**Algorithm Lab Assignment 01 (Answer Script)**

| Question No. 01` |
| --- |
| Write code to solve the single source shortest path problem on a **DAG** using **DFS**. Take both the **DAG** and the **source node** as input and output the **distance of each node**. (You can choose any graph representation or input format of your choice) |
| Answer:  #include<bits/stdc++.h>  using namespace std;  const int N=2e5;  int visited[N];  int level[N];  vector<int>adj\_list[N];  // sssp using dfs on DAG  void sssp(int node){  visited[node]=1;  // level[node]=0; //src node=0  for(int adj\_node: adj\_list[node]){  // cout<<"node: "<<node<<" adj\_node: "<<adj\_node<<endl;  if(visited[adj\_node]==0){  level[adj\_node]=level[node]+1;  //cout<<"node: "<<node<<" adj\_node: "<<adj\_node<<endl;  //cout<<"level[node] "<<level[node]<<" level[adj] "<< level[adj\_node]<<endl;  sssp(adj\_node);  }  }  }  int main(){  int n,m;// node, edge  cin>>n>>m;  for(int i=0; i<m; i++){  int u,v;  cin>>u>>v;  adj\_list[u].push\_back(v);//directed graph  }  int src=0;  sssp(src);  //print level of nodes  for(int i=0; i<n; i++){  cout<<"node: "<<i<<" level: "<<level[i]<<endl;  }  } |

| Question No. 02 |
| --- |
| Write code to solve the following grid traversal problem. You don’t need to print the path. <https://cses.fi/problemset/task/1194> |
| Answer:  #include<bits/stdc++.h>  using namespace std;  const int N=2002;  int maze[N][N];  int visited[N][N];  int level[N][N];  int n,m;  int dx[]={0, 0, -1, 1};  int dy[]={1, -1, 0, 0};  // within the boundary?  bool is\_inside(pair<int,int>coord){  int x=coord.first;  int y=coord.second;  if((x>=0 && x<n) && (y>=0 && y<m)){ // within boundaey  return true;  }  return false;  }  //forbidden cell- wall/ monster  bool is\_safe(pair<int,int>coord){  int x=coord.first;  int y= coord.second;  if(maze[x][y]==-1){  return false;  }  return true;  }  //bfs  bool flag=false;  void bfs(pair<int, int>src){  queue<pair<int, int>>q;  visited[src.first][src.second]=1;  level[src.first][src.second]=0;  q.push(src);  while(!q.empty()){  pair<int, int>head;  head=q.front();  q.pop();  int x=head.first;  int y=head.second;  // adj nodes in left, right, up, down  for(int i=0; i<4; i++){  int new\_x=x+dx[i]; // new co.ordinate  int new\_y=y+dy[i];  pair<int, int>adj\_node={new\_x,new\_y};  //push after checking  if(is\_inside(adj\_node) && is\_safe(adj\_node) && visited[new\_x][new\_y]==0){  visited[new\_x][new\_y]=1;  level[new\_x][new\_y]=level[x][y]+1;  q.push(adj\_node);  // if reached boundary, ans and break the loop  if((new\_x==0 || new\_x==n-1) || (new\_y==0 || new\_y==m-1)){  flag=true;  cout<<"possible"<<endl;  cout<<"steps: "<<level[new\_x][new\_y]<<endl;  break;  }  }  }  }  if(flag==false){  cout<<"not possible"<<endl;  }  }  int main(){  cin>>n>>m;  pair<int,int>src;  for(int i=0; i<n; i++){  for(int j=0; j<m; j++){  level[i][j]=-1;  }  }  //input  for(int i=0; i<n; i++){  string input;  cin>>input;  for(int j=0; j<m; j++){  if(input[j]=='#' || input[j]=='M'){  maze[i][j]=-1;  }  else if(input[j]=='A'){  src={i,j};  }  }  }  //input check  // for(int i = 0 ; i < n ; i++) {  // for(int j = 0; j < m ; j++) {  // cout<<maze[i][j]<<"\t";  // }  // cout<<endl;  // }  // cout<<endl;    //run bfs  bfs(src);  } |

