CSE 571 Fall 2022 HW3

Submitted By: Aman Peshin (1225476655)

Exercise 1.3

a. No, pruning is not possible in a max tree. The root node gets the max of all the values of the leaf nodes in the tree. It's not possible to prune because any of these leaves may have a maximum value that cannot be predetermined by any pruning methods.

b. No pruning is not possible in an expectimax tree. The problem statement defines that the leaves have a finite value and are unbounded and none of the probabilities from the chance node is zero. So there exists a node/leaf with a utility value and a probability that would stop any pruning methods from pruning it. Anyone unseen leaf may have a value that is higher or lower than an explored leaf node we can't estimate a bound on the value that we can see.

c. No, pruning opportunities are not present in a max tree if the leaves have non-negative values. It would still use the same logic as in all positive values of leaf nodes and will check all nodes in order to determine the root node value. Any pruning algorithms will not work on this configuration of the max tree.

d. No pruning opportunities in an expectimax tree with non-negative values. Let's say that if we get an exp value from the first chance node as 0.8, then this value for the max node will only be a lower bound. It is still going to check the other chance node's value in order to make the best decision.

e. Yes, pruning opportunities are present in a max tree if the leaves have values between 0 and 1. Let's take an example of any node, if we can see that the first leaf or utility is equal to 1 then it makes no sense to check other children since the max value that can be obtained has already been obtained.

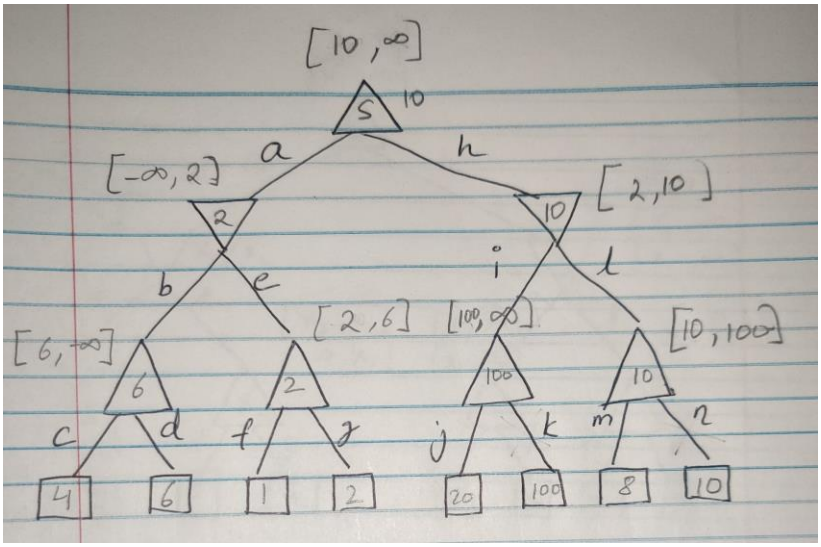
f. Yes, pruning is possible in an expectimax tree where the leaf values are in the range of 0 to 1. For example, a two branching factor game tree, if the left chance node gives us an exp value of 0.8 and the first leaf node of the second chance node has a terminal value of 0 with a probability of anything greater than 0.2 then we can safely prune the other leaf node of chance node 2

g. The highest probability first will give us the best shot at pruning if it is available. Hypothetically if the probability of the first action is very high and the terminal value is near the upper bound of the evaluation function value of the node. Then it is highly probable that there is a pruning opportunity. With higher probability we can set an

upper bound from which the other leaf nodes can be decided to be pruned or not, here it's the likelihood and not absolute pruning.

