**Muhammad Abdullah**

**SE(3A) | 19F-0916**

Data Structure Lab

Graph Algorithms

**QUESTION # 1: (DIJKSTRA’S ALGORITHM)**

**PROGRAM**

#include<iostream>

using namespace std;

int const V = 5; //As total Vertices are 5 hence initillized here

void Dijkstra(int Graph[V][V], int Source); // Working Algorithm of Dijkstra

int Minimun\_Distance(int Distance[], int Set[]); // Funtion to get Minimum Distance

void Print(int Distance[]); // To Print Algorithm

int main() // Main Code

{

cout << endl << " ! DIJKSTRA'S ALGORITHM !" << endl;

int graph[V][V] = // Hardcode (according to given in question)

{

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

{ 0, 0, 0, 0, 6 },

{ 0, 0, 0, 4, 2 },

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

{ 0, 0, 0, 0, 0}

};

Dijkstra(graph, 0); // Calling Function

cout << endl << endl;

system("pause");

}

void Dijkstra(int Graph[V][V], int Source) // Working Algorithm of Dijkstra

{

int Distance[V];

int Set[V];

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

{

Distance[i] = INT\_MAX, Set[i] = 0;

}

Distance[Source] = 0;

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

{

int temp = Minimun\_Distance(Distance, Set);

Set[temp] = 1;

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

if (Set[i] == 0 && Graph[temp][i] && Distance[temp] != INT\_MAX && Distance[temp] + Graph[temp][i] < Distance[i])

{

Distance[i] = Distance[temp] + Graph[temp][i];

}

}

Print(Distance);

}

int Minimun\_Distance(int Distance[], int Set[]) // Funtion to get Minimum Distance

{

int Min = INT\_MAX, Minimum\_Value;

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

{

if (Set[i] == 0 && Distance[i] <= Min)

{

Min = Distance[i]; // Exchanging Values

Minimum\_Value = i;

}

}

return Minimum\_Value; // Returning Minimum Value

}

void Print(int Distance[]) // To Print Algorithm

{

char abc[5] = { 'a','b','c','d','e' };

cout << endl << (" Vertex \t Distance from Vertex(Source) ") << endl;

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

{

cout << " " << abc[i] << "\t->\t " << Distance[i] << endl; //Indentation

}

}

**STARTING FROM C (CANNOT COUNT A BECAUSE THERE IS NO LINE GOING IN A):**

**A picture containing text, monitor, indoor, electronics

Description automatically generated**

**STARTING FROM A (ALL EDGES COUNTS):**

**A picture containing text, monitor, electronics, screenshot

Description automatically generated**

**QUESTION # 2: (BELLMAN FORD’S ALGORITHM)**

**PROGRAM:**

#include<iostream>

using namespace std;

struct Edge // Structure for creating Edge

{

int Source;

int Destination;

int Weight;

};

struct Graph // Strucure for creating Graph

{

int Vertex;

int Edge;

struct Edge\* edge;

};

Graph\* Create\_Graph(int V, int E); // Creating Graph

void Bellman\_Ford(Graph\* graph, int Source); // Implementing Bellman Fords Algo

void Print(int Distance[], int Vertex); // Printing the result

int main() // Main Code

{

int Vertex = 5;

int Edge = 7;

cout << endl << " ! BELLMEN FORD'S ALGORITHM !" << endl << endl;

Graph\* graph = Create\_Graph(Vertex, Edge); // Creating Graph according to Vertex and Edges

graph->edge[0].Source = 0;

graph->edge[0].Destination = 1;

graph->edge[0].Weight = -1;

graph->edge[1].Source = 0;

graph->edge[1].Destination = 2;

graph->edge[1].Weight = 3;

graph->edge[2].Source = 1;

graph->edge[2].Destination = 4;

graph->edge[2].Weight = 6;

graph->edge[3].Source = 2;

graph->edge[3].Destination = 3;

graph->edge[3].Weight = 4;

graph->edge[4].Source = 2;

graph->edge[4].Destination = 4;

graph->edge[4].Weight = 2;

graph->edge[5].Source = 3;

graph->edge[5].Destination = 1;

graph->edge[5].Weight = 5;

graph->edge[6].Source = 3;

graph->edge[6].Destination = 4;

graph->edge[6].Weight = 1;

