Minimax & Alpha-Beta Pruning Algorithms in Game Theory

Asst.Prof.Dr.Emre Özbilge

Department of Artificial Intelligence Engineering
Faculty of Engineering
Cyprus International University

December 11, 2024

CYPRUS INTERNATIONAL UNIVERSITY

Game Theory I

- Two-player games are games in which two players compete against each other. These are also known as zero-sum games. The goal then in playing a two-player game is choosing a move that maximizes the score of the player and/or minimizes the score of the competing player.
- Consider the two-player game Tic-Tac-Toe. Players alternate moves, and as each move is made, the possible moves are constrained. In this simple game, a move can be selected based on the move leading to a win by traversing all moves that are constrained by this move. Also, by traversing the tree for a given move, we can choose the move that leads to the win in the shallowest depth (minimal number of moves).
- Tic-Tac-Toe is an interesting case because the maximum number of moves is tiny when compared to more complex games such as Checkers or Chess.

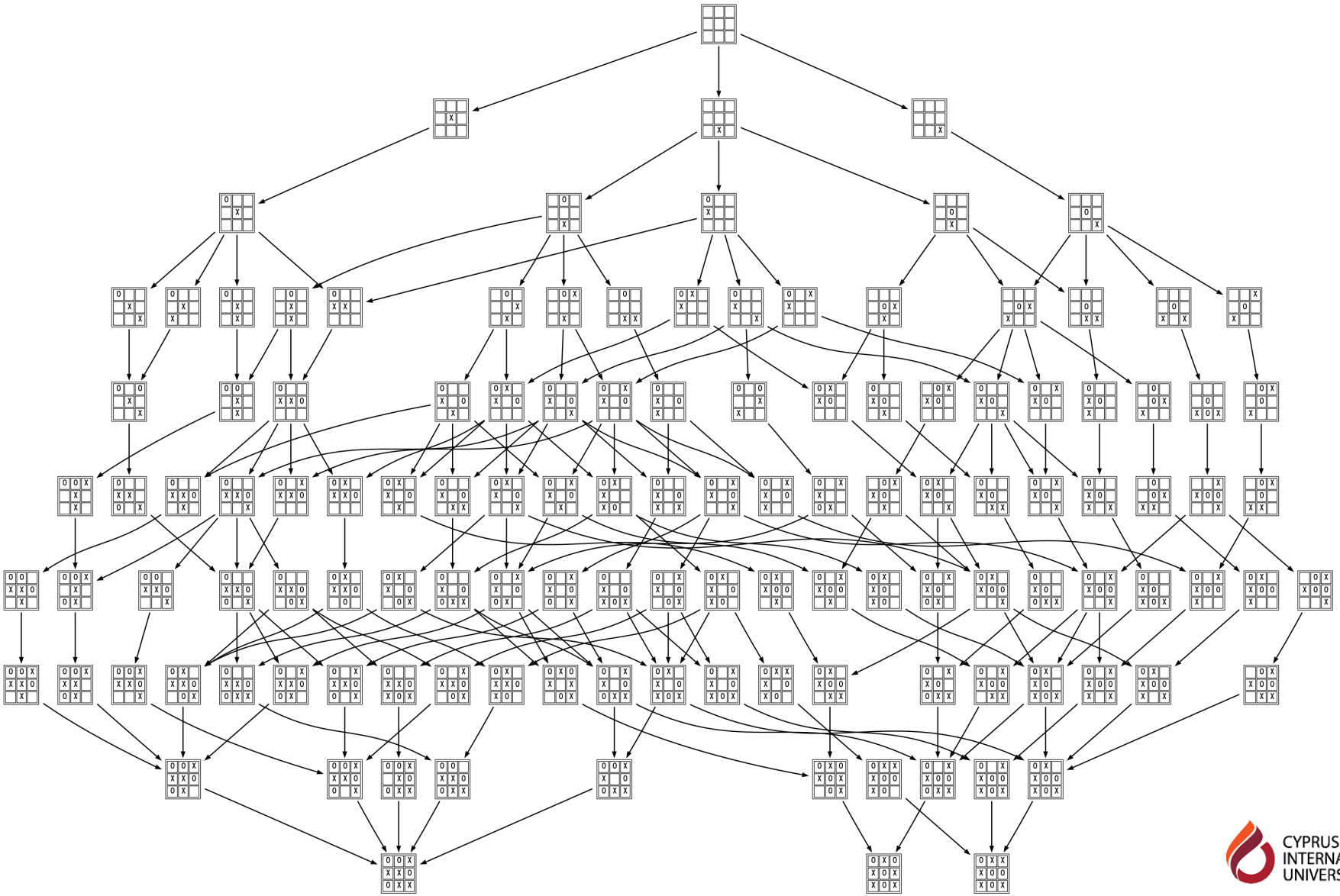
CYPRUS INTERNATIONAL UNIVERSITY

Game Theory II

