**TOPO SORT AND PROBLEMS GRAPHS**

Q.1 <https://www.naukri.com/code360/problems/topological-sort_982938>

* For topological sort can be happen only if graph is directed Acyclic graph and here we have take out vertex order wise if 0 is before 6 in path then 0 come before 6
* So here we used simple dfs also took stack when dfs is completed we just store that node
* At last copy stack element into vector and return it

|  |
| --- |
| #include <bits/stdc++.h>  void dfs(vector<int> adj[],vector<int> &vis,stack<int>& st,int node)  {      vis[node]=1;      for(auto it:adj[node])      {          if(!vis[it])          {              dfs(adj,vis,st,it);          }      }      st.push(node);  }  vector<int> topologicalSort(vector<vector<int>> &edges, int v, int e)  {      // Write your code here      stack<int> st;      vector<int> ans;      vector<int> vis(v,0);      vector<int> adj[v];      for(int i=0;i<e;i++)      {          adj[edges[i][0]].push\_back(edges[i][1]);      }        for(int i=0;i<v;i++)      {          if(!vis[i])          {              dfs(adj,vis,st,i);          }      }      while(!st.empty())      {          ans.push\_back(st.top());          st.pop();      }      return ans;    } |

TC: O(V+E)

SC:O(V)

Q.2 <https://www.geeksforgeeks.org/problems/topological-sort/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=topological-sort>

[Kahn's Algorithm](https://takeuforward.org/data-structure/kahns-algorithm-topological-sort-algorithm-bfs-g-22/)

* This is a topo sort only but we are going to solve it in BFS way that’s why this called kahn’s Algorithm
* Here we store indegree of each node in array and initialize array first with zero
* Then declare queue and iterate through degree array check who has degree zero push into queue
* Now simply use bfs pop out element store into ans vector after that iterate through adjacent node and decrement degree of adjacent and check it any of adjacent node degree gets zero then add into queue

|  |
| --- |
| class Solution  {  public:  //Function to return list containing vertices in Topological order.  vector<int> topoSort(int V, vector<int> adj[])  {  // code here  int indegree[V]={0};    for(int i=0;i<V;i++)  {  for(auto it:adj[i])  {  indegree[it]++;  }  }    queue<int> q;  vector<int> topo;    for(int i=0;i<V;i++)  {  if(indegree[i]==0)  {  q.push(i);  }  }    while(!q.empty())  {  int node=q.front();  q.pop();  topo.push\_back(node);      for(auto it:adj[node])  {  indegree[it]--;  if(indegree[it]==0) q.push(it);  }  }    return topo;  }  };  //{ Driver Code Starts.  /\* Function to check if elements returned by user  \* contains the elements in topological sorted form  \* V: number of vertices  \* \*res: array containing elements in topological sorted form  \* adj[]: graph input  \*/  int check(int V, vector <int> &res, vector<int> adj[]) {    if(V!=res.size())  return 0;    vector<int> map(V, -1);  for (int i = 0; i < V; i++) {  map[res[i]] = i;  }  for (int i = 0; i < V; i++) {  for (int v : adj[i]) {  if (map[i] > map[v]) return 0;  }  }  return 1;  }  int main() {  int T;  cin >> T;  while (T--) {  int N, E;  cin >> E >> N;  int u, v;  vector<int> adj[N];  for (int i = 0; i < E; i++) {  cin >> u >> v;  adj[u].push\_back(v);  }    Solution obj;  vector <int> res = obj.topoSort(N, adj);  cout << check(N, res, adj) << endl;  }    return 0;  }  // } Driver Code Ends |

TC: O(V+E)

SC:O(V)

Q.3 <https://www.naukri.com/code360/problems/detect-cycle-in-a-directed-graph_1062626?leftPanelTabValue=PROBLEM>

* Here we have to detect cycle in directed graph
* So here we are using same logic of topological sort using BFS or kahan’s Algorithm
* Just instead of using vector storing it just cnt how many timing we are poping it from queue
* At last if cnt==V return true or else false

|  |
| --- |
| #include<queue>  int detectCycleInDirectedGraph(int V, vector < pair < int, int >> & edges) {    // Write your code here.    vector<int> adj[V+1];    for(int i=0;i<edges.size();i++)    {      adj[edges[i].first].push\_back(edges[i].second);    }     int indegree[V+1]={0};          for(int i=0;i<V;i++)        {          for(auto it:adj[i+1])          {            indegree[it]++;          }          }          queue<int> q;        int cnt=0;          for(int i=0;i<V;i++)        {            if(indegree[i+1]==0)            {                q.push(i+1);            }        }              while(!q.empty())            {                int node=q.front();                q.pop();                cnt++;                    for(auto it:adj[node])                {                    indegree[it]--;                    if(indegree[it]==0) q.push(it);                }            }            if(cnt==V) return 0;            return 1;      } |