Bellman\_Ford(graph, 0);

cout << endl << endl;

system("pause");

}

Graph\* Create\_Graph(int V, int E) // Creating Graph

{

struct Graph\* graph = new Graph;

graph->Vertex = V;

graph->Edge = E;

graph->edge = new Edge[E];

return graph;

}

void Bellman\_Ford(Graph\* graph, int Source) // Implementing Bellman Fords Algo

{

int V = graph->Vertex;

int E = graph->Edge;

int \*distance = new int[V];

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

{

distance[i] = INT\_MAX;

}

distance[Source] = 0;

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

{

for (int j = 0; j < E; j++)

{

int u = graph->edge[j].Source;

int v = graph->edge[j].Destination;

int Weight = graph->edge[j].Weight;

if (distance[u] != INT\_MAX && distance[u] + Weight < distance[v])

{

distance[v] = distance[u] + Weight;

}

}

}

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

{

int u = graph->edge[i].Source;

int v = graph->edge[i].Destination;

int Weight = graph->edge[i].Weight;

if (distance[u] != INT\_MAX && distance[u] + Weight < distance[v])

{

cout << endl << ("Graph contains negative Weight cycle") << endl;

return;

}

}

Print(distance, V);

}

void Print(int Distance[], int Vertex) // Printing the result

{

char abc[5] = { 'a','b','c','d','e' };

cout << (" Vertex Distance from Source") << endl;

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

{

cout << " " << abc[i] << "\t->\t" << Distance[i] << endl;

}

}

**Conclusion:**

Hence at the end, if we start from C when A-B=-1 then there is no difference between Dijkstra and Bellmen Ford because A is not counting in it because no Edge going to A shown in below figure 1 and 2

But if we start from A, then there comes difference between Dijkstra and Bellmen Ford because A is now being counting in it. Shown in Below Figure 3 and 4

**STARTING FROM C (A🡪B = 1):**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**STARTING FROM C (A🡪B = -1):**

**A picture containing text, monitor, indoor, electronics

Description automatically generated**

**STARTING FROM A (A🡪B = -1):**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**STARTING FROM A (A🡪B = 1):**

**A picture containing text, monitor, screenshot, electronics

Description automatically generated**

**QUESTION # 3: (PRIM’S ALGORITHM)**

**PROGRAM:**

#include<iostream>

using namespace std;

int const V = 7; // Initillizing Total Vertices

void Prism(int Graph[V][V]); // Implementing Prim's Algo

int Minimim(int Key[], int MST[]); // Finding Minimum Path

void Print(int Parent[], int Graph[V][V]); // Printing End Result

int main() // Main Code

{

cout << endl << " !! PRIM'S ALGORITHM !!" << endl << endl;

int Graph[V][V] = // Hardcoding Given Graph To Solve

{

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

{ 1, 0, 4, 0, 0, 0, 0 },

{ 0, 4, 0, 6, 0, 2, 0 },

{ 0, 0, 6, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 3, 10 },

{ 0, 0, 2, 0, 3, 0, 0 },

{ 0, 0, 0, 0, 10, 0, 0 },

};

Prism(Graph); // Calling Function to Solve Given Graph

cout << endl << endl;

system("pause");

}

void Prism(int Graph[V][V]) // Implementing Prim's Algo

{

int Parent[V];

int Key[V];

int MST[V];

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

{

Key[i] = INT\_MAX, MST[i] = 0;

}

Key[0] = 0;

Parent[0] = -1;

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

{

int u = Minimim(Key, MST);

MST[u] = 1;

for (int j = 0; j < V; j++)

{

if (Graph[u][j] && MST[j] == 0 && Graph[u][j] < Key[j])

{

Parent[j] = u, Key[j] = Graph[u][j];

}

}

}

Print(Parent, Graph);

}

int Minimim(int Key[], int MST[]) // Finding Minimum Path

{

int Min = INT\_MAX, Minimim\_Index;

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

{

if (MST[i] == 0 && Key[i] < Min)

{

Min = Key[i], Minimim\_Index = i;

}

}

return Minimim\_Index;

}

void Print(int Parent[], int Graph[V][V]) // Printing End Result

{

char abc[7] = { 'a','b','c','d','e','f','g' };

cout << " Edge \t\tWeight" << endl << endl;

