Task 1

#include<iostream>

#include<limits>

using namespace std;

const int V = 6;

const int max\_val = numeric\_limits<int>::max();

int findMinDist(int dist[], bool visited[]) {

int minkey = max\_val;

int minIndex = -1;

for (int v = 0; v < V; ++v) {

if (!visited[v] && dist[v] < minkey) {

minkey = dist[v];

minIndex = v;

}

}

return minIndex;

}

void printPath(int parent[], int j) {

if (parent[j] == -1)

return;

printPath(parent, parent[j]);

cout << j << " ";

}

void printSolution(int dist[], int parent[], int src, int dest) {

cout << "Shortest Path from " << src << " to " << dest << " with distance " << dist[dest] << " : ";

printPath(parent, dest);

cout << endl;

}

void dijkstra(int graph[V][V], int src, int dest) {

int dist[V];

bool visited[V];

int parent[V];

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

dist[i] = max\_val;

visited[i] = false;

parent[i] = -1;

}

dist[src] = 0;

for (int count = 0; count < V - 1; ++count) {

int u = findMinDist(dist, visited);

visited[u] = true;

for (int v = 0; v < V; ++v) {

if (!visited[v] && graph[u][v] && dist[u] != max\_val && dist[u] + graph[u][v] < dist[v]) {

parent[v] = u;

dist[v] = dist[u] + graph[u][v];

}

}

}

printSolution(dist, parent, src, dest);

}

int main() {

int graph[V][V] = {

{0, 1, 0, 5, 0, 0},

{0, 0, 2, 2, 1, 0},

{0, 0, 0, 0, 3, 1},

{0, 0, 0, 0, 2, 0},

{0, 0, 0, 0, 0, 2},

{0, 0, 0, 0, 0, 0}

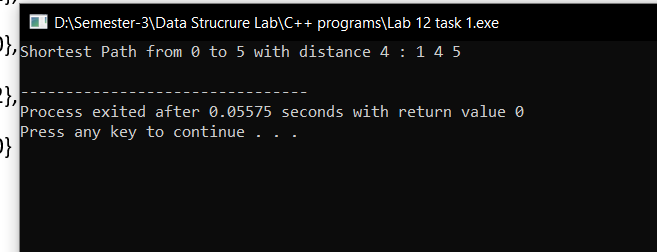
};

int src = 0;

int dest = 5;

dijkstra(graph, src, dest);

return 0;

}

Task 2

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* next;

Node(int v){

this->data=v;

this->next=NULL;

}

};

class Graph{

private:

int V;

Node\*\* adjList;

public:

Graph(int vertices){

this->V = vertices;

adjList = new Node\*[V];

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

adjList[i]=NULL;

}

}

void addEdge(int u,int v){

Node\* newNode = new Node(v);

newNode->next = adjList[u];

adjList[u] = newNode;

newNode = new Node(u);

newNode->next = adjList[v];

adjList[v] = newNode;

}

void printAdjList(){

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

cout<<i<<" ";

Node\* current = adjList[i];

while(current!=NULL){

cout<<current->data<<" ";

current=current->next;

}

cout<<endl;

}

}

};

int main(){

int V = 5;

int E =7;

int edges[][2]={{0,1}, {0,4}, {4,1}, {4,3}, {1,3}, {1,2}, {3,2}};

Graph graph(V);

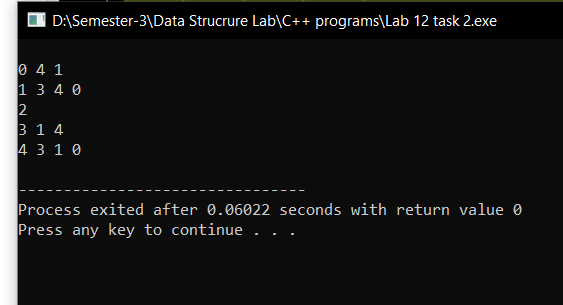
for(int i=0;i<V;++i){

graph.addEdge(edges[i][0],edges[i][1]);

}

cout<<endl;

graph.printAdjList();

}

Task 3

#include<iostream>

using namespace std;

const int V = 3;

const int max\_val = 1e9;

int minIndex;

int findMinDist(int key[], bool visited[]) {

int minkey = max\_val;

minIndex = -1;

for (int v = 0; v < V; ++v) {

if (!visited[v] && key[v] < minkey) {

minkey = key[v];

minIndex = v;

