ARTIFICIAL INTELLIGENCE I

FALL 2020

PROGRAMMING ASSIGNMENT 3

Name: Nudrat Nawal Saber

UTA ID :1001733394

Problem 1:Missionaries and Cannibals

Implementation:

For 15 cannibals, and 15 missionaries,the start state is (15,15,1,0,0,0) and the goal state is (0,0,0,15,15,1) .At first, the first three elements in the start state mean the missionaries, cannibals, and the boat, respectively, on the river's left side. The last three elements represent the number of missionaries, cannibals, and the boat. Our goal is to move all the missionaries, cannibals, and boat on the ride side of the river. In action state, it is represented by (m,c,b) [m=missionaries,c=cannibals,b=boat]the positive elements mean the action has occurred form the left to the right side of the river, and the negative elements mean action is occurred from the right to left side of the river.It is same for 20 cannibals and 20 missionaries.

In problem 1, I modified the code from the web site(https://github.com/aimacode/aima-lisp) to solve the problem 1.I changed the successor function as it is for boat capacity of 6,in the sample code there were successor function for boat capacity of 2.I also modified the function h-cost which was by default 0.I also made some changes to sample code to compile it .

In the CLISP environment, my code runs fine and it gets an optimal solution to the goal. To check and run the code,load the "Missionaries and Cannibals.lisp" and it will run correctly.

At first I used the Domain part, to define the cannibal problem, state, h-cost, g-cost, successors and so on. In order to do search,I have constructed the state as a node in the tree. The node structure has several fields here, such as state, parent and f-cost. There are some functions defined to manipulate node, such as expand, create-start-node,then I used a search algorithm. There are some simple algorithms and also some algorithms that avoid repeat,I.E. A*-search. For the missionary and cannibal problem breadth first search is very inefficient.I used A* search that handles repeated states.To do search, we need a queue data structure to store the nodes that we expanded. Besides, to do A* search, we need a function to insert the node by the priority of its f-cost. This is implemented by enqueue-by-priority and heap-insert.And then used function for printing solutions. The search algorithm returns the goal node containing its parent node and contains the information of the whole solution path and prints the solution.

The h-cost function that I modified and used is,h-cost=(m1+c1)/2.I used that as heuristic, it gives an optimal solution.At first ,I used (m1+c1)/boat capacity ,which is admissible ,but it expands the more nodes than (m1+c1)/2 .

Output:

The following result is for 15 cannibals, and 15 missionaries.

```
Action              --> State
================================
                    --> (15 15 1 0 0 0)
(0 6 1)             --> (15 9 0 0 6 1)
(0 -2 -1)           --> (15 11 1 0 4 0)
(5 1 1)             --> (10 10 0 5 5 1)
(-1 -1 -1)          --> (11 11 1 4 4 0)
(3 3 1)             --> (8 8 0 7 7 1)
(-1 -1 -1)          --> (9 9 1 6 6 0)
(3 3 1)             --> (6 6 0 9 9 1)
(-1 -1 -1)          --> (7 7 1 8 8 0)
(3 3 1)             --> (4 4 0 11 11 1)
(-1 -1 -1)          --> (5 5 1 10 10 0)
(3 3 1)             --> (2 2 0 13 13 1)
(-1 -1 -1)          --> (3 3 1 12 12 0)
(3 3 1)             --> (0 0 0 15 15 1)
========================================
Total of 35 nodes expanded.
```

The following result is for 20 cannibals and 20 missionaries.

```
Action              --> State
================================
                    --> (20 20 1 0 0 0)
(0 6 1)             --> (20 14 0 0 6 1)
(0 -2 -1)           --> (20 16 1 0 4 0)
(5 1 1)             --> (15 15 0 5 5 1)
(-1 -1 -1)          --> (16 16 1 4 4 0)
(3 3 1)             --> (13 13 0 7 7 1)
(-1 -1 -1)          --> (14 14 1 6 6 0)
(3 3 1)             --> (11 11 0 9 9 1)
(-1 -1 -1)          --> (12 12 1 8 8 0)
(3 3 1)             --> (9 9 0 11 11 1)
(-1 -1 -1)          --> (10 10 1 10 10 0)
(3 3 1)             --> (7 7 0 13 13 1)
(-1 -1 -1)          --> (8 8 1 12 12 0)
(3 3 1)             --> (5 5 0 15 15 1)
(-1 -1 -1)          --> (6 6 1 14 14 0)
(3 3 1)             --> (3 3 0 17 17 1)
(-1 -1 -1)          --> (4 4 1 16 16 0)
(3 3 1)             --> (1 1 0 19 19 1)
(-1 -1 -1)          --> (2 2 1 18 18 0)
(2 2 1)             --> (0 0 0 20 20 1)
========================================
Total of 44 nodes expanded.
```

Problem 2: 8-Puzzle

Implementation:

For problem 2, I have written my own code but use some structures from the web site(https://github.com/aimacode/aima-lisp) such as node and queue. I used a list, such as ((E 1 3) (4 2 5) (7 8 6)), to represent the start state. Besides,the four actions are represented by LEFT,RIGHT,DOWN,UP I took “move the space”. I used the function heuristic=number of misplaced tiles. In CLISP environment, load the “8_puzzle.lisp” and it will run perfectly.It reaches the goal state ((1, 2, 3),(4, 5, 6),(7, 8, E)) after some actions. I used A*-search algorithm for this problem.There are some cases when the program can’t find a solution in a certain amount of time,that is an infeasible puzzle.I.E.((8 1 2) (E 4 3) (7 6 5)).

Output:

```
((E 1 3) (4 2 5) (7 8 6))
((1 E 3) (4 2 5) (7 8 6))
((1 2 3) (4 E 5) (7 8 6))
((1 2 3) (4 5 E) (7 8 6))
((1 2 3) (4 5 6) (7 8 E))
Total of 4 nodes expanded.
```