**Institute of Engineering & Management**

**Department of Computer Science & Engineering**

**Design & Analysis of Algorithm Lab for 3rd year 5th semester 2018**

**Code: CS 591**

**Date:** 02/10/2018

**WEEK-4**

**Assignment-1**

**Problem Statement:** Implement BFS for a certain undirected graph and starting vertex.

**Algorithm:**

**Source code:**

#include <iostream>

#include <vector>

#include <list>

#include <unordered\_set>

#include <queue>

#include <sstream>

void bfs(std::vector<std::list<int>> adjList, std::vector<std::pair<int, int>> &path, int start)

{

std::unordered\_set<int> visited;

std::queue<int> yetoex;

yetoex.push(start);

visited.insert(start);

while(!yetoex.empty())

{

while(!adjList[yetoex.front()-1].empty())

{

if(visited.find(adjList[yetoex.front()-1].front()) == visited.end())

{

visited.insert(adjList[yetoex.front()-1].front());

yetoex.push(adjList[yetoex.front()-1].front());

path.push\_back(std::pair<int, int>(yetoex.front(), adjList[yetoex.front()-1].front()));

}

adjList[yetoex.front()-1].pop\_front();

}

yetoex.pop();

}

}

int main()

{

int n;

std::cout<<"Enter the no. of vertices: ";

std::cin>>n;

std::cin.get();

std::vector<std::list<int>> adjList(n);

std::vector<std::pair<int, int>> path;

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

{

std::string temp;

std::cout<<"Adjacent verteces of "<<i+1<<": ";

getline(std::cin, temp);

std::stringstream ss(temp);

int m;

while(ss>>m)

adjList[i].push\_back(m);

}

std::cout<<"Starting vertex: ";

int start;

std::cin>>start;

bfs(adjList, path, 1);

std::cout<<"Reachable vertices: ";

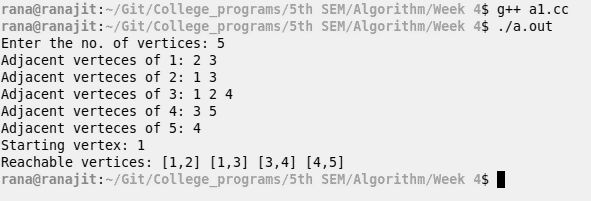
for(int i=0;i<path.size();i++)

std::cout<<"["<<path[i].first<<","<<path[i].second<<"] ";

std::cout<<"\n";

}

**Screen-Shot:**

****

**Time Complexity:**

**Source code:**

#include <iostream>

#include <vector>

#include <list>

#include <unordered\_set>

#include <sstream>

void dfs(std::vector<std::list<int>> &adjList, std::vector<std::pair<int, int>> &path, std::unordered\_set<int> &visited, int start)

{

visited.insert(start);

while(!adjList[start-1].empty())

{

if(visited.find(adjList[start-1].front()) == visited.end())

{

path.push\_back(std::pair<int, int>(start, adjList[start- 1].front()));

dfs(adjList, path, visited, adjList[start-1].front());

}

adjList[start-1].pop\_front();

}

}

int main()

{

int n;

std::cout<<"Enter the no. of vertices: ";

std::cin>>n;

std::cin.get();

std::vector<std::list<int>> adjList(n);

std::vector<std::pair<int, int>> path;

std::unordered\_set<int> visited;

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

{

std::string temp;

std::cout<<"Adjacent verteces of "<<i+1<<": ";

getline(std::cin, temp);

std::stringstream ss(temp);

int m;

while(ss>>m)

adjList[i].push\_back(m);

}

std::cout<<"Starting vertex: ";

int start;

std::cin>>start;

dfs(adjList, path, visited, 1);

std::cout<<"Reachable vertices: \n";

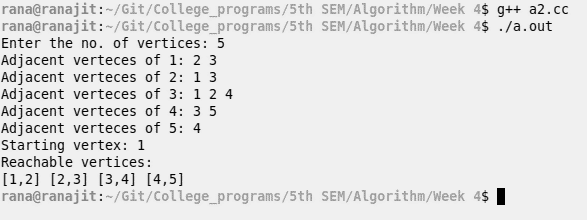
for(int i=0;i<path.size();i++)

std::cout<<"["<<path[i].first<<","<<path[i].second<<"] ";

std::cout<<"\n";

}

**Screen-Shot:**

****

**Time Complexity:**

**Source code:**

#include <iostream>

#include <vector>

void make\_set(std::vector<int> &set)

{

for(auto &i: set)

{

i = -1;

}

}

int find\_set(std::vector<int> &set, int elm)

{

std::vector<int> stack;

int temp = elm;

while(temp>0)

{

stack.push\_back(temp);

temp = set[temp-1];

}

elm = stack.back();

for(int i=0;i<stack.size()-1;i++)

set[stack[i]-1] = temp;

return elm;

}

bool union\_set(std::vector<int> &set, std::pair<int,int> &edge)

{

int n1 = find\_set(set, edge.first), n2 = find\_set(set, edge.second);

if(n1 == n2)

return false;

else{

if(set[n1-1] < set[n2-1])

{

set[n1-1] += set[n2-1];

set[n2-1] = n1;

}

else{

set[n2-1] += set[n1-1];

set[n1-1] = n2;

}

}

return true;

}

int main()

{

int v,e;

std::cout<<"Enter the no. of vertices & edges: ";

std::cin>>v>>e;

std::vector<int> set(v);

std::vector<std::pair<int,int>> edges(e);

make\_set(set);

std::cout<<"Enter the edges:\nsource\tdestination\n";

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

{

int n1,n2;

std::cin>>n1>>n2;

edges[i] = std::make\_pair(n1,n2);

}

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

{

if(union\_set(set, edges[i]))

std::cout<<"edge added: "<<edges[i].first<<" - "<<edges[i].second<<"\n";

else{

std::cout<<"Cycle detected at edge: "<<edges[i].first<<" - "<<edges[i].second<<"\n";

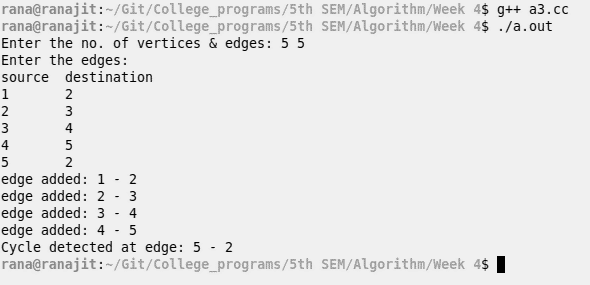
break;

}

}

}

**Screen-Shot:**

****

**Time Complexity:**