FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE - CS161

Programming Assignment 5 - Due Thursday, Feb 15, 11:59 p.m.

DO NOT CONSULT ANY OUTSIDE REFERENCES OR SOURCES FOR THIS ASSIGNMENT

Your task is to write a set of LISP functions to implement minimax search with alpha-beta pruning. A good solution to this problem is quite elegant.

The search tree will be an argument to your functions. There is a one-to-one mapping between trees and lists, and the search tree will be represented as a list. For example, a uniform binary tree of depth three with the terminal nodes labelled from left to right with the integers 1 through 8 is represented by the list: (((1 2) (3 4)) ((5 6) (7 8))). A uniform ternary tree of depth two with the terminal nodes labelled from left to right with the integers 1 through 9 is represented by the list: ((1 2 3) (4 5 6) (7 8 9)). In general your trees may not have uniform branching factor nor depth.

One solution to this assignment consists of just two short mutually recursive and entirely symmetric functions. In addition to any helper functions, your solution must define two top-level functions. One function, called MAXIMIN, assumes that the root of the tree is a MAX node and the other, called MINIMAX, assumes that the root of the tree is a MIN node. Each takes three arguments: the tree to be searched, an alpha value and a beta value. Alpha is the greatest lower bound among all the MAX ancestors of a node, while beta is the least upper bound among all the MIN ancestors of a node.

In order to verify that your functions are examining only the minimum number of terminal nodes, you need to include a counting mechanism. Instead of simply returning the minimax values of their subtrees, your functions should return a two-element list, where the first element is the minimax value, and the second element is the number of terminal or leaf nodes examined.

Be sure to thoroughly test your program, including the number of terminal nodes examined. Some sample test cases are given below and on the back. For each case, you should test your program with each player going first. Also, make sure you correctly handle the case where two nodes have the same value.

I strongly suggest that you first write and debug simple minimax functions, ignoring alpha-beta pruning and node counting, then add node counting, and finally add alpha-beta pruning.

```
(defconstant tree22 '((1 2) (3 4)))

(defconstant tree23 '(((1 2) (3 4)) ((5 6) (7 8))))

(defconstant tree24
'((((1 2) (3 4)) ((5 6) (7 8))) (((9 10) (11 12)) ((13 14) (15 16)))))

(defconstant tree25
'(((((1 2) (3 4)) ((5 6) (7 8))) (((9 10) (11 12)) ((13 14) (15 16))))
  ((((17 18) (19 20)) ((21 22) (23 24)))(((25 26) (27 28)) ((29 30) (31 32))))))

(defconstant tree26
'(((((((1 2)   (3 4))   ((5 6)   (7 8)))  (((9 10)  (11 12)) ((13 14) (15 16))))
   ((((17 18) (19 20)) ((21 22) (23 24)))(((25 26) (27 28)) ((29 30) (31 32)))))
  (((((33 34) (35 36)) ((37 38) (39 40)))(((41 42) (43 44)) ((45 46) (47 48))))
   ((((49 50) (51 52))((53 54) (55 56)))(((57 58) (59 60))((61 62) (63 64)))))))

(defconstant tree32 '((1 2 3) (4 5 6) (7 8 9)))

(defconstant tree33 '(((1 2 3)    (4 5 6)    (7 8 9))
                      ((10 11 12) (13 14 15) (16 17 18))
                      ((19 20 21) (22 23 24) (25 26 27))))

(defconstant tree34 '((((1 2 3)    (4 5 6)    (7 8 9))
                       ((10 11 12) (13 14 15) (16 17 18))
                       ((19 20 21) (22 23 24) (25 26 27)))
                      (((28 29 30) (31 32 33) (34 35 36))
```

```
                        ((37 38 39) (40 41 42) (43 44 45))
                        ((46 47 48) (49 50 51) (52 53 54)))
                       (((55 56 57) (58 59 60) (61 62 63))
                        ((64 65 66) (67 68 69) (70 71 72))
                        ((73 74 75) (76 77 78) (79 80 81)))))

(defconstant tree42 '((1 2 3 4) (5 6 7 8) (9 10 11 12) (13 14 15 16)))

(defconstant tree43
'(((1 2 3 4)     (5 6 7 8)     (9 10 11 12) (13 14 15 16))
  ((17 18 19 20) (21 22 23 24) (25 26 27 28) (29 30 31 32))
  ((33 34 35 36) (37 38 39 40) (41 42 43 44) (45 46 47 48))
  ((49 50 51 52) (53 54 55 56) (57 58 59 60) (61 62 63 64))))

(defconstant tree52
'((1 2 3 4 5) (6 7 8 9 10) (11 12 13 14 15) (16 17 18 19 20) (21 22 23 24 25)))

(defconstant tree53
'(((1 2 3 4 5) (6 7 8 9 10) (11 12 13 14 15)
   (16 17 18 19 20) (21 22 23 24 25))
  ((26 27 28 29 30) (31 32 33 34 35) (36 37 38 39 40)
   (41 42 43 44 45) (46 47 48 49 50))
  ((51 52 53 54 55) (56 57 58 59 60) (61 62 63 64 65)
   (66 67 68 69 70) (71 72 73 74 75))
  ((76 77 78 79 80) (81 82 83 84 85) (86 87 88 89 90)
   (91 92 93 94 95) (96 97 98 99 100))
  ((101 102 103 104 105) (106 107 108 109 110) (111 112 113 114 115)
   (116 117 118 119 120) (121 122 123 124 125))))

(defconstant tree62
'((1 2 3 4 5 6)       (7 8 9 10 11 12)    (13 14 15 16 17 18)
  (19 20 21 22 23 24) (25 26 27 28 29 30) (31 32 33 34 35 36)))
```