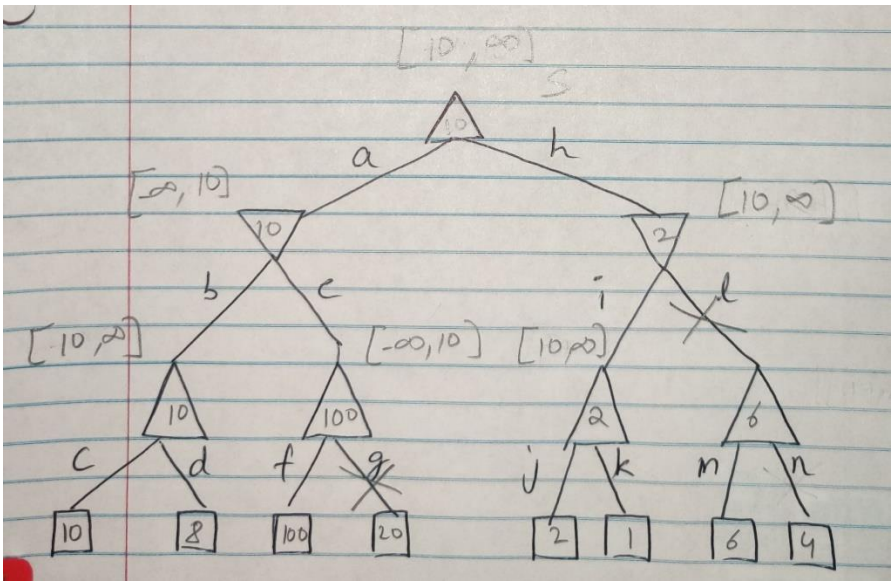
Exercise 1.1

a. The attached diagram shows the tree after alpha-beta pruning. The alpha and beta values at every node are as follows



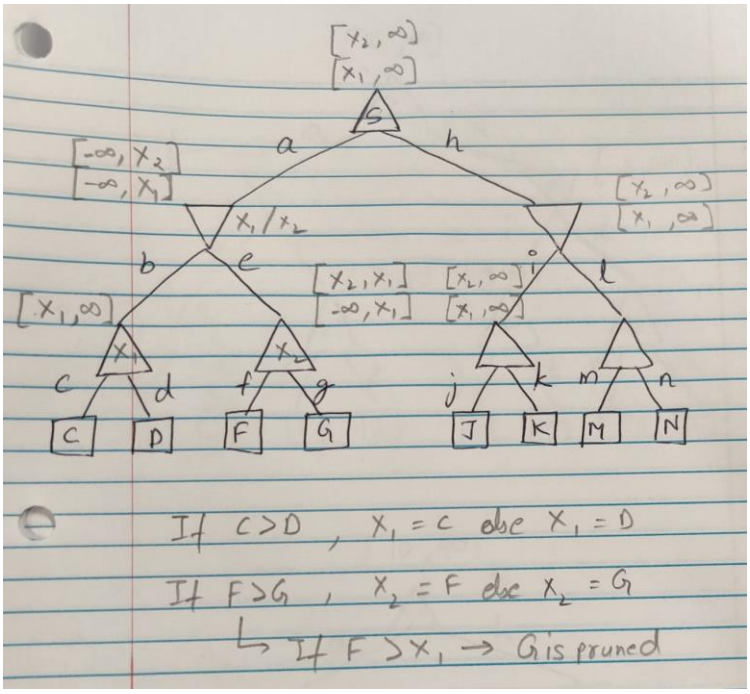
S -> [10, ∞]
A-> [-∞, 2]
H-> [2, 10]
B-> [6, ∞]
E-> [2, 6]
I-> [100, ∞]
L-> [10, 100]

b. The attached diagram shows the tree after alpha-beta pruning. The alpha and beta values at every node are as follows



S -> [10, ∞]
A- >[-∞, 10]
H-> [10, ∞]
B-> [10, ∞]
E-> [-∞, 10]
I-> [10, ∞]
L-> pruned

c. The attached diagram shows the tree after alpha-beta pruning. The alpha and beta values at every node are as follows



S -> [X2, ∞] or [X1, ∞]
A-> [-∞, X2] or [-∞, X1]
H-> [X2, ∞] or [X1, ∞]
B-> [X1, ∞]
E-> [X2, X1] or [-∞, X1]
I-> [X2, ∞] or [X1, ∞]
L-> NA

At node B:
if C > D, X1 = C else X1 = D
At node E:
If F>G, X2 = F else X2 = G

Also,
If F > X1 -> G is pruned (but this won't matter for pruning nodes J and K

Node J can't be pruned at any point because we need to visit it at least once to decide on pruning opportunities for node H's children. For the case of K, it can't be pruned since the value of J can't be greater than the best possible value for the min nodes till the root, i.e. B value. Hence since J can't be greater than ∞. We can't prune K
J and K can't be pruned. We can map one to one for any value and it will remain same, also with perfect reordering, it will remain the same.

Exercise 1.4

a. True. If both the agents play rationally in a fully observable, turn-taking, the zero-sum game then we already know how the game tree would be provided it finishes in finite moves.
   At every node/play, we already know the possible values the MIN agent would take to counter the MAX agent's moves. Strategy comes into play in partially observable games where a new card played or any new info, changes MAX's strategy to MIN's gameplay.

b. False, It helps the agent to decide its next move if the environment is partially observable. Any new move like picking up a card or replacing an existing one in the game of cards or UNO will require the MAX agent to alter its strategy. In fact, any new info that changes the game space would require both the agents to alter their strategy based on observed data.

