Experiment **#12**		Student ID	
Date		Student Name	[@KLWKS_BOT] THANOS

Experiment Title: To implement programs on problem solving using Graph Algorithms.

Aim/Objective: To understand the concept and implementation of programs on Graph Algorithms-based Problems.

Description: The students will understand All-Pairs Shortest path algorithms. Students will gain experience in implementing these algorithms, Minimum Spanning Trees (Prim's and Kruskal's Algorithm) , and applying them to solve real-world problems.

Pre-Requisites:

Knowledge: Before beginning this lab, students should have a foundational understanding of:

- The concepts of Prim's and Kruskal's Algorithm and All-Pairs Shortest path algorithms.
- Mathematical Background: Logarithmic complexity analysis for tree operations.
- Understanding of height-balanced properties.

Tools: Code Blocks/Eclipse IDE.

Pre-Lab:

- You are tasked with designing an optimal railway network to connect different cities in a newly developed region. Each pair of cities is connected by a possible railway line with a specified construction cost. Your objective is to construct a railway network that connects all the cities with the minimum total cost, ensuring that no cycles are formed. (Kruskal's Algorithm)

Input Format:

- A list of cities (nodes) and possible railway lines (edges) with their respective costs (weights).
- The graph is represented as an edge list.

Output Format:

- The edges included in the Minimum Spanning Tree (MST).
- The total cost of the MST.
- Steps involved in the selection of edges.

Sample Input:

- There are 5 cities: A, B, C, D, and E. The possible railway connections and their costs are:

Railway Line	Cost
A -- B	4
A -- C	3
B -- C	2
B -- D	6
C -- D	5
C -- E	7
D -- E	8

Sample Output:

- The MST should connect all cities with the minimum cost.
- Example output:

  Edges in MST:

  - B -- C (2)
  - A -- C (3)
  - C -- D (5)
  - C -- E (7)

  Total Cost: 17

- **Procedure/Program:**

```
import java.util.Arrays;

class Edge {
   int src, dest, weight;

   Edge(int src, int dest, int weight) {
      this.src = src;
      this.dest = dest;
      this.weight = weight;
   }
}

class Subset {
   int parent, rank;
}
```

```
public class Main {
   static int find(Subset subsets[], int i) {
      if (subsets[i].parent != i)
         subsets[i].parent = find(subsets, subsets[i].parent);
      return subsets[i].parent;
   }

   static void unionSets(Subset subsets[], int x, int y) {
      int rootX = find(subsets, x);
      int rootY = find(subsets, y);
      if (subsets[rootX].rank < subsets[rootY].rank)
         subsets[rootX].parent = rootY;
      else if (subsets[rootX].rank > subsets[rootY].rank)
         subsets[rootY].parent = rootX;
      else {
         subsets[rootY].parent = rootX;
         subsets[rootX].rank++;
      }
   }

   static int compareEdges(Edge a, Edge b) {
      return a.weight - b.weight;
   }

   static void kruskalMST(Edge edges[], int V, int E) {
      Arrays.sort(edges, Main::compareEdges);
      Subset subsets[] = new Subset[V];
      for (int v = 0; v < V; v++) {
         subsets[v] = new Subset();
         subsets[v].parent = v;
         subsets[v].rank = 0;
      }

      Edge result[] = new Edge[V];
      int e = 0, i = 0;
      while (e < V - 1 && i < E) {
         Edge nextEdge = edges[i++];
```

```
        int x = find(subsets, nextEdge.src);
        int y = find(subsets, nextEdge.dest);
        if (x != y) {
          result[e++] = nextEdge;
          unionSets(subsets, x, y);
        }
      }

      System.out.println("Edges in MST:");
      int totalCost = 0;
      for (i = 0; i < e; i++) {
        System.out.printf("%c -- %c (%d)\n", result[i].src + 'A', result[i].dest + 'A',
result[i].weight);
        totalCost += result[i].weight;
      }
      System.out.println("Total Cost: " + totalCost);
    }

    public static void main(String[] args) {
      int V = 5, E = 7;
      Edge edges[] = {
        new Edge(0, 1, 4), new Edge(0, 2, 3), new Edge(1, 2, 2), new Edge(1, 3, 6),
        new Edge(2, 3, 5), new Edge(2, 4, 7), new Edge(3, 4, 8)
      };
      kruskalMST(edges, V, E);
    }
}
```

- **Data and Results:**

Data: The graph has five cities with given railway connections costs.

