добавлено: 5 Jul 2008 22:26
редактировано: 6 Dec 2012 1:41

Поиск точек сочленения

Содержание [скрыть]

Пусть дан связный неориентированный граф. **Точкой сочленения** (или точкой артикуляции, англ. "cut vertex" или "articulation point") называется такая вершина, удаление которой делает граф несвязным.

Опишем алгоритм, основанный на поиске в глубину, работающий за $O(n+m)$, где n — количество вершин, m — рёбер.

Алгоритм

Запустим обход в глубину из произвольной вершины графа; обозначим её через root. Заметим следующий **факт** (который несложно доказать):

- Пусть мы находимся в обходе в глубину, просматривая сейчас все рёбра из вершины $v \neq \text{root}$. Тогда, если текущее ребро (v, to) таково, что из вершины to и из любого её потомка в дереве обхода в глубину нет обратного ребра в какого-либо предка вершины v, то вершина v является точкой сочленения. В противном случае, т.е. если обход в глубину просмотрел все рёбра из вершины v, и не нашёл удовлетворяющего вышеописанным условиям ребра, то вершина v не является точкой сочленения. (В самом деле, мы этим условием проверяем, нет ли другого пути из v в to)
- Рассмотрим теперь оставшийся случай: $v = \text{root}$. Тогда эта вершина является точкой сочленения тогда и только тогда, когда эта вершина имеет более одного сына в дереве обхода в глубину. (В самом деле, это означает, что, пройдя из root по произвольному ребру, мы не смогли обойти весь граф, откуда сразу следует, что root — точка сочленения).

(Ср. формулировку этого критерия с формулировкой критерия для алгоритма поиска мостов.)

Теперь осталось научиться проверять этот факт для каждой вершины эффективно. Для этого воспользуемся "временами входа в вершину", вычисляемыми алгоритмом поиска в глубину.

Итак, пусть $tin[v]$ — это время захода поиска в глубину в вершину v. Теперь введём массив $fup[v]$, который и позволит нам отвечать на вышеописанные запросы. Время $fup[v]$ равно минимуму из времени захода в саму вершину $tin[v]$, времён захода в каждую вершину p, являющуюся концом некоторого обратного ребра (v, p), а также из всех значений $fup[to]$ для каждой вершины to, являющейся непосредственным сыном v в дереве поиска:

$$fup[v] = \min \begin{cases} tin[v], & \\ tin[p], & \text{for all (v,p) — back edge} \\ fup[to], & \text{for all (v,to) — tree edge} \end{cases}$$

(здесь "back edge" — обратное ребро, "tree edge" — ребро дерева)

Тогда, из вершины v или её потомка есть обратное ребро в её предка тогда и только тогда, когда найдётся такой сын to, что $fup[to] < tin[v]$.

Таким образом, если для текущего ребра (v, to) (принадлежащего дереву поиска) выполняется $fup[to] \geq tin[v]$, то вершина v является точкой сочленения. Для начальной вершины $v = \text{root}$ критерий другой: для этой вершины надо посчитать число непосредственных сыновей в дереве обхода в глубину.

Реализация

Если говорить о самой реализации, то здесь нам нужно уметь различать три случая: когда мы идём по ребру дерева поиска в глубину, когда идём по обратному ребру, и когда пытаемся пойти по ребру дерева в обратную сторону. Это, соответственно, случаи $used[to] = false$, $used[to] = true \ \&\& \ to \neq parent$, и $to = parent$. Таким образом, нам надо передавать в функцию поиска в глубину вершину-предка текущей вершины.

```
vector<int> g[MAXN];
bool used[MAXN];
int timer, tin[MAXN], fup[MAXN];

void dfs (int v, int p = -1) {
        used[v] = true;
        tin[v] = fup[v] = timer++;
        int children = 0;
        for (size_t i=0; i<g[v].size(); ++i) {
                int to = g[v][i];
                if (to == p)  continue;
                if (used[to])
                        fup[v] = min (fup[v], tin[to]);
                else {
                        dfs (to, v);
                        fup[v] = min (fup[v], fup[to]);
                        if (fup[to] >= tin[v] && p != -1)
                                IS_CUTPOINT(v);
                        ++children;
                }
        }
        if (p == -1 && children > 1)
                IS_CUTPOINT(v);
}

int main() {
        int n;
        ... чтение n и g ...

        timer = 0;
        for (int i=0; i<n; ++i)
```

```
                used[i] = false;
        dfs (0);
}
```

Здесь константе MAXN должно быть задано значение, равное максимально возможному числу вершин во входном графе.

Функция $\mathrm{IS\_CUTPOINT}(v)$ в коде — это некая функция, которая будет реагировать на то, что вершина v является точкой сочленения, например, выводить эту вершины на экран (надо учитывать, что для одной и той же вершины эта функция может быть вызвана несколько раз).

Задачи в online judges

Список задач, в которых требуется искать точки сочленения:

- UVA #10199 **"Tourist Guide"** [сложность: низкая]
- UVA #315 **"Network"** [сложность: низкая]