- At each node in the tree (a possible move) a value defining the goodness of the move toward the player winning the game can be provided. So at a given node, the child nodes (possible moves from this state in the game) each have an attribute defining the relative goodness of the move. It's an easy task then to choose the best move given the current state. But given the alternating nature of two-player games, the next player makes a move that benefits himself.

CYPRUS INTERNATIONAL UNIVERSITY

Game Theory III



CYPRUS INTERNATIONAL UNIVERSITY

Minimax Algorithm I

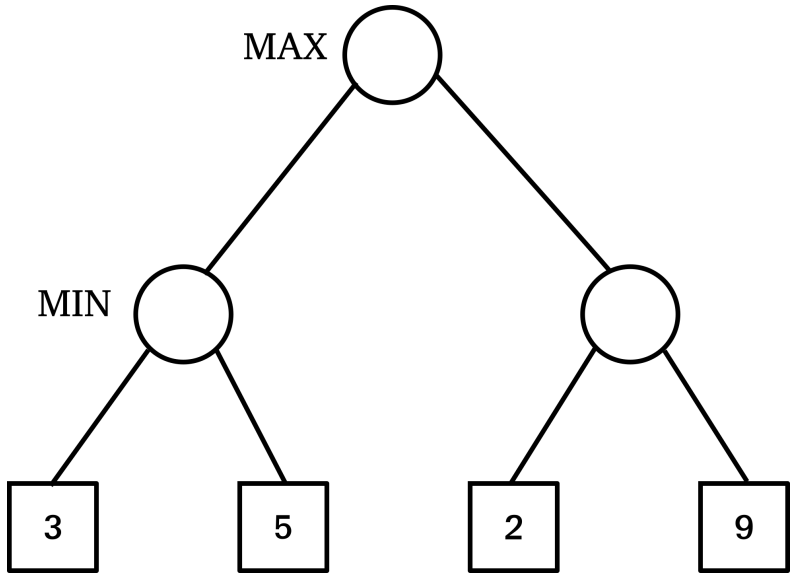
- Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.
- In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.
- Every board state has a value associated with it. The values of the board are calculated by some heuristics which are unique for every type of game.

Simple example:

- Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level.
- Which move you would make as a maximizing player considering that your opponent also plays optimally?

CYPRUS INTERNATIONAL UNIVERSITY

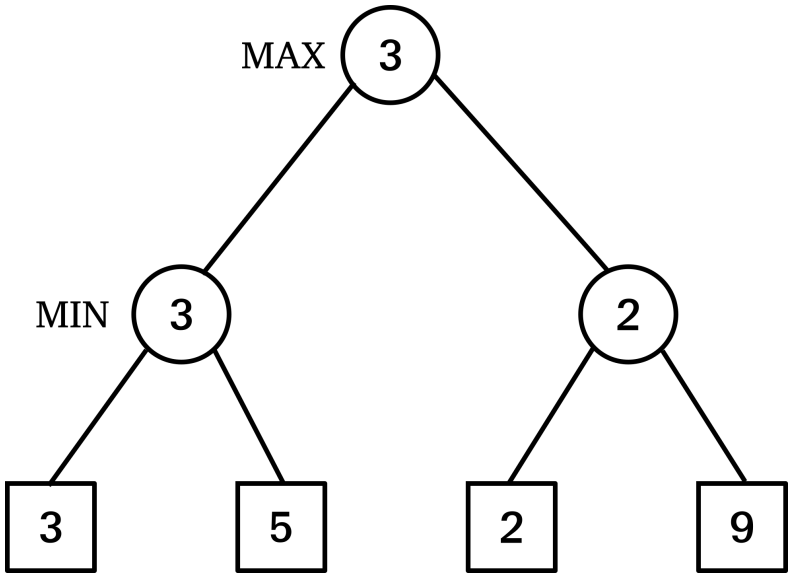
Minimax Algorithm II



- Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.
  - Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
  - Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

