CS 486 Assignment 1

Youngbin Kim (20543575)

Q1.

a) s h k c a b d m e n g

b) i : Yes. It is admissible since shortest path (when not blocked by the central barrier) from any state to the goal state is equal to the Manhattan distance from n. h (n) =< h'(n) for any n.

ii : s h k c a b d m g

iii : s h k c f p q r t g

-> Assumption for breaking ties : Node that is generated later gets expanded first.. (c was generated later than f, so expand c first)

Q2.

a) True. When all step costs are equal in Uniform-cost search, g(n) of n is directly proportional to the depth of n from the start state. Then, we are choosing which node to expand based on the depth, and this is Breadth-first search.

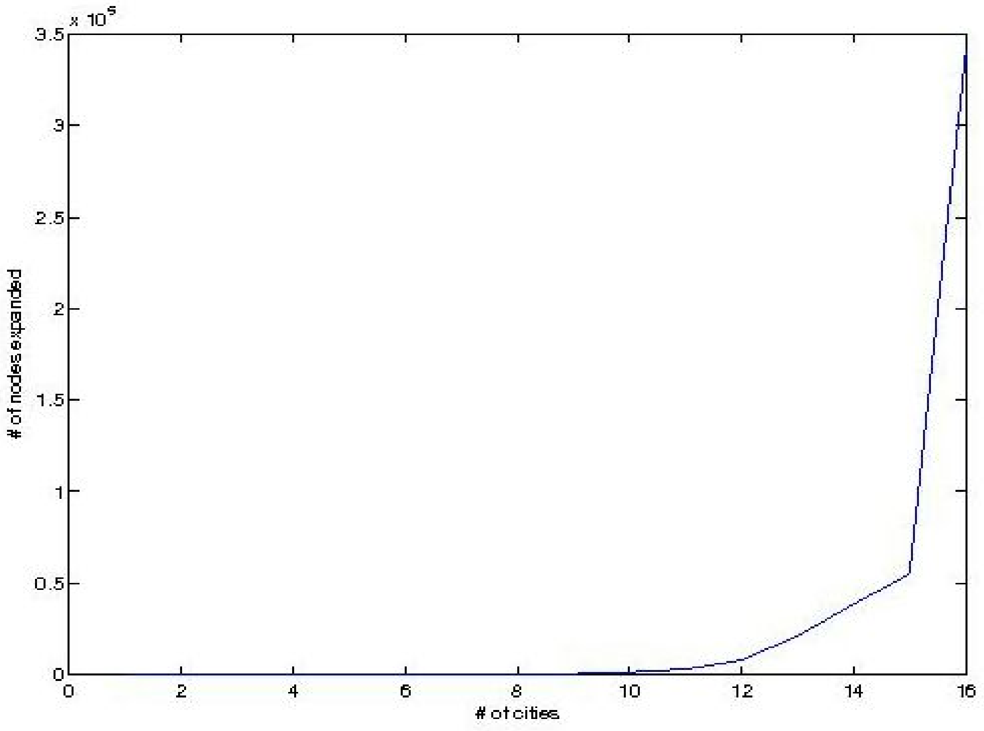
b) True. When we assign h(n) of Best-first search to be -depth(n), then it behaves same as Depth-first search.