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

{

cout << " " << abc[Parent[i]] << " -- " << abc[i] << " \t->" << Graph[i][Parent[i]] << endl;

if (i == 3)

{

cout<< "\t\t\t\t\t\t Minimum Spanning Tree By Prim's Algorithm" << endl;

}

}

}

**AS THE GRAPH GIVEN FOR PRIMS WAS ALREADY An MST APPLIED TREE HENCE THIS IS ONLY THE CODE OF THE MST PRIMS AND SCREESHOT IS ACCORDING TO IT**

**MST ACCORDING TO PRIM’S ALGORITHM:**

**A picture containing text, screenshot, monitor, indoor

Description automatically generated**

**QUESTION # 4: (KRUSKLE’S ALGORITHM)**

**NODE:( THIS PROGRAM IS DONE BY TAKING HELP FROM ONLINE RESOURCES. I’VE TRIED TO UNDERSTAND AND IMPLEMENT IT BY TAKING HELP FROM GOOGLE. IT WAS PRETTY TOUGH FOR ME THAT IS THE REASON)**

**PROGRAM:**

#include <iostream>

using namespace std;

int const Vertices = 9; // Total vertices according to given graph

int parent[Vertices]; // For parent Nodes (to store)

void Kruskal(int Matrix[][Vertices]); // Implementing Kruskal's Algorithm

int Find\_Parent(int temp); // For Finding Parent Vertex

void Exchanging(int i, int j); // For Exhacnging Min

int main() // Main Code

{

int Matrix[][Vertices] =

{

{ 9, INT\_MAX, INT\_MAX, INT\_MAX, 8,8,INT\_MAX,INT\_MAX,INT\_MAX },

{ 9, INT\_MAX, 2, INT\_MAX, INT\_MAX,3,INT\_MAX,INT\_MAX,INT\_MAX },

{ INT\_MAX, 2, INT\_MAX, 1, INT\_MAX,7,INT\_MAX,INT\_MAX,INT\_MAX },

{ INT\_MAX, INT\_MAX, 1, INT\_MAX, INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX,15 }, // Hardcoding Matrix according to given graph

{ 8, INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX,6,INT\_MAX,8,INT\_MAX },

{ 8, 3, 7, INT\_MAX, 6,INT\_MAX,1,INT\_MAX,9 },

{ INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX,1,INT\_MAX,INT\_MAX,5 },

{ INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX, 8,INT\_MAX,INT\_MAX,INT\_MAX,10 },

{ INT\_MAX, INT\_MAX, INT\_MAX, 15, INT\_MAX,9,5,INT\_MAX,10 },

};

cout << endl << " !! Kruskal's Algorithm !!" << endl << endl;

Kruskal(Matrix); // Calling Function

cout << endl << endl;

system("pause");

}

void Kruskal(int Matrix[][Vertices]) // Implementing Kruskal's Algorithm

{

int Minimum = 0;

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

{

parent[i] = i;

}

int edge\_count = 0;

while (edge\_count < Vertices - 1)

{

int Min = INT\_MAX, a = -1, b = -1;

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

{

for (int j = 0; j < Vertices; j++)

{

if (Find\_Parent(i) != Find\_Parent(j) && Matrix[i][j] < Min)

{

Min = Matrix[i][j];

a = i;

b = j;

}

}

}

Exchanging(a, b);

cout << " Edge: " << 1 + edge\_count++ << " ( " << 1 + a << "," << 1 + b << ") " << " Cost :" << Min << endl;

Minimum = Minimum + Min;

}

cout << endl << " Minimum Cost = " << Minimum << endl;

}

int Find\_Parent(int temp) // For Finding Parent Vertex

{

while (parent[temp] != temp)

{

temp = parent[temp];

}

return temp;

}

void Exchanging(int i, int j) // For Exhacnging Min

{

int a = Find\_Parent(i);

int b = Find\_Parent(j);

parent[a] = b;

}

**KRUSKAL’S ALGORITHM ACCORDING TO GRAPH**

**(GRAPH IN QUESTION)**

**A picture containing text, screenshot, monitor, indoor

Description automatically generated**

**QUESTION # 5: (DFS TRAVERSAL)**

**PROGRAM:**

#include <iostream>

using namespace std;

struct Arrayy //Implementaiting Stack

{

int Size;

int top = -1;

int \*Array = NULL;

