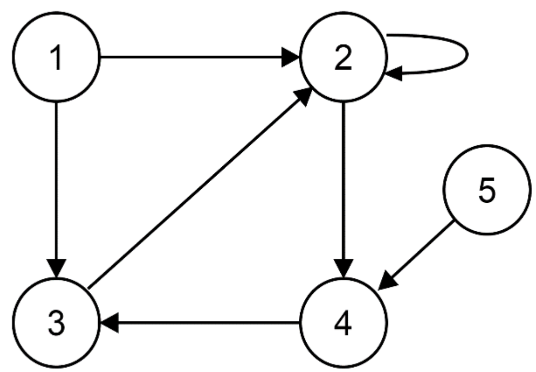
Содержание курса

- 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

Графы

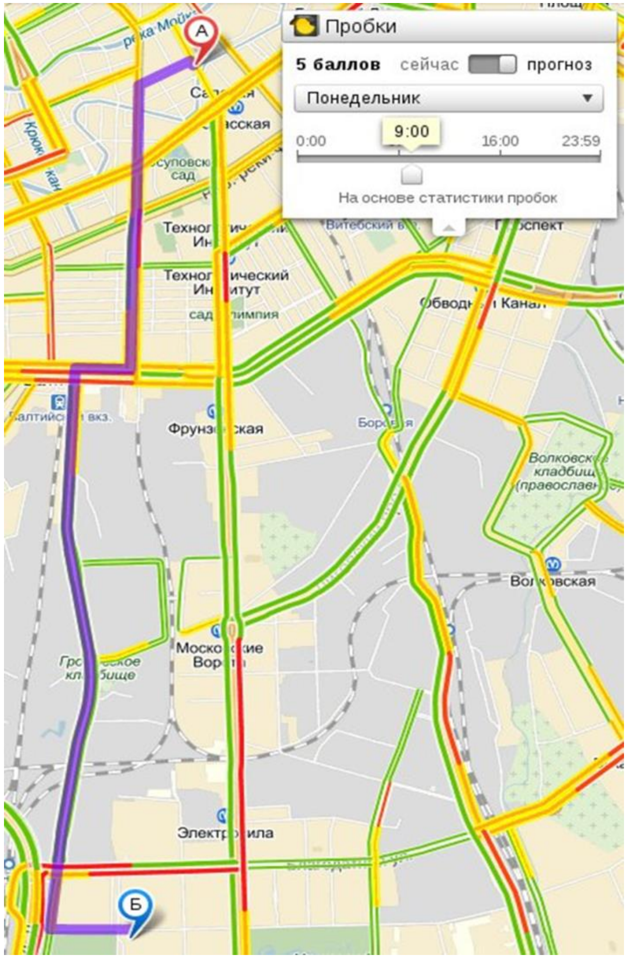
- Граф $G(V, E)$ описывает бинарное отношение на (непустом) множестве объектов
- Объекты: множество вершин V
- Отношение: множество ребер E (ребро = пара вершин)
  - неупорядоченные пары (неориентированный граф)
  - упорядоченные пары (ориентированный граф)





Примеры:
что есть вершины / ребра?

- Транспортные сети
  - кратчайшие пути
  - максимальная пропускная способность

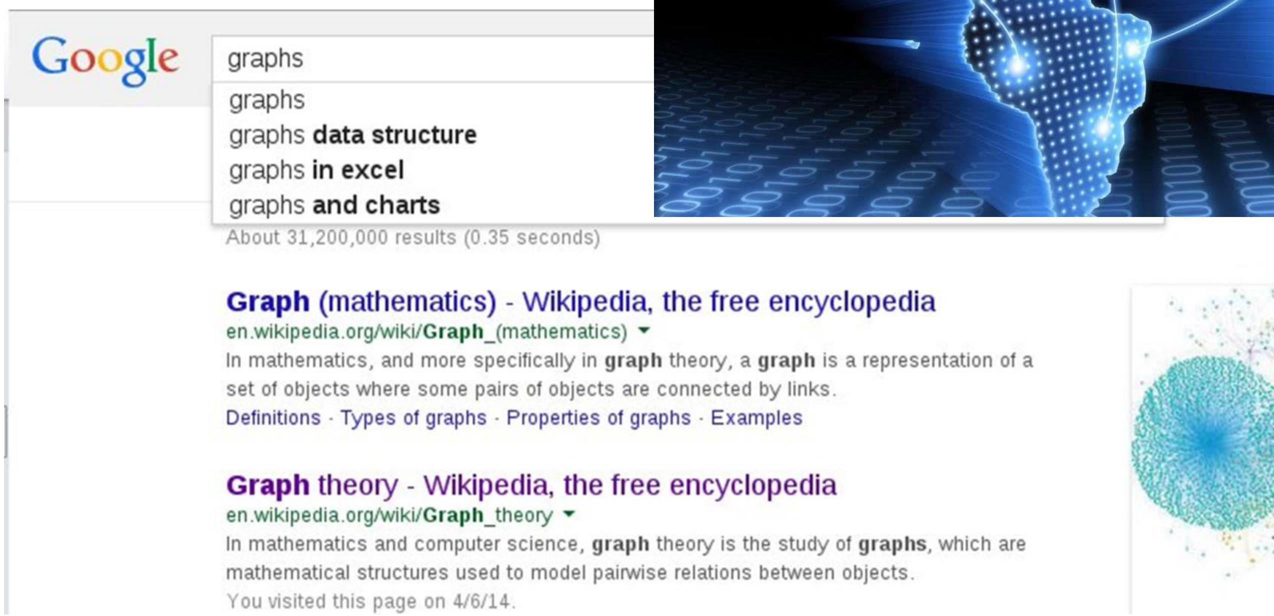


УНИВЕРСИТЕТ ИТМО

LET IT TRADE

Примеры: что есть вершины / ребра?

