Мосты. Конденсация графа

Denis Bakin

Определения

- Мост, точка сочленения

Определения

- Мост, точка сочленения
- **Мост** — ребро неориентированного графа, при удалении которого граф становится несвязным

Определения

- Мост, точка сочленения
- **Мост** — ребро неориентированного графа, при удалении которого граф становится несвязным
- **Точка сочленения** — вершина неориентированного графа, при удалении которой граф становится несвязным

Определения

- Мост, точка сочленения
- **Мост** — ребро неориентированного графа, при удалении которого граф становится несвязным
- **Точка сочленения** — вершина неориентированного графа, при удалении которой граф становится несвязным
- **Задача:** дана топология сети (компьютеры и соединения). Найти все единые точки отказа — узлы и связи, без которых сеть распадётся

Наивный подход

Для каждого ребра (u, v): удалить его, проверить связность, вернуть обратно

Наивный подход

Для каждого ребра (u, v): удалить его, проверить связность, вернуть обратно

Сложность: $O(m \cdot (n + m))$

Классификация рёбер при DFS

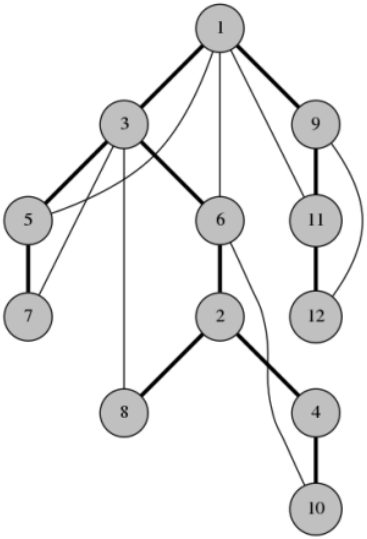
Запустим DFS из произвольной вершины. Все рёбра разделятся на два типа:

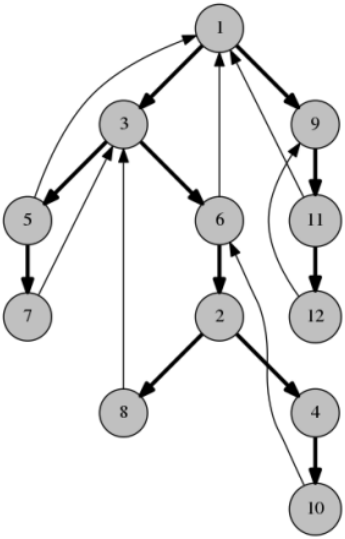
Классификация рёбер при DFS

Запустим DFS из произвольной вершины. Все рёбра разделятся на два типа:

Тип ребра	Описание
Ребро дерева (tree edge)	DFS перешёл по нему в новую вершину
Обратное ребро (back edge)	вело в уже посещённую вершину

Классификация рёбер при DFS



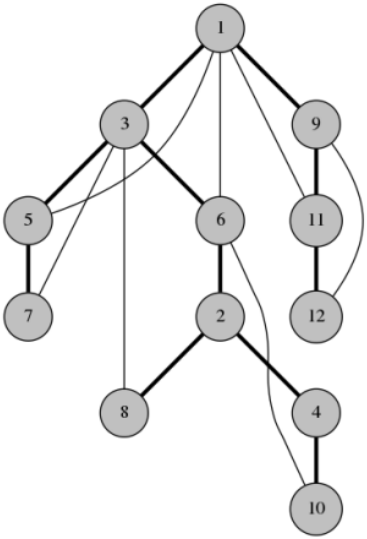


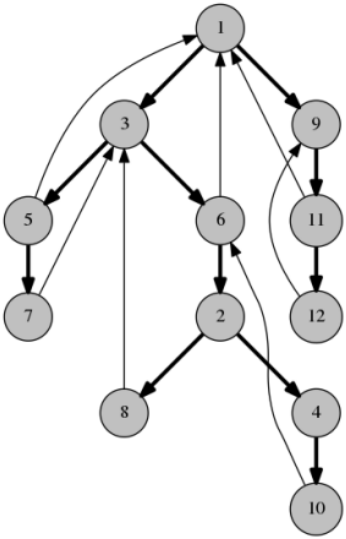
Обратные рёбра — не мосты

- обратное ребро не может быть мостом
- $\Rightarrow$ проверять нужно только рёбра дерева

Наблюдение: в неориентированном графе обратные рёбра всегда ведут к **предку** в дереве обхода (иначе DFS увидел бы это ребро раньше, и оно стало бы ребром дерева)

