Question 1

1. States: Represented by a list of cities with a search agent. In my program, these things are represented by their own unique classes. Cities maintain their own x and y coordinates while they are stored in an ArrayList. Agents keep track of the city they are visited, along with those that they have already visited and have yet to visit.

Initial State: Search agent starts on City A. Remaining cities remain unvisited.

Goal State: Search agent ends on City A. All cities are visited.

Successor Function: Generates all the remaining unvisited cities for the search agent. The g cost is the Euclidean distance between the search agent’s current city to any of these remaining cities.

1. My heuristic function generates the minimum spanning tree of the remaining nodes unvisited by the search agent. It does so by using an algorithm very similar to Kruskal’s algorithm. It iterates through all possible edges and stores vertices in disjointed sets to achieve a complexity of ~ 2 log V, where V is the number of cities in the problem.

This heuristic never overestimates the minimum remaining distance of the search agent. This is trivial as if the remaining path were cheaper, then you could construct a cheaper spanning tree using that path. Therefore, this function is admissible.

1. Asd
2. Asd

Question 2

1. <V, D, C>  
   V = {V00 … Vrc}, where V represents a Cell and rc represents the rows and columns in a 9x9 grid

D = {1 … 9}

C = {C1, C2, C3}, where

C1 = {}

C2 = {}

C3 = {}

1. Implemented
2. I found in general for all versions that the graph formed a bell curve that was skewed right. The more “balanced” the number of initial values were, the more time it took for the algorithms to solve the problem. I suppose this makes sense, because the less initial values there are, the more “freedom” there is to finding a solution. But having more initial values means less possibilities, which reduces the number of checks needed.