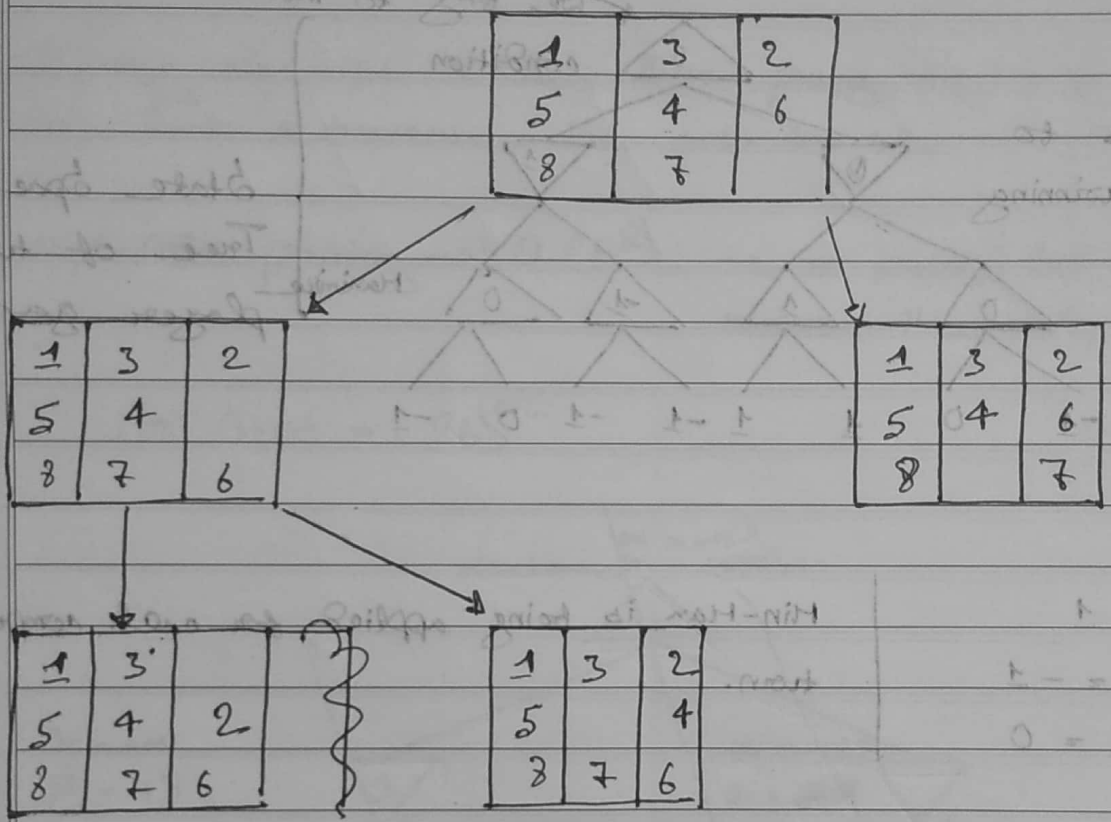
Subject : Games

Date :

1	2	3
4	5	6
7	8	

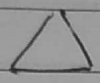
Goal State.



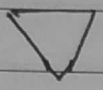
This is a state space Tree. Each of the node represents a state. The goal node will also be the leaf node.

Subject :

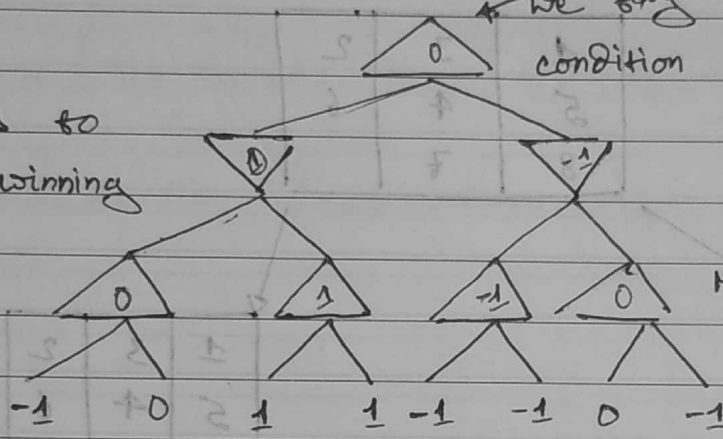
Date :



← Maximise



← Minimise



Win = 1
Lose = -1
Draw = 0

Min-Max is being applied for each computer turn.

