added: 10 Jun 2008 19:19
Edited: 10 Jun 2008 19:21

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)** .

Contents [hide]

Application of the algorithm

- Search for any path in the graph.
- Search lexicographically first path in the graph.
- Checking whether a tree node ancestor another:
  At the beginning and end of the iteration depth-first search will remember the "time" entry and exit at each vertex. Now 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 the vertices. Sort the peaks retention time descending - this will be the answer.
- Checking on the acyclic graph and finding a cycle
- Search Strongly Connected Components :
  First, do a topological sort, then transpose the graph and perform again a series of depth-first search in the manner determined by the topological sorting. Each search tree - strongly connected component.
- Search bridges :
  First convert the graph into a directed, making a series of depth-first search, and orienting each edge as we were trying to pass him. Then we find the strong-components. Bridges are those edges whose endpoints belong to different strongly connected components.

Implementation

```
vector <vector <int>> g; // Graph
int n; // Number of vertices

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

vector <int> time_in, time_out; // "Times" approach and exit from the top
int dfs_timer = 0; // "Timer" for the determination of the time of

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 it will be necessary to introduce a similar within the meaning of the boolean array used). Here are the most simple implementation:

```
vector <vector <int>> g; // Graph
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);
}
```