Search algorithm of connected components in a graph

Given an undirected graph $G$with $n$vertices and $m$edges. You want to find in it all the connected components, iesplit vertices into several groups so that within a group can be reached from one vertex to any other, and between different groups - the path does not exist.

An algorithm for solving

Alternatively you can use as a bypass in-depth and wide detour .

In fact, we will produce **a series of rounds** : first run of bypassing the first vertex and all vertices that while he walked - form the first connected component. Then find the first of the remaining vertices that have not yet been visited and run circumvention of it, thus finding a second connected component. And so on, until all the vertices will not be marked.

Total **asymptotics** be $O(n+m)$: in fact, such an algorithm will not start from the same vertex twice, which means that each edge will be seen exactly twice (at one end and the other end).

Implementation

To implement a little more convenient to bypass the in depth :

```
int n;
vector<int> g[MAXN];
bool used[MAXN];
vector<int> comp;

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

```
void find_comps() {
     for (int i=0; i<n; ++i)
          used[i] = false;
     for (int i=0; i<n; ++i)
          if (! used[i]) {
               comp.clear();
               dfs (i);

               cout << "Component:";
               for (size_t j=0; j<comp.size(); ++j)
                    cout << ' ' << comp[j];
               cout << endl;
          }
}
```

The main function to call - $\mathrm{find_comps}()$she finds and displays the components of the graph.

We believe that given the graph adjacency lists, ie $g[i]$a list of vertices that have edges from the top i. Constant MAXNvalue should be set equal to the maximum possible number of vertices in the graph.

Vector compcontains a list of vertices in the current connected component.