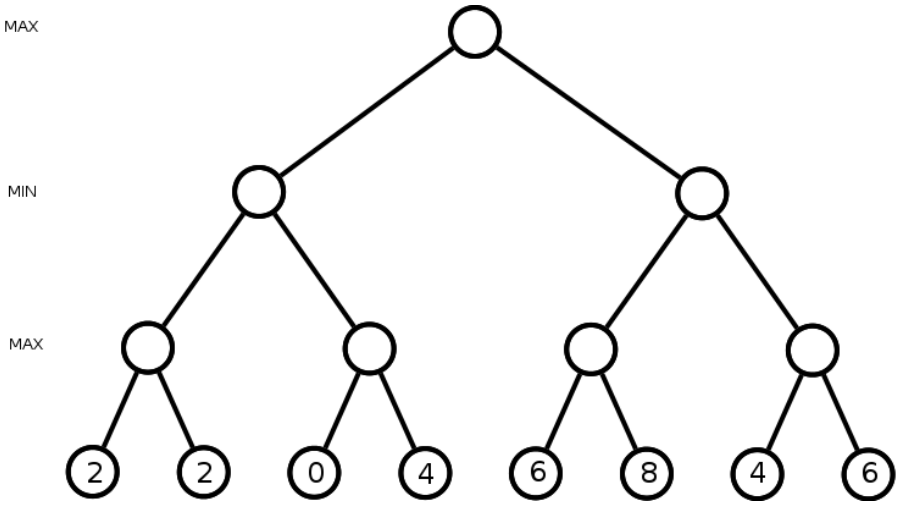
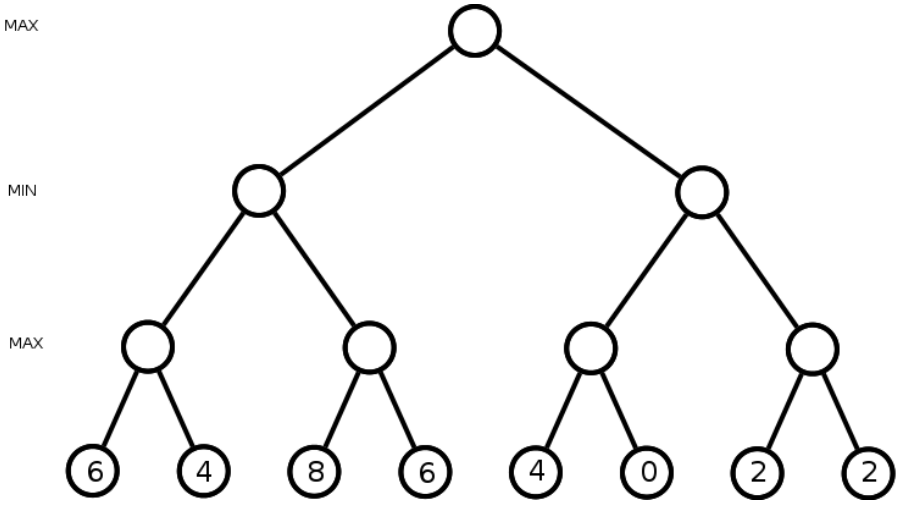
COMP 6721 Applied Artificial Intelligence (Fall 2023)

Lab Exercise #3: MiniMax and Alpha-Beta Pruning

Question 1 (a) Consider the game tree shown below. Explore the tree using Alpha-Beta. Indicate all parts of the tree that are pruned, and indicate the winning path or paths.

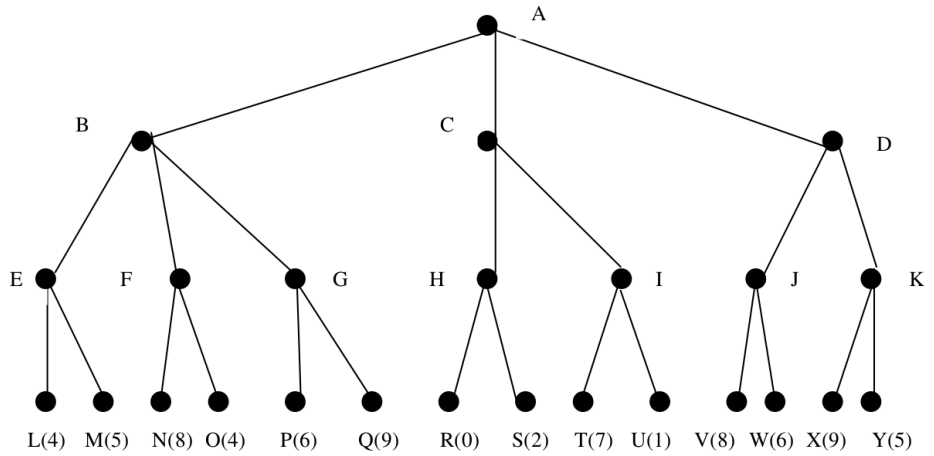


(b) Now do the same for the tree below, which is a mirror image of the tree shown above.



(c) Compare the amount of pruning in the above two trees. What do you notice about how the order of evaluation nodes affects the amount of Alpha-Beta pruning?

