

Лекция 01. DFS и его производные для ориентированных графов.

2 февраля 2026 г.

Автор конспекта: Гридчин Михаил

Понятие графа

Рассмотрим пару (V, E) , где V — множество вершин (здесь и далее $|V| < \infty$), а $E \subseteq V \times V$ — мульти множество рёбер.

Определение. Петлёй называют такое ребро $e \in E$, что $e = (v, v)$

Определение. Рёбра e_1 и e_2 называются кратными, если $e_1 = (u, v)$ и $e_2 = (u, v)$.

Определение. Пара (V, E) называется *графом*, если:

- $\forall v \in V \rightarrow e = (v, v) \notin E$, то есть нет петель
- $\forall u, v \in V \rightarrow |(u, v) \cap E| \leq 1$, то есть нет кратных рёбер
- $\forall u, v \in V : (u, v) \in E \Leftrightarrow (v, u) \in E$, то есть при наличии ребра в одну сторону обязательно есть ребро в другую

Определение. Пара (V, E) называется *орграфом* или *ориентированным графом*, если:

- нет петель
- нет кратных рёбер

Определение. Пара (V, E) называется *псевдографом*, если:

- нет кратных рёбер
- при наличии ребра в одну сторону обязательно есть ребро в другую

Определение. Пара (V, E) называется *мультиграфом*, если:

- нет петель
- при наличии ребра в одну сторону обязательно есть ребро в другую

Способы хранения графа

Рассмотрим следующие два способа хранения графа.

1. Матрица смежности $G[u][v] = I((u, v) \in E)$, то есть бинарная матрица, хранящая на пересечении u -й строки v -го столбца наличие ребра $u \rightarrow v$.

2. Список смежности. $G[u]$ — массив всех соседей вершины u .

Сравнение способов хранения.

Операция	Список смежности	Матрица смежности
$(u, v) \in E$	$O(\deg u)$	$O(1)$
<code>remove</code> $(u, v) \in E$	$O(\deg u)$	$O(1)$
Получить список соседей	$O(1)$	$O(V)$
Потребляемая память	$O(V + E)$	$O(V ^2)$

Алгоритм DFS

Определение. *DFS*, или *depth first search*, или *обход в глубину* — такой обход, при котором после входа в вершину v рекурсивно посещаются все ещё не посещённые соседи v до завершения обработки v .

```
void DFS(Graph<Vertex, Edge> graph, Vertex vertex, HashMap<Vertex, bool>
visited) {
    visited[vertex] = true;
    for (Vertex neighbour : graph.GetNeighbours(vertex)) {
        if (visited[neighbour]) {
            continue;
        }
        DFS(graph, neighbour, visited);
    }
}
```

Определение. Цветом вершины в ходе DFS назовём одно из трёх состояний:

- *Белый* цвет — вершина ещё не была посещена.
- *Серый* цвет — вершина была посещена, но рекурсия ещё не вышла из неё.
- *Чёрный* цвет — рекурсия вышла из вершины.

Определение. Временем входа или $t_{\text{in}}(v)$ назовём момент времени, когда вершина v была впервые посещена.

Определение. Временем выхода $t_{\text{out}}(v)$ назовём момент времени, когда вершина v была покрашена в чёрный.

Замечание. Время увеличивается каждый раз, когда мы посещаем какую-то вершину или же перекрашиваем вершину в новый цвет.

Лемма (о белых путях)

Пусть в процессе выполнения DFS на графе $G = (V, E)$ в момент времени $t_{\text{in}}(v)$ вершина v становится серой. Тогда для любой вершины u , достижимой из v по пути, состоящему исключительно из белых вершин (включая u , но исключая v) выполняется:

$$t_{\text{in}}(v) < t_{\text{in}}(u) < t_{\text{out}}(u) < t_{\text{out}}(v)$$

То есть u станет чёрной до завершения обработки v .

□

Проведём доказательство индукцией по длине белого пути $P = (v = v_0, v_1, \dots, v_k = u)$.

База ($k = 0$). Если $k = 0$, то $u = v$. Утверждение тривиально верно, так как $t_{\text{in}}(v) < t_{\text{out}}(v)$ по определению времён входа/выхода.

