Dfs

This is one of the basic graph algorithms.

As a result, depth-first search is lexicographically first path in the graph.

The algorithm works in **O (N + M)** .

Application of the algorithm

- Search any path in the graph.
- Search lexicographically first path in the graph.
- Checking whether a node of the tree another ancestor:
- At the beginning and end of the iteration depth-first search will remember the "time" entry and exit at each vertex. Now for the O (1) can find the answer: vertex i is an ancestor of node j if and only if the $start_i < start_j$ and $end_i > end_j$.
- Problem LCA (lowest common ancestor) .
- Topological Sort :
- Run a series of depth-first search to traverse all vertices of the graph. Sort the vertices by time descending exit - this will be the answer.
- Checking on the acyclic graph and finding cycle
- Search Strongly Connected Components :
- Please do topological sort, then transpose the graph again and hold a series of searches in depth in the order determined by the topological sorting. Each search tree - strongly connected component.
- Search bridges :
- first converted into a directed graph, making a series of searches in depth, and orienting each edge as we were trying to pass him. Then we find the strongly connected components. Bridges are those ribs, the ends of which belong to different strongly connected components.

Implementation

```
vector <vector <int>> g; / / Count
int n; / / number of vertices

vector <int> color; / / vertex color (0, 1, or 2)

vector <int> time_in, time_out; / / "times" approach and exit from the top
int dfs_timer = 0, / / "switch" to determine the time

void dfs (int v) {
        time_in [v] = dfs_timer + +;
        color [v] = 1;
        for (vector <int> :: iterator i = g [v]. begin (); i! = g [v]. end (); + + i)
                if (color [* i] == 0)
                        dfs (* i);
        color [v] = 2;
        time_out [v] = dfs_timer + +;
}
```

This is the most common code. In many cases, the time of entry and exit from the top is not important, as well as the vertex colors are not important (but then you have to enter the same within the meaning of the boolean array used). Here are the most simple implementation:

```
vector <vector <int>> g; / / Count
int n; / / number of vertices

vector <char> used;

void dfs (int v) {
        used [v] = true;
        for (vector <int> :: iterator i = g [v]. begin (); i! = g [v]. end (); + + i)
                if (! used [* i])
                        dfs (* i);
}
```