Лекция 02. DFS на неориентированных графах. BFS

9 февраля 2026 г.

Автор конспекта: Гридчин Михаил

Дерево обхода DFS

Определение. *Деревом обхода* DFS называют граф, состоящий из вершин, посещаемых в ходе обхода DFS и следующих рёбер:

- *Древесное ребро* — ребро, по которому DFS переходит напрямую (переходы в белые вершины из серых);
- *Обратное ребро* — ребро, которое DFS просматривает, но не идёт по нему (переходы в серые вершины).

Функция t_{up}

Определение. *Функцией $t_{\text{up}}(v)$* назовём функцию, определяемую следующим образом:

$$t_{\text{up}}(v) = \min \begin{cases} t_{\text{in}}(v), \\ \min\limits_{u} t_{\text{in}}(u) \end{cases}$$

где u — предок v и при этом u достижима по обратному ребру из w — вершины поддерева v.

```
void FillTUP(Graph graph, Vertex from, Vertex ancestor) {
  visited[from] = true;
  t_in[from] = t_up[from] = timer++;
  for (Vertex to : graph.GetNeighbours(from)) {
    if (to == ancestor) {
      continue;
    }
    if (visited[to]) {
      t_up[from] = min(t_up[from], t_in[to]);
    } else {
      FillTUP(graph, to, from);
      t_up[from] = min(t_up[from], t_up[to]);
    }
  }
}
```

Мосты и рёберная двусвязность

Определение. Две вершины в неориентированном графе *рёберно двусвязны*, если между ними есть два рёберно непересекающихся пути.

Теорема (б/д). Отношение рёберной двусвязности является отношением эквивалентности.
□
Обозначим отношение рёберной двусвязности как $(\sim)$. Покажем, что $(\sim)$ является отношением эквивалентности.
Рефлексивность: Положим по определению $u \sim u$.
Симметричность: Покажем, что если $u \sim v$, то $v \sim u$. Действительно, пути в неориентированном графе не имеют направления, а значит если есть два рёберно непересекающихся пути из u в v, то те же самые пути (пройденные в обратном порядке) являются двумя рёберно непересекающимися путями из v в u.
Транзитивность: Пусть $u \sim v$ и $v \sim w$. Тогда

- существуют два рёберно непересекающихся пути P_1, P_2 из u в v $(P_1 \cap P_2 = \varnothing)$
- существуют два рёберно непересекающихся пути Q_1, Q_2 из v в w $(Q_1 \cap Q_2 = \varnothing)$

Пусть $C = P_1 \cup P_2$. Заметим, что C — рёберно-простой цикл. Пусть вершины a и b — первые со стороны w вершины на пересечении $(Q_1 \cup Q_2) \cap C$. Рассмотрим пути $w \to a \to u$ и $w \to b \to u$, такие, что $a \to u$ и $b \to u$ идут по разные стороны по циклу C, а пути $w \to a$ и $w \to b$ идут по Q_1 и Q_2 соответственно. Приведённые пути не пересекаются, значит $u \sim w$.
■

Определение. *Компонентой рёберной двусвязности* в неориентированном графе называют класс эквивалентности по отношению рёберной двусвязности.

Определение *Мост* в неориентированном графе — это ребро, при удалении которого увеличивается число компонент связности.

Теорема (б/д). Рёбра графа конденсации по отношению рёберной двусвязности находятся во взаимно однозначном соответствии с мостами исходного графа: каждое ребро графа конденсации соответствует мосту исходного графа, и каждый мост исходного графа соединяет разные компоненты рёберной двусвязности, то есть является ребром графа конденсации.
□
Заметим сначала, что граф конденсации по отношению рёберной двусвязности ацикличен. Действительно, если бы существовал цикл, то он бы образовывал новую компоненту рёберной двусвязности. Значит граф конденсации относительно рёберной двусвязности — лес. Заметим, что каждое ребро в дереве является мостом, поэтому все рёбра в графе конденсации будут мостами. Покажем теперь, что других мостов не

существует. Действительно, пусть в компоненте рёберной двусвязности C есть ребро e, которое является мостом. Пусть e соединяет вершины x и y. Тогда x и y не рёберно двусвязны, поскольку граф конденсации ацикличен, а также между долями C, разделёнными мостом e, все пути проходят через e. Следовательно, в компонентах рёберной двусвязности не может быть мостов. Таким образом, все рёбра в графе конденсации по отношению рёберной двусвязности являются мостами в исходном графе, и все мосты в исходном графе являются рёбрами в графе конденсации, а значит, находятся во взаимно однозначном соответствии.
■

