AI
TT-1

JUNAID. GIRKAR
60004190057
TE COMPS A4

23/10/21

Q1
ANS "SeniorHelp"

~~PERFORMANCE MEASURE~~ :
Tasks to be performed :-
→ Heart rate check and notify accordingly
→ Blood Pressure check and notify accordingly
→ Blood sugar check and notify accordingly
→ Body temperature check and notify accordingly
→ Fall alert

Sensors to be used / actuators are speakers, Liquid Crystal Display (LCD) for notification, vibration motor, wifi connectivity module

PERFORMANCE ANALYSIS : → Correctness of readings
→ Notification alert time
→ Comfort level

ENVIRONMENTS :
• Arm of Human.
• Stochastic
• Single agent
• Partially observable as it cannot get correct blood sugar readings all the time
• Dynamic environment as BP, pulse. keep changing
• Continuous environment as 24/7 monitoring is required

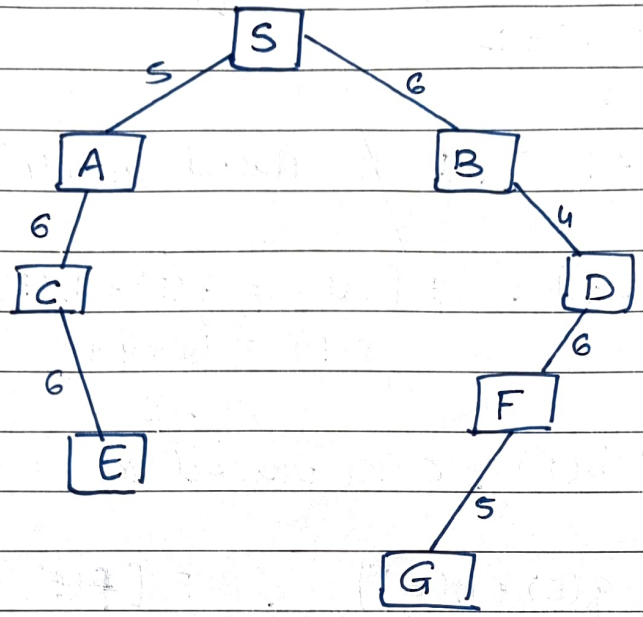
ACTUATORS : → Screen (LCD) → Vibration motor
→ speakers ~~→ wifi connectivity module~~

SENSOR : → Heart rate sensor → Blood sugar sensor
→ Blood Pressure sensor → temperature sensor
→ Fall alert (Gravity) sensor → Wifi module sensors.

JAGirkar

G000419005 ⇒

Q2



$h(S) = 17$ $h(B) = 14$ $h(D) = 12$ $h(F) = 12$

$h(A) = 16$ $h(C) = 12$ $h(E) = 10$

Initial state = S

Final state = G

For A* search :- $f(N) = g(h) + h(n)$

1) A $[f(n) = g(n) + h(n)]$, B $[f(B) = g(B) + h(B)]$

$f(A) = 5 + 16$ $f(B) = 6 + 14$

$= 21$ $= 20$

$\because f(B) < f(A)$, B is placed closed

2) A $[f(A) = 21]$, D $[f(D) = g(D) + h(D)]$

$f(D) = (6+4) + 12$

$= 22$

$\because f(A) < f(D)$, A is placed in closed

JAG

6000419005

3) C [f(C) = g(C) + h(C)] , D [f(D) = 22]

f(C) = (5+6) + 12

= 23

∵ f(D) < f(C), D is placed in closed

4) C [f(C) = 23] , F [f(F) = g(F) + h(F)]

f(F) = (6+4+6) + 12

= 28

∵ f(C) < b(F), C is placed in closed.

5) E [f(E) = g(E) + h(E)] , F [f(F) = 28]

f(E) = (5+6+8) + 10

= 27

∵ f(E) < f(F), E is placed in closed

6) F (f(F) = 28)

∵ E has no neighbours, F is placed in closed.

7) G [f(G) = g(G) + h(G)]