Шаг ($k \geq 1$). Пусть утверждение верно для всех белых путей длины $< k$. Рассмотрим белый путь $P = (v = v_0, v_1, \dots, v_k = u)$ длины k в момент $t_{\text{in}}(v)$. Заметим, что вершина v_1 белая в момент времени $t_{\text{in}}(v)$ (по определению белого пути все вершины v_1, \dots, v_k белые в момент $t_{\text{in}}(v)$). Также DFS посетит v_1 из v (при обработке вершины v алгоритм перебирает всех её соседей, поскольку v_1 белая при рассмотрении ребра (v, v_1) , будет выполнен рекурсивный вызов $\text{DFS}(v_1)$). Следовательно:

$$t_{\text{in}}(v) < t_{\text{in}}(v_1) < t_{\text{out}}(v_1) < t_{\text{out}}(v) \quad (1)$$

Последнее неравенство верно, потому что рекурсивный вызов $\text{DFS}(v_1)$ завершается до завершения обработки v . Заметим, что к моменту $t_{\text{in}}(v_1)$ единственная вершина из P , изменившая цвет с белого — это v (стала серой). Значит, все вершины из пути $P' = (v_1, \dots, v_k)$ остаются белыми. Путь P' имеет длину $k - 1$. По предположению индукции:

$$t_{\text{in}}(v_1) < t_{\text{in}}(u) < t_{\text{out}}(u) < t_{\text{out}}(v_1) \quad (2)$$

Комбинируя (1) и (2), получаем цепочку:

$$t_{\text{in}}(v) < t_{\text{in}}(v_1) < t_{\text{in}}(u) < t_{\text{out}}(u) < t_{\text{out}}(v_1) < t_{\text{out}}(v)$$

Что и требовалось доказать.

■

Утверждение

В графе есть цикл, достижимый из s , тогда и только тогда, когда DFS из вершины s нашёл ребро в серую вершину.

Замечание. У нас нет цели найти все циклы, а лишь проверить, есть ли он.

Замечание. Если хранить предка вершины в порядке DFS, то можно отыскать сам цикл.



Стек рекурсии — путь из серых вершин, так как чёрные там быть не могут (они удаляются из стека), а белые при попадании красятся в серый. При этом если найдено ребро в серую, то это ребро куда-то в вершину выше по стеку рекурсии, вот и цикл (стек рекурсии хранит его).

Пусть C — цикл, достижимый из s . Пусть v — первая посещённая вершина C , тогда весь цикл белый. Откуда к моменту $t_{\text{out}}(v)$ весь цикл станет чёрным, в частности будет посещена вершина u , предшествующая v по циклу, до $t_{\text{out}}(v)$, а значит v была серой на тот момент, вот и нашли ребро в серую вершину.



Виды связности

Определение. Неориентированный граф $G = (V, E)$ **связен**, если $\forall u, v \in V$ существует путь между u и v .

Определение. Ориентированный граф $G = (V, E)$ **слабо связен**, если его неориентированная копия связна.

Определение. Ориентированный граф $G = (V, E)$ **сильно связен**, если $\forall u, v \in V$ существует как путь из u в v , так и из v в u .

Проверка на слабую связность

Ориентированный граф слабо связен, если из каждой его вершины достижима каждая другая **в его неориентированной копии**. Запустим DFS от произвольной вершины. По лемме о белых путях все вершины должны окраситься в чёрный к концу работы DFS.

```
bool IsWeaklyConnected(Graph<Vertex, Edge> graph) {
    HashMap<Vertex, bool> visited;
    DFS(graph.Undirected(), graph.GetAnyVertex(), visited);
    return visited.size() == graph.GetVerticesCount();
}
```

Топологическая сортировка

Определение. Рассмотрим ориентированный граф. Присвоим каждой вершине номер.

Перестановка σ называется **топологической сортировкой** графа, если $\forall(u, v) \in E$ выполняется, что $\sigma(u) < \sigma(v)$.

Критерий существования

Топологическая сортировка ориентированного графа существует тогда и только тогда, когда граф ациклический.



➡ Так как порядок "меньше" ацикличен, если граф не ациклический, то топологическая сортировка не существует.

⬅ Приведём алгоритм, который находит топологическую сортировку графа, и докажем его корректность.

