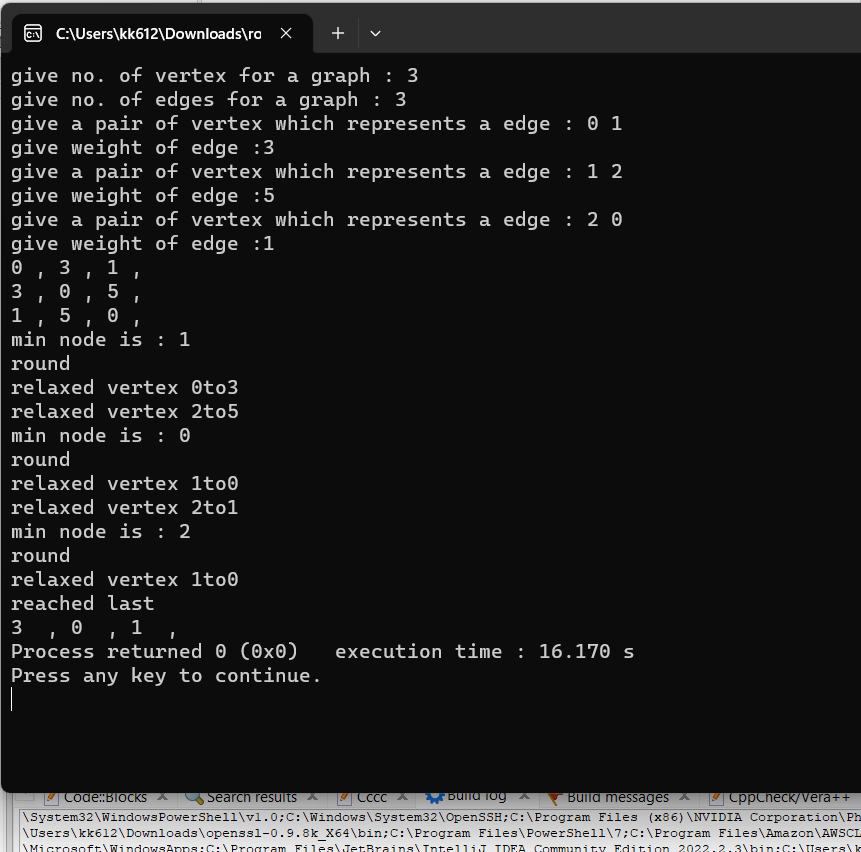
**9.) PRIMS**



#include <iostream>

#include <queue>

using namespace std;

class node{

int vertex;

int weight;

public:

node\* next;

node(){

next=0;

}

node(int v,int w){

vertex = v;

next=0;

weight =w;

}

node(int v){

vertex =v;

weight =0;

}

int get\_vertex(){

return vertex;

}

void set\_vertex(int a){

vertex = a;

}

};

int minimum\_fun(int a,int b){

if(a>b){

return b;

}else{

return a;

}

}

void print(int \*arr,int size){

for(int i=0;i<size;i++){

cout<<arr[i]<<" , ";

}

}

void prims\_algo(node \*\*adjan\_matrix,int size,int src){ // matrix , size

int \*dist = new int[size]; // distance of index from the source provided

int \*mst\_set = new int[size]; //

int \*parent = new int[size]; // parent of index[node] so that we can print mst

for(int i=0;i<size;i++){

dist[i]= 1000; //at starting all the distances of nodes are infinity , 1000 is considered infinity

mst\_set[i]=0;

parent[i]=0; //at starting parent is null for every node

}

parent[src]= -1; // for src it will be -1 so that we can know the root

dist[src]=0; //for src ditance is 0 at starting

//1st step is to pick minimum node value among all the given nodes

int min =1005,min\_node=-1; // i took 1005 so that at

int temp =src;

int pre=-1;

for(int k=0;k<size;k++){ // number of nodes

min =1005; // it is larger than infinte so that it does not interfare while choosing the minimum node

for(int i=0;i<size;i++){ //

if(min>dist[i] & mst\_set[i] !=1){ // if it is not included in mst and minimum distance is lower

//than previous distance of all nodes

min = dist[i];

min\_node=i; //3

}

}

parent[min\_node] = pre;

cout<<"min node is : "<<min\_node<<endl;