Теорема (критерий моста). В неориентированном графе древесное ребро $(p(v), v)$, где $p(v)$ — родитель v в дереве обхода DFS, является мостом, тогда и только тогда, когда $t_{\mathrm{up}}(v) = t_{\mathrm{in}}(v)$.
□
Обозначим ребро $e := (p(v), v)$.
$\boxed{\Rightarrow}$ Пусть ребро e — мост. Докажем, что $t_{\mathrm{up}}(v) = t_{\mathrm{in}}(v)$. Поскольку e — мост, его удаление нарушает связность графа: из любой вершины поддерева v невозможно достичь вершин вне этого поддерева без прохождения через ребро $(p(v), v)$. Следовательно, в поддереве v нет обратных рёбер выше $p(v)$. Следовательно, из определения $t_{\mathrm{up}}(v)$ второй случай никогда не выполняется: нет вершин u выше $p(v)$, которые достижимы по обратному ребру из поддерева v. Значит, $t_{\mathrm{up}}(v) = t_{\mathrm{in}}(v)$.
$\boxed{\Leftarrow}$ Пусть $t_{\mathrm{up}}(v) = t_{\mathrm{in}}(v)$. Докажем, что ребро e — мост. Предположим обратное: ребро e не является мостом. Тогда существует путь P из v в $p(v)$, не использующий ребро e. Рассмотрим первую вершину x на пути P, которая не лежит в поддереве v (такая найдётся, так как $p(v)$ вне поддерева v). Пусть y — предшествующая ей вершина на пути P, тогда y лежит в поддереве v, а ребро (y, x) — обратное. Поскольку в обходе DFS вершина x была посещена раньше v, то $t_{\mathrm{in}}(x) < t_{\mathrm{in}}(v)$. Но тогда поскольку (y, x) — обратное ребро, то $t_{\mathrm{up}}(v) \le t_{\mathrm{in}}(x) < t_{\mathrm{in}}(v)$. Противоречие, так как по условию $t_{\mathrm{up}}(v) = t_{\mathrm{in}}(v)$. Следовательно, предположение неверно, и ребро e является мостом.
■

Точки сочленения

Определение. Две вершины в неориентированном графе *вершинно двусвязны*, если между ними есть два вершинно непересекающихся пути.

Замечание. Отношение вершинной двусвязности на вершинах не является отношением эквивалентности.
□
Покажем, что транзитивность не выполняется. Пусть граф задан на вершинах $\{u, v, w\}$ и имеет следующие рёбра: $(u, v), (u, v), (v, w), (v, w)$. Вершины u и v, а также v и w вершинно двусвязны, но вершины u и w не вершинно двусвязны.
■

Определение. *Точка сочленения* в неориентированном графе — вершина, при удалении которой увеличивается число компонент связности.

Теорема (критерий точки сочленения). В неориентированном графе вершина v является точкой сочленения тогда и только тогда, когда:

1. v — корень дерева обхода DFS, и у v не менее двух детей в этом дереве.
2. v — не корень дерева обхода, и существует ребёнок w вершины v такой, что $t_{\text{up}}(w) \geq t_{\text{in}}(v)$.

□
Случай 1: v — корень DFS.
$\boxed{\Rightarrow}$ Пусть v — точка сочленения и корень. После удаления v граф распадётся на ≥ 2 компоненты связности. Каждая компонента содержит хотя бы одну вершину, достижимую из v первым шагом обхода, то есть ребёнка v. Если бы у v был только один ребёнок, все вершины бы лежали в одном поддереве, и после удаления v связность бы не нарушилась — противоречие. Следовательно, у v не менее двух детей.
$\boxed{\Leftarrow}$ Пусть у корня не менее двух детей w_1, w_2. Пусть v не точка сочленения. Тогда после удаления v существует путь P между поддеревьями w_1, w_2, не проходящий через v. Но при обходе DFS из v сначала в w_1 мы бы достигли поддерева w_2, пройдя по пути P и добавили бы его как часть поддерева w_1, а не как отдельного ребёнка — противоречие. Значит, v — точка сочленения.