Алгоритм.

1. Запустим DFS на всём графике.
2. В момент выхода DFS из вершины будем добавлять её в конец массива
3. Развернём полученный массив. Он содержит ответ.

Корректность. В DFS если u достижима из v (и v была посещена первой), то $t_{\text{out}}(v) > t_{\text{out}}(u)$. Таким образом, искомая топологическая сортировка — это сортировка в порядке убывания времён выхода.



Будем считать, что уже есть DFS, который позволяет обрабатывать произвольную логику, например, добавлять в конец массива рассматриваемую вершину.

```
Array<Vertex> GetTopologicalSort(Graph<Vertex, Edge> graph) {
    assert(graph.IsAcyclic());
    HashMap<Vertex, bool> visited;
    Array<Vertex> topsort;
    for (Vertex vertex : graph.GetVertices()) {
        if (!visited[vertex]) {
            DFS(graph, vertex, visited, topsort);
        }
    }
    return topsort.reversed();
}
```

Компоненты сильной связности

Определение. Две вершины $u, v \in V$ *сильно связаны* в орграфе G , если есть путь как из u в v , так и наоборот.

Замечание. Данное отношение является отношением эквивалентности.

Определение. *Компоненты сильной связности (КСС)* — классы эквивалентности по отношению сильной связности.

Определение. Графом конденсации называют граф, где все компоненты сильной связности скаты до одной вершины, а рёбра между ними получаются как рёбра между компонентами.

Утверждение. Граф конденсации ацикличен.

Лемма о временах выхода

Пусть C и C' — различные компоненты сильной связности орграфа G , и в графе конденсации существует ребро (C, C') . Тогда

$$\max_{v \in C} t_{\text{out}}(v) > \max_{v \in C'} t_{\text{out}}(v)$$

□

Рассмотрим первую по времени входа вершину среди $C \cup C'$. Возможны два случая:

1. Первой посещена вершина $v \in C$. При обработке v DFS обнаружит ребро в C' (поскольку $(C, C') \in E_{\text{cond}}$). Все вершины C' белые (иначе C и C' были бы в одной КСС), поэтому по лемме о белых путях весь C' будет посещён и завершён до выхода из v . Следовательно:

$$\max_{u \in C'} t_{\text{out}}(u) < t_{\text{out}}(v) \leq \max_{w \in C} t_{\text{out}}(w)$$

2. Первой посещена вершина $v' \in C'$. Если бы из C' существовал путь в C , то C и C' образовывали бы одну КСС — противоречие. Значит, при обходе из v' вершины C останутся белыми. Тогда весь C' завершится до начала обхода C , и:

$$\max_{u \in C'} t_{\text{out}}(u) < \min_{w \in C} t_{\text{in}}(w) \leq \max_{w \in C} t_{\text{out}}(w)$$

В обоих случаях $\max_C t_{\text{out}} > \max_{C'} t_{\text{out}}$.

■

Алгоритм Косарайю

Поиск компонент сильной связности (КСС):

1. Запустить DFS на исходном графе G , запоминая времена выхода $t_{\text{out}}(v)$.
2. Построить транспонированный граф G^T (все рёбра развернуть).
3. Запустить DFS на G^T , перебирая вершины в порядке убывания t_{out} из шага 1.
Каждое дерево, построенное в этой фазе, соответствует одной КСС.

```

HashMap<Vertex, int> Cosaraju(Graph graph) {
    Arra<Vertex> second_phase_starts; // ascending by t_out
    second_phase_starts = GetTopologicalSort(graph);
    Graph transponed = graph.GetTransponed();
    HashMap<Vertex, int> scc2color;
    int color = 0;
    while (scc2color.size() < graph.GetVerticesCount()) {
        // inserts all visited verices to scc2color
        DFS(transponed, scc2color.FirstNotFound(), scc2color, color++);
    }
}

```



Корректность. По лемме о временах выхода, компонента с максимальным $\max t_{\text{out}}$ является стоком в графе конденсации G_{cond} . В G_{cond}^T она становится истоком (нет входящих рёбер). Поэтому при старте DFS из любой её вершины в G^T будут посещены только вершины этой КСС. После удаления этой компоненты рассуждение повторяется по индукции.

