CS5800: Algorithms Spring 2018
Assignment 8.1

Saptaparna Das

April 20, 2018

Algorithm:
The given problem can be framed as a graph problem where the nodes indicate landmarks in map. The edges between nodes can indicate the paths between landmarks. The time taken to travel one landmark to another can be weight of edges. It is also given, there are n deadly landmarks.
The underlying problem is given a directed weighted graph, a source and a destination, we need to find quickest path with at most k-1 deadly nodes.

The idea is to calculate for each vertex v , for each deadly node $0 <= j < k-1$, the quickest path from source to v using at most k-1 deadly nodes.
This problem shows Optimal substructure property. Because, optimal solution contains within it optiomal solution of the subproblems. That is to calculate quickest path from source to v, using j intermediate nodes, we need to calculate quickest path from source using j-1 intermediate deadly nodes. So, the recurrence relation looks like,
$dist(v, j) = min(dist(v, j-1), min(dist(d, j-1) + S(d, v)))$
where dist(v,j) denotes lowest time taken to go from source to v using at most j deadly nodes in original graph.
d is the deadly node.
S(d,v) is shortest distance from d to v without using deadly nodes.
Also, the problem shows overlapping subproblem property. As seen in the recurrence relation dist(u,v) is calculated multiple times for same set of u and v.
Hence we can dynamic programming here.

```
Step 1: Remove all outgoing edges from all deadly nodes and construct a graph G'.
Step 2: Calculate shortest paths S(u,v) for all vertices on this graph G', using any
          standard algorithm.
Step 3: Let dist(v,j) denotes lowest time taken to go from source to v using
          at most j deadly nodes in original graph.
          Calculate dist(v,j) by taking minimum of
          either quickest path from source to v using j-1 intermediate deadly nodes
          or quickest path from source to a deadly node d with j-1 intermediate
          deadly nodes + quickest path from d to v
          using no deadly nodes.
Step 4: Store the intermediate nodes that produce shortest path.
```

Analysis: Let total number of vertices be V in the graph. Then we have to store V*k entries in order to get the result. Saving each entry takes O(V) time. So, step 3 takes $O(V^2k)$ time. In step 2, if we use Dijkstra's algorithm to calculate shortest path, we need $O(|V||E|log|V|)$. Otherwise it takes $O(|V|^3)$ for Floyd. So total time is $O(|V||E|log|V| + V^2k)$
Space complexity: $O(|V|^2)$ to store the results of V*k entries and store quickest paths in graph G'.