CYPRUS INTERNATIONAL UNIVERSITY

Minimax Algorithm III



- Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

Evaluation Function of Minimax Algorithm I

- Each leaf node had a value associated with it. We need to implement a function that calculates the value of the board depending on the placement of pieces on the board.
- This function is often known as Evaluation Function. It is sometimes also called Heuristic Function. The evaluation function is unique for every type of game.
- The basic idea behind the evaluation function is to give a high value for a board if maximizer's turn or a low value for the board if minimizer's turn.
- For this scenario let us consider X as the maximizer and O as the minimizer:
  1. If X wins on the board we give it a positive value of $+10$.

X	O	X
O	O	X
		X

+10

CYPRUS INTERNATIONAL UNIVERSITY

Evaluation Function of Minimax Algorithm II

2. If O wins on the board we give it a negative value of -10.

X	O	X
O	O	X
X	O	

-10

3. If no one has won or the game results in a draw then we give a value of $+0$.

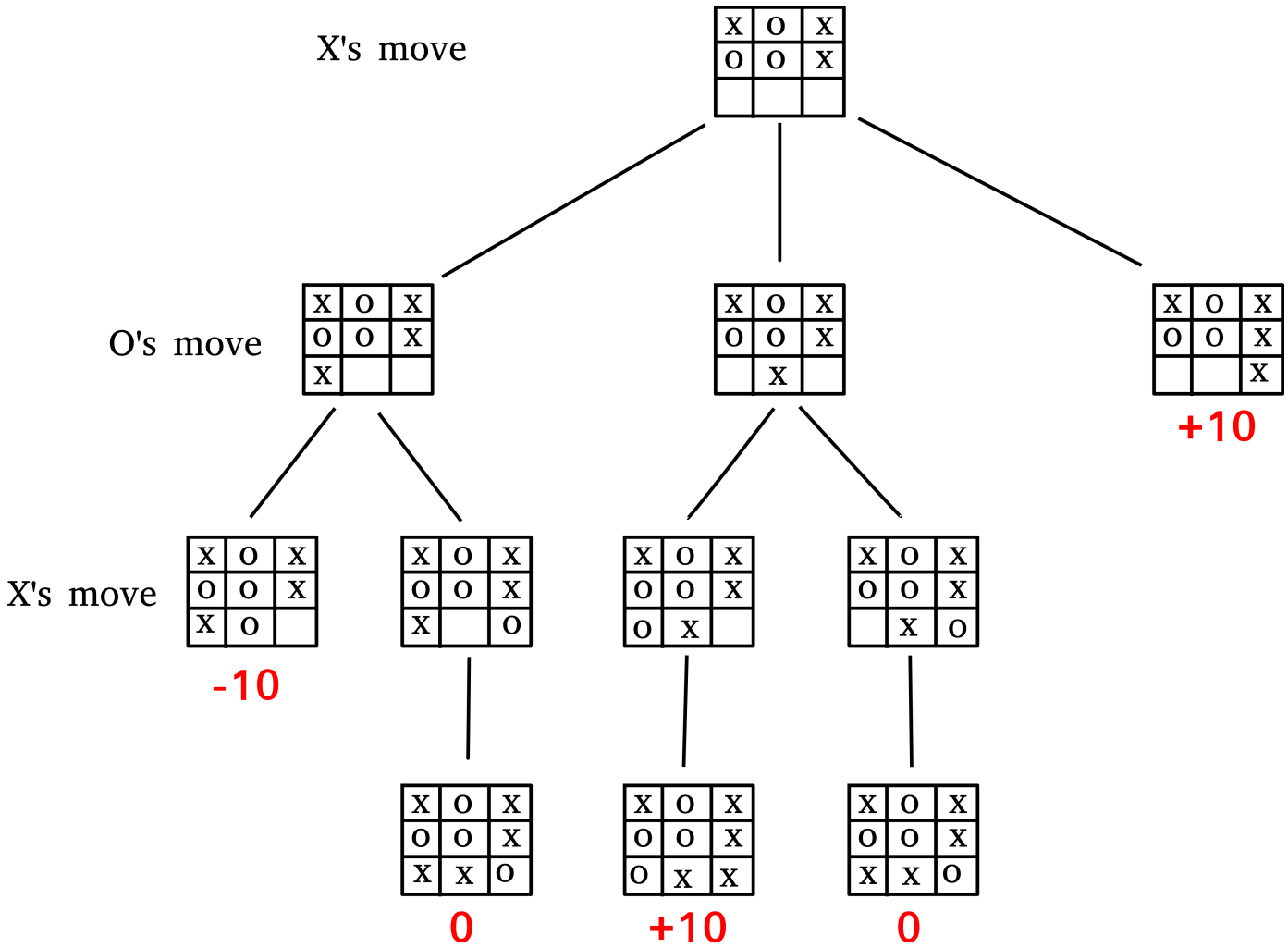
X	O	X
O	O	X
X	X	O

0

- We could have chosen any positive/negative value other than 10.

CYPRUS INTERNATIONAL UNIVERSITY

Finding Optimal Move I



- This image depicts all the possible paths that the game can take from the root board state. It is often called the Game Tree.

Finding Optimal Move II

- The 3 possible scenarios in the above example are:
  - Left Move: If X plays [2,0]. Then O will play [2,1] and win the game. The value of this move is -10
  - Middle Move: If X plays [2,1]. Then O will play [2,2] which draws the game. The value of this move is 0