Случай 2: v — не корень DFS.
$\boxed{\Rightarrow}$ Пусть v — точка сочленения. После удаления v граф распадётся на компоненты. Рассмотрим компоненту C, которая не содержит родителя $p(v)$. Все вершины C достижимы из v через некоторого ребёнка w (по древесному ребру (v, w)). Покажем, что $t_{\text{up}}(w) \geq t_{\text{in}}(v)$. Предположим противное: $t_{\text{up}}(w) < t_{\text{in}}(v)$. Тогда существует путь из поддерева w (то есть из C) в вершину u с $t_{\text{in}}(u) < t_{\text{in}}(v)$, не использующий ребро $(v, p(v))$. Поскольку $t_{\text{in}}(u) < t_{\text{in}}(v)$, вершина u была посещена до v, значит $u \notin C \cup \{v\}$. Этот путь соединяет C с внешней частью графа без прохождения через v — противоречие с тем, что C отдельная компонента после удаления v. Следовательно, $t_{\text{up}}(w) \geq t_{\text{in}}(v)$.
$\boxed{\Leftarrow}$ Пусть существует ребёнок w такой, что $t_{\text{up}}(w) \geq t_{\text{in}}(v)$. Следовательно, из поддерева w нет обратных рёбер в вершину u с $t_{\text{in}}(u) < t_{\text{in}}(v)$. Покажем, что после удаления v поддерево w станет новой компонентой связности. Предположим противное: существует путь P из вершины x в поддереве w в вершину $y \notin \text{subtree}(w) \cup \{v\}$, не проходящий через v. Рассмотрим первое ребро (a, b) на P, где $a \in \text{subtree}(w), b \notin \text{subtree}(w) \cup \{v\}$. Тогда $b \neq v$ и $t_{\text{in}}(b) < t_{\text{in}}(v)$ (иначе b вошла бы в поддерево w при DFS). Но тогда $t_{\text{up}}(w) \leq t_{\text{in}}(b) < t_{\text{in}}(v)$ — противоречие. Следовательно, поддерево w образует отдельную компоненту после удаления v, то есть v — точка сочленения.
■

Взвешенные графы

Определение. *Взвешенным графом* будем называть тройку $G = (V, E, w)$, где V — множество вершин ($|V| < \infty$), $E \subseteq V \times V$ — мультимножество рёбер, $w : E \to K \subseteq \mathbb{R}$ — весовая функция.

Определение. *Весом (длиной) пути* $p = v_1, \ldots, v_k$ будем называть величину $w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$.

Def. *Кратчайшим путём* от вершины s до вершины t, $\mathrm{dist}(s, t)$ будем называть путь $s = v_1, \ldots, v_k = t$ такой, что его вес минимален среди всех возможных путей от s до t.

BFS

Определение. *Обход в ширину* (*BFS*, *breadth-first search*) — алгоритм обхода графа в порядке неубывания расстояния от заданной стартовой вершины s. В случае неориентированного графа без весов (либо с единичными весами рёбер) обход в ширину находит длины кратчайших путей от s до всех достижимых из s вершин.

Алгоритм (BFS). Пусть даны граф $G = (V, E, w), w : E \to K = \{1\}$ и стартовая вершина $s \in V$. Требуется заполнить массив расстояний $\mathrm{dist}[v]$ — длина кратчайшего пути от s до v (или ∞, если v недостижима).

Инициализация:

- Для всех $v \in V$ положить $\mathrm{dist}[v] = \infty, \mathrm{used}[v] = \mathrm{false}$.
- $\mathrm{dist}[s] = 0, \mathrm{used}[s] = \mathrm{true}$
- Создать очередь Q и поместить в неё s.
  Основной цикл. Пока очередь Q не пуста:
- Извлечь вершину v из начала очереди.
- Для каждого соседа u вершины v если $\mathrm{used}[u] = \mathrm{false}$, то изменить $\mathrm{used}[u] = \mathrm{true}$, $\mathrm{dist}[u] = \mathrm{dist}[v] + 1$ и добавить u в конец очереди Q.

Псевдокод:

```
void BFS(Graph graph, Vertex start) {
  Queue<Vertex> bfs_queue;
  HashMap<Vertex, int> dist;
  dist[start] = 0;
  bfs_queue.push(start);
  while (!bfs_queue.empty()) {
    Vertex from = bfs_queue.pop();
    for (Vertex to : graph.GetNeighbours(from)) {
      if (dist.HasKey(to)) {
        continue;
```

```
      }
      dist[to] = dist[from] + 1;
      bfs_queue.push(to);
    }
  }
}
```