Идея





Функция fup

Для каждой вершины v определим:

$$fup_v = \text{минимальная глубина вершины, достижимой из поддерева } v$$

(с использованием не более одного обратного ребра)

Функция fup

Для каждой вершины v определим:

$$fup_v = \text{минимальная глубина вершины, достижимой из поддерева } v$$

(с использованием не более одного обратного ребра)

$$fup_v = \min \begin{cases} h_v, & \\ fup_u, & \text{ребро } (v,u) \text{ — ребро дерева} \\ h_u, & \text{ребро } (v,u) \text{ — обратное} \end{cases}$$

где h_v — глубина вершины v в дереве обхода

Критерий моста

Рассмотрим ребро дерева (v, u), где v — родитель, u — ребёнок

Критерий моста

Рассмотрим ребро дерева (v, u), где v — родитель, u — ребёнок

Критерий: ребро (v, u) является мостом $\iff$

$$h_v < fup_u$$

То есть из поддерева u нельзя добраться до v или выше по обратным рёбрам

$\Rightarrow$ удаление (v, u) разрывает связь между v и u

Мосты: реализация

```
// ...
void dfs(int v, int parent = -1) {
    visited[v] = true;
    depth[v] = fup[v] = (parent == -1 ? 0 : depth[parent] + 1);
    for (int u : adj[v]) {
        if (u == parent) continue;
        if (visited[u]) { // обратное ребро - обновляем fup глубиной предка
            fup[v] = std::min(fup[v], depth[u]);
        } else { // ребро дерева - рекурсивно обходим
            dfs(u, v);
            fup[v] = std::min(fup[v], fup[u]);
            if (depth[v] < fup[u]) {
                // ребро (v, u) - мост
            }
        }
    }
}
// Временная сложность: $O(n + m)$
```

Критерий точки сочленения

Вершина v — точка сочленения, если для какого-то ребёнка u из поддерева u нельзя добраться до строгого предка v

Критерий точки сочленения

Вершина v — точка сочленения, если для какого-то ребёнка u из поддерева u нельзя добраться до строгого предка v

Критерий (не корень): v — точка сочленения, если существует ребёнок u:

$$h_v \leq fup_u$$

- Неравенство **нестрогое:** если $fup_u = h_v$, из поддерева u можно добраться только до v, но не выше — при удалении v поддерево u отсоединится

Критерий точки сочленения

Вершина v — точка сочленения, если для какого-то ребёнка u из поддерева u нельзя добраться до строгого предка v

Критерий (не корень): v — точка сочленения, если существует ребёнок u:

$$h_v \leq fup_u$$

- Неравенство **нестрогое:** если $fup_u = h_v$, из поддерева u можно добраться только до v, но не выше — при удалении v поддерево u отсоединится

Критерий (корень): корень DFS-дерева является точкой сочленения $\iff$ у него **более одного ребёнка** в дереве обхода

Сильная связность: определение

- **Определение.** Две вершины **ориентированного** графа *сильно связаны*, если существует направленный путь из первой во вторую **и** из второй в первую
- Транзитивность: если $a \leftrightarrow b$ и $b \leftrightarrow c$, то $a \leftrightarrow c$
- Все вершины разбиваются на **компоненты сильной связности (КСС)** — максимальные подмножества, внутри которых все вершины попарно сильно связаны

Конденсация графа

Конденсация — граф, в котором каждая КСС сжата в одну вершину. Это **DAG** (направленный ациклический граф)

Почему граф ациклический?

Алгоритм Косарайю

Вход: ориентированный граф $G = (V, E)$

Выход: для каждой вершины — номер её КСС

Цель: $O(n + m)$

Лемма

Лемма. Пусть A и B — две различные КСС, и есть ребро $A \to B$ в графе конденсации. Тогда:

$$\max_{a \in A}(tout_a) > \max_{b \in B}(tout_b)$$

Компонента, из которой **выходят** рёбра, имеет **больший** максимальный $tout$

Доказательство

Случай 1: DFS первой достигает A

- DFS заходит в вершину $v \in A$, остальные вершины A и B не посещены

Доказательство

Случай 1: DFS первой достигает A

- DFS заходит в вершину $v \in A$, остальные вершины A и B не посещены
- Из A есть ребро в $B \Rightarrow$ из v достижима вся B

Доказательство

Случай 1: DFS первой достигает A

- DFS заходит в вершину $v \in A$, остальные вершины A и B не посещены
- Из A есть ребро в $B \Rightarrow$ из v достижима вся B
- DFS из v посетит все вершины A и B

