CP-Algorithms

Page Authors

Search

Topological Sorting

Table of Contents

You are given a directed graph with n vertices and m edges. You have to **number the vertices** so that every edge leads from the vertex with a smaller number assigned to the vertex with a larger one.

In other words, you want to find a permutation of the vertices (**topological order**) which corresponds to the order defined by all edges of the graph.

Topological order can be **non-unique** (for example, if the graph is empty; or if there exist three vertices a, b, c for which there exist paths from a to b and from a to c but not paths from b to c or from c to b).

Topological order may **not exist** at all if the graph contains cycles (because there is a contradiction: there is a path from a to b and vice versa).

A common problem in which topological sorting occurs is the following. There are n variables with unknown values. For some variables we know that one of them is less than the other. You have to check whether these constraints are contradictory, and if not, output the variables in ascending order (if several answers are possible, output any of them). It is easy to notice that this is exactly the problem of finding topological order of a graph with n vertices.

The Algorithm

To solve this problem we will use depth-first search.

Let's assume that the graph is acyclic, i.e. there is a solution. What does the depth-first search do? When started from some vertex v, it tries to run along all edges outgoing from v. It fails to run along the edges for which the opposite ends have been visited previously, and runs along the rest of the edges and starts from their ends.

Thus, by the time of the call $dfs(v)$ is ended, all vertices that are reachable from v either directly (via one edge) or indirectly are already visited by the search. Therefore, if at the time of exit from $dfs(v)$ we add vertex v to the beginning of a certain list, in the end this list will store a topological ordering of all vertices.

These explanations can also be presented in terms of time of exit from DFS routine. Exit time for vertex v is the time at which $dfs(v)$ finished work (the times can be numbered from 1 to n). It is easy to understand that exit time of any vertex v is always greater than exit time of any vertex reachable from it (since they were visited either before the call $dfs(v)$ or during it). Thus, the desired topological ordering is sorting vertices in descending order of their exit times.

Implementation

Here is an implementation which assumes that the graph is acyclic, i.e. the desired topological ordering exists. If necessary, you can easily check that the graph is acyclic, as described in the article on depth-first search.

C++ implementation Show/Hide

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

The main function of the solution is `topological_sort`, which initializes DFS variables, launches DFS and receives the answer in the vector `ans`.

Practice Problems

- SPOJ TOPOSORT "Topological Sorting" [difficulty: easy]
- UVA #10305 "Ordering Tasks" [difficulty: easy]
- UVA #124 "Following Orders" [difficulty: easy]
- UVA #200 "Rare Order" [difficulty: easy]
- Codeforces 510C "Fox and Names" [difficulty: easy]