};

class Stack // Stack Class

{

public:

Stack(int size)

{

ARRAY.Size = size;

ARRAY.Array = new int[size];

}

Arrayy ARRAY;

void push(int Value) //push function

{

if (ARRAY.top != ARRAY.Size - 1)

{

ARRAY.top++;

ARRAY.Array[ARRAY.top] = Value;

}

else

cout << endl << "Stack is FULL !!" << endl;

}

int pop() //pop function

{

if (ARRAY.top != -1)

{

int temp = 0;

temp = ARRAY.Array[ARRAY.top];

ARRAY.top--;

return temp;

}

else

cout << endl << "Stack is Empty !!";

}

};

class Graph // Class For Graph

{

public:

int \*\* Matrix; //2D Array for Adjecency Matrix

int Visited;

Graph(int Visited) // Constructor for Making Matrix

{

this->Visited = Visited;

Matrix = new int\*[Visited];

for (int i = 1; i <= Visited; i++)

{

Matrix[i] = new int[Visited];

for (int j = 1; j <= Visited; j++)

Matrix[i][j] = 0;

}

}

void Add\_Edge(int i, int j) // Adding Edge

{

Matrix[i][j] = 1;

Matrix[j][i] = 1;

}

void Remove\_Edge(int i, int j) //Removing Edge (if needed)

{

Matrix[i][j] = 0;

Matrix[j][i] = 0;

}

void DFS(int Vertex) // BFS Traversal and Displaying

{

int count = 1;

Stack stack(Visited);

cout << endl << " BFS is : ";

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

{

if (count == stack.ARRAY.top)

{

cout << stack.pop() << " ";

}

if (Matrix[i][j] == 1)

{

stack.push(count);

count++;

}

}

}

while (stack.ARRAY.top != -1)

{

cout << stack.pop() << " ";

}

cout << endl;

}

void Adjencency\_Matrix() // Displaying Adjececy Matrix

{

cout << endl;

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

cout << " " << Matrix[i][j] << " ";

cout << endl << endl;;

}

}

};

int main() // Main Code

{

int size = 0, i = 1, opt = 0;

cout << endl << " Enter Size of the Matrix : ";

cin >> size;

Graph Adj\_Matrix(size);

while (i == 1)

{

system("cls");

cout << "----------------------------------------------------" << endl; //Menu Driven Program

cout << " Press 1 to Make a Graph with Adjacency Matrix !" << endl;

cout << " Press 2 to Remove an Edge From the Graph !" << endl;

cout << " Press 3 to Display the DFS Traversal !" << endl;

cout << " Press 4 to Display the Adjecency Matrix !" << endl;

cout << " Press 0 to Exit from the System !" << endl;

cout << "----------------------------------------------------" << endl;

cout << endl << " Enter Choice : ";

cin >> opt;

switch (opt)

{

case 1: // To Add Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to add Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to add Node : ";

cin >> col;

Adj\_Matrix.Add\_Edge(row, col);

system("pause");

break;

}

case 2: // To Delete Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to Remove Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to Remove Node : ";

cin >> col;

Adj\_Matrix.Remove\_Edge(row, col);

system("pause");

break;

}

case 3: // To Show Matrix

{

cout << endl;

Adj\_Matrix.DFS(0);

cout << endl;

system("pause");

break;

}

case 4: // To Show Matrix

{

cout << endl << " Adjeceny Matrix is : " << endl;

Adj\_Matrix.Adjencency\_Matrix();

cout << endl;

system("pause");

break;

}

case 0: // To Exit from System

{

cout << endl << " You have exited from the System !" << endl;

i = 0;

system("pause");

break;

}

default:

{

cout << endl << " Invalid Option, Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**MAIN MENU**

**A picture containing text, monitor, screenshot, indoor

Description automatically generated**

**MATRIX ACCORDING TO GRAPH**

**A picture containing text, monitor, electronics, screenshot

Description automatically generated**

**QUESTION # 5: (BFS TRAVERSAL)**

**PROGRAM:**

#include <iostream>

using namespace std;

struct Arrayy //Implementaiting Stack

{

int Size;

int top = -1;

int \*Array = NULL;

