Assignment 4

**Question 1:** Implement BFS a;gorithm for a given undirected graph and starting vertex

**Algorithm:**

Start

Algorithm for subroutine bfs(adjList, path, start):

Start

Step 1: Declare an integer set variable visited and an integer queue yetoex

Step 2: yetoex.push(start)

Step 3: visited.insert(start)

Step 4: while sizeof(yetoex) > 0

1. while sizeof(adjList[yetoex.front() - 1]) > 0
   1. if visited.find(adjList[yetoex.front() - 1].front() == false)
      * 1. visited.insert(adjList[yetoex.front() - 1].front())
        2. yetoex.push(adjList[yetoex.front() - 1].front())
   2. adjList[yetoex.front() - 1].pop\_front()
2. yetoex.pop()

Stop

Algorithm for method main()

Start

Step 1: Declare an integer variable n and input the number of vertices in n

Step 2: Declare an adjacency list adjList of size n and a vector of integral pairs path

Step 3: for i = 0 to n-1

1. Declare the vertex numbers and push them at adjList[i]

Step 4: Declare an integer variable start and input an integer

Step 5: bfs(adjList, path, 1)

Step 6: Print the contents of variable path

Stop

Stop

**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";

}

**Output:**

Enter the no. of vertices: 4

Adjacent verteces of 1: 2 3

Adjacent verteces of 2: 2 3

Adjacent verteces of 3: 1 2 4

Adjacent verteces of 4: 3

Starting vertex: 1

Reachable vertices: [1,2] [1,3] [3,4]

**Question 2:** Implement DFS for a given undirected graph and starting vertex

**Algorithm**

Start

Algorithm for subroutine dfs(adjList, path, visited, start)

Start

Step 1: visited.insert(start)

Step 2: while sizeof(adjList[start – 1]) > 0

1. if visited.find(adjList[start – 1].front()) == false
   * 1. path.push\_back({start, adjList[start – 1].front()})
     2. dfs(adjList, path, visited, adjList[start – 1].front())
2. adjList[start – 1].pop\_front()

Stop

Algorithm for method main()

Start

Step 1: Declare an integer variable n and input the number of vertices in n

Step 2: Declare an adjacency list adjList of size n, a vector of integral pairs path and an integral set variable visited

Step 3: for i = 0 to n-1

1. Declare the vertex numbers and push them at adjList[i]

Step 4: Declare an integer variable start and input an integer

Step 5: dfs(adjList, path, visited, 1)

Step 6: Print the contents of variable path

Stop

Stop

**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";

}

**Output:**

Enter the no. of vertices: 4

Adjacent verteces of 1: 2 3

Adjacent verteces of 2: 2 3

Adjacent verteces of 3: 1 2 4

Adjacent verteces of 4: 3

Starting vertex: 1

Reachable vertices: [1,2] [1,3] [3,4]

**Question 3:** Detect whether an undirected graph contains a cycle

**Algorithm**

Start

Algorithm for subroutine make\_set(set):

Start

Step 1: Initialise all the elements of set with -1

Stop

Algorithm for subroutine find\_set(set, elem):

Start

Step 1: Declare a vector of integer variable stack

Step 2: temp ← elem

Step 3: while temp > 0

1. stack.push\_back(temp)
2. temp = set[temp – 1]

Step 4: for i = 0 to stack.size() - 1

1. set[stack[i] – 1] ← temp

Step 5: Return elem

Stop

Algorithm for subroutine union\_set(set, edge)

Start

Step 1: n1 ← find\_set(set, edge.first), n2 ← find\_set(set, edge.second)

Step 2: if n1 == n2

return false

Step 3: Else

1. if set[n1 -1] < set[n2 – 1]
   1. set[n1 – 1] += set[n2 – 1]
   2. set[n2 – 1] ← n1
2. Else
   1. set[n2 – 1] += set[n1 – 1]
   2. set[n1 – 1] ← n2

Step 4: Return true

Stop

Algorithm for method main()

Start

Step 1: Input the number of vertices and edges and store them in integer variables v and e

Step 2: Declare an integer vector variable set of length v

Step 3: Declare a vector of integer pair variable edges of length e

Step 4: make\_set(set)

Step 5: Input the edges of the graph in variable edges

Step 6: for i = 0 to e – 1

1. if union\_set(set, edges[i])
   1. print(edges[i].first, edges[i].second)
2. else
   1. print “cycle detected at edge :” edges[i].first, edges[i].second
   2. break

Stop

Stop

**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;

}

}

}

**Output:**

Enter the no. of vertices & edges: 4 5

Enter the edges:

source destination

1 2

1 3

1 4

2 3

3 4

edge added: 1 - 2

edge added: 1 - 3

edge added: 1 - 4

Cycle detected at edge: 2 - 3