- Интернет
  - задача о связности
  - поисковые системы

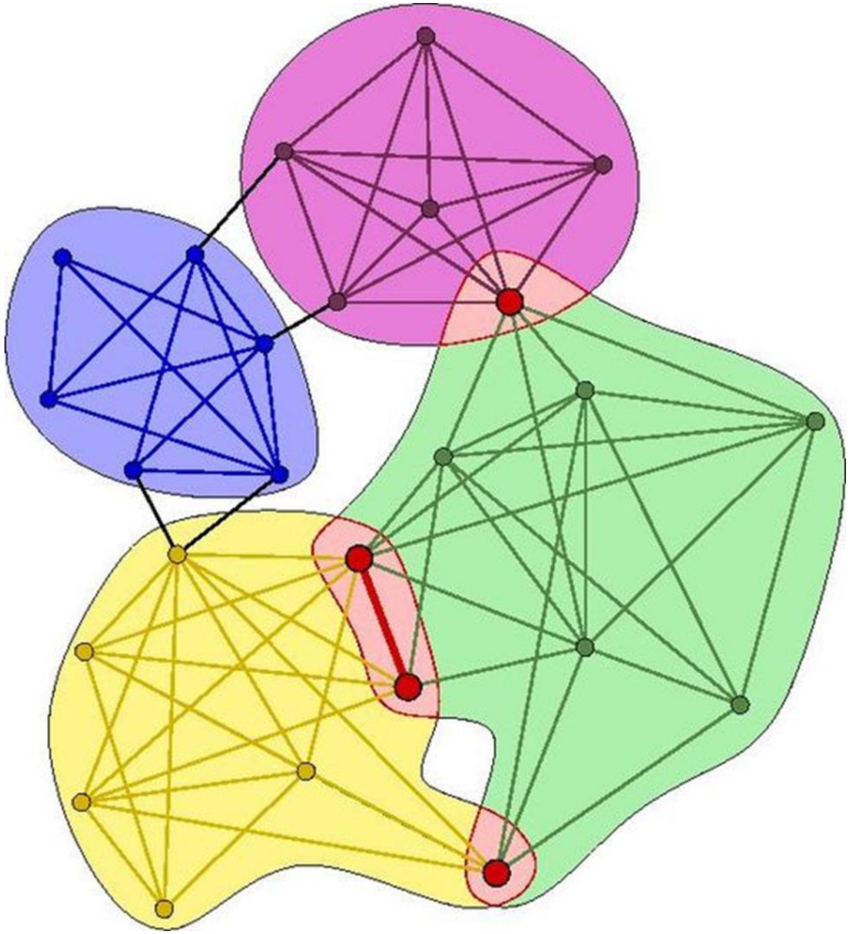


Примеры: что есть вершины / ребра?

ITIVITI

LET IT TRADE

- Социальные сети
  - выявление сообществ
  - анализ распространения информации
- Топологическая сортировка (графы зависимостей)
  - порядок компиляции и компоновки при сборке (GNU make)
  - порядок загрузки и инициализации разделяемых библиотек
  - граф вызовов



УНИВЕРСИТЕТ ИТМО

Неориентированные графы
Некоторые определения

LET IT TRADE

- Смежные вершины (соседи)
- Ребро инцидентно этим вершинам
- Степень $\deg(u)$ вершины u = число ребер, инцидентных этой вершине
- Лемма о рукопожатиях: $\sum \deg(u) = ?$

УНИВЕРСИТЕТ ИТМО

Неориентированные графы
Некоторые определения

LET IT TRADE

- Путь в графе – последовательность вершин, в которой каждая следующая вершина является смежной с предыдущей (путь из одной вершины?)
- Длиной пути называется количество составляющих его ребер (путь длиной ноль?)
- Путь называется простым, если ни одно ребро и ни одна вершина не посещается дважды (за исключением начала и конца)
- Циклом графа называется простой путь, у которого первая и последняя вершина одна и та же (запретили ходить по одному ребру туда-обратно)
- Граф называется ациклическим, если не содержит циклов

УНИВЕРСИТЕТ ИТМО

Связность в неориентированных графах

- Достижимость вершин
  - v достижима из u => u достижима из v (симметричность)
  - v достижима из u, w достижима из v => w достижима из u (транзитивность)
  - u достижима сама из себя?
- Компонент связности – максимальное множество вершин, где каждая вершина достижима из каждой (класс эквивалентности по отношению достижимости)
- Связный граф – один компонент связности
- Несвязный граф – множество компонент связности
  - если нет ребер – граф связный?

