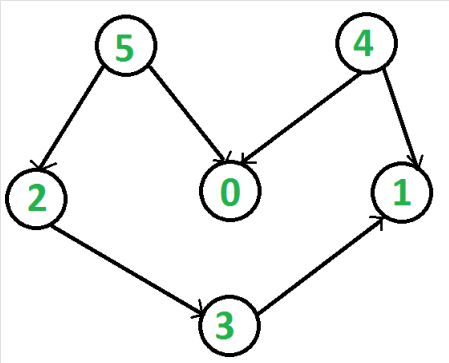


All Topological Sorts of a Directed Acyclic Graph

Topological sorting for **D**irected **A**cyclic **G**raph (DAG) is a linear ordering of vertices such that for every directed edge uv , vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

Given a DAG, print all topological sorts of the graph.

For example, consider the below graph.



All topological sorts of the given graph are:

```
4 5 0 2 3 1
4 5 2 0 3 1
4 5 2 3 0 1
4 5 2 3 1 0
4 5 2 3 1 0
5 2 3 4 0 1
5 2 3 4 1 0
5 2 3 4 1 0
5 2 4 0 3 1
5 2 4 3 0 1
5 2 4 3 0 1
5 2 4 3 1 0
5 4 0 2 3 1
5 4 2 0 3 1
5 4 2 3 0 1
5 4 2 3 1 0
```

In a Directed acyclic graph many a times we can have vertices which are unrelated to each other because of which we can order them in many ways. These various topological sorting is important in many cases, for example if some relative weight is also available between the vertices, which is to minimize then we need to take care of relative ordering as well as their relative weight, which creates the need of checking through all possible topological ordering.

We can go through all possible ordering via backtracking, the algorithm step are as follows :

1. Initialize all vertices as unvisited.
2. Now choose vertex which is unvisited and has zero indegree and decrease indegree of all those vertices by 1 (corresponding to removing edges) now add this vertex to result and call the recursive function again and backtrack.
3. After returning from function reset values of visited, result and indegree for enumeration of other possibilities.

Below is C++ implementation of above steps.

```
// C++ program to print all topological sorts of a graph
#include <bits/stdc++.h>
using namespace std;

class Graph
{
    int V;    // No. of vertices

    // Pointer to an array containing adjacency list
```

```

list<int> *adj;

// Vector to store indegree of vertices
vector<int> indegree;

// A function used by alltopologicalSort
void alltopologicalSortUtil(vector<int>& res,
                           bool visited[]);

public:
    Graph(int V);    // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // Prints all Topological Sorts
    void alltopologicalSort();
};

// Constructor of graph
Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];

    // Initialising all indegree with 0
    for (int i = 0; i < V; i++)
        indegree.push_back(0);
}

// Utility function to add edge
void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.

    // increasing inner degree of w by 1
    indegree[w]++;
}

// Main recursive function to print all possible
// topological sorts
void Graph::alltopologicalSortUtil(vector<int>& res,
                                   bool visited[])
{
    // To indicate whether all topological are found
    // or not
    bool flag = false;

    for (int i = 0; i < V; i++)
    {
        // If indegree is 0 and not yet visited then
        // only choose that vertex
        if (indegree[i] == 0 && !visited[i])
        {
            // reducing indegree of adjacent vertices
            list<int>::iterator j;
            for (j = adj[i].begin(); j != adj[i].end(); j++)
                indegree[*j]--;

            // including in result
            res.push_back(i);
            visited[i] = true;
            alltopologicalSortUtil(res, visited);

            // resetting visited, res and indegree for
            // backtracking
            visited[i] = false;
            res.erase(res.end() - 1);
            for (j = adj[i].begin(); j != adj[i].end(); j++)
                indegree[*j]++;

            flag = true;
        }
    }
}

```

```

    }
}

// We reach here if all vertices are visited.
// So we print the solution here
if (!flag)
{
    for (int i = 0; i < res.size(); i++)
        cout << res[i] << " ";
    cout << endl;
}
}

// The function does all Topological Sort.
// It uses recursive alltopologicalSortUtil()
void Graph::alltopologicalSort()
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    vector<int> res;
    alltopologicalSortUtil(res, visited);
}

// Driver program to test above functions
int main()
{
    // Create a graph given in the above diagram
    Graph g(6);
    g.addEdge(5, 2);
    g.addEdge(5, 0);
    g.addEdge(4, 0);
    g.addEdge(4, 1);
    g.addEdge(2, 3);
    g.addEdge(3, 1);

    cout << "All Topological sorts\n";

    g.alltopologicalSort();

    return 0;
}

```

Output :

```

All Topological sorts
4 5 0 2 3 1
4 5 0 3 1
4 5 2 0 3 1
4 5 2 3 0 1
4 5 2 3 1 0
5 2 3 4 0 1
5 2 3 4 1 0
5 2 4 0 3 1
5 2 4 3 0 1
5 2 4 3 1 0
5 4 0 2 3 1
5 4 2 0 3 1
5 4 2 3 0 1
5 4 2 3 1 0

```