| Question No. 03 |
| --- |
| Write code to solve **cycle detection** in a **directed graph** using **BFS.** |
| Answer:  #include<bits/stdc++.h>  using namespace std;  const int N=2e5;  int visited[N];  vector<int>adj\_list[N];  // vector<int>indegree[N];  void top\_sort(int nodes, vector<int>adj\_list[N]){    // checking  // for(int i=1; i<=nodes; i++){  // cout<<i<<" -> ";  // for(int j=0; j<adj\_list[i].size(); j++){  // cout<<adj\_list[i][j]<<" ";  // }  // cout<<endl;  // }  queue<int>q;  vector<int>indegree(nodes+1,0); // initial values to zero  for(int i=1; i<=nodes; i++){  for(int j=0; j<adj\_list[i].size(); j++){  indegree[adj\_list[i][j]]++;  }  }  // // check top sorting  // for(int i=1; i<=nodes; i++){  // cout<<indegree[i]<<" ";  // }  // insert first node into queue which has indegress==0  for(int i=1; i<=nodes; i++){  if(indegree[i]==0){  q.push(i);  // cout<<i<<endl;  }  }  stack<int>st; // for counting nodes  queue<int>output; // for output  while(!q.empty()){  int head=q.front();  q.pop();  st.push(head); // this is for counting  output.push(head); // this is for output print  // cout<<head<<" ";  // decrement indegree by 1 of child nodes  for(int adj\_node: adj\_list[head]){  // cout<<adj\_node<<" ";  indegree[adj\_node]--;  if(indegree[adj\_node]==0){  q.push(adj\_node);  }  }  // again check indegree  // for(int i=1; i<=nodes; i++){  // cout<<indegree[i]<<" ";  // }  // cout<<endl;  }  // count nodes to check cycle exists or not  int count=0;  while(!st.empty()){  int head=st.top();  count++;  st.pop();  // cout<<head<<" ";  }  if(count==nodes){ // print nodes  cout<<"no cycle detected"<<endl;  while(!output.empty()){  int head=output.front();  output.pop();  cout<<head<<" ";  }  }  else{  cout<<"cycle detected"<<endl;  }  }  int main(){  int n,m;// node, edge  cin>>n>>m;  for(int i=0; i<m; i++){  int u,v;  cin>>u>>v;  adj\_list[u].push\_back(v); // directed  }  top\_sort(n,adj\_list);  } |

| Question No. 04 |
| --- |
| Write code to solve cycle detection in a undirected graph using DFS.  name: round trip, https://cses.fi/problemset/task/1669 |
| Answer:  /\*  logic:  needs to keep track of the parent node.  - visited[adj\_node] == 0 -> unexplored node |apply dfs  - visited[adj\_node] == 1 -> "paused" node | cycle detected  - visited[adj\_node] == 1 and adj\_node==parent node -> still no cycle  - visited[adj\_node] == 1 and adj\_node!=parent node -> cycle detected  \*/  #include<bits/stdc++.h>  using namespace std;  const int N=2e5;  vector<int>adj\_list[N];  int visited[N];  bool detect\_cycle(int node,int parent){  // cout<<node<<" "<<parent<<endl;  visited[node]=1; // selected node=1  for(int adj\_node: adj\_list[node]){  //cout<<"parent: "<<parent<<" node:"<<node<<" adj\_node: "<<adj\_node<<endl;  // if new node  if(visited[adj\_node]==0){  bool got\_cycle=detect\_cycle(adj\_node,node); // return true if childs has cycle  if(got\_cycle){  return true;  }  }  //if paused node,already visited node and adj\_node !=parent node: detect cycle  else if(visited[adj\_node]==1 && adj\_node!=parent){  // cout<<"true condition matched: "<<parent<<" "<<node<<" "<<adj\_node<<endl;  return true;  }  // // if done node  // else if(visited[adj\_node]==2){  // continue;  // }  }  // if for loop ends, no "true" is returned, return false, no cycle  // visited[node]=2;  return false;  }  int main(){  int n,m;  cin>>n>>m;  for(int i=0; i<m; i++){  int u,v;  cin>>u>>v;  adj\_list[u].push\_back(v);  adj\_list[v].push\_back(u);// undirected graph  }  // detect cycle  bool has\_cycle=false;  for(int i=1; i<=n; i++){  if(visited[i]==0){ // if new node, call DSF  cout<<endl;  bool got\_cycle=detect\_cycle(i,-1); //true, if cycle, (node, parent\_node=-1)  if(got\_cycle){  has\_cycle=true;  break;  }  }  }  if(has\_cycle){  cout<<"cycle exists"<<endl;  }  else{  cout<<"no cycle"<<endl;  }  } |

