```java
class Dijkstra {
    11 usages
    private int[] dist;
    1 usage
    private boolean[] visited;
    4 usages
    private int[] prev;
    5 usages
    private int[][] graph;

    6 usages
    public Dijkstra(int[][] graph) {
        this.graph = graph;
        int n = graph.length;
        this.dist = new int[n];
        this.visited = new boolean[n];
        this.prev = new int[n];
    }

    6 usages
    public void computeShortestPaths(int start) {
        Arrays.fill(dist, Integer.MAX_VALUE);
        Arrays.fill(prev, val: -1);
        dist[start] = 0;

        PriorityQueue<Integer> queue = new PriorityQueue<>((a, b) -> Integer.compare(dist[a], dist[b]));
        queue.add(start);

        while (!queue.isEmpty()) {
            int u = queue.poll();

            for (int v = 0; v < graph.length; v++) {
                if (graph[u][v] != 0 && dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] < dist[v]) {
                    dist[v] = dist[u] + graph[u][v];
                    prev[v] = u;
                    queue.add(v);
                }
            }
        }
    }

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

    6 usages
    public int getPathCost(int target) {
        return dist[target];
```

```java
private static int findNearestAP(Graph graph, int startPoint) {
    int nearestAP = -1;
    int minCost = Integer.MAX_VALUE;
    Dijkstra dijkstra = new Dijkstra(graph.getWeightMatrix());

    for (Integer ap : graph.getAssemblyPoints()) {
        dijkstra.computeShortestPaths(ap);
        int cost = dijkstra.getPathCost(startPoint);
        if (cost < minCost) {
            minCost = cost;
            nearestAP = ap;
        }
    }
    return nearestAP;
```