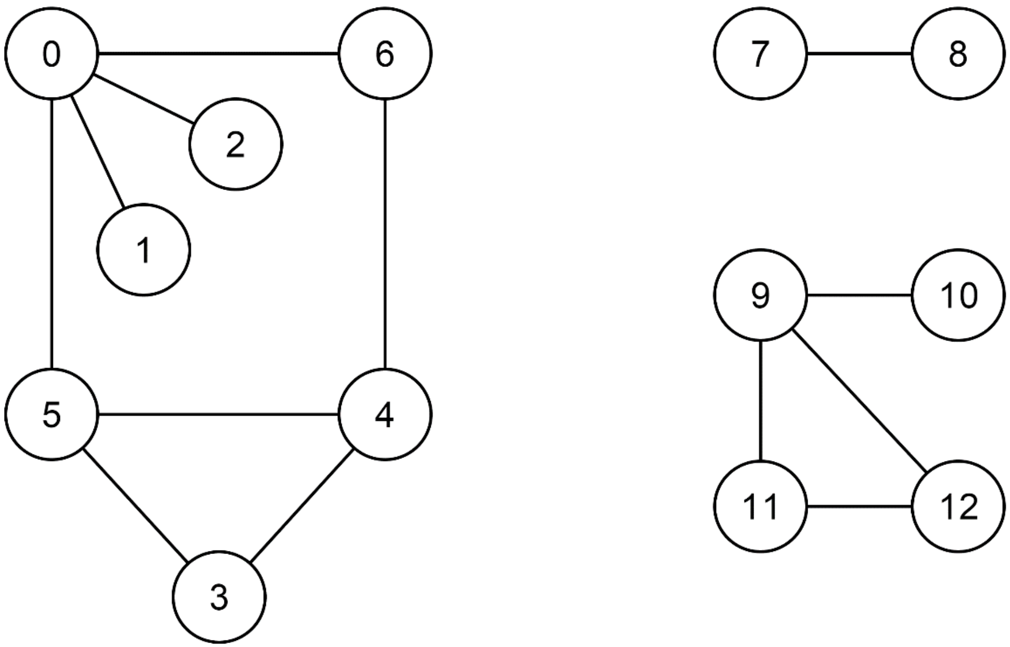
LET IT TRADE

ITIVITI

УНИВЕРСИТЕТ ИТМО

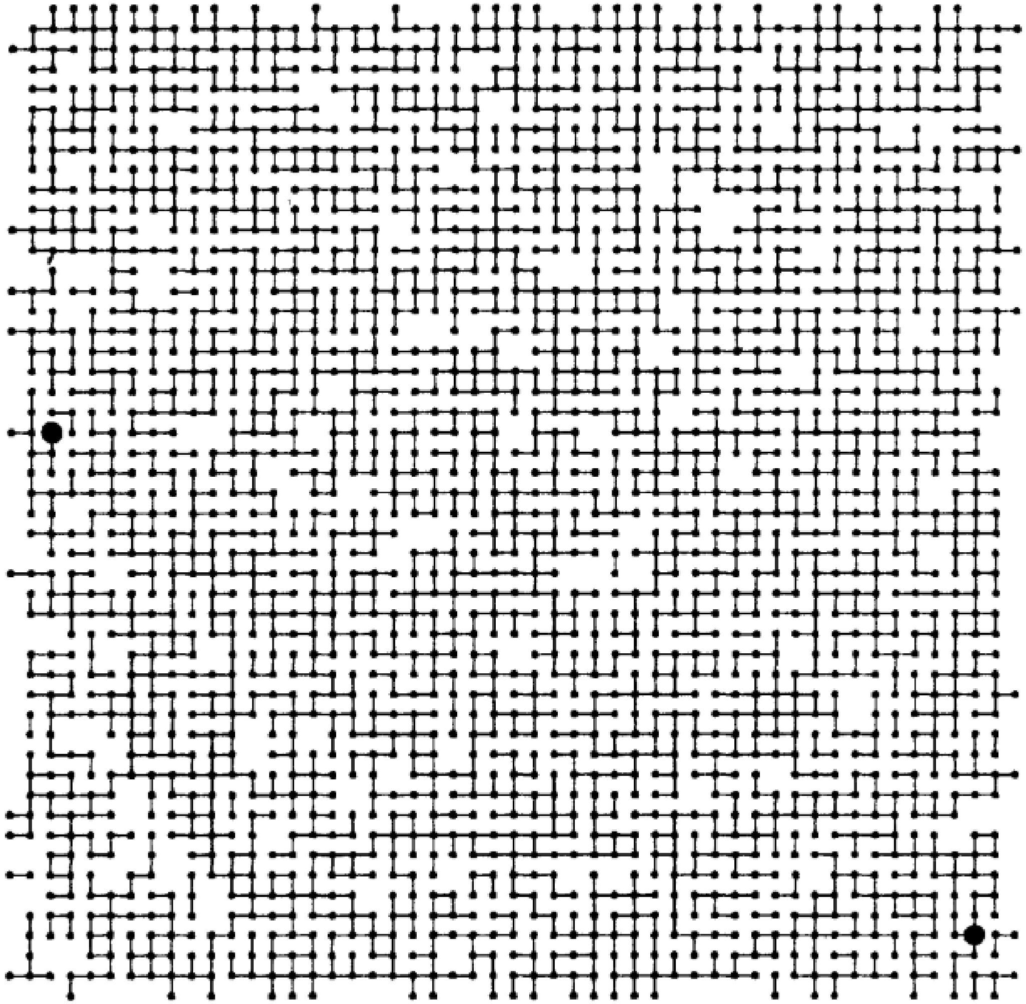
Неориентированные графы

- Сколько компонент связности? Какого размера?
- Вершина 10 достижима из вершины 3?
- Сколько циклов?



Неориентированные графы

- Сколько компонент связности? Какого размера?



Деревья и лес

- Деревом называется ациклический связный граф
- Лес – множество деревьев
- Пень – одна вершина, ребер нет
- Остовное дерево связного графа – подграф, который содержит все вершины и представляет собой единое дерево
- Остовный лес – подграф, который содержит все вершины и является лесом

Деревья и лес

- Граф с n вершинами есть дерево тогда и только тогда, когда он удовлетворяет одному из четырех условий:
- граф имеет $n - 1$ ребер и является связным (почему?)
- граф имеет $n - 1$ ребер и ни одного цикла
- ровно один простой путь соединяет каждую пару вершин в графе (почему?)
- граф является связным, но перестает быть таковым при удалении любого ребра

Ориентированные графы
Некоторые определения

ITIVITI

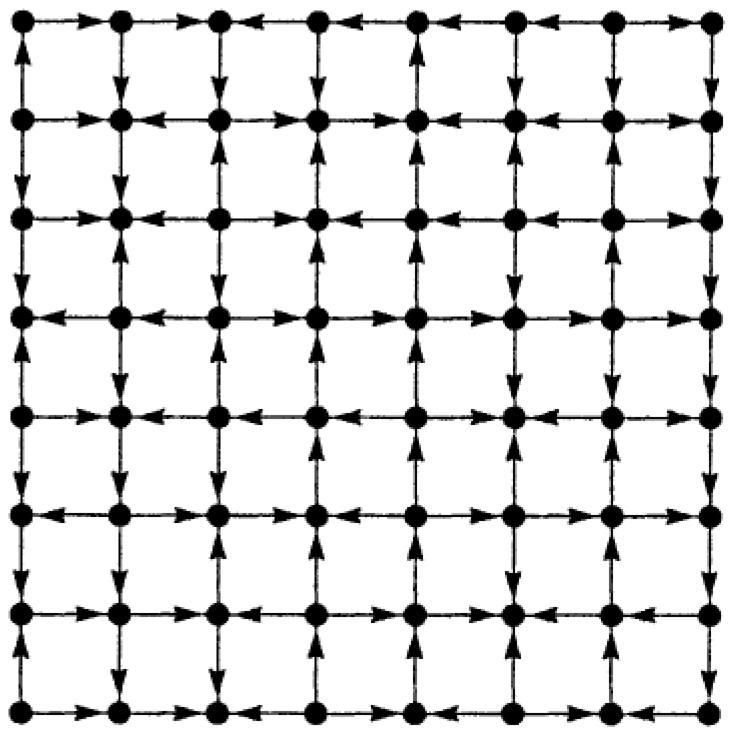
LET IT TRADE

- «Однонаправленная» смежность: из начала ребра в конец ребра (не означает, что есть обратный переход)
- Исходящая (полу)степень deg-(*u*) и входящая (полу)степень deg+(*u*) вершины *u*
- Петля (как влияет на степени вершины?)
- Лемма о рукопожатиях: ∑ deg-(u) ? ∑ deg+(u) = ?
- Ориентированный путь в орграфе – последовательность вершин, в которой каждая следующая вершина соединена ориентированным ребром с предыдущей (путь из одной вершины?)
- Направленный цикл (ациклический орграф – не дерево!)

УНИВЕРСИТЕТ ИТМО

Ориентированные графы

- Сколько циклов?
- Сколько компонент связности? Какого размера?

