1. Print adjacency list

vector<vector<int>> printGraph(int V, vector<int> adj[]) {

vector<vector<int>> r;

for(int i=0; i<V; i++)

{

r.push\_back(adj[i]);

r[i].insert(r[i].begin(), i);

}

return r;

}

1. BFS of graph

vector<int> bfsOfGraph(int V, vector<int> adj[]) {

vector<int> vec;

vector<int> vis(V,0);

for(int i=0;i<V;i++)

{

if(vis[i]==0)

{

vis[i]=1;

queue<int> q;

q.push(i);

while(q.size()!=0)

{

int node=q.front();

q.pop();

vec.push\_back(node);

for(auto it:adj[node])

{

if(vis[it]==0)

{

q.push(it);

vis[it]=1;

}

}

}

}

break;

}

return vec;

}

1. DFS of graph

void sol(int i, vector<int>\* adj, vector<int>& ans, vector<bool>& vis)

{

ans.push\_back(i);

vis[i] = true;

for(auto it: adj[i])

{

if(!vis[it])

{

sol(it, adj, ans, vis);

}

}

}

vector<int> dfsOfGraph(int V, vector<int> adj[]) {

vector<int> ans;

vector<bool> vis(V, false);

sol(0, adj, ans, vis);

return ans;

}

1. S
2. Detect cycle in undirected grapg

bool isCycle(int V, vector<int> adj[]) {

vector<int> vis(V, 0);

for(int i=0; i<V; i++)

{

if(vis[i] == 0)

{

queue<pair<int,int>> q;

q.push({i, -1});

vis[i] = 1;

while(!q.empty())

{

pair<int,int> p = q.front();

q.pop();

int node = p.first;

int prev = p.second;

for(auto x : adj[node])

{

if(vis[x] == 0)

{

vis[x] = 1;

q.push({x, node});

}

else

{

if(x != prev) return true;

}

}

}

}

}

return false;

}