```
package pt.ipp.isep.dei.mdisc.SprintC;

import java.util.*;

public class EmergencyPathFinder {
    private int[][] weights;
    private String[] points;
    private List<Integer> assemblyPoints;

    public EmergencyPathFinder(int[][] weights, String[] points,
List<Integer> assemblyPoints) {
        this.weights = weights;
        this.points = points;
        this.assemblyPoints = assemblyPoints;
    }

    public List<Integer> dijkstra(int startIdx, int endIdx) {
        int n = weights.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        int[] prev = new int[n];

        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[startIdx] = 0;
        Arrays.fill(prev, -1);

        PriorityQueue<Integer> pq = new
PriorityQueue<>(Comparator.comparingInt(i -> dist[i]));
        pq.add(startIdx);

        while (!pq.isEmpty()) {
            int u = pq.poll();
            if (u == endIdx) break;
            if (visited[u]) continue;
            visited[u] = true;

            for (int v = 0; v < n; v++) {
                if (weights[u][v] > 0 && !visited[v]) {
                    int newDist = dist[u] + weights[u][v];
                    if (newDist < dist[v]) {
                        dist[v] = newDist;
                        prev[v] = u;
                        pq.add(v);
                    }
                }
            }
        }

        List<Integer> path = new ArrayList<>();
        for (int at = endIdx; at != -1; at = prev[at]) {
            path.add(at);
        }
        Collections.reverse(path);
        return path;
    }

    public List<List<Integer>> calculateAllPaths() {
        List<List<Integer>> paths = new ArrayList<>();
```

```
        for (int i = 0; i < points.length; i++) {
            if (!assemblyPoints.contains(i)) {
                int closestAPIdx = -1;
                int minDist = Integer.MAX_VALUE;
                for (int apIdx : assemblyPoints) {
                    List<Integer> path = dijkstra(i, apIdx);
                    int dist = path.size() - 1; // assuming each step has
uniform cost
                    if (dist < minDist) {
                        minDist = dist;
                        closestAPIdx = apIdx;
                    }
                }
                paths.add(dijkstra(i, closestAPIdx));
            } else {
                paths.add(new ArrayList<>());
            }
        }
        return paths;
    }
}
```