TC:O(V+E)

SC:O(2V)

* Same question we can do it using BFS

Q <https://www.geeksforgeeks.org/problems/detect-cycle-in-a-directed-graph/1>

* Here we are going to maintain vis and pathvis array and call for each vertex if any of this return true then true or else after for it false
* So in dfs first mark vis and pathvis node =1
* Then call for adjacent check if not vis then call and check if it’s return true then you return true

Else if if any node visited vis and also in Pathvis[i]==1 then return true that means it’s coming from same path

* After for loop just do pathvis[node]=0 so will erase the path when we go
* At last return false

|  |
| --- |
| * class Solution { * public: * bool dfs(vector<int> adj[],int vis[],int pathVis[],int node) * { * vis[node]=1; * pathVis[node]=1; * for(auto it:adj[node]) * { * if(!vis[it]) * { * if(dfs(adj,vis,pathVis,it)==true) return true; * } * else if(pathVis[it]) * { * return true; * } * } * pathVis[node]=0; * return false; * } * // Function to detect cycle in a directed graph. * bool isCyclic(int V, vector<int> adj[]) { * // code here * int vis[V]={0}; * int pathVis[V]={0}; * for(int i=0;i<V;i++) * { * if(!vis[i]) * { * if(dfs(adj,vis,pathVis,i)==true) return true; * } * } * return false; * } * }; * //{ Driver Code Starts. * int main() { * int t; * cin >> t; * while (t--) { * int V, E; * cin >> V >> E; * vector<int> adj[V]; * for (int i = 0; i < E; i++) { * int u, v; * cin >> u >> v; * adj[u].push\_back(v); * } * Solution obj; * cout << obj.isCyclic(V, adj) << "\n"; * } * return 0; * } * // } Driver Code Ends |

TC:O(V+E)

SC:O(2V)

Q.4 <https://leetcode.com/problems/course-schedule/>

* There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you **must** take course bi first if you want to take course ai.
* For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.
* Return true if you can finish all courses. Otherwise, return false.
* Logic
* So here we used detect cycle in DAG DFS approach

|  |
| --- |
| class Solution {  public:      bool dfs(vector<int> adj[],vector<int> &vis,vector<int>& pathVis,int node)      {          vis[node]=1;          pathVis[node]=1;          for(auto it:adj[node])          {              if(!vis[it])              {                  if(dfs(adj,vis,pathVis,it)==true) return true;              }              else if(pathVis[it]==1) return true;          }          pathVis[node]=0;          return false;      }      bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {          vector<int> vis(numCourses,0);          vector<int> pathVis(numCourses,0);            vector<int> adj[numCourses];          int n=prerequisites.size();          for(int i=0;i<n;i++)          {              adj[prerequisites[i][0]].push\_back(prerequisites[i][1]);          }          for(int i=0;i<numCourses;i++)          {              if(!vis[i])              {                  // if graph has cycle it won't complete course that's why false                  if(dfs(adj,vis,pathVis,i)==true) return false;              }          }          return true;        }  }; |

TC:O(V+E) SC:O(2V)

Q.5 <https://leetcode.com/problems/course-schedule-ii/description/>

🡪 same question like course 1 but here we have to return sorted list of course

🡪 Return *the ordering of courses you should take to finish all courses*. If there are many valid answers, return **any** of them

* Here I have used BFS approach of topological sort that is kahn’s Algorithm
* Just while coverting into adjacency list just make sure to take in reverse order because for completing [a,b] (a) course you must have completed b

That means b is before a b->a

|  |
| --- |
| class Solution {  public:      vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {          vector<int> adj[numCourses];          int n=prerequisites.size();          vector<int> indegree(numCourses,0);          queue<int> q;          vector<int> topo;          for(int i=0;i<n;i++)          {              adj[prerequisites[i][1]].push\_back(prerequisites[i][0]);          }          for(int i=0;i<numCourses;i++)          {              for(auto it:adj[i])              {                  indegree[it]++;              }          }          for(int i=0;i<numCourses;i++)          {              if(indegree[i]==0) q.push(i);          }          while(!q.empty())          {              int node=q.front();              q.pop();              topo.push\_back(node);              for(auto it:adj[node])              {                  indegree[it]--;                  if(indegree[it]==0) q.push(it);              }          }          if(topo.size()==numCourses) return topo;          return {};        }  }; |