  - Right Move: If X plays [2,2]. Then he will win the game. The value of this move is $+10$
- Remember, even though X has a possibility of winning if he plays the middle move, O will never let that happen and will choose to draw instead.
- Therefore the best choice for X, is to play [2,2], which will guarantee a victory for him.
- Even though the following AI plays perfectly, it might choose to make a move which will result in a slower victory or a faster loss.

CYPRUS INTERNATIONAL UNIVERSITY

Finding Optimal Move III

Considering depth information:

- Assume that there are 2 possible ways for X to win the game from a give board state.
  - Move A : X can win in 2 moves
  - Move B : X can win in 4 moves
- Our evaluation function will return a value of +10 for both moves A and B. Even though the move A is better because it ensures a faster victory, our AI may choose B sometimes. To overcome this problem we subtract the depth value from the evaluated score. This means that in case of a victory it will choose a the victory which takes least number of moves and in case of a loss it will try to prolong the game and play as many moves as possible.
  - Move A will have a value of +10 - 2 = 8
  - Move B will have a value of +10 - 4 = 6

CYPRUS INTERNATIONAL UNIVERSITY

Finding Optimal Move IV

- Now since move A has a higher score compared to move B our AI will choose move A over move B. The same thing must be applied to the minimizer. Instead of subtracting the depth we add the depth value as the minimizer always tries to get, as negative a value as possible.
  - Maximizer has won: WIN_SCORE - depth
  - Minimizer has won: LOOSE_SCORE + depth