};

class Stack // Stack Class

{

public:

Stack(int size)

{

ARRAY.Size = size;

ARRAY.Array = new int[size];

}

Arrayy ARRAY;

void push(int Value) //push function

{

if (ARRAY.top != ARRAY.Size - 1)

{

ARRAY.top++;

ARRAY.Array[ARRAY.top] = Value;

}

else

cout << endl << "Stack is FULL !!" << endl;

}

int pop() //pop function

{

if (ARRAY.top != -1)

{

int temp = 0;

temp = ARRAY.Array[ARRAY.top];

ARRAY.top--;

return temp;

}

else

cout << endl << "Stack is Empty !!";

}

};

class Graph // Class For Graph

{

public:

int \*\* Matrix; //2D Array for Adjecency Matrix

int Visited;

Graph(int Visited) // Constructor for Making Matrix

{

this->Visited = Visited;

Matrix = new int\*[Visited];

for (int i = 1; i <= Visited; i++)

{

Matrix[i] = new int[Visited];

for (int j = 1; j <= Visited; j++)

Matrix[i][j] = 0;

}

}

void Add\_Edge(int i, int j) // Adding Edge

{

Matrix[i][j] = 1;

Matrix[j][i] = 1;

}

void Remove\_Edge(int i, int j) //Removing Edge (if needed)

{

Matrix[i][j] = 0;

Matrix[j][i] = 0;

}

void BFS(int Vertex) // BFS Traversal and Displaying

{

int count = 1;

Stack stack(Visited);

cout << endl << " BFS is : ";

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

{

if (count == stack.ARRAY.top)

{

cout << stack.pop() << " ";

}

if (Matrix[i][j] == 1)

{

stack.push(count);

count++;

}

}

}

while (stack.ARRAY.top != -1)

{

cout << stack.pop() << " ";

}

cout << endl;

}

void Adjencency\_Matrix() // Displaying Adjececy Matrix

{

cout << endl;

for (int i = 1; i <= Visited; i++)

{

for (int j = 1; j <= Visited; j++)

cout << " " << Matrix[i][j] << " ";

cout << endl << endl;;

}

}

};

int main() // Main Code

{

int size = 0, i = 1, opt = 0;

cout << endl << " Enter Size of the Matrix : ";

cin >> size;

Graph Adj\_Matrix(size);

while (i == 1)

{

system("cls");

cout << "----------------------------------------------------" << endl; //Menu Driven Program

cout << " Press 1 to Make a Graph with Adjacency Matrix !" << endl;

cout << " Press 2 to Remove an Edge From the Graph !" << endl;

cout << " Press 3 to Display the BFS Traversal !" << endl;

cout << " Press 4 to Display the Adjecency Matrix !" << endl;

cout << " Press 0 to Exit from the System !" << endl;

cout << "----------------------------------------------------" << endl;

cout << endl << " Enter Choice : ";

cin >> opt;

switch (opt)

{

case 1: // To Add Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to add Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to add Node : ";

cin >> col;

Adj\_Matrix.Add\_Edge(row, col);

system("pause");

break;

}

case 2: // To Delete Edge

{

cout << endl;

int row = 0, col = 0;

cout << " Enter Number of Row in which you want to Remove Node : ";

cin >> row;

cout << " Enter Number of Coloumn in which you want to Remove Node : ";

cin >> col;

Adj\_Matrix.Remove\_Edge(row, col);

system("pause");

break;

}

case 3: // To Show Matrix

{

cout << endl;

Adj\_Matrix.BFS(0);

cout << endl;

system("pause");

break;

}

case 4: // To Show Matrix

{

cout << endl << " Adjeceny Matrix is : " << endl;

Adj\_Matrix.Adjencency\_Matrix();

cout << endl;

system("pause");

break;

}

case 0: // To Exit from System

{

cout << endl << " You have exited from the System !" << endl;

i = 0;

system("pause");

break;

}

default:

{

cout << endl << " Invalid Option, Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**MAIN MENU**

**A picture containing text, monitor, screenshot, indoor

Description automatically generated**

**ADJECENCY MATRIX ALONG GRAPH GIVEN**

**A picture containing text, monitor, electronics, screenshot

Description automatically generated**