Meghna Malhotra
CS 510
HW 1
1.17.2016

Written Problems:

1.A.

Small Towers of Hanoi:

- Performance:
  1. State Representation: (Da, Rx) – a disk and a rod, with a = disk number and x = rod number
  2. Initial State: all 3 disks are on the same rod Rx
  3. Final State: all 3 disks are on the same rod Rz (where z != x)
  4. Completion Check: to check if the final state has been reached, we check if any of the rod stacks (other than the one we started on, in this case Rx) have the length 3

- Environment:
  1. 3 Disks – (Da, Db, Dc); where Dc is bigger than Db which is bigger than Da
  2. 3 Rods – (Rx, Ry, Rz); where each rod is a stack containing the numbers of all the disks that are on it; the stack can also be empty

- Actions:
  1. When (Da, Rx) wants to move to (Da, Ry) – check if Ry is empty or if the topmost element on the stack is larger than Da – and perform the move only if one of the above conditions is true
  2. To perform a move we need to pop the disk from its current stack and push it onto the stack representing the rod we wish to move it to

- Sensors:
  1. The game is perceived as 3 rods (Rx, Ry, Rz), each of which is either empty or has 1, 2 or 3 disks on it
  2. Each rod is a stack that contains either no disks, Da, Db or Dc or any combination of those, where Dc > Db > Da

Pac Man:

- Performance:
  1. State Representation: a rectangular grid of dimensions N x M; each position on the grid is represented as a set of co-ordinates (i, j)
  2. Initial State: there is a grid (of size N x M) that contains Pac Man, pellets and walls
  3. Final State: there is a grid (of size N x M) that contains Pac Man and walls but no pellets; when the list called pellets is empty, the game is over

- Environment:
  1. A rectangular grid with each position represented as a set of co-ordinates (i, j)
  2. The co-ordinates that represent Pac Man's position on the grid are called "man," the co-ordinates that represent the walls are contained in a list called "wall," the co-ordinates that represent pellets are contained in a list called "pells," and the co-ordinates of all the empty spaces are contained in a list called "open"

- Actions:
  1. Pac Man can move through the grid by moving one step up, down, right or left at a time
  2. He can step into empty spaces and onto pellets, but he cannot step over the walls
  3. When he steps onto a position that contains a pellet, the co-ordinates of that position are deleted from the list called "pellets" and added to the list called "open"
  4. Whenever Pac Man steps onto a valid position, the co-ordinates of that position – no matter which list it's in (except "walls") – should temporarily be changed to the co-ordinates contained in the variable "man"
  5. This will ensure that at any given time if we decide to go through all the lists we have on order to find "man," we will always get a valid result

- Sensors:
  1. The game is perceived as a board of size N x M that contains various elements like the Pac Man, pellets, walls or nothing
  2. When Pac Man runs into a wall, his position should not change and a message should be printed to the screen signaling an incorrect action
  3. When Pac Man wants to move to an empty space, he should be allowed to execute that action
  4. When Pac Man wants to move over a position containing a pellet, he should be allowed to execute that action and the co-ordinates of the pellet should be deleted from the list called "pellets" and added to the list called "open." When Pac Man moves on to another position, the position which previously contained a pellet, should now be perceived as empty

---

1.B.

Forward-Backward Search:

It would be easier to verify that Franklin is one of Eloise's ancestors.

Let us consider this problem as having a tree structure, with Franklin as the root and Eloise as one of the nodes. If we choose to search through the tree using one of the traditional searching algorithms like Depth-First Search (DFS) or Breadth-First Search (BFS), it will take us a lot longer to find the node containing Eloise, if there is one.

The worst-case complexity of DFS is $O(b^d)$ where b is the branching factor and d is the depth of a node. The worst-case complexity of BFS is $O(b^{d+1})$. Both these algorithms require more time and space because they traverse through each node in the tree till they find the one that they are looking for, or until they have searched all the nodes present.

But we decide to start at the node "Eloise" and work our way up to the root, we will be able to arrive at the conclusion much faster and utilize our time and space more efficiently. The space complexity for such an algorithm would be linear because we will not need to store any of the nodes in our path as we are looking for a node that cannot have a duplicate, i.e. the root. The time complexity for such an algorithm would be O(n) where n is the total number of nodes in the tree.