```
#include "Graph.h"

void Graph::dfsUtil(int v, bool* visited)
{
	visited[v] = true;
	cout << v << " , ";

	list<int> ::iterator i;
	for (i = adj[v].begin(); i != adj[v].end(); ++i)
		if (!visited[*i])
			dfsUtil(*i, visited);

}

void Graph::fillOrder(int v, bool* visited, stack<int>& s)
{
	visited[v] = true;

	list<int> ::iterator i;
	for (i = adj[v].begin(); i != adj[v].end(); ++i)
		if (!visited[*i])
			fillOrder(*i, visited,s);

	s.push(v);

}

/*Graph Graph::transpose()
{
	Graph tg = Graph(V);
	list<int>::iterator j;
	for (int i = 0; i < V; i++)
		for (j = this->adj[i].begin(); j != this->adj[i].end(); ++j)
			tg.adj[*j].push_back(i);

	return tg;
}
*/
Graph Graph::transpose()
{
	Graph tg = Graph(V);
	list<int>::iterator j;
	for (int i = 0; i < V; i++)
```

```
                for (j = this->adj[i].begin(); j != this->adj[i].end(); ++j)
                        tg.addEdge(*j, i);

        return tg;
}
Graph::Graph(int v)
{
        V = v;
        adj = new list<int>[v];
}

void Graph::addEdge(int u, int v)
{
        adj[u].push_back(v);
}

void Graph::SCCs()
{
        bool* visited = new bool[V];
        for (int i = 0; i < V; i++)
                visited[i] = false;
        stack<int> s;

        for (int i = 0; i < V; i++)
                if (!visited[i])
                        fillOrder(i, visited, s);
        Graph t = this->transpose();

        for (int i = 0; i < V; i++)
                visited[i] = false;
        while (!s.empty()) {
                int u = s.top();
                s.pop();
                if (!visited[u]){
                        t.dfsUtil(u, visited);
                        cout << endl;
                        }
        }



}
```

```
void Graph::topSort()
{
	int noVV = 0;
	int* in_deg = new int[V];
	int* top = new int[V];
	Graph t = this->transpose();
	queue<int> q;
	for (int i = 0; i < V; i++) {
		in_deg[i] = t.adj[i].size();
		if (in_deg[i] == 0)
			q.push(i);

	}

	while (!q.empty()) {
		int u = q.front();
		q.pop();
		top[noVV] = u;
		noVV++;

		list<int>::iterator i;
		for (i = adj[u].begin(); i != adj[u].end(); ++i) {
			in_deg[*i]--;
			if (in_deg[*i] == 0)
				q.push(*i);
		}
	}
	if (noVV < V - 1)
		cout << "Nu s-a putut realiza sortarea topologica";
	else
		for (int i = 0; i < noVV; i++)
			cout << top[i] << " , ";
}
```