}

}

return minIndex;

}

int printMST(int parent[], int graph[V][V]) {

int totalCount = 0;

for (int i = 1; i < V; ++i) {

cout << "Edge " << parent[i] << " - " << i << " Weight " << graph[i][parent[i]] << endl;

totalCount += graph[i][parent[i]];

}

cout << "Total Weight of Minimum Spanning Tree: " << totalCount << endl;

return totalCount;

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V];

bool visited[V];

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

key[i] = max\_val;

visited[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; ++count) {

int u = findMinDist(key, visited);

visited[u] = true;

for (int v = 0; v < V; ++v) {

if (graph[u][v] && !visited[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

int totalWeight = printMST(parent, graph);

}

int main() {

int graph[V][V] = {

{0, 1, 5},

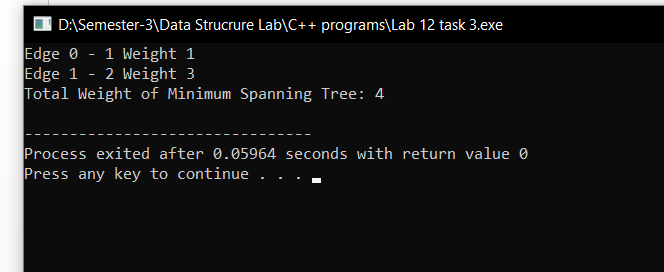
{1, 0, 3},

{5, 3, 0}

};

primMST(graph);

return 0;

}

Task 4

#include<iostream>

#include <climits>

using namespace std;

class findPath{

int v,e;

int \*dist;

public:

findPath(int ver, int edges){

v = ver;

e = edges;

dist = new int[ver];

}

void solve(int arr[50][3],int scr){

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

dist[i]= INT\_MAX;

}

dist[scr]=0;

for(int i=0;i<v-1;++i){

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

int u = arr[j][0];

int v = arr[j][1];

int weight = arr[j][2];

if (dist[u]!=INT\_MAX && dist[u]+weight<dist[v]) {

dist[v] = dist[u]+weight;

}

}

}

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

int u = arr[i][0];

int v = arr[i][1];

int weight =arr[i][2];

if (dist[u]!= INT\_MAX && dist[u]+weight<dist[v]) {

cout << -1;

return;

}

}

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

if (dist[i] == INT\_MAX) {

cout << 10e8 << " ";

} else {

cout << dist[i] << " ";

}

}

}

};

int main(){

int v=3;

int e =4;

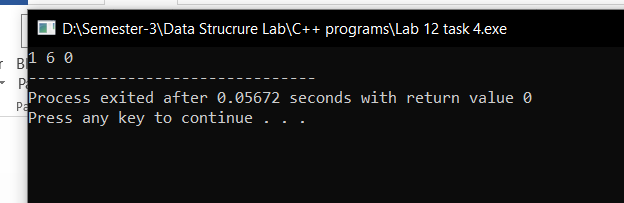
int graph[50][3]={{0,1,5},{1,0,3},{1,2,-1},{2,0,1}};

int sor = 2;

findPath sp(v,e);

sp.solve(graph,sor);

}



Task 5

#include<iostream>

using namespace std;

class search{

public:

int islands(int arr[50][50],int row,int col){

int count =0;

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

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

if(arr[i][j]==1){

visited\_island(arr,i,j,row,col);

count = count +1;

}

}

}

return count;

}

void visited\_island(int arr[50][50],int x,int y, int row, int col){

if (x< 0 || x >= row||y< 0||y >= col || arr[x][y] != 1)

return;

arr[x][y]=2;

visited\_island(arr,x+1,y,row,col);

visited\_island(arr,x,y+1,row,col);

visited\_island(arr,x-1,y,row,col);

visited\_island(arr,x,y-1,row,col);

visited\_island(arr,x+1,y+1,row,col);

visited\_island(arr,x-1,y-1,row,col);

visited\_island(arr,x-1,y+1,row,col);

visited\_island(arr,x+1,y-1,row,col);

}

};

int main(){

int grid[50][50]={{0,1},{1,0},{1,1},{1,0}};

int row =4;

int col=2;

search s;

int num = s.islands(grid,row,col);

cout<<num;

}

