Графы и обход в глубину (DFS)

Denis Bakin

Что такое граф?

Графом называется пара множеств $G = (V, E)$, где:

- V — множество **вершин** (vertices)
- $E \subseteq V \times V$ — множество **рёбер** (edges), пар вершин (u, v)

Что такое граф?

Графом называется пара множеств $G = (V, E)$, где:

- V — множество **вершин** (vertices)
- $E \subseteq V \times V$ — множество **рёбер** (edges), пар вершин (u, v)

Пример: $V = \{1, 2, 3, 4\}$, $E = \{(1, 2), (1, 3), (1, 4), (2, 4), (3, 4)\}$

Зачем нужны графы?

Графы моделируют **связи между объектами**:

Предметная область	Вершины	Рёбра
Социальная сеть	пользователи	дружба
Карта города	перекрёстки	дороги
Интернет	сайты	гиперссылки
Молекула	атомы	химические связи
Расписание	задачи	зависимости

Ориентированные и неориентированные графы

Неориентированный граф — граф, в котором рёбра не имеют направления: $(u, v) = (v, u)$

Ориентированный граф (орграф) — граф, в котором каждое ребро имеет направление: $(u, v) \neq (v, u)$

Примеры:

- Подписки в Twitter — ориентированный (A подписан на B $\not\Rightarrow$ B подписан на A)
- Дружба в VK — неориентированный (дружба взаимна)

Терминология: смежность и инцидентность

- Вершины u и v называются **смежными**, если существует ребро $(u, v) \in E$
- Ребро $e = (u, v)$ называется **инцидентным** вершинам u и v
- **Степенью** вершины v (обозначается $\deg(v)$) называется количество инцидентных ей рёбер

Для ориентированных графов различают:

- **Исходящая степень** $\deg^+(v)$ — количество рёбер, выходящих из v
- **Входящая степень** $\deg^-(v)$ — количество рёбер, входящих в v

Определения

- **Петля** — ребро, соединяющее вершину саму с собой: (v, v)
- **Кратные рёбра** — несколько рёбер между одной и той же парой вершин
- **Простой граф** — граф без петель и кратных рёбер
- **Мультиграф** — граф, допускающий кратные рёбра

Терминология: пути

- **Путь** из v_0 в v_k — последовательность вершин $(v_0, v_1, \dots, v_k)$, где каждые две соседние вершины соединены ребром: $(v_i, v_{i+1}) \in E$
- **Длина пути** — количество рёбер в пути (равно k)
- **Простой путь** — путь без повторяющихся вершин

Терминология: пути

- **Путь** из v_0 в v_k — последовательность вершин $(v_0, v_1, \dots, v_k)$, где каждые две соседние вершины соединены ребром: $(v_i, v_{i+1}) \in E$
- **Длина пути** — количество рёбер в пути (равно k)
- **Простой путь** — путь без повторяющихся вершин

Вершина u **достижима** из v, если существует путь из v в u

Терминология: связность

- Неориентированный граф называется **связным**, если из любой вершины достижима любая другая
- **Компонента связности** — максимальное по включению подмножество вершин, в котором любые две вершины достижимы друг из друга

Терминология: связность

- Неориентированный граф называется **связным**, если из любой вершины достижима любая другая
- **Компонента связности** — максимальное по включению подмножество вершин, в котором любые две вершины достижимы друг из друга

Для ориентированных графов:

- **Слабо связный** — связный, если забыть про направления рёбер
- **Сильно связный** — из любой вершины достижима любая другая (с учётом направлений)

Терминология: циклы

- **Цикл** — путь $(v_0, v_1, \dots, v_k)$, где $v_0 = v_k$ (начальная и конечная вершины совпадают)
- **Простой цикл** — цикл, в котором все вершины, кроме первой и последней, различны
- **Ацикличный граф** — граф, не содержащий простых циклов

Деревья

Дерево — связный ацикличный неориентированный граф

Деревья

Дерево — связный ацикличный неориентированный граф

Эквивалентные определения (для графа с n вершинами):

- Связный граф с $(n-1)$ ребром
- Ацикличный граф с $(n-1)$ ребром
- Граф, в котором между любыми двумя вершинами существует ровно один простой путь

Лес — ацикличный граф (не обязательно связный), т.е. набор деревьев

Хранение графов: обзор

Вершины нумеруются от 0 до $(n-1)$

Три основных способа:

Способ	Память	Проверка ребра	Соседи вершины
Список рёбер	$O(m)$	$O(m)$	$O(m)$
Матрица смежности	$O(n^2)$	$O(1)$	$O(n)$
Список смежности	$O(n+m)$	$O(\deg)$	$O(\deg)$

где $n = |V|$, $m = |E|$, $\deg$ — степень вершины

Способ 1: Список рёбер

Список рёбер — представление графа в виде массива пар (u, v) для каждого ребра

```
std::vector<std::pair<int, int>> edges;
// или
struct Edge { int u, v; };
std::vector<Edge> edges;
```

