Lab - 01 Date: 4/10/2024

1 Tic - Tac - Toe implementation using python

Pseudocode

```
function minimax (node, depth, isMaximizingPlayer)
    if node is a terminal state:
        return evaluate(node)
    if isMaximizingPlayer:
        bestValue = -inf
        for each child in node:
            value = minimax (child, depth, false)
            bestvalue = max(bestvalue, value)
        return bestvalue
    else:
        bestvalue = +inf
        for each child in node
            value = minimax(child, depth, true)
            bestvalue = min(bestvalue, value)
        return bestvalue
```

R 4/10/24

Output:

```
Player: 'O'      Bot: 'x'

x |_|_
_ |_|_
_ |_|_
-> enter position for O: 2
x |O|_
_ |_|_
_ |_|_

x |O|_
x |_|_
_ |_|_
-> enter position for O: 7
x |O|_
x |_|_
O |_|_

x |O|_
x |x|_
O |_|_

-> enter position for O: 9
x |O|_
x |x|_
O |_|O

x |O|_
x |x|x
O |_|O

Bot Wins!
```

Lab-02

2. Implement Vaccum Cleaner Agent

Pseudocode

```
Function vaccum_world(){
    initialize goal_state = {'A':'0', 'B':'0'}
    initialize cost = 0
    Input location
    Input status for location
    Input status for other location
    Print Initial condition for Location, goal state

If location input = 'A' and status_input = '1' then {
    print location A is dirty
    goal_state['A'] = '0'
    cost += 1
    print cost for ~~the~~ cleaning 'A' - cost

If status for other location = '1' then {
    print location B is Dirty
    cost += 1
    print cost for moving right as cost }

~~else~~
    ~~print "Vaccum is placed in Location B"~~
    If status_input == 1 then Location B is dirty

else {
    print Location A is already clean
    If status of other location = 1 then {
        print Location B is dirty
        cost += 1
        Print cost for moving right = cost
        cost += 1
        Print total cost of cleaning, cost
}}

Else { ~~print~~ print vaccum is at Location B {
    If status == '1' then print 'Location B is dirty'
    cost += 1
    cost for cleaning, cost
    If status of other location = 1, then {
        print Location = 1 then,
        cost += 1 ~~print~~
        print cost for moving Left, cost
        goal state['A'] = '0'
        cost += 1
        print cost for cleaning, cost }

    else {
        print location B is already clean
        If status other location = '1' then {
            print Location A is dirty
            cost += 1
            print cost for moving Left. cost
            goal state['A'] = '0'
            cost += 1
            print cost for clean, cost
    }
    print performance Measurement cost.

}
```

Output

Locations: A-0 B-1
Enter Location of vaccum: 1 (at B)
Enter status of Room (0 for clean, 1 for dirty): 1
Enter status of other room (0 for clean, 1 for dirty): 0
Initial Location Condition

Vaccum is placed in B
Location B is Dirty
Cost for cleaning: 1

Location B has been cleaned.

Location A is already clean

Goal state {A 'O', B: 'O'}

Performance measurement: 1

3. Implement puzzle problem using BFS algorithm

Algorithm:

Let fringe be a list containing the initial state

Loop If fringe is empty return failure

Node ← remove-first (fringe)

If Node is a goal

then return path from initial state to node

else generate all successors of Node and add generated nodes to the back of fringe.