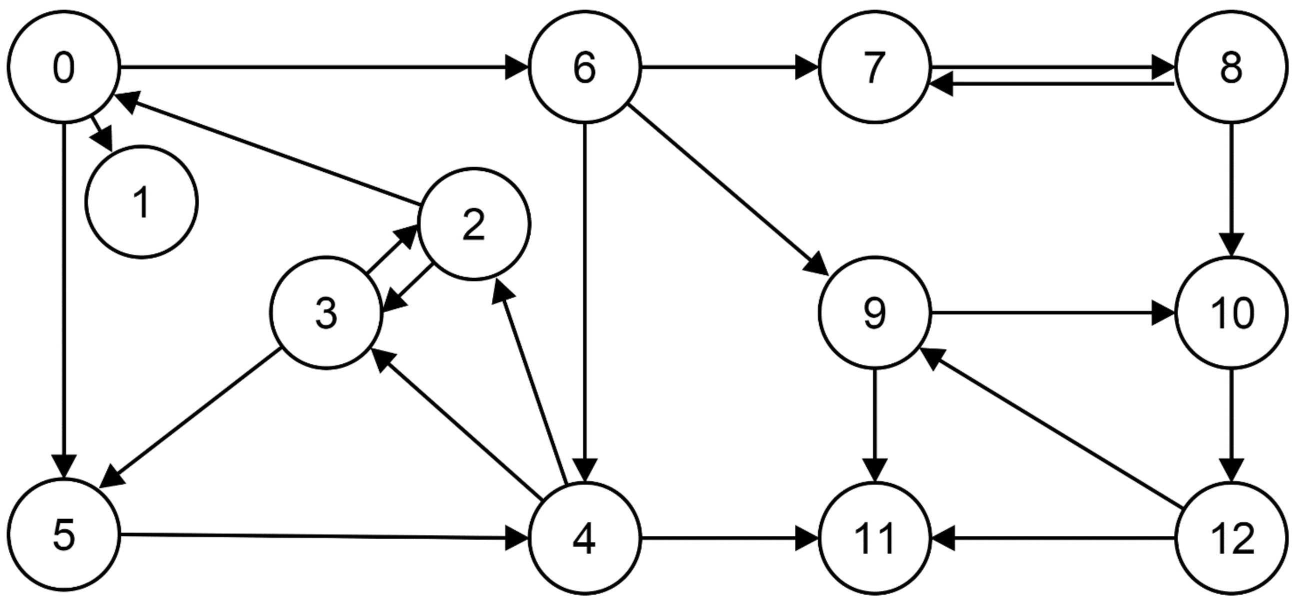


Связность в ориентированных графах

- Достижимость вершин в неориентированных графах
  - *v* достижима из *u* => *u* достижима из *v* (симметричность) – не выполняется для орграфов!
  - …
- В орграфах достижимые вершины не обязательно разбиваются на компоненты (могут пересекаться)
- Орграф называется сильно связанным, если каждая его вершина достижима из любой другой вершины
- Орграф содержит набор сильно связанных компонент и **набор ориентированных ребер**, идущих от одной компоненты к другой

Связность в ориентированных графах

- Сколько сильно связных компонент? Какого размера?



Алгоритмы на графах

- Как описать время работы алгоритма на графах?
  - Что такое размер задачи ?
  - Что лучше $O(m^2)$ или $O(n^3)$?
- Пусть неориентированный связный граф состоит из $n = |V|$ вершин. Какое минимальное и максимальное число ребер $m = |E|$ возможно в таком графе?
  - $(n-1)$ и n^2
  - n и n^n
  - n и 2^n
  - $(n-1)$ и $n(n-1)/2$
- Почему?

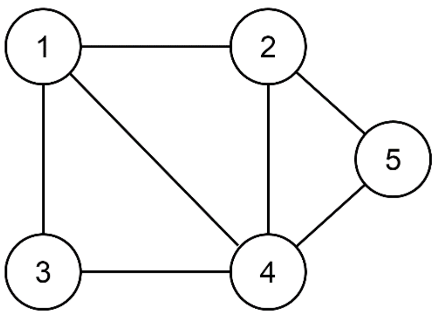
Представление графов в программе

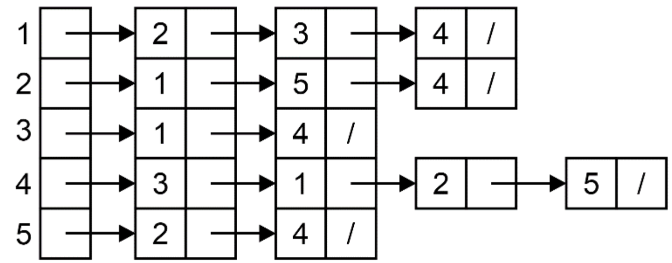
- Плотный (dense) граф: $m \sim n^2$
- Матрица смежности (число вершин невелико)
  - $A_{uv} = 1$, если между u и v существует ребро, иначе $A_{uv} = 0$
  - $A_{uv} = w$, где w – вес ребра
  - $A_{uv} = 1$, если ребро направлено от u к v
    $A_{uv} = -1$, если ребро направлено от v к u
- Какая структура у матрицы смежности неориентированного графа?

Представление графов в программе

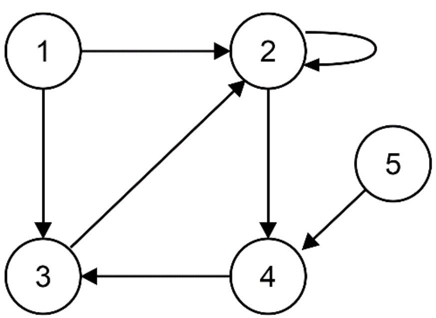
- Разреженный (sparse) граф: $m \sim n$
- Списки смежных вершин
  - Массив Adj из n списков (или мультисписок / мультистек: какого размера?)
  - Для каждой вершины u список Adj[u] содержит все смежные с u вершины
  - Вес w ребра (u, v) хранится вместе с v
- Произвольный порядок вершин в списке => порядок обработки вершин зависит от конкретного представления
- Орграфы: два списка для вершины (дети и родители)

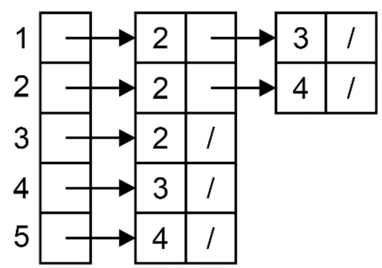
Представление графов в программе





	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	1
3	1	0	0	1	0
4	1	1	1	0	1
5	0	1	0	1	0





	1	2	3	4	5
1	0	1	1	0	0
2	0	1	0	1	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	0	1	0

- Что будет, если транспонировать матрицу смежности?

Представление графов в программе

LET IT TRADE

- Сколько памяти требуется для хранения матрицы смежности?
  - $\Theta(n)$
  - $\Theta(m)$
  - $\Theta(n + m)$
  - $\Theta(n^2)$
- Как быстро можно проверить наличие ребра между вершинами в матрице смежности?
  - $O(n)$
  - $O(m)$
  - $O(1)$
  - $O(n^2)$

УНИВЕРСИТЕТ ИТМО

Представление графов в программе

- Сколько памяти требуется для списков смежности?
  - $\Theta(n)$
  - $\Theta(m)$
  - $\Theta(n+m)$
  - $\Theta(n^2)$
- Как быстро можно проверить наличие ребра между вершинами в списках смежности?
  - $O(n)$
  - $O(m)$
  - $O(1)$
  - $O(n^2)$

Обход графов в ширину и в глубину

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Поиск на графе

- Поиск путей и изучение структуры графа по мере продвижения по нему
  - Поиск связных компонент
  - Поиск двухсвязных компонент
  - Поиск циклов
  - Поиск остовных деревьев
  - Поиск кратчайших путей
  - и т.д.

Поиск на графе

- Примеры использования:
  - Проверка связности компьютерной / телефонной сети (можем ли мы связаться с любой точкой из любой точки)
  - Нахождение числа Исенбаева (linkedin и т.п.)
  - Поиск выхода из лабиринта, оптимального маршрута, достижимости в транспортных сетях
  - Поиск последовательности решений, приводящей из начального состояния в желаемое конечное (например, при решении головоломки)
  - Кластеризация (ставим ребро только если объекты «сильно похожи»)
  - и т.д.

