WEEK-3

Name: B.THARUN REDDY
Roll No: 422116
SECTION: A

1. Find MST (minimum spanning tree) and SPT (shortest path tree) for the following Graph G (V1, V2, W) where G is a connected undirected nonnegative weighted graph with V as vertices, E as edges, and W as Weight.
(a, c,6),(a, b,6),(a, d,6),(b, d,2),(c, d,2)

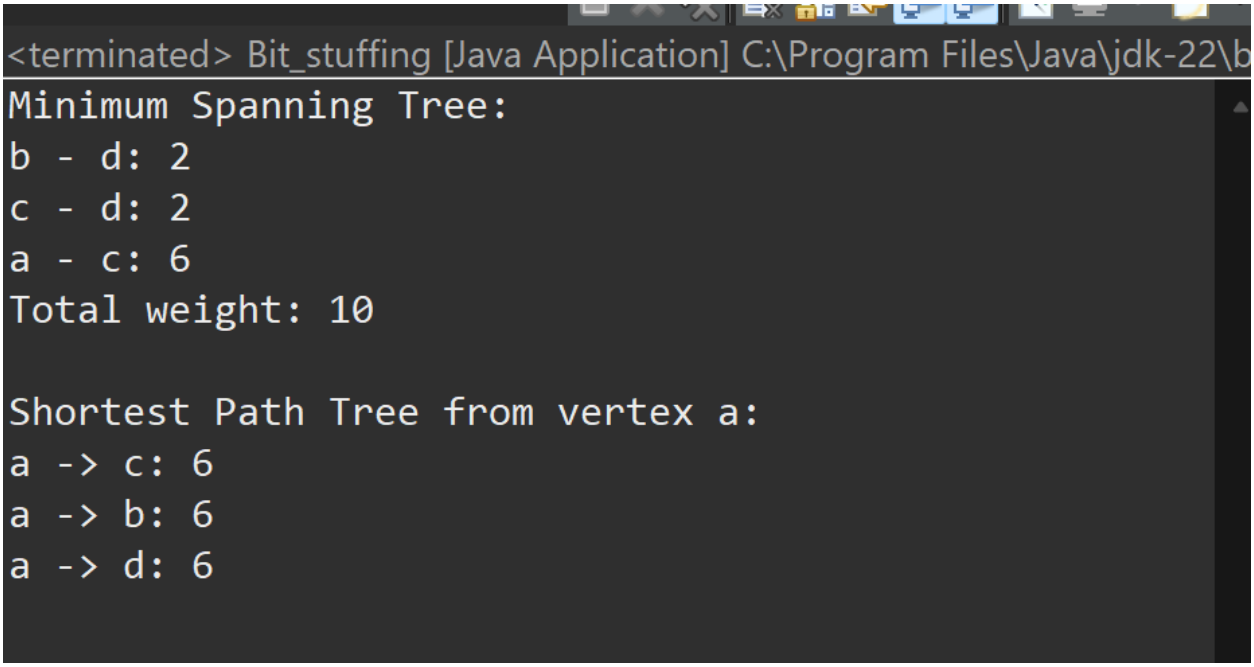
CODE:

```
package week3;
import java.util.*;
class Edge implements Comparable<Edge> {
  int src, dest, weight;
  Edge(int src, int dest, int weight) {
     this.src = src;
     this.dest = dest;
     this.weight = weight;
  }
  @Override
  public int compareTo(Edge other) {
     return this.weight - other.weight;
  }
}
class Graph {
  int vertices;
  List<Edge> edges;
  Graph(int vertices) {
     this.vertices = vertices;
     this.edges = new ArrayList<>();
  }
  void addEdge(int src, int dest, int weight) {
     edges.add(new Edge(src, dest, weight));
  }
  void findMST() {
     Collections.sort(edges);
     int[] parent = new int[vertices];
     for (int i = 0; i < vertices; i++) {
        parent[i] = i;
     }
     List<Edge> mst = new ArrayList<>();
```

```
        int mstWeight = 0;
        for (Edge edge : edges) {
            int rootSrc = find(parent, edge.src);
            int rootDest = find(parent, edge.dest);
            if (rootSrc != rootDest) {
                mst.add(edge);
                mstWeight += edge.weight;
                union(parent, rootSrc, rootDest);
            }
        }
        System.out.println("Minimum Spanning Tree:");
        for (Edge edge : mst) {
            System.out.println((char) (edge.src + 'a') + " - " + (char) (edge.dest + 'a') + ": " +
edge.weight);
        }
        System.out.println("Total weight: " + mstWeight);
    }
    private int find(int[] parent, int vertex) {
        if (parent[vertex] != vertex) {
            parent[vertex] = find(parent, parent[vertex]);
        }
        return parent[vertex];
    }
    private void union(int[] parent, int srcRoot, int destRoot) {
        parent[srcRoot] = destRoot;
    }
    void findSPT(int start) {
        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a ->
a[1]));
        int[] dist = new int[vertices];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[start] = 0;
        pq.add(new int[]{start, 0});
        boolean[] visited = new boolean[vertices];
        System.out.println("Shortest Path Tree from vertex " + (char) (start + 'a') + ":");
        while (!pq.isEmpty()) {
            int[] current = pq.poll();
            int u = current[0];
            if (visited[u]) continue;
            visited[u] = true;
            for (Edge edge : edges) {
                if (edge.src == u || edge.dest == u) {
                    int v = edge.src == u ? edge.dest : edge.src;
                    if (!visited[v] && dist[u] + edge.weight < dist[v]) {
                        dist[v] = dist[u] + edge.weight;
                        pq.add(new int[]{v, dist[v]});
                        System.out.println((char) (u + 'a') + " -> " + (char) (v + 'a') + ": " + dist[v]);
                    }
```

```
        }
      }
    }
  }
}
public class GraphMSTSPT {
  public static void main(String[] args) {
    Graph graph = new Graph(4);
    graph.addEdge(0, 2, 6); // a - c
    graph.addEdge(0, 1, 6); // a - b
    graph.addEdge(0, 3, 6); // a - d
    graph.addEdge(1, 3, 2); // b - d
    graph.addEdge(2, 3, 2); // c - d
    graph.findMST();
    System.out.println();
    graph.findSPT(0); // Start SPT from vertex 'a'
  }
}
```