Доказательство

Случай 1: DFS первой достигает A

- DFS заходит в вершину $v \in A$, остальные вершины A и B не посещены
- Из A есть ребро в $B \Rightarrow$ из v достижима вся B
- DFS из v посетит все вершины A и B
- $tout[v] > tout[u]$ $\forall u \in A \cup B$, $u \neq v$

Доказательство

Случай 1: DFS первой достигает A

- DFS заходит в вершину $v \in A$, остальные вершины A и B не посещены
- Из A есть ребро в $B \Rightarrow$ из v достижима вся B
- DFS из v посетит все вершины A и B
- $tout[v] > tout[u]$ $\forall u \in A \cup B$, $u \neq v$

Доказательство

Случай 1: DFS первой достигает A

- DFS заходит в вершину $v \in A$, остальные вершины A и B не посещены
- Из A есть ребро в $B \Rightarrow$ из v достижима вся B
- DFS из v посетит все вершины A и B
- $tout[v] > tout[u]$ $\forall u \in A \cup B$, $u \neq v$

Случай 2: DFS первой достигает B

- Из B нельзя попасть в A (иначе одна КСС, т. к. по условию есть ребро $A \to B$)
- DFS выйдет из всех вершин B **до** входа в A

Следствие леммы

Вершина с **максимальным** $tout$ принадлежит КСС, в которую **не входят** рёбра из других компонент

Если развернуть все рёбра (*транспонировать* граф), то из этой вершины по транспонированному графу достижима **ровно её КСС** — и ничего больше

Алгоритм Косарайю: шаги

1. Запускаем DFS на исходном графе, записываем вершины по убыванию $tout$

Алгоритм Косарайю: шаги

1. Запускаем DFS на исходном графе, записываем вершины по убыванию $tout$
2. Строим **транспонированный** граф (разворачиваем все рёбра)

Алгоритм Косарайю: шаги

1. Запускаем DFS на исходном графе, записываем вершины по убыванию $tout$
2. Строим **транспонированный** граф (разворачиваем все рёбра)
3. Проходим по вершинам в порядке убывания $tout$. Для каждой непомеченной вершины запускаем DFS на транспонированном графе — все достижимые вершины образуют одну КСС

Реализация: переменные

```
std::vector<std::vector<int>> adj(n);      // исходный граф
std::vector<std::vector<int>> adj_rev(n);  // транспонированный граф
std::vector<bool> visited(n, false);
std::vector<int> order;             // вершины по убыванию tout
std::vector<int> component(n, -1);  // номер КСС
int num_components = 0;
```

Реализация: функции DFS

```
void dfs1(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u]) {
            dfs1(u);
        }
    }
    order.push_back(v);
}

void dfs2(int v) {
    component[v] = num_components;
    for (int u : adj_rev[v]) {
        if (component[u] == -1) {
            dfs2(u);
        }
    }
}
```

Реализация: использование обходов

```
// построение транспонированного графа
for (int v = 0; v < n; ++v) {
    for (int u : adj[v]) {
        adj_rev[u].push_back(v);
    }
}
```

Реализация: использование обходов

```
// первый проход: определяем порядок по tout
for (int v = 0; v < n; ++v) {
    if (!visited[v]) {
        dfs1(v);
    }
}
```

Реализация: использование обходов

```
// второй проход: выделяем КСС
std::reverse(order.begin(), order.end());
for (int v : order) {
    if (component[v] == -1) {
        dfs2(v);
        ++num_components;
    }
}
```

Свойство результата

- После выполнения алгоритма номера компонент `component[v]` **топологически упорядочены:**
- Если в графе конденсации есть ребро из компоненты i в компоненту j, то $i < j$

Временная сложность: $O(n + m)$

Итоги: мосты

- Ребро, при удалении которого граф распадается
- Обратные рёбра — не мосты, проверяем только рёбра дерева
- Функция fup_v: минимальная глубина, достижимая из поддерева v
- Критерий: $h_v < fup_u$ (строгое неравенство)
- Сложность: $O(n + m)$

Итоги: точки сочленения

- Вершина, при удалении которой граф распадается
- Критерий: $h_v \leq fup_u$ (нестрогое неравенство)
- Особый случай: корень — точка сочленения $\iff > 1$ ребёнка в DFS-дереве
- Сложность: $O(n + m)$

Итоги: КСС и конденсация

- КСС — максимальное множество попарно сильно связанных вершин
- Конденсация — сжатие каждой КСС в вершину, всегда DAG
- Алгоритм Косарайю: два DFS (прямой + на транспонированном графе)
- Номера КСС автоматически топологически упорядочены
- Сложность: $O(n + m)$