1. Written Problems
    A. Forward-Backward Search
        - Suppose we have a family tree of Benjamin Franklin family, and one the node is Franklin, then showing Franklin is one of Eloise's ancestors is backward search(base on Eloise and search from bottom to top), and showing Eloise is one of Franklin's descendants forward search(base on Franklin and search from top to bottom). The backward search is a easier way to verify this. It is because Franklin has more than one child, if we use forward search, we need to visit all the leaves under Franklin node and check if it is Eloise. However, if we use backward search, we only need to visits the nodes which are ‘Eloise’ leaf’s ancestors, and check if the node is Franklin. The data size in backward search is obviously smaller than forward search, therefore the backward search is a easier way.

    B. Uninformed Search with Negative Action costs
        - Because if the actions are large negatives costs, the path cost can be extremely reduced with these actions. Therefore an optimal uninformed search algorithm can be applied, with least path cost.
        - No. Because for both trees and graphs, the algorithm will search the entire space anyway. So insist the step cost greater or equal to a negative number won’t help, but affect the optimal algorithm.
        - A loop which using this set will lead to the least cost path, which is a optimal behavior of this set.
        - As human, they avoid infinite loop because the cost of fuel of time will be high if they do so. To define a better state space and actions for artificial agent, fuel and time cost must be also considered, to avoid the outweighed scenery lead to a indefinite loop.
        - When the step costs are all the same, and they never change the state with the ordered results. Then we can use a loop.

    C. Backtracking

```
list L
int i=-1
backTrack (stateList , depthBound) {
 state = first element of stateList
 if state is a member of the rest of stateList, return FAIL
 if goal(state), return NULL
 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, depthBound)
        if path != FAILED {
          i++;
          return L[i].append(path,m)
        }
      }
    }
```

D. Baskets of Marbles
    - backTrack 3:
      Path: (4,4,4),(6,4,2),(6,5,1)
      Backtrack calls: 7
      Failures: 3

    - backTrack 4:
      Path: (4,4,4),(6,2,4),(6,4,2),(6,5,1)
      Backtrack calls: 12
      Failures: 6

    - Modified backTrack 4:
      Path: (4,4,4),(6,2,4),(6,4,2),(6,5,1)
      Backtrack calls: 12
      Failures: 6

    - depth-first:
      Path: (6,5,1),(6,4,2),(4,4,4)
      Nodes generated: 12
      Failures: 3

    - breadth-first:
      Path: (6,5,1),(6,4,2),(4,4,4)
      Nodes generated: 12
      Failures: 3