| Question No. 05 |
| --- |
| Write code to solve the problem https://cses.fi/problemset/task/1669 again using BFS. Can you come up with your own algorithm? |
| Answer:  #include<bits/stdc++.h>  using namespace std;  const int N=2e5;  vector<int>adj\_list[N];  int visited[N];  bool detect\_cycle\_bfs(int node, int parent){  queue<pair<int, int>>q; // node, paraent/previous node  visited[node]=1;  q.push({node,-1}); //node, parent  while(!q.empty()){  int head=q.front().first; //node  int parent=q.front().second; //parent  q.pop();  for(int adj\_node: adj\_list[head]){  if(visited[adj\_node]==0){// new node  visited[adj\_node]=1; // make visited, then push  q.push({adj\_node, head}); // node, parent or previous node  // cout<<"parent: "<<parent<<" head: "<<head<<" adj\_node: "<<adj\_node<<endl;  }  // if finds, already visited node and node != parent node: cycle detected  else if(visited[adj\_node]==1 && adj\_node!=parent){  // cout<<"true condition "<<parent<<" "<<head<<" "<<adj\_node<<endl;  return true;  }  }  }  // visited[node]=2;  // cout<<node<<"- "<<visited[node]<<endl;  return false; // no cycle  }  int main(){  int n,m;  cin>>n>>m;  for(int i=0; i<m; i++){  int u,v;  cin>>u>>v;  adj\_list[u].push\_back(v);  adj\_list[v].push\_back(u);// undirected graph  }  // detect cycle  bool has\_cycle=false;  for(int i=1; i<=n; i++){  if(visited[i]==0){ // if new node, call BFS  cout<<endl;  bool got\_cycle=detect\_cycle\_bfs(i,-1); //true, if cycle, (node, parent\_node=-1)  if(got\_cycle){  has\_cycle=true;  break;  }  }  }  if(has\_cycle){  cout<<"cycle exists"<<endl;  }  else{  cout<<"no cycle"<<endl;  }  } |

| Question No. 06 |
| --- |
| Write code to solve the topological sorting problem using BFS.  <https://cses.fi/problemset/task/1679/>  Can you give an intuitive description of why this algorithm works? |
| Answer:  #include<bits/stdc++.h>  using namespace std;  const int N=2e5;  int visited[N];  vector<int>adj\_list[N];  // vector<int>indegree[N];  void top\_sort(int nodes, vector<int>adj\_list[N]){    // checking  // for(int i=1; i<=nodes; i++){  // cout<<i<<" -> ";  // for(int j=0; j<adj\_list[i].size(); j++){  // cout<<adj\_list[i][j]<<" ";  // }  // cout<<endl;  // }  queue<int>q;  vector<int>indegree(nodes+1,0); // initial values to zero  for(int i=1; i<=nodes; i++){  for(int j=0; j<adj\_list[i].size(); j++){  indegree[adj\_list[i][j]]++;  }  }  // // check top sorting  // for(int i=1; i<=nodes; i++){  // cout<<indegree[i]<<" ";  // }  // insert first node into queue which has indegress==0  for(int i=1; i<=nodes; i++){  if(indegree[i]==0){  q.push(i);  // cout<<i<<endl;  }  }  stack<int>st; // for counting nodes  queue<int>output; // for output  while(!q.empty()){  int head=q.front();  q.pop();  st.push(head); // this is for counting  output.push(head); // this is for output print  // cout<<head<<" ";  // decrement indegree by 1 of child nodes  for(int adj\_node: adj\_list[head]){  // cout<<adj\_node<<" ";  indegree[adj\_node]--;  if(indegree[adj\_node]==0){  q.push(adj\_node);  }  }  // again check indegree  // for(int i=1; i<=nodes; i++){  // cout<<indegree[i]<<" ";  // }  // cout<<endl;    }  // count nodes to check cycle exists or not  int count=0;  while(!st.empty()){  int head=st.top();  count++;  st.pop();  // cout<<head<<" ";  }  if(count==nodes){ // print nodes  // cout<<count<<endl;  while(!output.empty()){  int head=output.front();  output.pop();  cout<<head<<" ";  }  }  else{  cout<<"IMPOSSIBLE"<<endl;  }  }  int main(){  int n,m;// node, edge  cin>>n>>m;  for(int i=0; i<m; i++){  int u,v;  cin>>u>>v;  adj\_list[u].push\_back(v);  }  top\_sort(n,adj\_list);  } |
| Intuitive description:  In this implementation with BFS, we first calculate the in degrees of all the nodes. That means we count how many inbound nodes are there to a particular node. When we find a node with in degree 0, we insert it to a queue. And decrement all the adjacent nodes of that particular parent node by 1. That means, we are collecting the parent nodes first and then the child node. That’s the topological sort.  Also if we compare to DFS, in DFS we collect the nodes who has 0 outbound nodes means, we collect child nodes first and then the parent nodes. So we need to use a stack, which follows LIFO. |