f(G) = (6+4+6+5) + 0

= 21

∴ PATH : S → B → D → F → G

COST = 21

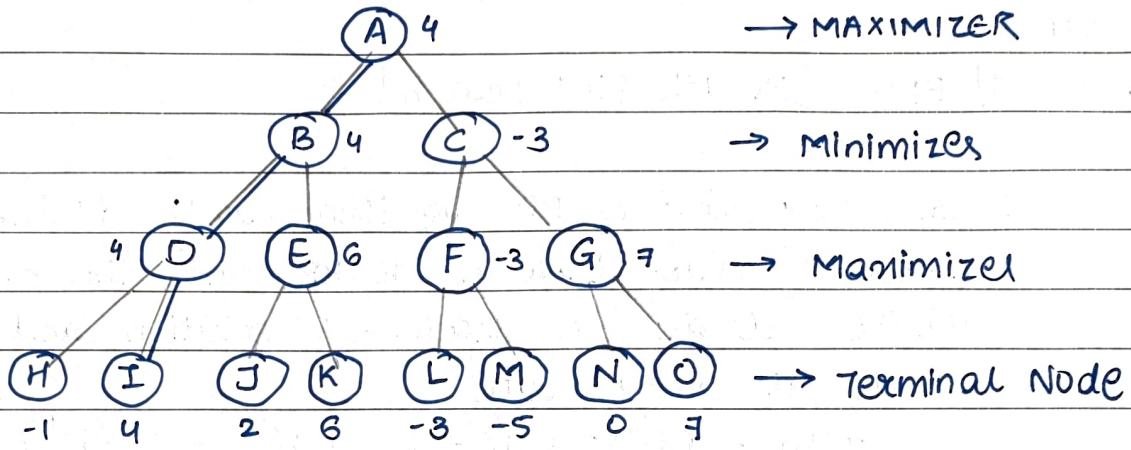
@00041900 57

Q3

ANS
- Minmax is a ~~type~~ kind of backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that oppernant is also playing optimally.
- Min-Max algorithm uses recursion to search through the game-tree and is mostly used for game playing in AI such as chess, checkers, and other two player games. This algorithm computes the minmax decision for the current state.
- In this algorithm, two players play the game, one is called MAX and the other is called MIN. Both the players fight to get the maximum benefit and are oppernent of each other where MAX will select the maximized value and MIN will select the minimized value
- the Min-Max algorithm peryorms a depth-first search algorithm for the enploration of the complete game tree and proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

EXAMPLE : consider a game which has 8 final states and paths to reach final state are from root to 8 leaves of a peryect binary tree as shown below

6000419005?



STEP 1: let A be initial state. Suppose maximizer takes first turn which has ~~first~~ worst case worst value = $-\infty$, and minimizer has worst-case inital value = $+\infty$

STEP 2: we find utilities value for the maximizer and we compare each value in the terminal state with initial value and determine higher node value.

- For node D $\max(-1,-\infty) \Rightarrow \max(-1,4) = 4$
- For node E $\max(2,-\infty) \Rightarrow \max(2,6) = 6$
- For node F $\max(-3,-\infty) \Rightarrow \max(-3,-5) = -3$
- For node G $\max(0,-\infty) \Rightarrow \max(0,7) = 7$

STEP 3: In next step, minimizer will compare all node values with $+\infty$ and will find 3rd layer values

- For node B = $\min(4,6) = 4$
- For node C = $\min(-3,7) = -3$

STEP 4: Now maximizer will again choose the maximum of all nodes and find the maximum value for root node.

- For node A $\Rightarrow \max(4,-3) = 4$

This was the complete workflow for a minmax two player game.

6000U190057

Q4

ANS 1] BFS [Breadth First Search]

i) Time complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest node. where d = depth of shallowest solution and b is a node at every state

$$T(b) = b + b^2 + b^3 + \dots + b^d = O(b^d)$$

ii) Space complexity is given by the memory size of frontier which is $O(b^d)$

iii) Completeness : BFS is complete which means that if the shallowest node is at some finite depth then BFS will find it.

iv) Optimality : BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2] DFS [Depth First Search]

i) Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by :

$$T(n) = n + n^2 + n^3 + \dots + n^m = O(n^m)$$

where m = max depth of any node and this can be much larger than d (shallowest depth solution)

ii) Space complexity : DFS algorithm needs to store only

6000419005 7

single path from the root node, hence space complexity of DFS is equivalent to size of the fringe set which is o(bm)

iii) completeness: DFS search algorithm is complete with finite state spaces as it will expand every node within a limited search tree

iv) optimal: DFS search algorithm is non-optimal as it may generate a large number of steps or high cost to reach goal node.

3] DLS [Depth limit search]

i) Time complexity of DLS is $o(b^l)$

ii) space complexity of DLS is $O(b \times l)$

iii) Completeness: DLS algorithm is complete if the solution is above the depth limit.

iv) optimal: DLS can be viewed as a special case of DFS and it is also not optimal if $l > d$.

JAGISKAS

8

6000419005 7

4] DFID [Iterative Deepening Depth First search]

i) Time Complexity : Lets assume 'b' is the branching factor and depth is 'd' then worst case time complexity is $O(b^d)$

ii) Space complexity is $O(b \times d)$

iii) Completeness : This algorithm is complete if the branching factor is finite

iv) optimal : DFID algorithm is optimal if path cost is a non-decreasing function of the depth of the node

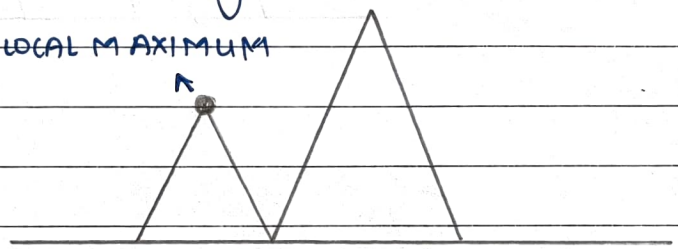
JAG

G0001190057

Q5

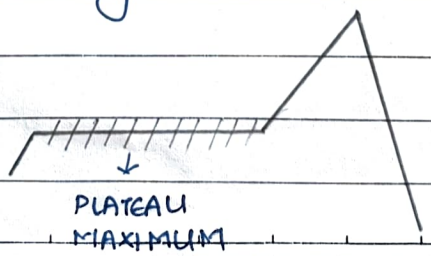
Ans Problems associated with Hill climbing Algorithms

1) LOCAL MAXIMUM: A local maximum is a peak state in the landscape which is better than each of its neighbouring states, but there is another state also present which is higher than the local maxima.



SOLUTION: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.

2) PLATEAU: A plateau is the flat area of the search space in which all the neighbour states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau



B0004190057

3) RIDGES: A ridge is a special form of the local maximum It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.