Chess has a total of 10^{120} possible states. So, a state space tree for a game of chess would be gigantic to the point that we can not comprehend. To reach the goal node every time, traversing depth would require huge amount of computation time.

Pruning is achieved using alpha-beta pruning.

Subject :

Date :

α starts from $-\infty$

β " " ∞

~~max~~

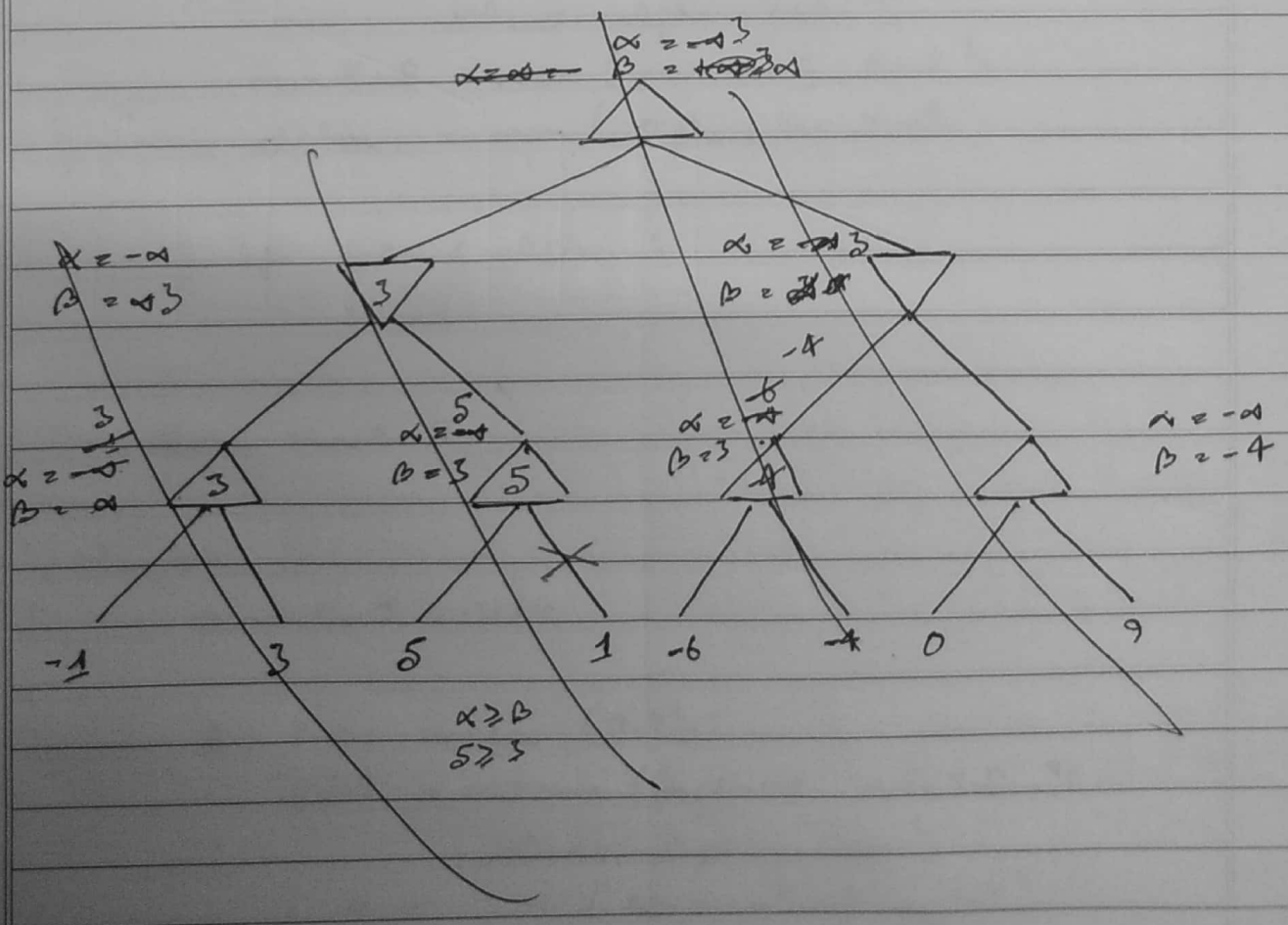
In max depth we always update α with max value

" min " " " " β " min "

In any case if $\alpha \geq \beta$, we prune the sub branches i.e we dont traverse the sub branch.

