CS510 HW Assignment 2

Christopher Geib

Due: Jan 30th 11:59pm

Part 1: Written Problems (60 points)

1.A: Forward-Backward Search (5 points)
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?

1.B: Uninformed Search with Negative Action costs (15 points)
(this is problem 3.8 of the text book)
In class, we always assumed that the cost of each action is positive. Let us explore this decision in more depth.

- 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.
- 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.
- Suppose that a set of actions forms a loop in the state space such that executing the set in some order results in no net change to the state. If all of these actions have negative cost, what does this imply about the optimal behavior for an agent in such an environment?
- 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, within 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.
- Can you think of a real domain in which step costs are such as to cause looping?

1.C: Backtracking (10 points)
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), 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 return append(path,m)
  }
}
```

1.D: Baskets of Marbles (30 points)
Consider the "Baskets of Marbles" problem, described as follows:

Each of three baskets contains a certain number of marbles. You may move from one basket into another basket as many marbles as are already there, thus doubling the quantity in the basket that received the marbles. You must find a sequence of moves that will yield the same number of marbles in the three baskets (or decide that no such sequence exists).

For each of the exercises that follow, you are to solve this problem using a particular algorithm and initial state. Moves are generated in an uninformed order. In order to be consistent regarding the order in which you determine and apply moves, consider them in lexicographic order; that is, the move (-1,0,1) (move one marble from basket 1 to basket 3) comes before (0,-1,1) (move one marble from basket 2 to basket 3). Show the solution path found and report on the number of calls to backTrack, and number of failures before finding the solution. You do not need to write a program for this exercise.

- Solve this problem using backTrack (shown above), with initial state (6,5,1), and a depth bound of 3.
- Solve this problem again using backTrack, with the same initial state and a depth bound of 5.
- Solve this problem again using your modified version of backTrack, with the same initial state and a depth bound of 5.
- Solve this problem again using a depth-first search approach, with the same initial state. (Instead of reporting the number of calls to backTrack, show the number of nodes generated.)
- Solve this problem again using a breadth-first search approach, with the same initial state. (Instead of reporting the number of calls to backTrack, show the number of nodes generated.

Part 2: Programming Assignment (40 points)
Using the code you wrote for assignment 1, write:

- A function that solves a given sliding bricks puzzle using a breadth-first strategy.
- A function that solves a given sliding bricks puzzle using a depth-first strategy.

Notice that the search space is a graph, so you will have to keep track of all the states visited so far, and make sure your algorithm does not get stuck in loops.

When the solution is found, it should be printed to screen. Print the list of moves required to solve the state, and the final state of the puzzle, for example:

```
(2,left)
(4,down)
(3,right)
(2,up)
(2,up)
5,4,
1,2,2,1,1,
1,0,0,3,1,
1,0,0,4,1,
1,1,1,1,1,
```

Together with the source code, turn in (in a plain text file called 'output-part2.txt') the output that your program generates for the following four levels: SBP-level0.txt, SBP-level1.txt, SBP-level2.txt, SBP-level3.txt. Also, report how many nodes are explored, how much time does the search take, and the length of the solution found

Using these search strategies, it is unlikely that your program handles puzzles much larger than the ones linked above (in assignment 3 you will implement much better strategies!). But in case you want to test out the limits of your program, you can use these more complex puzzles: SBP-bricks-level1.txt, SBP-bricks-level2.txt, SBP-bricks-level3.txt, SBP-bricks-level4.txt, SBP-bricks-level5.txt, SBP-bricks-level6.txt, SBP-bricks-level7.txt.

Part 3: Extra Credit (10 points)
Using the code you wrote for assignment 1, write:

- A function that solves a given sliding bricks puzzle using an iterative deepening search strategy. Provide the same output as requested for part 2, and turn it in in a text file called 'output-part3.txt'.

What to Submit
All homework for this course must be submitted electronically using Blackboard. Do not e-mail your assignment to a TA or Instructor! If you are having difficulty with your Blackboard account, you are responsible for resolving these problems with a TA, an Instructor, or someone from IRT, before the assignment it due. It is suggested you complete your work early so that a TA can help you if you have difficulty with this process.

For this assignment, you must submit:

- A PDF document with your answers to the "Written problems to be turned in" (Do **NOT** submit a Microsoft Word, OpenOffice document, or any other format that is not a PDF, you will lose points if you do so!).
- Your C/C++/Java/Lisp,/Python/Javascript/... source code, written documentation for your program.
- Use a compression utility to compress your files into a single file (with a .zip extension) and upload it to the assignment page.

Important: Assignment 3 will build on top of this assignment. So, make sure that you complete it adequately. Otherwise, you will have problems completing assignments 3.

Academic Honesty
You must compose all program and written material yourself, including answers to book questions. All material taken from outside sources must be appropriately cited. If you need assistance with this aspect of the assignment, see a consultant during consulting hours