End Loop

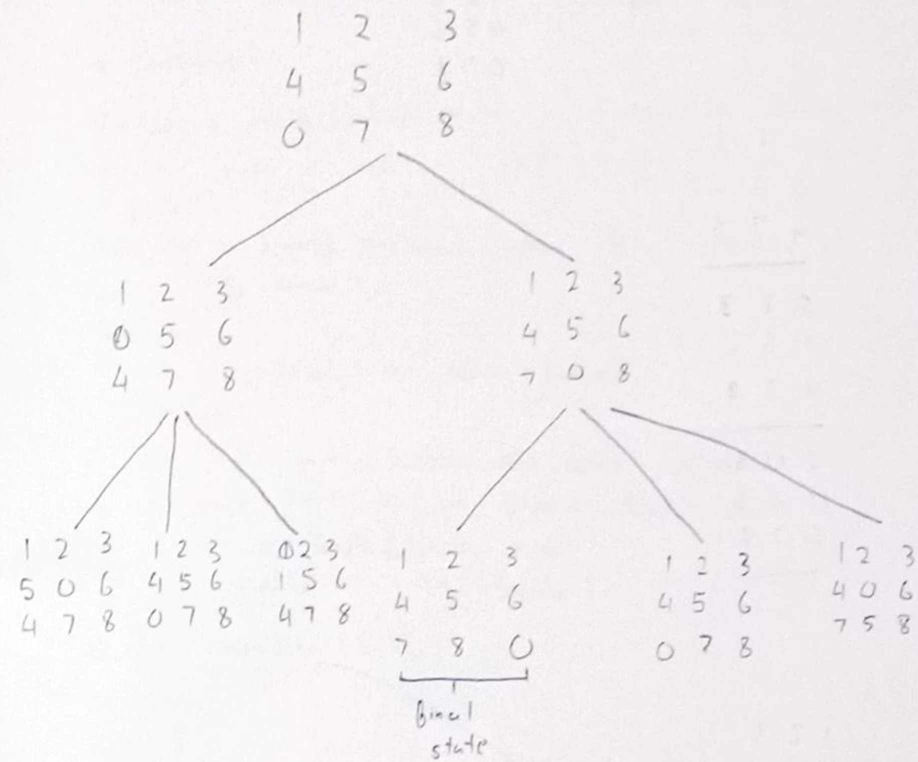
Consider initial state and final state

Initial

1	2	3
4	5	6
0	7	8

Goal

1	2	3
4	5	6
7	8	0



ii) Implement 8 puzzle problem using DFS Algorithm

Algorithm:

Let fringe be the list containing the initial state

Loop If fringe is empty, return failure

Node ← remove-first (fringe)

If Node is a goal

then return path from initial state to ~~final state~~ node

else generate all successors of Node & E add generated Nodes to the front of the fringe

Initial Stagte :

1 2 3
4 5 6
0 7 8

1 2 3
0 5 6
4 7 8

---

0 2 3
1 5 6
4 7 8

---

2 0 3
1 5 6
4 7 8

---

1 2 3
4 5 6
7 8 0

} final stat

Lab 04 : A* Search Algorithm

Pseudocode :

function A* search(problem) returns a solution or failure
node ← a node n with n. states = problem. initial state
n.g = 0
frontier ← a priority ~~que~~ queue ordered by ascending g+h,
only element n
loop do
if empty? (frontier) then return failure
n ← pop
if problem. goalTest (n. State) then return solution(n)
for each action a in problem. action(n. state) do
n' ← childNode (problem n. a)
insert (n', g(n') + h(n'), frontier)

Output : (Manhatten Distance)

Start State:

2 8 3
1 6 4
7   5

Goal State:

1 2 3
8   4
7 6 5

Solution found in 5 moves using manhatten heuristic

2 8 3
1 6 4
7   5

↓

~~Move 1~~

2 8 3
1   4
7 6 5

↓

Move 2

2   3
1 8 4
7 6 5

Move 3

  2 3
1 8 4
7 6 5

Move 4

1 2 3
  8 4
7 6 5

Move 5

1 2 3
8   4
7 6 5

2 8 3
1 6 4
7 5

Misplaced Tiles

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix} \quad g(n)=0,\ h(n)=4,\ f(n)=4 \qquad \begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

(initial) (goal)

$g(n)=1$

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 5 & 0 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 0 & 7 & 5 \end{bmatrix}$$

$h(n)=3$, $f(n)=4$ — $h(n)=5$, $f(n)=6$ — $h(n)=5$, $f(n)=6$

$g(n)=2$

$$\begin{bmatrix} 2 & 0 & 3 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 4 & 0 \\ 7 & 6 & 5 \end{bmatrix}$$

$h(n)=3$
$f(n)=5$

$g(n)=3$

$$\begin{bmatrix} 0 & 2 & 3 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 3 & 0 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$h(n)=2$
$f(n)=5$

$g(n)=4$

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$h(n)=1$
$f(n)=5$ — $f(n)=5$

~~Manhattan Distance~~

Manhattan Distan

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 0 & 5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

initial goal

$g(n)=1$

$$\begin{bmatrix} 2 & 8 & 3 \\ 1 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & 5 & 0 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 0 & 7 & 5 \end{bmatrix}$$

$f(n)=5$ — $f(n)=7$ — $f(n)=7$

$g(n)=2$

$$\begin{bmatrix} 2 & 0 & 3 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 0 & 1 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 8 & 3 \\ 1 & 4 & 0 \\ 7 & 6 & 5 \end{bmatrix}$$

$f(n)=5$ — $f(n)=6$ — $f(n)=7$

$g(n)=3$

$$\begin{bmatrix} 0 & 2 & 3 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 3 & 0 \\ 1 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$f(n)=5$ — $f(n)=7$

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 8 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$f(n)=5$

9

$$\begin{bmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

$f(n)=5$

Lab - 03 Date 8/11/2024

N-Queen Implementation Using hill-Climbing algorithm

Algorithm for hill Climbing Algorithm

function hill-Climbing (Problem) returns a state that is local maximum

current ← Make-Node (Problem, Initial-State)

loop do

neighbour ← a highest valued function

if neighbour . value ≤ current . value then return state

end if

current ← neighbour

end loop.

Execute

State Space Tree