//now relax the adjacent edges

cout<<"round"<<endl;

for(int i=0;i<size;i++){

//0 0 , 0 1 , 0 2

if(adjan\_matrix[min\_node][i].get\_vertex() !=0 & mst\_set[parent[i]] ==0){

//has edge

int minimum = minimum\_fun(adjan\_matrix[min\_node][i].get\_vertex() , dist[i]);

dist[i] = minimum;

cout<<"relaxed vertex "<<i <<"to"<<dist[i]<<endl;

}

}

pre = min\_node;

mst\_set[min\_node] = 1;

}

cout<<"reached last"<<endl;

print(dist,size);

}

void print\_matrix(node \*\*adjan\_matrix,int size){

for(int i=0;i<size;i++){

for(int j=0;j<size;j++){

cout<<adjan\_matrix[i][j].get\_vertex()<<" , ";

}

cout<<endl;

}

}

int main()

{

cout<<"give no. of vertex for a graph : ";

int no\_vertex=0;

cin>>no\_vertex;

cout<<"give no. of edges for a graph : ";

int no\_edges=0;

cin>>no\_edges;

node \*\*adjan\_matrix = new node\*[no\_vertex];

for(int i=0;i<no\_vertex;i++){

adjan\_matrix[i] = new node[no\_vertex];

}

for(int i=0;i<no\_vertex;i++){

for(int j=0;j<no\_vertex;j++){

adjan\_matrix[i][j].set\_vertex(0);

}

}

//building a Graph :: Adjancent list

for(int i=0;i<no\_edges;i++){

cout<<"give a pair of vertex which represents a edge : ";

int u=0,v=0;

cin>>u>>v;

cout<<"give weight of edge :";

int weight=0;

cin>>weight;

adjan\_matrix[u][v].set\_vertex(weight);

adjan\_matrix[v][u].set\_vertex(weight);

}

print\_matrix(adjan\_matrix,no\_vertex);

prims\_algo(adjan\_matrix,no\_vertex,1);

return 0;

}

**b.) KRUSKALS**

#include <iostream>

#include <iterator>

#include <list>

#include <string>

using namespace std;

class edge{

public:

int u;

int v;

int weight;

edge(int a,int b,int w){

u= a;

v= b;

weight = w;

}

bool operator<(const edge &r) const

{

if (weight != r.weight) {

return weight < r.weight;

}

return weight < r.weight;

}

// overload the operator> (if required)

bool operator>(const edge &r) const

{

if (weight != r.weight) {

return weight > r.weight;

}

return weight > r.weight;

}

};

int Find(int \*parent, int i)

{

if (parent[i] == -1)

return i;

return Find(parent, parent[i]);

}

void Union(int \*parent, int x, int y) {

parent[x] = Find(parent,y);

//parent[x] =y;

}

void print(list<edge> li){

for (auto i : li) {

cout << i.weight<<" " << i.u<<" "<<i.v<<" || ";

}

}

void kruskal\_algo(int \*parent,list <edge> edge\_list,int no\_edges){

for(int i=0;i<no\_edges;i++){

if(Find(parent ,edge\_list.front().u ) != Find(parent,edge\_list.front().v )){

//have different absolute root

//therefore they must belong to different set

Union(parent,edge\_list.front().u,edge\_list.front().v);

}else{

//they already have path between them

};

edge\_list.pop\_front();

}}

int main()

{

cout<<"enter the number of vertex : ";

int no\_vertex=0;

cin>>no\_vertex;

cout<<"enter the number of edges : ";

int no\_edges=0;

cin>>no\_edges;

list<edge> edge\_list;

int \*parent = new int[no\_vertex];

for(int i=0;i<no\_vertex;i++){

parent[i] = -1;

}

for(int i=0;i<no\_edges;i++){

cout<<"enter edges : ";

int a,b,w;

cin>>a>>b;

cout<<"enter the weight : ";

cin>>w;

edge\_list.push\_back(edge(a,b,w));

}

print(edge\_list);

edge\_list.sort();

cout<<endl;

print(edge\_list);

kruskal\_algo(parent,edge\_list,no\_edges);

for(int i=1;i<=no\_edges;i++){

cout<<parent[i]<<" , ";

}

return 0;

}