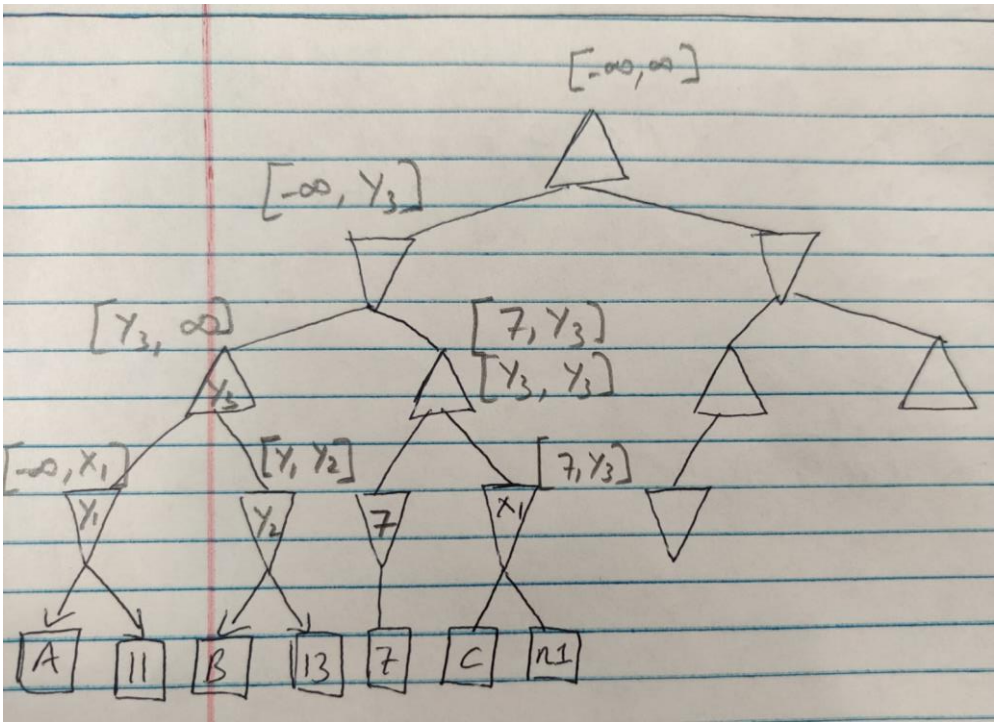
c. A rational minimax Pacman would always win against a rational ghost or a random ghost. But an Expectimax Pacman would lose against a rational ghost and win against a random ghost. Here the expectation of all games would seem that a minimax Pacman is better fared against any type of ghost since we don't take into consideration the probabilities with which the ghost can move to a new state. Hence the only thing our Pacman needs to worry about is increasing its utility. But the same can't be told about an expectimax Pacman, since it relies on the probability of the action the ghost may take, this introduces uncertainty in whether the Pacman can always win.

Exercise 1.2

a. To determine the values of A, B, C such that X1 and its leaves nodes are not pruned. Based on the diagram given below we get the following equations

Y1 = A if A<11 else 11
Y2 = B if B<13 else 13
Y3 = Y4 if Y1>Y2 else Y2
Y4 = Y3 if Y3<7 else 7

Now,
If Y4<Y3 node Y will not be pruned
-> 7<Y3 -> 7<Y1 or 7<Y2
-> 7<A or 7<11 and 7<B or 7<13
We get A, B >7
Also,
C< Y3
C<Y1, Y2
➔ C <11 or C< 13, A, B
Hence the values of **A > 7 OR B > 7 AND C > 7** for X1 node is not to be pruned



b. For the node n1 to be pruned, we can consider if X1 is pruned then n1 is pruned as well continuing from the previous question

   we get,
   7 > Y3 and
   Y3 = Y1 if Y1 >Y2 else Y2 and
   Y1 = A if A < 11 else 11
   Y2 =B if B < 13 else 13

   -> 7> Y1 -> 7 > A or 7 > 11 (not possible)
   -> 7 > Y2 -> 7 > B or 7 > 13 (not possible)

