Sunny Patel
CS380
Assignment 2

1.A
Eloise claims to be a descendant of Benjamin Franklin. Which would be the easier way to verify Eloise's claim: By showing that Franklin is one of Eloise's ancestors or by showing that Eloise is one of Franklin's descendants? Why?

Thinking of this question in terms of a tree, Eloise would be a leaf in the tree and Franklin somewhere above Eloise's depth in the tree, let's say depth F. Therefore, it would be much easier for us to traverse up from Eloise's node , checking every node up to depth F than for us to check every node of Franklin's children, most likely taking a breadth-first approach from depth F down to height of the tree.

1.B: Uninformed Search with Negative Action costs

- Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal uninformed search algorithm to explore the entire state space.

  With negative costs the algorithm will always keep choosing the lowest option, so it would be better for the algorithm to choose a state with -10 cost then to choose a state with 1 cost and this could cause a loop because no matter how many times you choose -10 it's always going to be less than 1 or 0. It can also search the entire space if with each new state it finds a more negative cost.

- Does it help if we insist that step costs must be greater than or equal to some negative constant c? Consider both trees and graphs.

  It doesn't help for a graph because if it finds the state with the most negative cost then it will be stuck in an infinite loop choosing that cost over and over again. For a tree, looping wouldn't be a factor but it would still choose the –Max over any other option even if the goal step is only 1 cost away.

- One can easily imagine actions with high negative cost, even in domains such as route finding. For example, some stretches of road might have such a beautiful scenery as to far outweigh the normal coasts in terms of time and fuel. Explain, in precise terms, whithin the context of state-space search, why humans do not drive around scenic loops indefinitely,

and explain how to define the state space and actions for route finding so that artificial agents can also avoid looping.

Well for humans, rewarding actions such as scenic routes depreciate with repetition of the same action. For example, if we say a particular scenic route provides 10pts reward the first time, the second time it might be 8pts and the third time around it would be 5pts, and continue to decrease. Putting this in artificial agents context, it would be -10 cost (10 points reward for choosing this route because of its scenery) and after selecting that route we would change the cost to a value greater than the current value of -10. Therefore, we could have a depreciation function do this for us. In a graph, next time around the agent may still choose the scenic route but eventually it will choose the more optimal route in terms of distance and time because the scenery will have no value.

1.C: Backtracking

Modify the backTrack algorithm (shown below) so that instead of returning the first solution path it finds, it continues and finds all solution paths.

```
backTrack (stateList , depthBound)
{
      state = first element of stateList
      if state is a member of the rest of stateList, return FAIL
      if goal(state), save this state by appending it to goalsSoFar
      array.
      if length(stateList) > depthBound, return FAIL
      moves = getPossibleMoves(state)

      for each move m in ruleSet {
        newState = applyMoveCloning(state,m)
        newStateList = addToFront(newState,stateList)
        path = backTrack(newStateList)
        if path != FAILED return append(path,m)
      }
}
```

Solve this problem using backTrack (shown above), with initial state (6,5,1), and a depth bound of 3.

Total BackTrack calls : 30
Failure backtrack calls : 28
Solution path:

Calling backtrack w/ initial state of (6,5,1) depth 3.
Then apply move (0, -1, 1) call bT w/ state (6,4,2)
Then apply move (-2, 0, 2) arrive at goal state (4, 4, 4)

Solve this problem using backTrack (shown above), with initial state (6,5,1), and a depth bound of 5.
Total Calls : 33
Failures : 27

Solution Path =

(-5,5,0)
(+1,-1,0)
(+2,-2,0)
(0,-1,+1)
(0,-2,+2)