**Muhammad Abdullah**

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

DS Assignment 6

Graphs

**Question # 1:**

**MST BY PRMIM’S ALGORITHM**

**A picture containing letter

Description automatically generated**

**Question # 1:**

**MST BY KURSKAL’S ALGORITHM**

**Diagram

Description automatically generated**

**Question # 1:**

**SHORTEST DISTANCE BY DIJKSTRA’S ALGORITHM**

**A picture containing diagram

Description automatically generated**

**Question # 2: (DIJKSTRA’S ALGORITHM)**

**PROGRAM:**

#include<iostream>

#include <limits.h>

#include <stdio.h>

using namespace std;

int const V = 9;

int Minimun\_Distance(int Distance[], int Set[])

{

int Min = INT\_MAX, Minimum\_Value;

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

{

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

{

Min = Distance[i], Minimum\_Value = i;

}

}

return Minimum\_Value;

}

void Print(int Distance[])

{

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

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

{

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

}

}

void Dijkstra(int Graph[V][V], int Source)

{

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 main()

{

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

int graph[V][V] =

{

{ 0, 6, 0, 0, 0, 0, 0, 5, 0 },

{ 6, 0, 7, 0, 0, 0, 0, 13, 0 },

{ 0, 7, 0, 9, 9, 2, 0, 0, 3 },

{ 0, 0, 9, 0, 9, 11, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 8, 0, 0, 0 },

{ 0, 0, 2, 11, 8, 0, 3, 0, 0},

{ 0, 0, 0, 0, 0, 3, 0, 2, 9 },

{ 5, 13, 0, 0, 0, 0, 2, 0, 8 },

{ 0, 0, 3, 0, 0, 0, 9, 8, 0 }

};

Dijkstra(graph, 0);

cout << endl << endl;

system("pause");

}

A picture containing text, monitor, indoor, screenshot

Description automatically generated

**Question # 2: (BELLMEN FORD’S ALGORITHM)**

**PROGRAM:**

#include<iostream>

#include <limits.h>

#include <stdio.h>

using namespace std;

struct Edge

{

int Source, Distance, Weight;

};

struct Graph

{

int Vertex, Edge;

struct Edge\* edge;

};

struct Graph\* Create\_Graph(int V, int E)

{

struct Graph\* graph = new Graph;

graph->Vertex = V;

graph->Edge = E;

graph->edge = new Edge[E];

return graph;

}

void Print(int Distance[], int Vertex)

{

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

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

{

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

}

}

void Bellman\_Ford(Graph\* graph, int Source)

{

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].Distance;

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].Distance;

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);

}

int main()

{

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

int Vertex = 5;

int Edge = 8;

struct Graph\* graph = Create\_Graph(Vertex, Edge);

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

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

graph->edge[0].Weight = -3;

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

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

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

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

graph->edge[2].Distance = 2;

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

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

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

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

graph->edge[4].Source = 1;

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

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

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

graph->edge[5].Distance = 2;

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

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

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

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

graph->edge[7].Source = 4;

graph->edge[7].Distance = 3;

graph->edge[7].Weight = -3;

Bellman\_Ford(graph, 0);

cout << endl << endl;

system("pause");

}

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

Description automatically generated**

**Question # 2: (PRIMS ALGORITHM)**

**PROGRAM:**

#include<iostream>

#include <limits.h>

#include <stdio.h>

using namespace std;

int const V = 5;

int Minimim(int Key[], int MST[])

{

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])

{

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

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

{

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

}

}

void Prism(int Graph[V][V])

{

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 main()

{

cout << " ! PRIMS ALGORITHM !" << endl << endl;

int Graph[V][V] =

{

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

{ 4, 0, 5, 7, 6 },

{ 0, 5, 0, 0, 7 },

{ 4, 7, 0, 0, 9 },

{ 0, 6, 7, 9, 0 }

};

Prism(Graph);

cout << endl << endl;

system("pause");

}

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

Description automatically generated**