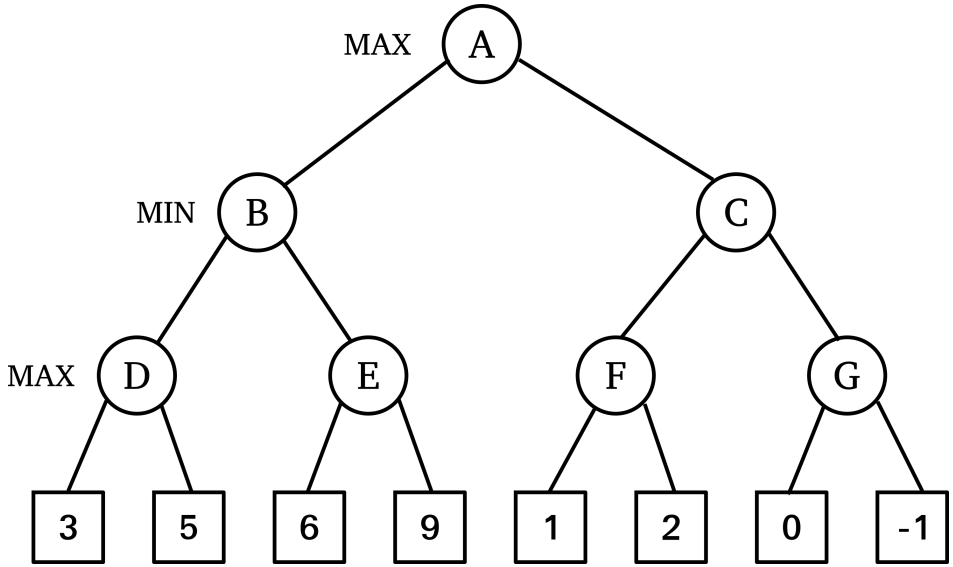
CYPRUS INTERNATIONAL UNIVERSITY

Alpha-Beta Pruning I

- Alpha-Beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor.
- This allows us to search much faster and even go into deeper levels in the game tree.
- It cuts off branches in the game tree which need not be searched because there already exists a better move available.
- It is called Alpha-Beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.
- **Alpha** is the best value that the maximizer currently can guarantee at that level or above.
- **Beta** is the best value that the minimizer currently can guarantee at that level or above.

CYPRUS INTERNATIONAL UNIVERSITY

Alpha-Beta Pruning II



- The initial call starts from A. The value of alpha here is $-\infty$ and the value of beta is $+\infty$. These values are passed down to subsequent nodes in the tree. At A the maximizer must choose max of B and C, so A calls B first.
- At B it the minimizer must choose min of D and E and hence calls D first.
- At D, it looks at its left child which is a leaf node. This node returns a value of 3. Now the value of alpha at D is max($-\infty$,3) which is 3.