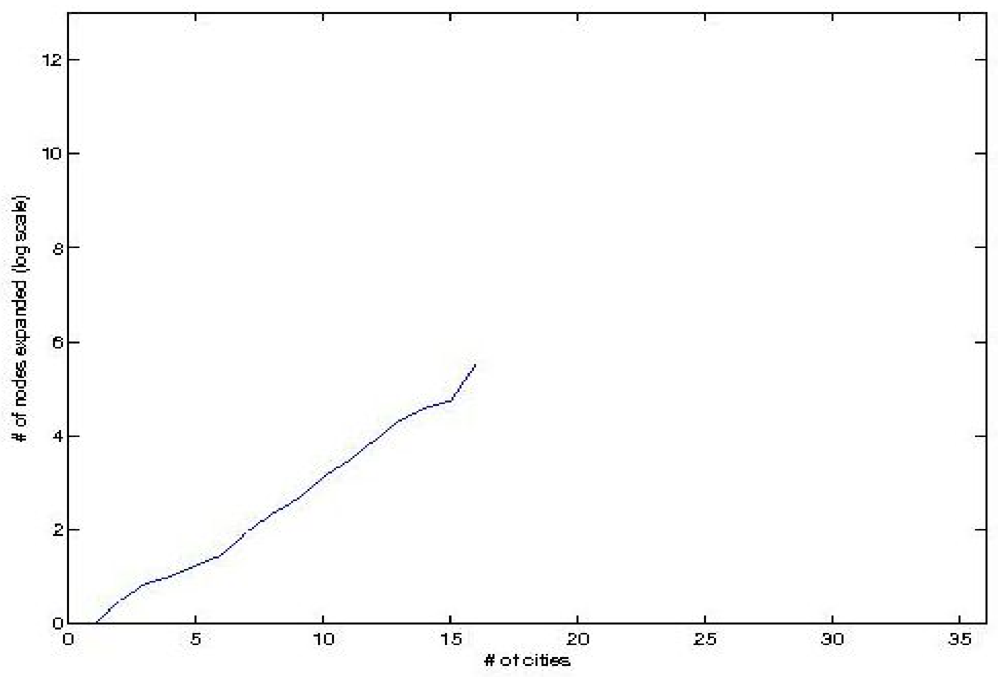
c) True. If we let h(n) of A* search to be 0 for any node, f(n) = 0 + g(n) = g(n). Then it behaves just like a Uniform-cost search.

Q3.

1. Representation of the problem
   a. States: Each state has a list of visited cities & current city.
   b. State representation: {C, arr}, where C is the current city and arr is the list of visited cities in visited order.
   c. Initial State: {A, []} (No city has been visited yet)
   d. Goal State: {A, [A , ….]} (where the array contains every city exactly once)
   e. Operators: Move to unvisited city. Operator({C1, {A, ….}}) = {C2, {A, … , C1}} for each unvisited city C2.
      So, for the state,
      1) find the unvisited cities.
      2) loop through each of them
      3) Each of those cities becomes the current city of new state
      4) put the current city of previous state in the path of new state.
      5) return list of states.
   f. cost: Euclidean distance between current cities of parent and successor state. dist(C2, C1 for the above example)
2. Heuristic Function
   a. h(n) = dist(n, closest unvisited city) + dist(A, closest unvisited city) + sum of edges of MST (minimum spanning tree) constructed from unvisited cities other than n.
   b. sum of edges of MST is used to estimate the length of path needed to visit all unvisited cities in TSP problem.
   c. It is admissible since the weight of path of optimal solution in TSP >= sum of MST. (Both soln path for TSP and edges in MST connect all unvisited cities, and by definition of MST, MST uses the edges with minimum sum). (Also, it's the solution of relaxed problem)

d. Relaxation: First, consider the problem restrictions. The solution path of TSP can be represented by the graph, where cities represent node, and path represent edges. Then, any node should be connected to another node, and these edges must have been attached so that it's possible to follow the edges without lifting the pen to visit all cities and come back to the first city. Then, relax the problem by removing the constraint, 'these edges must have been attached so that it's possible to follow the edges without lifting the pen to visit all cities and come back to the first city'. The solution of this relaxed problem is to find MST.

3. Result
   a. Average number of nodes generated
      i. 1 city : 1.0
      ii. 2 cities : 3.0
      iii. 3 cities : 7.0
      iv. 4 cities : 10.0
      v. 5 cities : 17.3
      vi. 6 cities : 28.3
      vii. 7 cities : 89.4
      viii. 8 cities : 217.3
      ix. 9 cities : 454.4
      x. 10 cities : 1342.8
      xi. 11 cities : 2961.0
      xii. 12 cities : 7942.5
      xiii. 13 cities : 21051.7
      xiv. 14 cities : 38644.3
      xv. 15 cities : 54861.4
      xvi. 16 cities : 340984.0
   b. Plot
      i. 



ii.

(log scale on y)

c. Expectation of A * search performance on 36 cities problem
    i. As shown above, the graph line is pretty linear for log scale. Thus, by extending the straight line up to 36 cities on x axis, we can see that the line will approximately reach 12 on y axis. That is, It's expected that around 10^12 number of nodes will be expanded for 36 cities problem.

d. Code
    i. compile : gradle build
    ii. run : gradle run