Result: The MST includes minimum-cost edges connecting all cities efficiently.

- **Analysis and Inferences:**

Analysis: Kruskal's Algorithm sorts edges and uses Union-Find for MST.

Inferences: The algorithm ensures the minimum spanning tree with no cycles.

In-Lab:

You are a network engineer tasked with designing a communication network for a new office campus. The campus consists of several buildings (nodes), and you need to lay out network cables (edges) between these buildings to ensure all buildings are connected. The cost of laying cables between two buildings depends on the distance between them. Your objective is to design the network with the minimum total cost while ensuring all buildings are connected. (Prim's Algorithm).

Input Format:

- The number of buildings (nodes) and the distances (weights) between them.
- The graph is represented by an adjacency matrix.

Output Format:

- The edges included in the Minimum Spanning Tree (MST).
- The total cost of the MST.
- The sequence of steps taken to build the MST.

Example:

The campus has 5 buildings: A, B, C, D, and E. The distances between buildings are represented in the table below:

	A	B	C	D	E
A	0	4	3	INF	INF
B	4	0	2	6	INF
C	3	2	0	5	7
D	INF	6	5	0	8
E	INF	INF	7	8	0

(Note: INF denotes that no direct connection exists between those buildings.)

Expected Output:

- The MST should connect all buildings with minimum cost.

Edges in MST:

- A -- C (3)
- C -- B (2)
- C -- D (5)
- D -- E (8)

Total Cost: 18

- **Procedure/Program**:

```
import java.util.Arrays;

public class Main {
   static final int V = 5;
   static final int INF = 99999;

   int minKey(int key[], Boolean mstSet[]) {
      int min = INF, min_index = -1;
      for (int v = 0; v < V; v++)
         if (!mstSet[v] && key[v] < min) {
            min = key[v];
            min_index = v;
         }
      return min_index;
   }

   void printMST(int parent[], int graph[][]) {
      int totalCost = 0;
      System.out.println("Edges in MST:");
      for (int i = 1; i < V; i++) {
         System.out.printf("%c -- %c (%d)\n", (char)(parent[i] + 'A'), (char)(i + 'A'),
graph[i][parent[i]]);
         totalCost += graph[i][parent[i]];
      }
```

```
    System.out.println("Total Cost: " + totalCost);
  }

  void primMST(int graph[][]) {
    int parent[] = new int[V];
    int key[] = new int[V];
    Boolean mstSet[] = new Boolean[V];
    Arrays.fill(key, INF);
    Arrays.fill(mstSet, false);

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
      int u = minKey(key, mstSet);
      mstSet[u] = true;

      for (int v = 0; v < V; v++)
        if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
          parent[v] = u;
          key[v] = graph[u][v];
        }
    }
    printMST(parent, graph);
  }
  public static void main(String[] args) {
    int graph[][] = new int[][] {
      {0, 4, 3, INF, INF},
      {4, 0, 2, 6, INF},
      {3, 2, 0, 5, 7},
      {INF, 6, 5, 0, 8},
      {INF, INF, 7, 8, 0}
    };

    Main t = new Main();
    t.primMST(graph);
  }
}
```

- **Data and Results:**

Data

The adjacency matrix represents distances between five buildings (nodes).

Result

Minimum Spanning Tree connects all buildings with the lowest cost.

- **Analysis and Inferences:**

Analysis

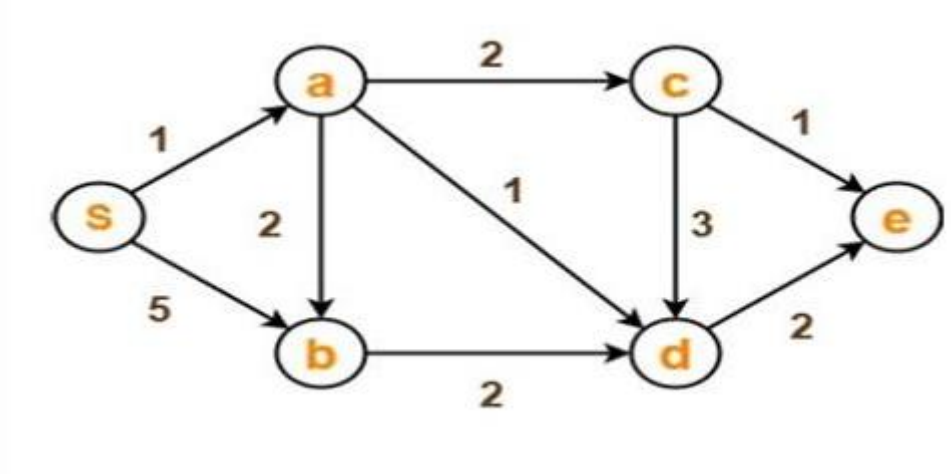
Prim's algorithm selects the shortest edge iteratively, ensuring connectivity efficiently.

