MAXimal

added: 10 Jun 2008 19:28
Edited: 23 Aug 2011 11:23

Search bridges

Suppose we are given an undirected graph. A bridge is an edge whose removal makes the graph disconnected (or, rather, increases the number of connected components). You want to find all the bridges in a given graph.

Informally, this problem is formulated as follows: it is required to find on a given road map all the "important" roads, ie such way that removing either of them will lead to the disappearance of the path between any pair of cities.

Below we describe an algorithm based on DFS , and the running time $O(n+m)$, where n- the number of vertices m- edges in the graph.

Note that the site also describes an online search algorithm bridges - in contrast to the algorithm described here, an online algorithm is able to maintain all the bridges of the graph in a changing graph (refers to adding new edges).

Algorithm

Start the tour in depth from an arbitrary vertex of the graph; denote it through root. We note the following **fact** (which is not difficult to prove):

- Suppose we are in a circuit in depth, looking now all edges from the top v. Then, if the current edge (v, to)such that from the top $to$and from any of its descendants in the tree traversal depth no reverse edges in the top $v$or any of its parent, then this edge is bridge. Otherwise, it is not a bridge. (In fact, we have this condition, check if there are other ways of $v$to toother than the descent along the edge $(v, to)$of the tree traversal in depth.)

Now it is necessary to learn how to verify this fact for each vertex effectively. To do this, use the "time of entry into the top," is calculated by the search algorithm in depth .

So, let $tin[v]$- this time call DFS at the top v. Now we introduce an array $fup[v]$, which will allow us to answer the above questions. Time $fup[v]$is the minimum time of entry into the very top $tin[v]$, the time of entering into each vertex p, which is the end of a reverse edges (v, p), as well as of all the values $fup[to]$for each vertex to, which is a direct son $v$of the search tree:

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

(Here "back edge" - the opposite edge, "tree edge" - edge of the tree)

Then, from the top $v$or her offspring have the opposite edge in its parent if and only if there is such a son tothat $fup[to] \le tin[v]$. (If $fup[to] = tin[v]$, this means that there exists a reverse edge coming into precisely v, but if $fup[to] < tin[v]$it means the presence of the reverse edge in any ancestor vertex v.)

Thus, if the current edge (v, to)(belonging to the tree search) is performed $fup[to] > tin[v]$, then this edge is a bridge; otherwise it is not a bridge.

Implementation

If we talk about the actual implementation, here we need to be able to distinguish three cases: when we are on the edge of the tree depth-first search, when we go to the opposite edge, and when we try to go along the edge of the tree in the opposite direction. It is, accordingly, the cases:

- $used[to] = false$ - Criterion of edges in the search;
- $used[to] = true \; \&\& \; to \neq parent$ - Criterion of reverse rib;
- $to = parent$ - Criterion of passage along the edge of the search tree in the opposite direction.

Thus, for the implementation of these criteria, we need to pass in the search function in the top of the depth of the parent of the current node.

```
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);
}
```

Here, the main function to call - it $\mathrm{find\_bridges}$- it makes the necessary initialization and start crawling in depth for each connected component of the graph.

At the same time $\mathrm{IS\_BRIDGE}(a,b)$- it's a function that will respond to the fact that the edge of (a,b)a bridge, for example, to display it on the screen edge.

Constant MAXNat the beginning of the code should be set equal to the maximum possible number of vertices in the input graph.

It should be noted that this implementation does not work correctly in the presence of the column **multiple edges** : it actually does not pay attention, if multiple edge or it is unique. Of course, multiple edges do not have to go back, so if you call $\mathrm{IS\_BRIDGE}$, you can check further, if not a multiple edge, we want to add to the answer. Another way - a more accurate work with the ancestors, ie transmit to dfs the top is not the parent, and the number of edges, on which we have entered into the top (you will need to additionally store the numbers of all the edges).

Problem in online judges

A list of tasks that need to search for bridges:

- UVA # 796 **"Critical Links"** [Difficulty: Easy]
- UVA # 610 **"Street Directions"** [Difficulty: Medium]

6 Комментариев e-maxx Войти

Лучшее вначале Поделиться Избранный ★

Присоединиться к обсуждению...

mamuka • 6 месяцев назад

ia smatrel na kod i sichas vsio paniatna :)

• Ответить • Поделиться ›

mamuka • 6 месяцев назад

ia vso eshio ne ponial, shto aznachaet fup[v].. esli ktota znaet, ne mojete abiasnic. spasiba :)
sorry for english letters.

• Ответить • Поделиться ›

Аааа • 8 месяцев назад

помогите исправить код так, чтобы он работал правильно при наличии кратных ребер! очень надо

• Ответить • Поделиться ›

veschii_nevstruii → Аааа • 4 месяца назад

Может, вместо кратного ребра ввести фиктивную вершину, которая заменит одно из рёбер.
Алгоритм всё равно не посчитает это мостом, а от кратных рёбер т.о. избавляемся.

• Ответить • Поделиться ›

ffff • год назад

Что описает fup[] можете сказать подробно?

• Ответить • Поделиться ›

Дмитрий → ffff • год назад

По-моему, когда мы рассматриваем ребро (v, to), мы в качестве fup[to] получаем минимальное время входа из всех тех вершин, до которых можно добраться, выйдя из to по любому направлению, отличному от (to, v). Поэтому, если fup[to] <= tin[v], то мы сможем