Поиск на графе: общий алгоритм

- Дано:
  - граф G
  - стартовая вершина s
- Задача:
  - найти все вершины, достижимые из заданной стартовой вершины s, т.е. те, для которых есть (направленный) путь из s
  - не посещать никакую вершину дважды – время $O(n+m)$ при представлении графа в виде списков смежности

Поиск на графе: общий алгоритм

отметить s , как посещенную, остальные вершины еще не посетили;

while (true)

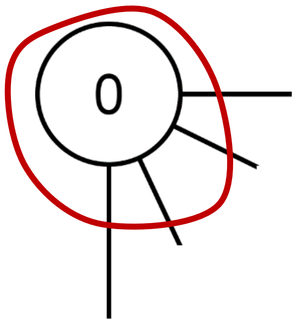
{

выбрать ребро (u, v), в котором вершина u из числа посещенных, v – еще не посещали, если такого ребра нет, прекратить выполнение;

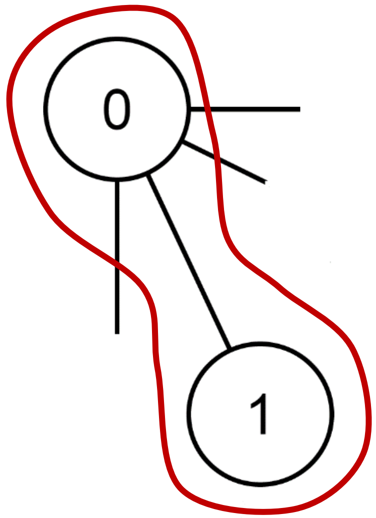
отметить вершину v , как посещенную;

