## LAB-5

Print all the nodes reachable from a given starting node in a graph using BFS method. Time analysis to be done by varying the input graph size.

### Code:

#include<math.h>

#include <iostream>

#include <list>

int count=0,count1=0;

using namespace std;

int c = 1;

class Graph {

  int n;

  list<int>\* adj;

  public:

  int\* visited;

  Graph(int v) {

  n = v;

  adj = new list<int>[v];

}

  void addEdge(int src, int dest);

  void BFS(int start,int \* visited);

  void BFSrec(int start,int \* visited);

  void BFSdisc(int start,int \* visited);

  void reachibility(int start , int\* visited);

};

void Graph::addEdge(int src, int dest) {

  adj[src].push\_back(dest);

  adj[dest].push\_back(src);

}

void Graph::BFS(int start,int\* visited) {

  visited = new int[n];

  for (int i = 0; i < n; i++)

    visited[i] = 0;

  list<int> q;

  visited[start] = 1;

  q.push\_back(start);

  while (!q.empty()) {

    int cur = q.front();

    cout << " "<< cur << " ";

    q.pop\_front();

    for(auto i:adj[cur]) {

       count1++;

      if (visited[i]==0) {

        visited[i] = 1;

        q.push\_back(i);

      }

    }

  }

}

// void Graph:: BFSdisc(int s,int\* visited) {

//   visited = new int[n];

//   for (int i = 0; i < n; i++)

//     visited[i] = 0;

//    for (int u=0; u<n; u++)

//      if (visited[u] == 0)

//    BFS(u,visited);

// }

// void Graph::reachibility(int start,int\* visited) {

//   visited = new int[n];

//   for (int i = 0; i < n; i++)

//     visited[i] = 0;

//   list<int> q;

//   visited[start] = 1;

//   q.push\_back(start);

//   while (!q.empty()) {

//     int cur = q.front();

//     cout << " "<< cur << " ";

//     q.pop\_front();

//     for(auto i:adj[cur]) {

//       if (visited[i]==0) {

//         visited[i] = 1;

//         q.push\_back(i);

//       }

//     }

//   }

// }

list<int> q;

void Graph::BFSrec(int start ,int \* visited){

// list<int> q;

count++;

if(visited[start]==0){

  visited[start] = 1;

  cout<<" "<<start<<" ";

  q.push\_back(start);

}

  int flag=0;

  for(auto i:adj[start]){

      count++;

    if(visited[i]==0){

      cout<<" "<<i<<" ";

      q.push\_back(i);

      visited[i]=1;

      flag=1;

    }

  }

  if(flag==0){

    q.pop\_front();

    if(q.empty()){

    for(int i=0;i!=n;i++){

    //    count++;

    if(visited[i]==0){

         c++;

      // cout<<"\ncomponent "<<c<<" found\n";

      BFSrec(i,visited);

      return;

      }

    }

    }

    else{

       BFSrec(q.front(),visited);

       return;

    }

  }

  if(!q.empty()){

    q.pop\_front();

    BFSrec(q.front(),visited);

  }

}

int main() {

    // int n;

    cout<<"number of vertices  ";

    int v,e;

    cin>>v;

    // cin>>e;

    e=v\*(v-1);

  Graph g(v);

  g.visited = new int[v];

  for (int i = 0; i < v; i++)

    g.visited[i] = 0;

// for(int i=0;i<e;i++){

//     cout<<"Enter src and dest of edge "<<i+1<<" : ";

//     int src,dest;

//     cin>>src;

//     cin>>dest;

//     g.addEdge(src,dest);

// }

for(int i=0;i<v;i++)

    {

        for(int j=0;j<v-1;j++)

        {

            g.addEdge(i,j);

        }

    }

  cout<<"Enter starting vertex : ";

  int s ;

  cin>>s;

  cout<<"iterative BFS....\n";

  g.BFS(s,g.visited);

  cout<<"\nrecursive BFS....\n";

  g.BFSrec(s,g.visited);

  if(c>1){

    cout<<"\nGraph is disconnected\n";

     cout<<"\nTotal components in graph is "<<c;

  }

  else{

    cout<<"\nGraph is connected\n";

  }

//   cout<<"\nReachibility of given nodes : \n";

//   g.reachibility(s,g.visited);

cout<<"Count : "<<count;

 cout<<"\ne^2 =>"<<(float)count/(e\*e)<<"\neloge =>"<<(float)count/(e\*log(e))<<"\ne =>"<<(float)count/(e)<<"\nvloge=>"<<v\*log(e);

cout<<"\n\n";

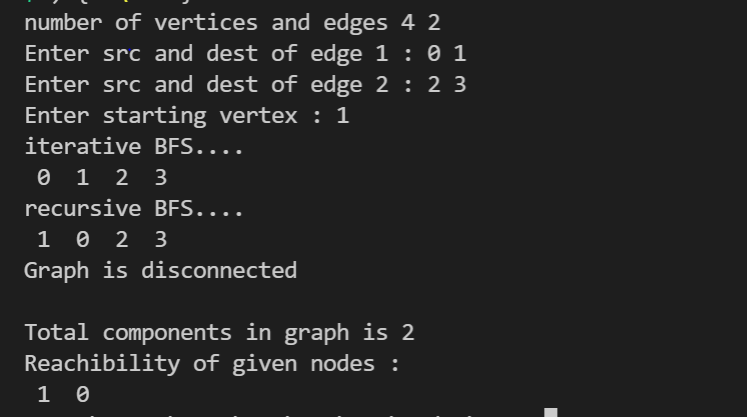
// cout<<"Count1 : "<<count1;

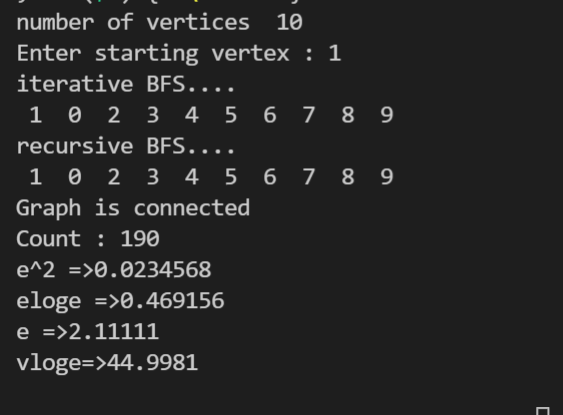
//  cout<<"\ne^2 =>"<<(float)count1/(e\*e)<<"\neloge =>"<<(float)count1/(e\*log(e))<<"\ne =>"<<(float)count1/(e)<<"\nvloge=>"<<v\*log(e);

  return 0;

}

# Output:





## Analysis:

