Алгоритм 6

Алгоритм Хопкрофта-Тарьяна поиска мостов

Мостом называется такое ребро, удаление которого делает граф несвязным (или, точнее, увеличивает число компонент связности). Требуется найти все мосты в заданном графе.

**Алгоритм**

Запустим [обход в глубину](http://e-maxx.ru/algo/dfs) из произвольной вершины графа; обозначим её через \rm root. Заметим следующий **факт** (который несложно доказать):

* Пусть мы находимся в обходе в глубину, просматривая сейчас все рёбра из вершины v. 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 вершину v или какого-либо её предка, то это ребро является мостом. В противном случае оно мостом не является. (В самом деле, мы этим условием проверяем, нет ли другого пути из v в to, кроме как спуск по ребру (v,to)дерева обхода в глубину.)

Теперь осталось научиться проверять этот факт для каждой вершины эффективно. Для этого воспользуемся "временами входа в вершину", вычисляемыми [алгоритмом поиска в глубину](http://e-maxx.ru/algo/dfs).

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 \cases{
tin[v], & \cr
tin[p], & {[...]

(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] \le tin[v]. (Если fup[to] = tin[v], то это означает, что найдётся обратное ребро, приходящее точно в v; если же fup[to] < tin[v], то это означает наличие обратного ребра в какого-либо предка вершины v.)

Таким образом, если для текущего ребра (v,to) (принадлежащего дереву поиска) выполняется fup[to] > tin[v], то это ребро является мостом; в противном случае оно мостом не является.

**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 — критерий ребра дерева поиска;
* used[to]=true\ \&\&\ to \ne parent — критерий обратного ребра;
* to=parent — критерий прохода по ребру дерева поиска в обратную сторону.

Таким образом, для реализации этих критериев нам надо передавать в функцию поиска в глубину вершину-предка текущей вершины.

const int MAXN = ...;

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++;

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])

IS\_BRIDGE(v,to);

}

}

}

void find\_bridges() {

timer = 0;

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

used[i] = **false**;

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

if (!used[i])

dfs (i);

}

Здесь основная функция для вызова — это {\rm find\_bridges} — она производит необходимую инициализацию и запуск обхода в глубину для каждой компоненты связности графа.

При этом {\rm IS\_BRIDGE}(a,b) — это некая функция, которая будет реагировать на то, что ребро (a,b) является мостом, например, выводить это ребро на экран.

Константе {\rm MAXN} в самом начале кода следует задать значение, равное максимально возможному числу вершин во входном графе.

Стоит заметить, что эта реализация некорректно работает при наличии в графе **кратных рёбер**: она фактически не обращает внимания, кратное ли ребро или оно единственно. Разумеется, кратные рёбра не должны входить в ответ, поэтому при вызове \rm IS\_BRIDGE можно проверять дополнительно, не кратное ли ребро мы хотим добавить в ответ. Другой способ — более аккуратная работа с предками, т.е. передавать в \rm dfs не вершину-предка, а номер ребра, по которому мы вошли в вершину (для этого надо будет дополнительно хранить номера всех рёбер).