}

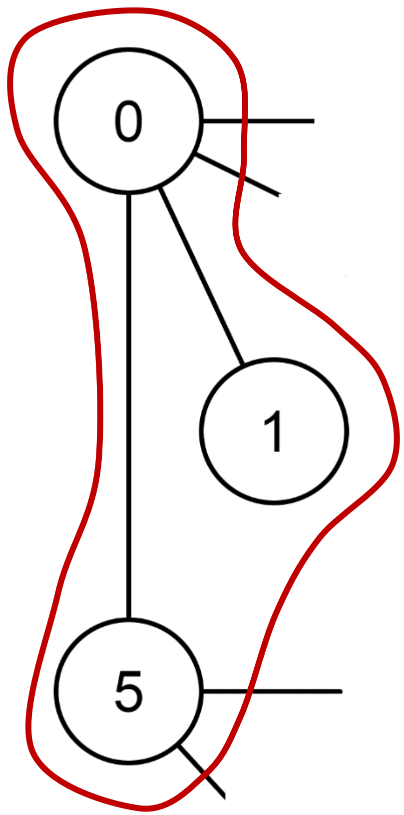
Поиск на графе: общий алгоритм



Поиск на графе: общий алгоритм



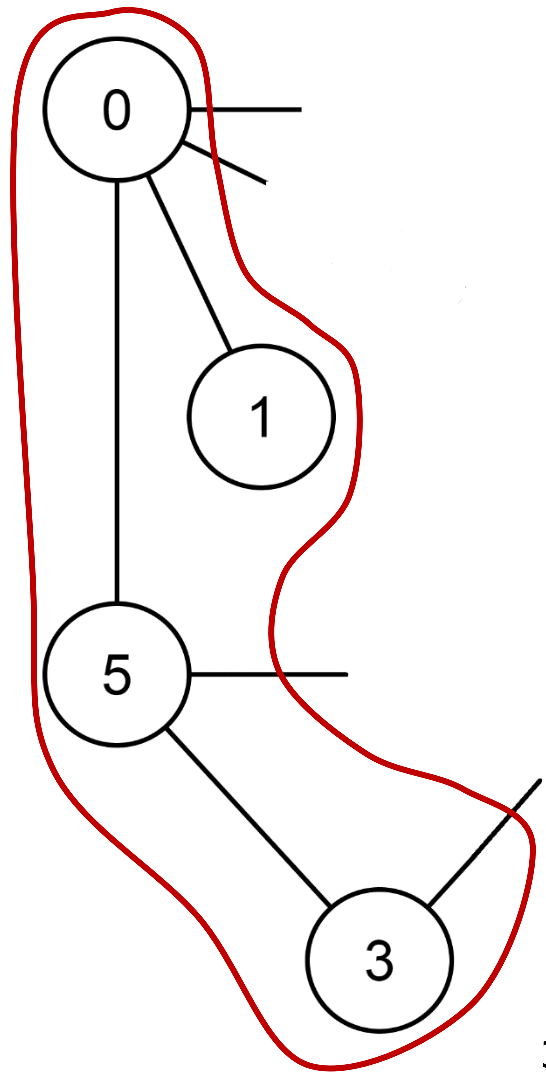
Поиск на графе: общий алгоритм



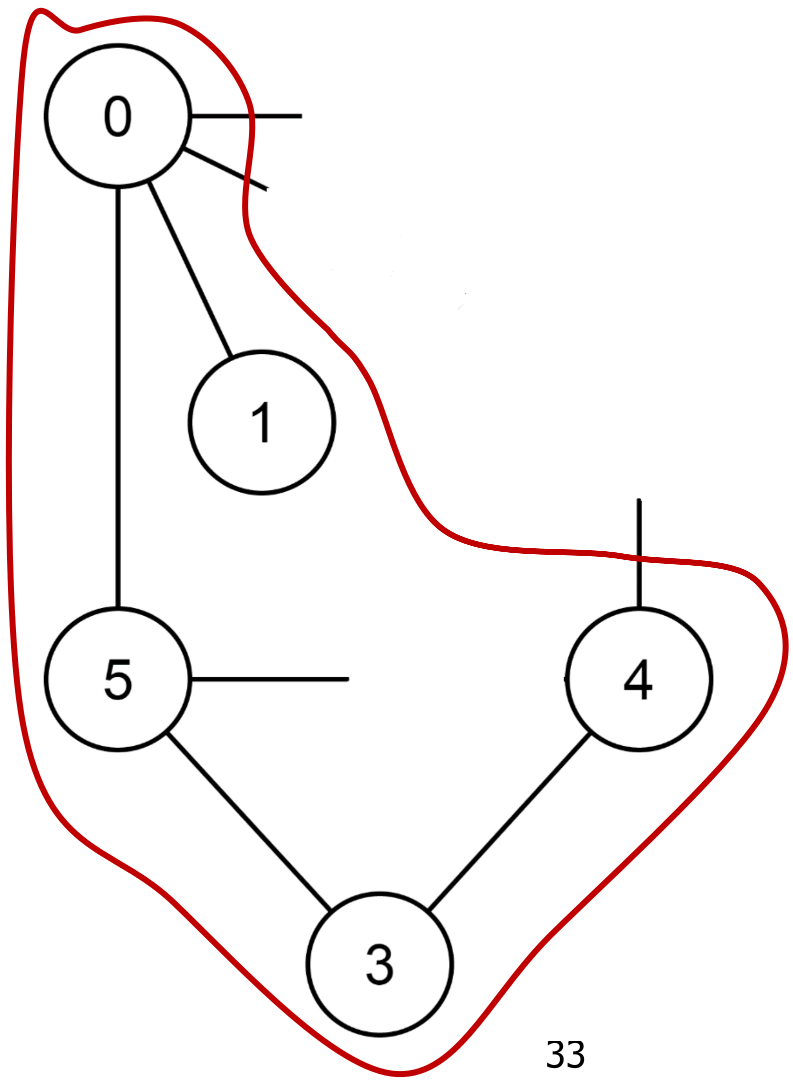
УНИВЕРСИТЕТ ИТМО

LET IT TRADE

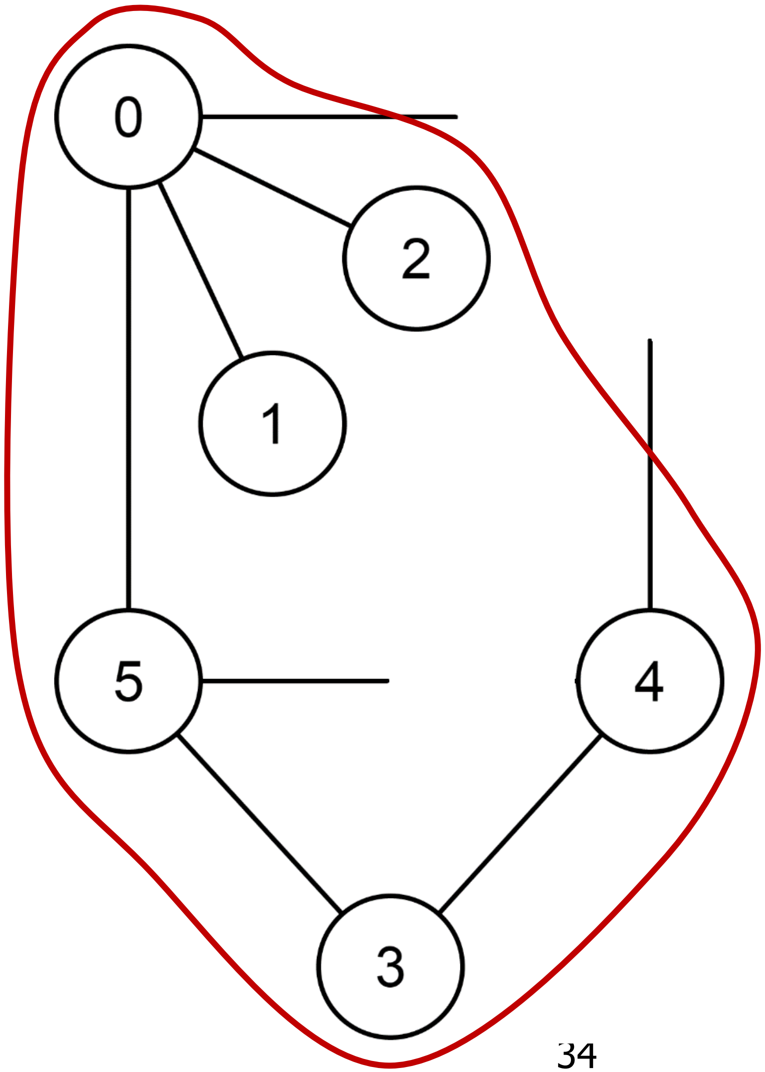
Поиск на графе: общий алгоритм



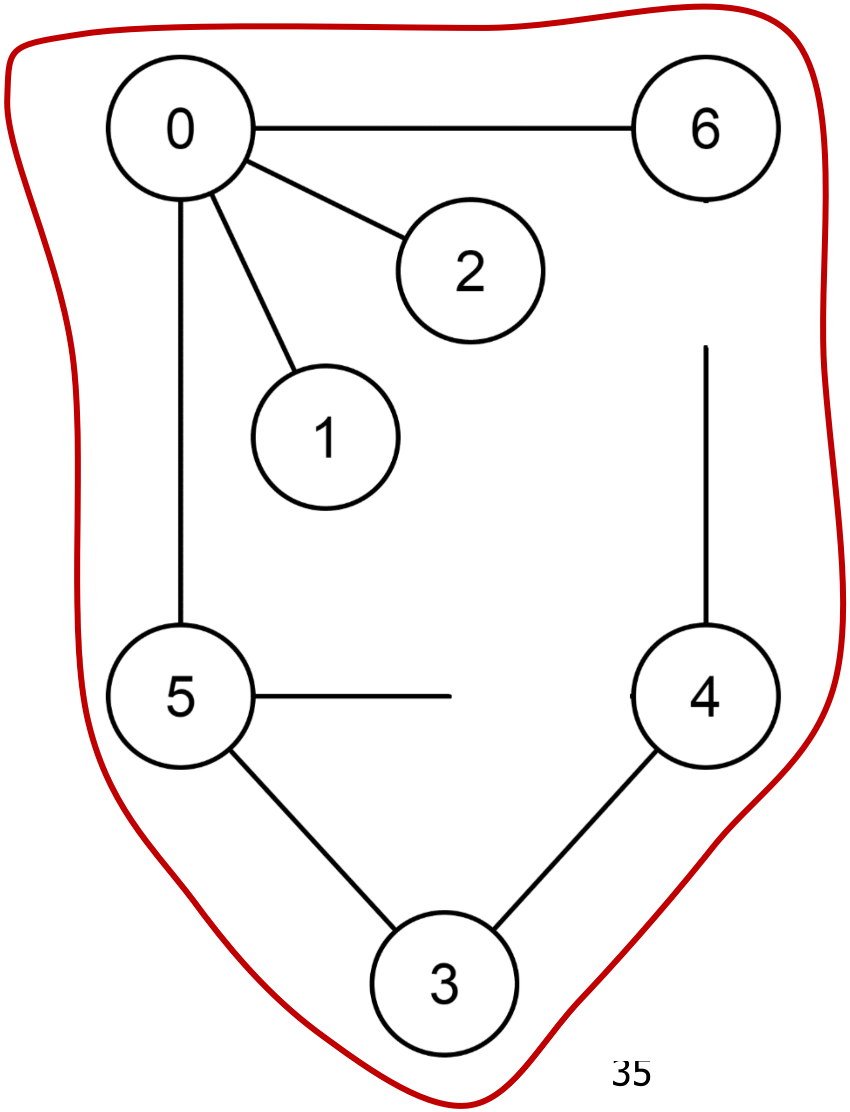
Поиск на графе: общий алгоритм



Поиск на графе: общий алгоритм



Поиск на графе: общий алгоритм



Поиск на графе: общий алгоритм

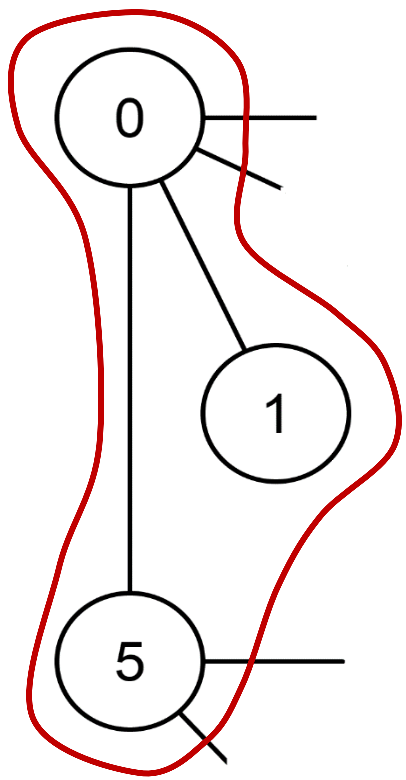
Почему любая вершина v будет отмечена как посещенная тогда и только тогда когда существует путь из s в v?

