Topological Sort

Given a directed graph with $n$vertices and $m$edges. Required **to renumber** the vertices so that each edge is led from the top with a lower number in the top with a lot.

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

Topological sorting can be **not only** (for example, if the graph - empty, or if there are three such peaks a, b, $c$that of $a$there way to $b$and in c, but none of the $b$in $c$or out of ca $b$can not be reached).

Topological sorting may **not exist** at all - if the graph contains cycles (as there is a contradiction: there is a way and from one vertex to another, and vice versa).

A common problem on a topological sort - next. There are $n$variables that are unknown to us. We only know about some of the pairs of variables that one variable is less than another. Need to check whether these are not contradictory inequality, and if not, give the variables in ascending order (if there are several solutions - to give any). Easy to see that this is exactly is the problem of finding topological sort of the graph n vertices.

Algorithm

Solutions for use in bypass depth .

Assume that the graph is acyclic, ie solution exists. What makes a detour into the depths? When you run out of some vertex $v$he tries to run along all edges emanating from v. Along the edges, the ends of which have already been visited before, it does not pass, and along all the others - calls itself and passes on their ends.

Thus, by the time the call $\mathrm{dfs}(v)$all the vertices reachable from $v$both directly (one edge) and indirectly (by the way) - all such vertices already visited bypass.Consequently, if we are at the moment out of $\mathrm{dfs}(v)$our top add to the beginning of a list, then in the end of this list will **be a topological sorting** .

These explanations can be presented in a slightly different light, using the concept of **"time release"** crawl deep. Retention time for each vertex v- this is the point in time at which the call ended up working $\mathrm{dfs}(v)$in the crawl depth of it (retention times can be numbered from $1$before n). Easy to understand that when crawling into the depths of time to some vertex $v$is always greater than the output of all the vertices reachable from it (as they have been visited or to call $\mathrm{dfs}(v)$or during it).Thus, the desired topological sorting - sorting in descending order of retention times.

Implementation

Show implementation, which implies that the graph is acyclic, ie desired topological sort exists. If necessary check on the acyclic graph easily inserted into the bypass in depth, as described in the article on the circuit in depth .

```
int n; // число вершин
vector<int> g[MAXN]; // граф
bool used[MAXN];
vector<int> ans;

void dfs (int v) {
	used[v] = true;
	for (size_t i=0; i<g[v].size(); ++i) {
		int to = g[v][i];
		if (!used[to])
			dfs (to);
	}
	ans.push_back (v);
}

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

Here the constant MAXNvalue should be set equal to the maximum possible number of vertices in the graph.

The main function of a solution - it topological_sort, it initializes tagging crawl deep, runs it, and the result is a response vector ans.

Problem in online judges

List of tasks that require topological sort search:

- UVA # 10305 **"Ordering Tasks"** [Difficulty: Easy]
- UVA # 124 **"Following Orders"** [Difficulty: Easy]
- UVA # 200 **"Rare Order"** [Difficulty: Easy]