Experiment #7		Student ID	
Date		Student Name	

Experiment Title: Implementation of Programs on Greedy method Problems – MST and Single Source Shortest Path.

Aim/Objective:

To implement algorithms for solving Minimum Spanning Tree (MST) and Single Source Shortest Path (SSSP) problems using greedy methods.

Description:

The goal is to understand and apply greedy algorithms to solve MST and SSSP problems efficiently. For MST, we will use Kruskal's and Prim's algorithms. For SSSP, we will use Dijkstra's algorithm.

Pre-Requisites:

- Basic understanding of graphs and their representations (adjacency matrix/list).
- Familiarity with greedy algorithm concepts.
- Knowledge of data structures like heaps and disjoint-set data structures.

Pre-Lab:

Given a graph with vertices A, B, C, D, and E, and the following edges with weights: (A-B, 1), (A-C, 3), (B-C, 2), (B-D, 4), (C-D, 5), (C-E, 6), (D-E, 7). Perform Prim's algorithm step-by-step to find the MST. Illustrate your steps and show the final MST.

- **Procedure/Program:**

Graph e

1) Initialize:

- start at vertex A.

- MST edges : []

- visited vertices : {A}

2) Add minimum Edge (A-B, 1):

- MST edges: [(A-B, 1)]
- visited vertices: {A, B}

3) Add minimum Edge (B-C, 2):

- MST edges: [(A-B, 1), (B-C, 2)]
- visited vertices: {A, B, C}

4) Add minimum Edge (B-D, 4):

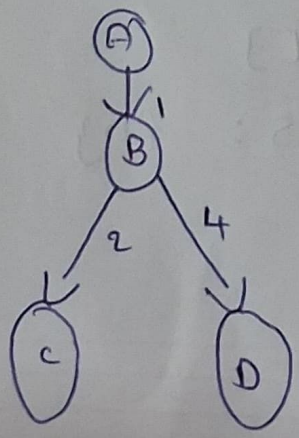
- MST edges: [(A-B, 1), (B-C, 2), (B-D, 4)]
- visited vertices: {A, B, C, D}

5) Add minimum Edge (C-E, 6):

- MST edges: [(A-B, 1), (B-C, 2), (B-D, 4), (C-E, 6)]
- visited vertices: {A, B, C, D, E}

Final MST

- Edges: (A-B, 1), (B-C, 2), (B-D, 4), (C-E, 6)
- Total weight: 13



- **Data and Results:**

Data: Graph representation with vertices and edges for Prim's algorithm.

results: Minimum spanning Tree edges and total weight calculated

- **Analysis and Inferences:**

Analysis: Prim's algorithm efficiently connects successfully vertices with minimal total edge weight

inferences: MST reduces costs while ensuring optimal connections between locations.

In-Lab:

In a city consider the Apartments as nodes and the possible cable connections as edges with costs. Implement Kruskal's algorithm to determine the optimal way to connect all Apartments. Write a program the implications of using a greedy algorithm for this task.

Input:

Apartments = ["Aparna Amaravati ", "Jayabheri", "Vajra Residency", "Sunrise Towers", "Rams enclave"]

cable_connections = [("Aparna Amaravati ", "Jayabheri", 10), ("Aparna Amaravati ", " Vajra

Residency ", 20), ("Jayabheri ", "Vajra Residency", 30), ("Jayabheri ", "Sunrise Towers ", 40), ("Vajra Residency", "Sunrise Towers ", 50), ("Sunrise Towers ", "Rams enclave ", 60), ("Vajra Residency ", " Rams enclave ", 70)]

Output:

Total cost: 130

MST: [('Aparna Amaravati', 'Jayabheri', 10), ('Aparna Amaravati', 'Vajra Residency', 20), ('Jayabheri', 'Sunrise Towers', 40), ('Sunrise Towers', 'Rams enclave', 60)]