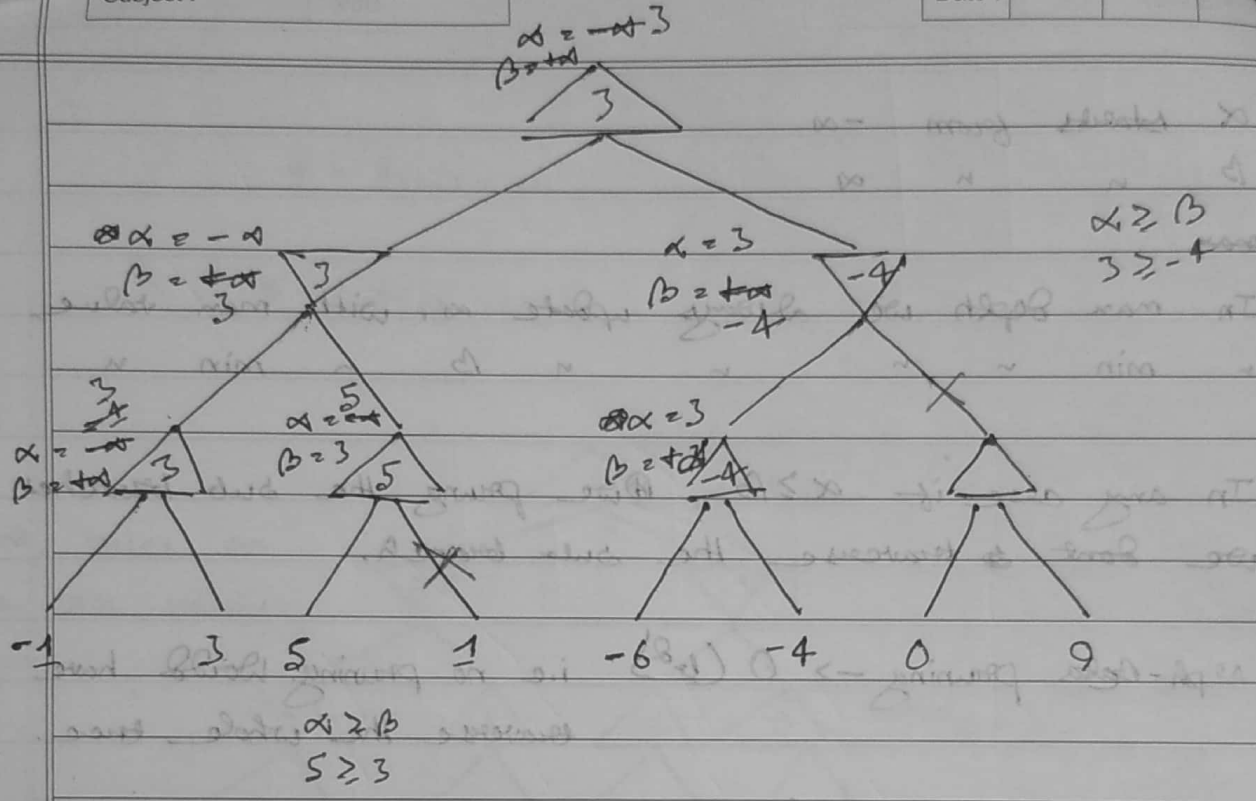
Alpha-Beta pruning -> $O(b^{d/2})$ i.e no pruning would have to traverse the whole tree.

For Best = $(2b)^{d/2}$



Subject :

Date :



Subject: MINIMAX algo using (Alpha-Beta Pruning) Date:

MIN, MAX = -1000, 1000

```
def minimax (depth, nodeIndex, maximizingPlayer, values,
             alpha, beta):

    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        maxEval = MIN

        for i in range(0, 2):
            Eval = minimax(depth+1, nodeIndex*2+i, False,
                           values, alpha, beta)
            maxEval = max(maxEval, Eval)
            alpha = max(alpha, maxEval)

            if beta <= alpha:
                break

        return maxEval

    else:
        maxEval = MAX

        for i in range(0,2):
            Eval = minimax(depth+1, nodeIndex*2+i, True,
                           values, alpha, beta)
            maxEval = min(maxEval, eval)
            beta = min(beta, maxEval)
```

Subject :

Date :

if beta <= alpha :

break

return MaxEval

values = [-1, 3, 5, 1, -6, -4, 0, 9]

minimax (0, 0, True, values, MIN, MAX)