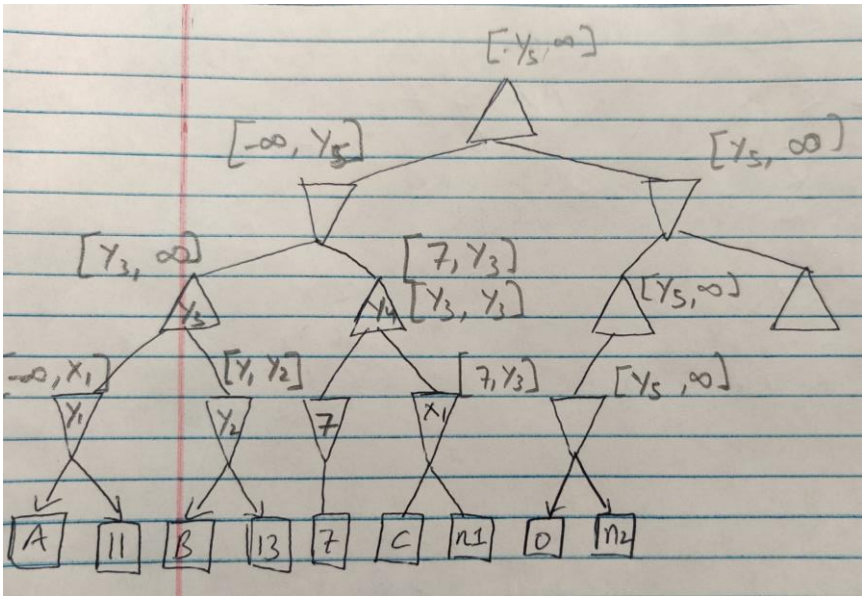
   The values are **A and B < = 7** when only X1 is pruned (In turn n1 is also pruned)

   If only node n1 is pruned then **A OR B > 7** (has to come till node X1) and C < Y3
   i.e. C < A, B <= 7 -> **C <= 7**

c. The value of node X1 = C if C> n1 else n1
   Y4 = Y1 if Y1>7 else 7
   Y5 = Y3 if Y3>Y4 else Y4

   now for n2 to get pruned
   0 < Y5
   -> 0< Y3 or 0 <Y4
   -> 0< y1 or 0<Y2 or 0 < X1
   -> 0 < A, 11 or 0 < B, 13 or 0 < C

   This means **that A Or B >= 0 (both can't be negative)**
   If thr value of alpha is negative then node n2 will not be pruned, but if it is positive n1 will be pruned.

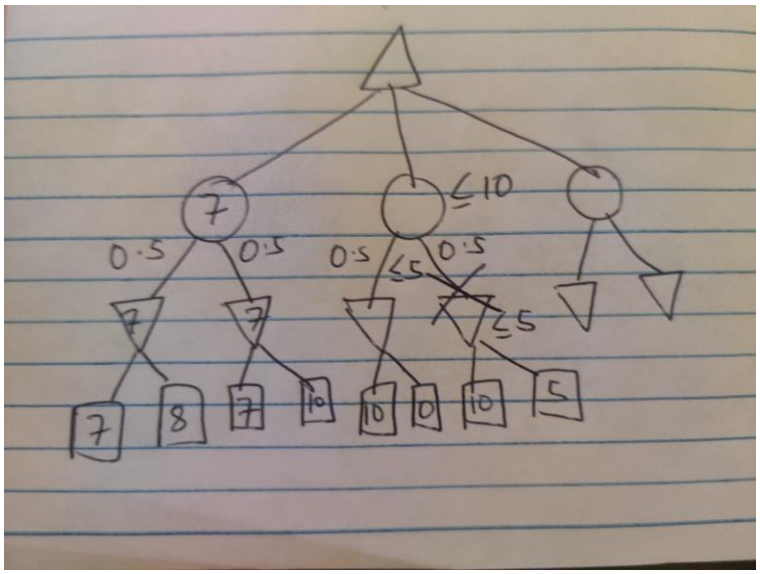


Exercise 1.5

a. Yes an expectiminimax tree can lead to pruning opportunities, an example is given below.

We would run the same alpha-beta pruning algorithm but the branch that would be pruned would be either the max or min node, we would compare the alpha-beta value while updating it from the leaf nodes to the root node.

let us take one layer of max, chance and min nodes. Here we see that from the min nodes of the left most branch we get a value of 7 and from the branch on the right of that we get again 7 after multiplying it with the chance probability we get the exp value as 7. Now let's go over to the 2nd chance node. From the first min node, we get a value of 10, after multiplying it with 0.5 probability we know that the probabilistic value as greater than >= 5, but the actual value that the min node return is 0 this in turn. By looking at the second min node we can say safely that with even the max value 10 we are smaller than the other chance node and hence the second chance node does not add anything and can be pruned.



c. The algorithm for alpha beta pruning in similar to the one in minimax tree.

def value (state):
if state == max node
return the highest expectiminimax value from all of its successors

else if state == min node
return the lowest expectiminimax value of successor nodes

else if state == chance node
Then return aveage of expectiminimax value of successor node.
Def exp-state:
initialize value = 0
probs = p(succseesor)

V = + p*value(successor)
return v

b. In the tree given in part 1 we can prune MIN MAX nodes which are consecutive and similar to alpha beta pruning in MINIMAX tree