- **Procedure/Program:**

```
#include<stdio.h>
#include<string.h>
int main(){
char *apartments[]={"Aparna Amaravati",
"Jayabheri", "Vajra Residency", "Sunrise
Towers", "Rams enclave"};
int connections[][3] ={
{0, 1, 10}, {0, 2, 20}, {1, 2, 30}, {1, 3, 40},
{2, 3, 50}, {3, 4, 60}, {2, 4, 70}
};
int Parent[5], totalcost=0, mst[7][3],
mstIndex =0;
for(int i= 0; i<5; i++) Parent[i] = i;
```

```
for(int i =0; i<6; i++){
for(int j = 0; j<6-i -1; j++){
if(connections[j][2] > connections[j+1][2]){
int temp[3]= {connetions[j][0], connections[j][1],
        connections[j][2]};
connections[j][0] = connections[j+1][0];
connections[j][1] = connections[j+1][1];
connections[j][2] = connections[j+1][2];
connections[j+1][0]= temp[0];
connections[j+1][1]= temp[1];
connections[j+1][2]= temp[2];
}}}
for(int i=0;  i<7; i++){
int u = connections[i][0], v = connections[i][1];
while(Parent[u] != u) u= Parent[u];
while(Parent[v] != v) v= Parent[v];
  if(u!=v){
  Parent[u] = v;
mst[mstIndex][0] = connections[i][0];
mst[mstIndex][1]= connections[i][1];
mst[mstIndex][2]= connections[i][2];
totalcost += connections[i][2];
   mstIndex++;
  }}
Printf("Total cost: %d\n", totalcost);
 Printf("MST:\n");
for(int i=0; i<mstIndex; i++)
Printf("(%s, %s, %d)\n", apartments[mst[i][0]],
```

```
apartments[mst[i][1]],
    mst[i][2]);
    return 0;
}
```

```
char *delivery_points[] = {"w", "c1", "c2",
                           "c3", "c4"};
int travel_times [5][5] = {
  {0, 3, 6, INI_MAX, INI_MAX},
  {3, 0, INI_MAX, 2, INI_MAX},
  {INI_MAX, 2, 4, 0, 1},
  {6, INT_MAX, 0, 4, 2},
  {INT_MAX, INI_MAX, 2, 1, 0}
};
int dist[5], visited[5] = {0};
for(int i = 0; i<5; i++)
  dist[i] = INT_MAX;
  dist[0] = 0;

for(int i = 0; i<5; i++){
int min_index = -1, min_dist = INT_MAX;
for(int j=0; j<5; j++){
  if(!visited[j] && disit[j] < min_dist){
    min_dist = dist[j];
      min_index = j;
  }
}

if(min_index = =-1) break;
    visited[min_index] =
```

```
for(int j = 0; j < 5; j++){
if (!visited[j] && travel_times[min_index][j]
    != INT_MAX && dist[min_index] +
travel_times[min_index][j] < dist[j]) {
 dist[j] = dist[min_index] + travel_times[min_
                                  index][j];
}}}

Printf("Shortest Paths from W: \n");
  for(int i = 0; i<5; i++)
  Printf("W -> %s : %d \n", delivery_Points[i],
                              dist[i]);

   return 0;
}
```

- **Data and Results:**

Data

Delivery Points and travel times are modeled for optimization.

Result

Shortest Paths calculated for efficient delivery from the warehouse

Experiment #7		Student ID	
Date		Student Name	

- **Analysis and Inferences:**

Analysis: Dijkstra's algorithm minimizes delivery times and optimizes route efficiency

Inference: optimized routes enhance delivery speed and improve customer satisfaction.

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is a greedy algorithm?
2. What are the characteristics of problems suitable for greedy algorithms?
3. What is the time complexity of Prim's algorithm, Kruskal's algorithm and Dijkstra's algorithm?
4. What are the limitations or constraints of Greedy Method approaches in solving MST and SSSP problems?
5. Explain how the time complexity is derived.

Evaluator Remark (if Any):	**Marks Secured:___out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

1) A greedy algorithm builds solutions piece by piece, choosing the best option at each step

2) suitable problems exhibit optimal substructure and greedy choice property, ensuring local choices lead to global solutions.

3) Prim's: O(E log V), Kruskal's : O(E log E), Dijkstra's: O(V^2) or O(E log V).

4) Limitations include non-optimal solutions for some problems and dependency on Problem structure characteristics.

5) Time complexity is derived from analyzing algorithm steps and their respective input sizes and operations.