| Question No. 07 |
| --- |
| We solved the flood fill problem https://cses.fi/problemset/task/1192 with the following code:  https://github.com/phitronio/Algorithm-Batch1/blob/main/module%2009/flood\_fill.cpp  What is the time complexity of this code? Can you come up with something better?  Hint: maybe the while(true) loop for finding an unvisited cell is overkill? |
| Answer:  Time complexity: O(m\*n), m is the number of rows and, n is the number of columns.  Optimization:  In this code an infinite while loop is used which uses two nested for loops to check the unvisited node.  We can check the unvisited node without using the infinite loop by simply using two nested loops and each time we find an unvisited node, call DFS or BFS algorithm on that coordinate.  Pseudocode:  for(int i=0; i<n; i++){  for(int j=0; j<m; j++){  // if the cell is unvisited and not a wall(#)  if(visited[i][j]==0 && maze[i][j]==0){  BFS(i,j);  }  } |

| Question No. 08 |
| --- |
| Write code to solve the following problem: https://cses.fi/problemset/task/1666 |
| Answer:  #include<bits/stdc++.h>  using namespace std;  const int N=2e5;  int visited[N];  vector<int>adj\_list[N];  vector<int>f\_nodes; // first node of each connected component  void dfs(int node){  visited[node]=1;  for(int adj\_node: adj\_list[node]){  // cout<<"node: "<<node<<" adj\_node: "<<adj\_node<<endl;  if(visited[adj\_node]==0){  dfs(adj\_node);  }  }  }  int main(){  int n,m;// node, edge  cin>>n>>m;  for(int i=0; i<m; i++){  int u,v;  cin>>u>>v;  adj\_list[u].push\_back(v); //undirected graph  adj\_list[v].push\_back(u);  }  int count=0;  for(int i=1; i<=n; i++){  if(visited[i]==0){ // if new node, call DFS  count++; // count the connected components  dfs(i);  f\_nodes.push\_back(i); // first node each connected component  }  }  // output  int roads=count-1;  cout<<roads<<endl;// new roads=connected components-1  if(count>1){  int city1=f\_nodes[0];  int city2;  for(int i=1; i<=roads; i++){  city2=f\_nodes[i];  cout<<city1<<" "<<city2<<endl;  city1=city2;  }  }  } |

| Question No. 09 |
| --- |
| blank |
|  |

| Question No. 10 |
| --- |
| blank |
|  |

github: https://github.com/subrataBAEC/Programming-Fundamentals-Nano-Degree/tree/main/algorithm\_with\_c%2B%2B/11\_module\_lab\_assignment\_01

Submitted by: Subrata Saha

Email:[Subratabaec@gmail.com](mailto:Subratabaec@gmail.com)

Date: .01.2023

THE END