Способ 2: Матрица смежности

Матрица смежности — квадратная матрица A размера $n \times n$, где

$$A[i][j] = \begin{cases} 1, & \text{если } (i,j) \in E \\ 0, & \text{иначе} \end{cases}$$

```
std::vector<std::vector<int>> A(n, std::vector<int>(n, 0));
```

Способ 2: Матрица смежности

Матрица смежности — квадратная матрица A размера $n \times n$, где

$$A[i][j] = \begin{cases} 1, & \text{если } (i,j) \in E \\ 0, & \text{иначе} \end{cases}$$

```
std::vector<std::vector<int>> A(n, std::vector<int>(n, 0));
```

Свойство: для неориентированного графа матрица симметрична: $A[i][j] = A[j][i]$

Матрица смежности: особенности

- Проверка ребра за $O(1)$: `if (A[u][v])`
- $O(n^2)$ памяти — даже для разреженных графов
- Перебор соседей за $O(n)$

Способ 3: Список смежности

Список смежности — представление графа, где для каждой вершины v хранится список всех смежных с ней вершин

```
std::vector<std::vector<int>> adj(n);
// adj[v] = {u : (v, u) ∈ E}
```

Способ 3: Список смежности

Список смежности — представление графа, где для каждой вершины v хранится список всех смежных с ней вершин

```
std::vector<std::vector<int>> adj(n);
// adj[v] = {u : (v, u) ∈ E}
```

- $O(n + m)$ памяти, перебор соседей за $O(\deg(v))$
- проверка ребра за $O(\deg)$

Список смежности: добавление рёбер

Неориентированный граф:

```
void add_edge(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);  // обратное ребро
}
```

Ориентированный граф:

```
void add_edge(int u, int v) {
    adj[u].push_back(v);  // только одно направление
}
```

Обход в глубину: определение

Поиск в глубину (DFS, Depth-First Search) — алгоритм обхода графа, который:

1. Начинает с заданной стартовой вершины
2. Посещает вершину и рекурсивно обходит всех непосещённых соседей
3. Возвращается, когда все соседи посещены

DFS: базовая реализация

```cpp
std::vector<std::vector<int>> adj;  // граф
std::vector<bool> visited;           // посещена ли вершина

void dfs(int v) {
    visited[v] = true;

    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs(u);
        }
    }
}
```

Времена входа и выхода: определение

При обходе DFS для каждой вершины v определяются:

- $tin[v]$ — **время входа** — момент, когда алгоритм впервые посетил v
- $tout[v]$ — **время выхода** — момент, когда алгоритм полностью обработал v и всё её поддерево

Времена входа и выхода: определение

При обходе DFS для каждой вершины v определяются:

- $tin[v]$ — **время входа** — момент, когда алгоритм впервые посетил v
- $tout[v]$ — **время выхода** — момент, когда алгоритм полностью обработал v и всё её поддерево

Времена измеряются глобальным счётчиком (таймером), который увеличивается при каждом входе в новую вершину

tin и tout: реализация

```
std::vector<int> tin, tout;
int timer = 0;

void dfs(int v) {
    tin[v] = timer++;   // фиксируем время входа
    visited[v] = true;

    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs(u);
        }
    }

    tout[v] = timer;    // фиксируем время выхода
}
```

tin и tout: пример

Вершина	tin	tout	Интерпретация
0	0	5	корень, обработка всего дерева
1	1	4	поддерево с 3 вершинами
2	4	5	лист, обработан последним
3	2	3	лист внутри поддерева 1
4	3	4	лист внутри поддерева 1

Полуинтервал $[tin_v, tout_v)$ — “время жизни” вершины в стеке рекурсии

Свойство 1: проверка предка

Вершина u является предком вершины v в дереве DFS тогда и только тогда, когда

$$tin[u] \leq tin[v] \text{ и } tout[v] \leq tout[u]$$

Свойство 2: вложенность интервалов

Для любых двух вершин u и v полуинтервалы $[tin_u, tout_u)$ и $[tin_v, tout_v)$:

- либо **не пересекаются** (вершины в разных поддеревьях)
- либо **один вложен в другой** (одна вершина — предок другой)

Свойство 3: размер поддерева

Размер поддерева вершины v (количество вершин, включая саму v):

$$size(v) = tout[v] - tin[v]$$

Применения DFS

- **Проверка связности** — запустить DFS, проверить все ли вершины посещены
- **Поиск компонент связности** — запускать DFS из непосещённых вершин
- **Топологическая сортировка** — порядок $tout$ в обратном порядке
- **Поиск циклов** — если встретили посещённую вершину (не родителя)
- **Мосты и точки сочленения** — с использованием tin и fup

Сложность DFS

Временная сложность DFS: $O(n + m)$

Пространственная сложность: $O(n)$

Итоги

- **Граф** $G = (V, E)$ — модель связей между объектами
- **Хранение:** список смежности — универсальный выбор, $O(n + m)$
- **DFS** — рекурсивный обход графа “вглубь”, сложность $O(n + m)$
- **tin/tout** — мощный инструмент:
  - проверка “предок ли u для v” за $O(1)$
  - размер поддерева за $O(1)$
  - сведение запросов на поддереве к запросам на отрезке