… по правилу останова алгоритма

Т.о. найдем, все что можно найти, и ничего не пропустим!

BFS vs DFS

Какое ребро из совокупности ребер, пересекающих фронт, выбрать следующим?



LET IT TRADE

ITIVITI

УНИВЕРСИТЕТ ИТМО

BFS

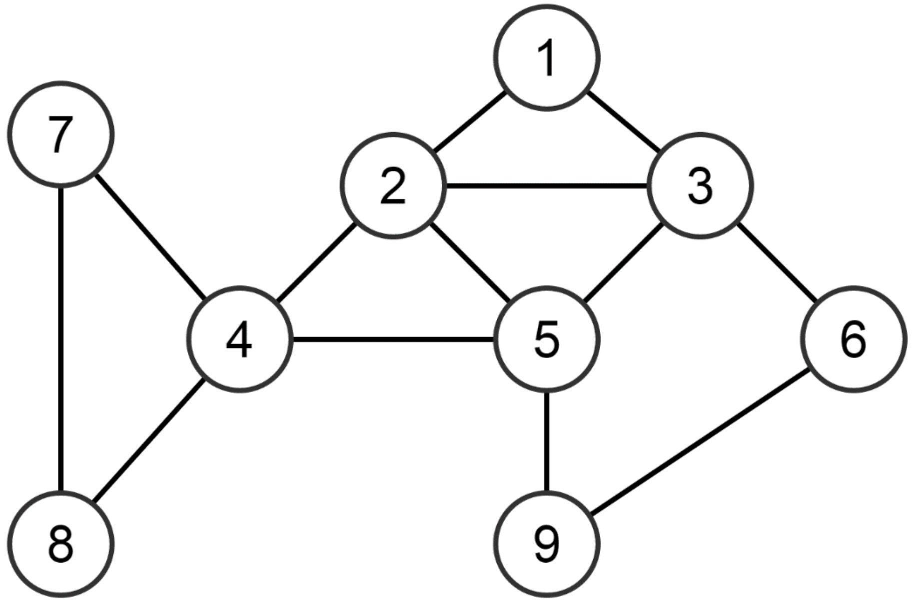
- Посещаем вершины по «уровням»
- Уровень L_0 – вершина s
- Уровень L_1 – все соседи s
- Уровень L_{k+1} – вершины, не принадлежащие предыдущим уровням и смежные с вершинами из L_k
  - С вершинами каких уровней могут быть смежными вершины из L_k ?

BFS: реализация

```
typedef std::vector<int> EdgeVec;
typedef std::vector<EdgeVec> Graph;

// Assumes nodes are numbered starting from zero
void BFS(const Graph & g, int s)
{
    std::vector<int> visited(g.size()); visited[s] = 1;
    Queue queue; queue.enqueue(s);
    while (!queue.empty()) {
        int u = queue.dequeue();
        for (int i = 0; i < g[u].size(); ++i) {
            int v = g[u][i];
            if (visited[v] == 0) {
                visited[v] = 1; queue.enqueue(v);
            }
        }
    }
}
```

BFS: пример процесса поиска и изменения очереди



- Если $u \in L_i$ и $v \in Lj$ и в графе G существует ребро (u, v), то как соотносятся i и j?
- Какие вершины находятся в очереди queue?
- Как формировать уровни L_k?

BFS: кратчайшие пути

LET IT TRADE

```
class Node {
public:
    Node() : d(-1), predecessor(-1), visited(false) { /* */ }
    int d;
    int predecessor;
    bool visited;
};

typedef std::vector<int> EdgeVec;
typedef std::vector<EdgeVec> Graph;
typedef std::vector<Node> VisitedMap;

```


УНИВЕРСИТЕТ ИТМО

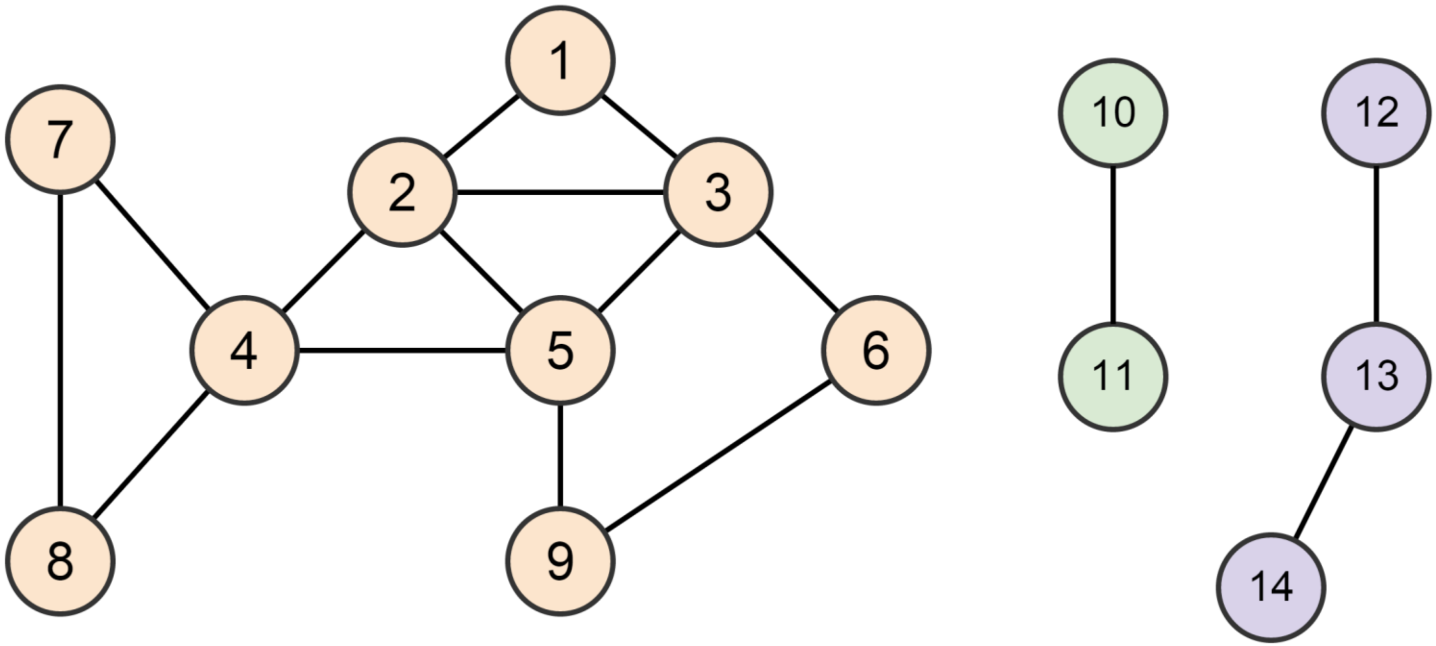
BFS: кратчайшие пути


```
// Assumes nodes are numbered starting from zero
void BFS(const Graph & g, int s, VisitedMap & vmap)
{
    vmap[s].d = 0; vmap[s].predecessor = s; vmap[s].visited = true;
    Queue queue; queue.enqueue(s);
    while (!queue.empty()) {
        int u = queue.dequeue();
        for (int i = 0; i < g[u].size(); ++i) {
            int v = g[u][i];
            if (!vmap[v].visited) {
                vmap[v].d = vmap[u].d + 1; vmap[v].predecessor = u;
                vmap[v].visited = true; queue.enqueue(v);
            }
        }
    }
}
```