OUTPUT:



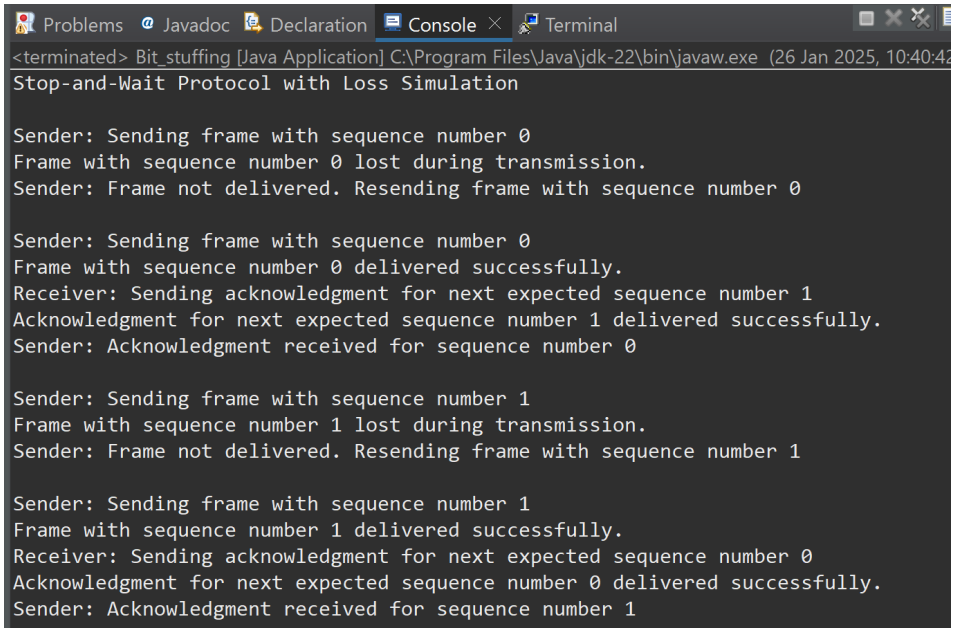
2. Construct a JAVA program for Stop and Wait Protocol. (Hint: The sender sends

a message and Receiver should give acknowledgement)

CODE:

```
package week3;
import java.util.Random;
class Sender {
  private final Random random = new Random();
  private final double lossProbability;
  Sender(double lossProbability) {
    this.lossProbability = lossProbability;
  }
  public boolean sendFrame(int seqNo) {
    System.out.println("Sender: Sending frame with sequence number " + seqNo);
    boolean isLost = random.nextDouble() < lossProbability;
    if (isLost) {
      System.out.println("Frame with sequence number " + seqNo + " lost during
transmission.");
      return false;
    }
    System.out.println("Frame with sequence number " + seqNo + " delivered
successfully.");
    return true;
  }
}
class Receiver {
  private final Random random = new Random();
  private final double lossProbability;
  Receiver(double lossProbability) {
    this.lossProbability = lossProbability;
  }
  public boolean sendAcknowledgment(int expectedSeqNo) {
    System.out.println("Receiver: Sending acknowledgment for next expected sequence
number " + expectedSeqNo);
    boolean isLost = random.nextDouble() < lossProbability;
    if (isLost) {
      System.out.println("Acknowledgment for next expected sequence number " +
expectedSeqNo + " lost during transmission.");
      return false;
    }
    System.out.println("Acknowledgment for next expected sequence number " +
expectedSeqNo + " delivered successfully.");
    return true;
  }
}
public class StopAndWaitProtocol {
  public static void main(String[] args) {
    final double lossProbability = 0.5;
    Sender sender = new Sender(lossProbability);
```

```
    Receiver receiver = new Receiver(lossProbability);
    int seqNo = 0;
    boolean frameSent, ackReceived;
    System.out.println("Stop-and-Wait Protocol with Loss Simulation\n");
    while (true) {
        frameSent = sender.sendFrame(seqNo);
        if (frameSent) {
            int nextExpectedSeqNo = 1 - seqNo;
            ackReceived = receiver.sendAcknowledgment(nextExpectedSeqNo);
            if (ackReceived) {
                System.out.println("Sender: Acknowledgment received for sequence number "
+ seqNo);
                seqNo = nextExpectedSeqNo;
            } else {
                System.out.println("Sender: Acknowledgment not received. Resending frame
with sequence number " + seqNo);
            }
        } else {
            System.out.println("Sender: Frame not delivered. Resending frame with sequence
number " + seqNo);
        }
        System.out.println();
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            break;
        }
    }
  }
}
```



OUTPUT