TC:O(V+E) SC:O(2V)

Q.6 <https://leetcode.com/problems/find-eventual-safe-states/description/>

* here is a directed graph of n nodes with each node labeled from 0 to n - 1. The graph is represented by a **0-indexed** 2D integer array graph where graph[i] is an integer array of nodes adjacent to node i, meaning there is an edge from node i to each node in graph[i].
* A node is a **terminal node** if there are no outgoing edges. A node is a **safe node** if every possible path starting from that node leads to a **terminal node** (or another safe node).
* Return *an array containing all the****safe nodes****of the graph*. The answer should be sorted in **ascending** order.
* So The logic is
* Here we have using topological sort that is kahn’s Algorithm
* But there is slight change we have to reverse the direction and then store indegree and it’s quite simple while storing adjacency list just instead of stroing adj[node].push\_back(it) you should do adj[it].push\_back(I or node)

And do indegree[i]++

|  |
| --- |
| class Solution {  public:      vector<int> eventualSafeNodes(vector<vector<int>>& graph) {          int n=graph.size();          vector<int> adjRev[n];          vector<int> indegree(n,0);          for(int i=0;i<n;i++)          {              // edge i-->it              // for rev it-->i              for(auto it:graph[i])              {                  adjRev[it].push\_back(i);                  indegree[i]++;              }          }          queue<int> q;          vector<int> safeNode;          for(int i=0;i<n;i++)          {              if(indegree[i]==0) q.push(i);          }          while(!q.empty())          {              int node=q.front();              q.pop();              safeNode.push\_back(node);              for(auto it: adjRev[node])              {                  indegree[it]--;                  if(indegree[it]==0) q.push(it);              }          }          sort(safeNode.begin(),safeNode.end());          return safeNode;        }  }; |

TC:O(V+E) SC:O(2V)

Q.7 <https://www.naukri.com/code360/problems/alien-dictionary_630423>

You have been given a sorted (lexical order) dictionary of an alien language.

Write a function that returns the order of characters as a string in the alien language. This dictionary will be given to you as an array of strings called ***'dictionary'***, of size ***'N'***.

**Example :**

If the dictionary consists of the following words:-

["caa", "aaa", "aab"], and 'K' is 3.

Then, the order of the alphabet is -

['c', 'a', 'b']

* This alien dict is given to us from that we have get alphabet order
* Here we’ll use kahn’s algorithm topo sort but some other logic as well
* For iterate through dictionary till N-1 then store two word in s1=word[i]

S2=word[i+1]

* Take out min length from both string
* Now compare each char of string if any of the first char is not equal
* Then store into adj list adj[str1[ptr]-‘a’].push\_back(str2[prt]-‘a’)

And do break we don’t want compare it further

* Like that store adj list and adj[k] size
* Call topo sort of BFS
* It will return vector
* Iterate through that vector and ans=ans+char(‘a’+it)
* Return ans (string)

|  |
| --- |
| #include<queue>  vector<int> topoSort(vector<int> adj[],int V)  {      int indegree[V]={0};      queue<int> q;      vector<int> topo;      for(int i=0;i<V;i++)      {          for(auto it:adj[i])          {              indegree[it]++;          }      }      for(int i=0;i<V;i++)      {          if(indegree[i]==0)          {              q.push(i);          }      }      while(!q.empty())      {          int node=q.front();          q.pop();          topo.push\_back(node);          for(auto it:adj[node])          {              indegree[it]--;              if(indegree[it]==0) q.push(it);          }      }      return topo;  }  string getAlienLanguage(vector<string> &dictionary, int k)  {      // Write your code here.      int n=dictionary.size();      vector<int> adj[k];      for (int i = 0; i < n - 1; i++) {        string s1 = dictionary[i];        string s2 = dictionary[i + 1];        int len = min(s1.size(), s2.size());        for (int ptr = 0; ptr < len; ptr++)        {            if(s1[ptr]!=s2[ptr])            {                adj[s1[ptr]-'a'].push\_back(s2[ptr]-'a');                break;            }        }      }      vector<int> topo=topoSort(adj,k);      string ans="";      for(auto it:topo)      {          ans=ans+char(it+'a');      }      return ans;  } |

TC: O(n\*len) +O(V+E) +O(K)

SC:O(2v)