Основные свойства BFS

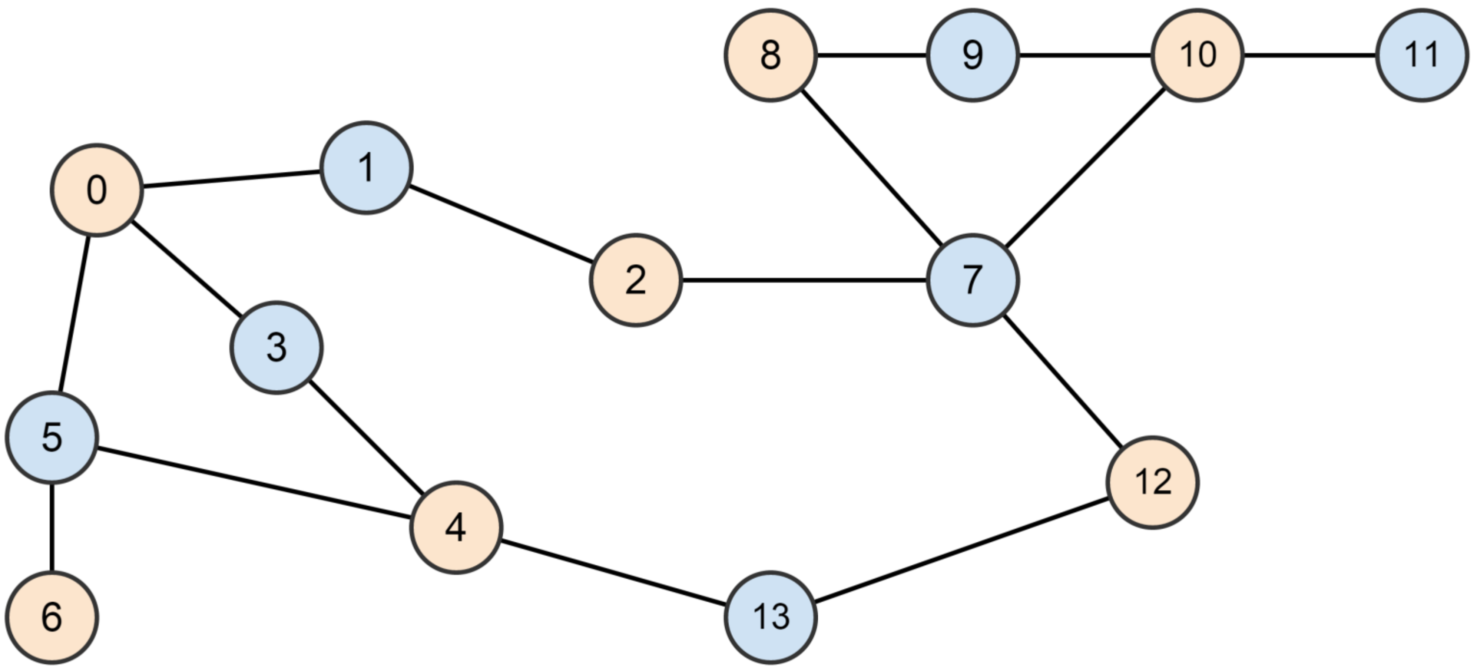
- **Свойство 1**: после выполнения алгоритма, вершина v будет отмечена, как посещенная, тогда и только тогда, когда существует путь от s к v
- **Свойство 2:** время выполнения $O(n_s + m_s)$
- **Свойство 3:** BFS позволяет найти кратчайшие пути от s ко всем достижимым из s вершинам
- **Свойство 4:** BFS позволяет найти связные компоненты неориентированного графа

BFS: поиск компонент связности



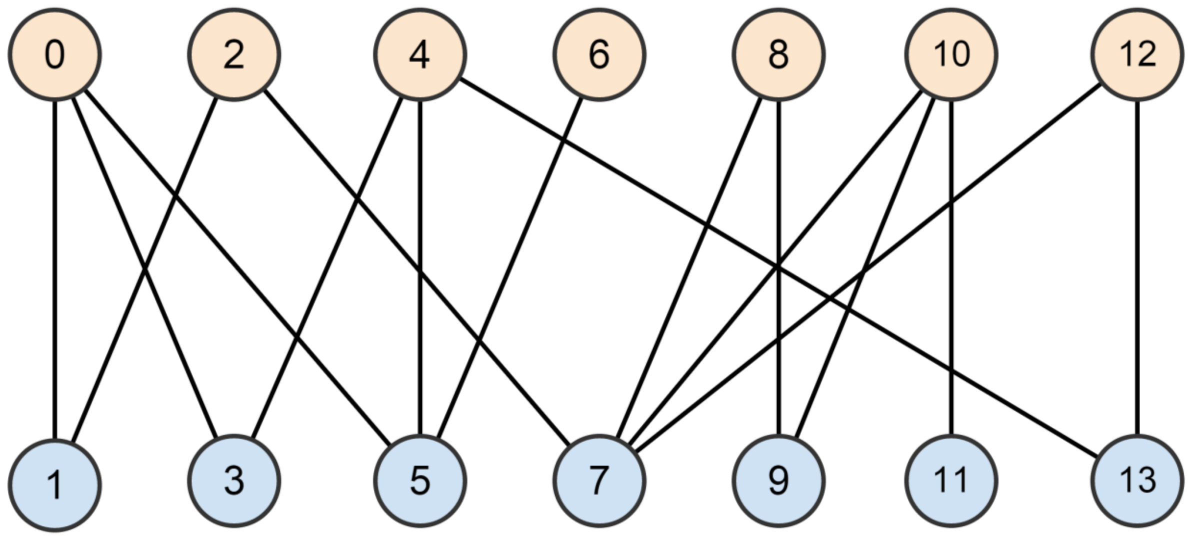
- Как найти все компоненты связности с помощью BFS?
- Какое будет время выполнения?

BFS: проверка на двудольность



- Может ли граф, содержащий цикл нечетной длины, быть двудольным?
- Цикл нечетной длины – единственное препятствие для двудольности?

BFS: проверка на двудольность



- Как проверить на двудольность с помощью BFS?
  - Чередуем раскраску уровней
  - Проверяем есть ли ребро с вершинами одного цвета
  - Если да, значит у нас есть цикл нечетной длины! Почему?

LET IT TRADE

Спасибо за внимание!

УНИВЕРСИТЕТ ИТМО