Question 2 Consider the game tree below. Each node is labelled with a letter, and the evaluation function for each leaf is indicated in parentheses. Assume that the MAX player goes first.

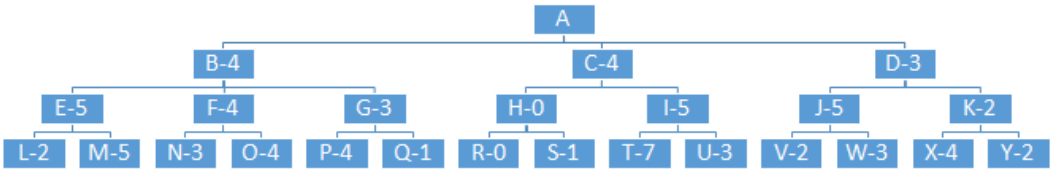


(a) Compute the minimax game value of nodes A, B, C, and D using the minimax algorithm. Show all values that are brought up to the internal nodes. What should MAX do?

(b) Cross out the branches of all the nodes that are *not* visited by alpha-beta pruning. Show all your work.

(c) Draw a new game tree by re-ordering the children of each internal nodes (B to M), such that the new game tree is equivalent to the tree above, but alpha-beta pruning will prune as many nodes as possible.

Question 3 Now consider the following game tree.



The value of the evaluation function at each node is shown next to its name. For example, B-4 indicates that node B has an evaluation function of 4. All evaluations are from the point of view of the first player.

(a) Assume that the first player is the maximizing player MAX and she looks at all levels (ie, to the level labeled L, M, N, O, ...). List in order the states that will **NOT** be examined when using alpha-beta pruning.

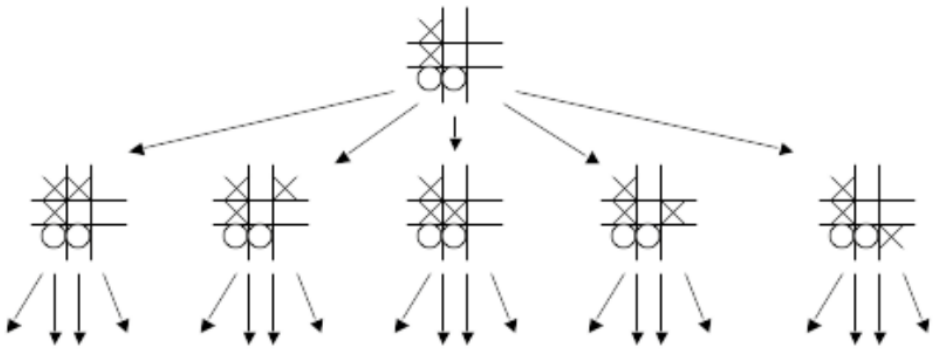
(b) What move should MAX choose?

(c) Suppose that instead of looking down all levels, MAX can only afford to look at level 2 (ie, the level with E, F, G, H, ... instead of the level with L, M, N, O...). In theory, could that change MAX's move? Explain.

Question 4 Consider state space search for the game of Tic-Tac-Toe. You are the X player, looking at the board shown below, with five possible moves. You want to look ahead to find your best move and decide to use the following evaluation function for rating board configurations:

value $V = 0$
for all rows, columns, diagonals R **do**:
    if R contains three Xs **then**:
        $V = V + 1000$
    else if R contains three 0s **then**:
        $V = V - 1000$
    else if R contains two Xs **then**:
        $V = V + 100$
    else if R contains two Os **then**:
        $V = V - 100$
    else if R contains one X **then**:
        $V = V + 10$
    else if R contains one O **then**:
        $V = V - 10$
    end if
end for
return V

(a) Draw the four possible configurations for the leftmost and the rightmost board configurations below. Use the evaluation function above to rate these 8 board configurations and choose X's best move.



Question 5 Let's create a minimal Python implementation for the alpha-beta algorithm to better understand how it works. We'll use a simple tree representation using integers:

```
# ASCII Diagram of the game tree
#
#           MAX
#         /  |  \
#        /   |   \
#       /    |    \
#      /     |     \
#    MIN    MIN    MIN
#    / \    / \    / \
#   5   8  4   2  3   7
```

```
tree = [[5, 8], [4, 2], [3, 7]]
```

We want to be able to find the next best move on this tree by calling:

```
best_move = alpha_beta_search(tree, 2, float('-inf'), float('inf'), True)
print(f"The optimal value (best move) for the root node is: {best_move}")
```

Here's part of the implementation. Note that it adds some debug output so that you can follow its execution on a given tree. Feel free to add more `print` statements:

```
def alpha_beta_search(node, depth, alpha, beta, maximizing_player):
    if depth == 0 or not isinstance(node, list):
        print(f"Evaluating node with value {node}. Alpha: {alpha}, Beta: {
            ↪ beta}")
        return node

    if maximizing_player:
        max_eval = float('-inf')
        for child in node:
            eval = alpha_beta_search(child, depth-1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                print(f"Pruning branch at node with max_eval {max_eval} due to
                    ↪ beta <= alpha. Alpha: {alpha}, Beta: {beta}")
                break
        return max_eval
    else:
        # YOUR_CODE_HERE
```

Complete the code and run it on the provided example tree. Is the result what you would expect?