1.
It is difficult to write a recurrence relation (of the style we used in prior DP problems) for Dijkstra because, there isn't a good way to incorporate the graph structure. The graph structure is unknown and so we can't concretely state which array values will depend on which ones. Also some distance values may or may not be overridden later on in the algorithm. Overall there are a lot of dependencies on a specific graph structure to create a recurrence relation.

2.
Assuming that we were successful in creating DP with a 2D array we would likely get a runtime of O(n^2) where n is the number of vertices in the graph. This is because we would be computing each of the cells in the 2D array a constant number of times on average (even with overriding values). Our current approach is also O(n^2) assuming the worst possible graph. This is because we have every node in the priority queue and for each we go through all of the neighboring vertices which is at worst (n-1). n*(n-1) is approximately O(n^2).

In terms of space the DP method would only need a 2D array (in theory). We currently use double because we have both a priority queue and a distance map which need to hold up to the total number of vertices. Each of these vertices also holds an internal list and parent object, and the priority queue and distance map also hold doubles for each vertex. The DP method could be saving a lot of space which would be useful with searching large graphs.

3.
Problems that are most suitable for dynamic programming have some clear recursive element. This way the run time of the recursive component can be cut down by saving their values. Dijkstra's algorithm doesn't have a clear recursive component. This means there isn't a clear way to implement it with DP. Although there may be a way to cleverly use a matrix to store values which would be reminiscent of DP, there isn't a clear cut method that reveals a recursive element in the algorithm (as far as I can see).