Теорема (корректность BFS). В любой момент работы алгоритма BFS в очереди содержатся вершины, расстояния которых от стартовой вершины s равны только l или $l+1$, причём все вершины с расстоянием l расположены перед вершинами с расстоянием $l+1$, где l — расстояние от s до последней обработанной вершины (или $l=0$, если очередь содержит только s).
□
Докажем утверждение по индукции по количеству обработанных вершин (т.е. вершин, извлечённых из очереди и помеченных как посещённые).
База. Изначально очередь содержит только вершину s, для которой $\mathrm{dist}(s,s)=0$. Таким образом, $l=0$, и в очереди содержатся вершины с расстоянием $l=0$. Утверждение выполнено.
Шаг. Пусть после обработки k вершин в очереди содержатся все вершины с расстоянием l (если они есть), все вершины с расстоянием $l+1$ (если они есть), причём вершины с расстоянием l расположены перед вершинами с расстоянием $l+1$. Рассмотрим обработку $(k+1)$-й вершины v. По предположению индукции, v имеет расстояние l (поскольку вершины с расстоянием l расположены в начале очереди). При обработке v мы добавляем в очередь все ещё не посещённые соседние вершины u, для которых $\mathrm{dist}(s,u)=\mathrm{dist}(s,v)+1=l+1$. Так как добавление происходит в конец очереди, новые вершины с расстоянием $l+1$ размещаются после уже существующих вершин с расстоянием $l+1$. Таким образом, упорядоченность очереди сохраняется: сначала вершины с расстоянием l, затем с расстоянием $l+1$.
■

Теорема (асимптотика BFS). Алгоритм BFS работает за $O(|V|+|E|)$.
□
Оценим, сколько раз в ходе алгоритма происходит перебор соседей вершины v. Каждое ребро $\{u,v\}$ хранится в списках смежности обеих вершин. Оно просматривается дважды: при обработке v и при обработке u. Общее количество просмотров $O(|E|)$. Также каждая вершина будет добавлена в очередь (и извлечена для перебора соседей) не более одного раза. Таким образом, операций с очередью $O(|V|)$ и суммарно $O(|V|)+O(|E|)=O(|V|+|E|)$.
■

0-1 BFS

Определение. *0-1 BFS* — это модификация обхода в ширину для взвешенных графов, в которых вес каждого ребра принадлежит множеству $\{0,1\}$. Алгоритм находит длины кратчайших путей от заданной стартовой вершины s до всех остальных.

Алгоритм (0-1 BFS). В отличие от обычного BFS (который использует очередь), 0-1 BFS применяет дек:

- При переходе по ребру веса 0 вершина добавляется в начало дека — её расстояние не увеличивается, поэтому она должна быть обработана раньше текущих вершин того же уровня.
- При переходе по ребру веса 1 вершина добавляется в конец дека — её расстояние увеличивается на 1, поэтому она обрабатывается позже.

Псевдокод:

```
Vertex from = bfs_queue.pop_front();
for (Vertex to : graph.GetNeighbours(from)) {
  if (dist.HasKey(to)) {
    continue;
  }
  dist[to] = dist[from] + w(from, to);
  if (w(from, to) == 0) {
    bfs_queue.push_front(to);
  } else {
    bfs_queue.push_back(to);
  }
}
```

Теорема (корректность 0-1 BFS). В любой момент времени работы алгоритма дек D содержит вершины, упорядоченные по неубыванию их текущих расстояний $\mathrm{dist}[\cdot]$ от s.
□
Доказательство по индукции.
База. Изначально $D = [s], \mathrm{dist}(s, s) = 0$ — упорядоченность выполнена.
Шаг. Предположим, что перед обработкой вершины v дек упорядочен: все вершины в нём имеют расстояния $l, l, \ldots, l, l+1, l+1, \ldots, l+1$. При обработке v с $\mathrm{dist}(s, v) = l$:

- Для ребра веса 0 получаем вершину w с $\mathrm{dist}(s, w) = l$. Она добавляется в начало дека, где уже находятся вершины с расстоянием l — порядок сохраняется.
- Для ребра веса 1 получаем вершину w с $\mathrm{dist}(s, w) = l + 1$. Она добавляется в конец дека, где находятся вершины с расстоянием $> l$ — порядок сохраняется.
  После извлечения v из начала дека упорядоченность также сохраняется.
  ■

Теорема (асимптотика 0-1 BFS). Алгоритм 0-1 BFS работает за $O(|V| + |E|)$.
□

Анализ временной сложности аналогичен обычному BFS. Заметим, что каждая вершина может быть добавлена в дек несколько раз, но не более чем дважды:

- Первый раз — при получении расстояния $l + 1$ через ребро веса 1.
- Второй раз — при улучшении расстояния до l через ребро веса 0.
- Больше двух раз вершина добавляться не может, так как расстояния только уменьшаются, а минимальное расстояние фиксировано.