Inferences

The MST minimizes cabling costs while maintaining full network connectivity.

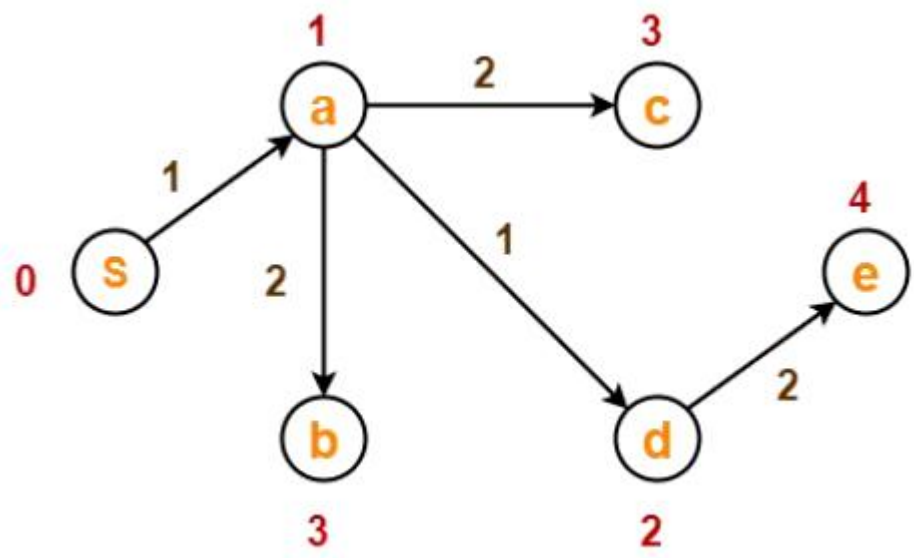
Post-Lab:

Using Dijkstra's algorithm, find out the shortest distance from the source vertex 's' to the rest of the vertices in the given graph. Also write the order in which all the vertices of the graph are visited.



- **Procedure/Program**:

Iteration	S	Vertex Selected	Distance					
			S	a	b	c	d	e
Initial	-	-	0	1	5	∞	∞	∞
1	{s}	a	0	1	3	3	2	∞
2	{s,a}	d	0	1	3	3	2	4
3	{s,a,d}	b	0	1	3	3	2	4
4	{s,a,d,b}	c	0	1	3	3	2	4
5	{s,a,d,b,c}	e	0	1	3	3	2	4
6	{s,a,d,b,c,e}							



- **Data and Results:**

Data

Dijkstra's algorithm finds shortest paths from source vertex efficiently.

Result

Final shortest distances: S=0, A=1, B=3, C=3, D=2, E=4.

- **Analysis and Inferences :**

Analysis

Algorithm selects the nearest vertex and updates path distances iteratively.

Inferences

Dijkstra's ensures optimal path selection using priority-based vertex expansion.

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

1. What is the difference between the Single-Source Shortest Path (SSSP) and the All-Pairs Shortest Path (APSP) problem?

- SSSP: Shortest path from one source (e.g., Dijkstra).
- APSP: Shortest paths between all pairs (e.g., Floyd-Warshall).

2. What are some real-world applications of All-Pairs Shortest Path algorithms?

- GPS, network routing, social networks, urban planning.

3. How does Prim's algorithm ensure that no cycles are formed in the MST?

- Expands MST by adding the smallest edge without forming cycles.

4. What data structure is commonly used to implement Kruskal's algorithm?

- **Disjoint Set (Union-Find)** for cycle detection.

5. How does the choice of data structure affect the efficiency of Prim's and Kruskal's algorithms?

- **Prim**: Binary heap → $O(E \log V)$, Fibonacci heap → $O(V \log V + E)$.
- **Kruskal**: Sorting + Union-Find → $O(E \log E)$.

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.