Таким образом, учитывая, что каждое ребро просматривается не более двух раз, и каждая вершина добавляется в дек не более чем дважды, время работы равно $O(|V| + |E|)$.
■

1-k BFS

Определение. *1-k BFS* — это модификация обхода в ширину для взвешенных графов, в которых вес каждого ребра принадлежит множеству $\{1, 2, \ldots, k\}$, где k — фиксированная константа. Алгоритм находит длины кратчайших путей от заданной стартовой вершины s до всех остальных вершин.

Алгоритм (1-k BFS). В отличие от обычного BFS (который использует одну очередь), 1-k BFS применяет массив очередей `at_dist`, где `at_dist[d]` содержит очередь вершин с текущим расстоянием d. Это позволяет обрабатывать вершины в порядке неубывания расстояний, гарантируя корректность вычисления кратчайших путей.

Псевдокод:

```
int max_dist = k * (graph.GetVerticesCount() - 1);
Array<Queue<Vertex>> at_dist(max_dist);
at_dist[0].push(start);
HashMap<Vertex, int> dist;
dist[start] = 0;
for (int dist = 0; dist < max_dist; ++dist) {
  while (!at_dist[dist].empty()) {
    Vertex from = at_dist[dist].pop();
    if (dist[from] < dist) {
      continue;
    }
    for (Vertex to : graph.GetNeighbours(from)) {
      if (dist[to] > dist[from] + w(from, to)) {
        dist[to] = dist[from] + w(from, to);
        at_dist[dist[to]].push(to);
      }
    }
```

```
    }
  }
```

Теорема (корректность 1-k BFS). В любой момент работы алгоритма вершины в массиве очередей `at_dist` упорядочены по неубыванию их расстояний от s.
□
Доказательство по индукции.
База. Изначально `at_dist[0] = [s]`, $\text{dist}(s, s) = 0$ — упорядоченность выполнена.
Шаг. Предположим, что перед обработкой расстояния l все вершины с расстоянием $< l$ уже обработаны, а `at_dist[l]` содержит вершины с расстоянием l. При обработке вершины v с $\text{dist}(s, v) = l$:

- Для ребра веса $w \in \{1, \ldots, k\}$ получаем вершину u с $\text{dist}(s, u) = l + w$.
- Вершина u добавляется в очередь `at_dist[l + w]`, которая будет обработана позже (так как $l + w > l$).
- Если $\text{dist}(s, u)$ уже меньше $l + w$, вершина пропускается.

Следовательно, вершины всегда обрабатываются в порядке неубывания расстояний, что гарантирует корректность значений $\text{dist}(s, v)$ как длин кратчайших путей.
■

Теорема (асимптотика 1-k BFS). Алгоритм 1-k BFS работает за $O(k \cdot |V| + |E|)$.
□
Заметим, что каждое ребро просматривается один раз. Для ребра веса w вершина u может быть добавлена в очередь не более одного раза (после первого улучшения расстояния). Поскольку веса ограничены k, максимальное расстояние не превосходит $k \cdot (|V| - 1)$, что позволяет использовать массив фиксированного размера. Таким образом, асимптотика $O(k \cdot |V| + |E|)$.
■

Замечание. В каждый момент только k очередей не пусты (при обработке расстояния l нас интересуют только расстояния в диапазоне $[l, l + k]$), можно хранить k очередей вместо $k \cdot |V|$, и тогда потребление памяти составит $O(k + |V|)$.

0-k BFS

Определение. *0-k BFS* — это модификация обхода в ширину для взвешенных графов, в которых вес каждого ребра принадлежит множеству $\{0, 1, \ldots, k\}$, где k — фиксированная константа. Алгоритм находит длины кратчайших путей от заданной стартовой вершины s до всех остальных вершин.

Алгоритм (0-k BFS). В отличие от 1-k BFS, 0-k BFS использует массив деков вершин `at_dist`, где `at_dist[l]` содержит вершины с текущим расстоянием l от s. Отличие от 1-k BFS — обработка рёбер веса 0:

- При переходе по ребру веса 0 вершина добавляется в начало текущего дека `at_dist[l]` , чтобы быть обработанной раньше текущих.
- При переходе по ребру веса $w > 0$ вершина добавляется в конец дека `at_dist[l + w]` .

Теорема (корректность 0-k BFS). При обработке дека `at_dist[l]` все вершины с расстоянием $< l$ уже обработаны окончательно, а вершины в `at_dist[l]` обрабатываются в порядке неубывания их окончательных расстояний.
□
Доказательство аналогично доказательству 1-k BFS и 0-1 BFS.
■

Теорема (асимптотика 0-k BFS). Алгоритм 0-k BFS работает за $O(k \cdot |V| + |E|)$.
□
Доказательство аналогично доказательству 1-k BFS и 0-1 BFS.
■