CYPRUS INTERNATIONAL UNIVERSITY

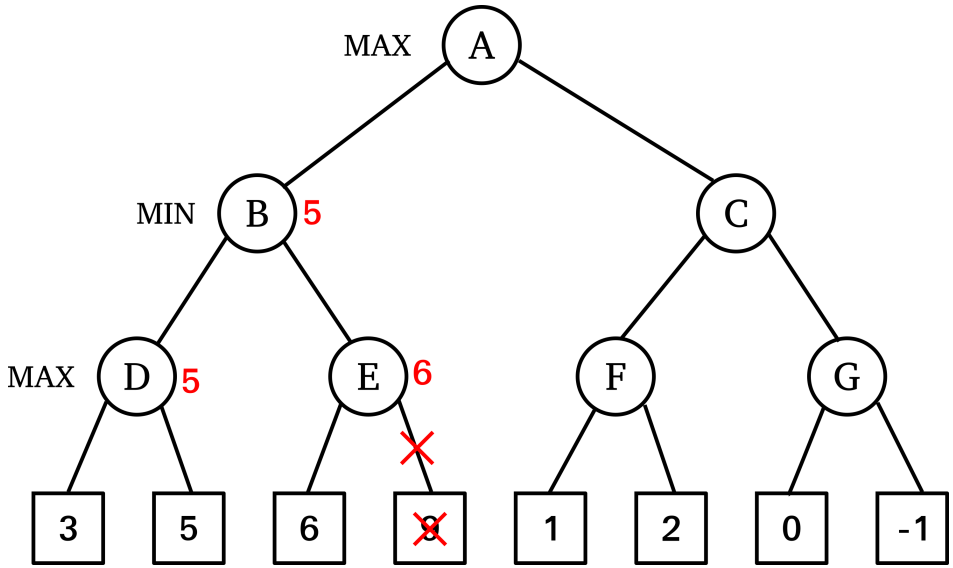
Alpha-Beta Pruning III

- To decide whether its worth looking at its right node or not, it checks the condition beta<=alpha. This is false since beta $= +\infty$ and alpha $= 3$. So it continues the search.
- D now looks at its right child which returns a value of 5. At D, alpha $=$ max(3,5) which is 5. Now the value of node D is 5.
- D returns a value of 5 to B. At B, beta $=$ min($+\infty$,5) which is 5. The minimizer is now guaranteed a value of 5 or lesser. B now calls E to see if he can get a lower value than 5.
- At E the values of alpha and beta is not $-\infty$ and $+\infty$ but instead $-\infty$ and 5 respectively, because the value of beta was changed at B and that is what B passed down to E.
- Now E looks at its left child which is 6. At E, alpha $=$ max($-\infty$,6) which is 6. Here the condition becomes true. beta is 5 and alpha is 6. So beta<=alpha is true. Hence it breaks and E returns 6 to B.

CYPRUS INTERNATIONAL UNIVERSITY

Alpha-Beta Pruning IV

- Note how it did not matter what the value of E's right child is. It could have been $+\infty$ or $-\infty$, it still wouldn't matter, We never even had to look at it because the minimizer was guaranteed a value of 5 or lesser. So as soon as the maximizer saw the 6 he knew the minimizer would never come this way because he can get a 5 on the left side of B. This way we didn't have to look at that 9 and hence saved computation time.
- E returns a value of 6 to B. At B, beta = min(5,6) which is 5.The value of node B is also 5.

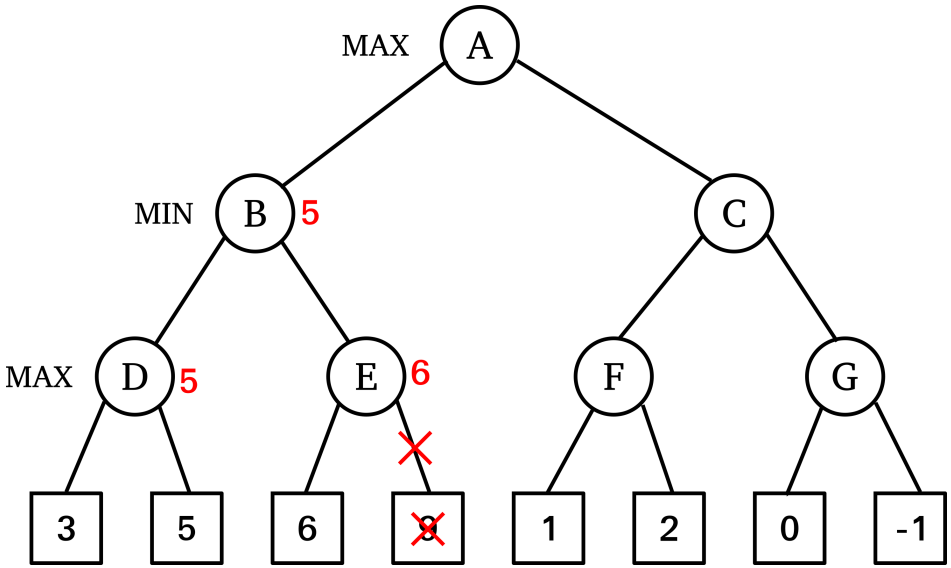


Alpha-Beta Pruning V

- B returns 5 to A. At A, alpha = max($-\infty$,5) which is 5. Now the maximizer is guaranteed a value of 5 or greater. A now calls C to see if it can get a higher value than 5.
- At C, alpha = 5 and beta = $+\infty$. C calls F.
- At F, alpha = 5 and beta = $+\infty$. F looks at its left child which is a 1. alpha = max(5,1) which is still 5.
- F looks at its right child which is a 2. Hence the best value of this node is 2. Alpha still remains 5.
- F returns a value of 2 to C. At C, beta = min($+\infty$,2). The condition beta <= alpha becomes true as beta = 2 and alpha = 5. So it breaks and it does not even have to compute the entire sub-tree of G.

CYPRUS INTERNATIONAL UNIVERSITY

Alpha-Beta Pruning VI

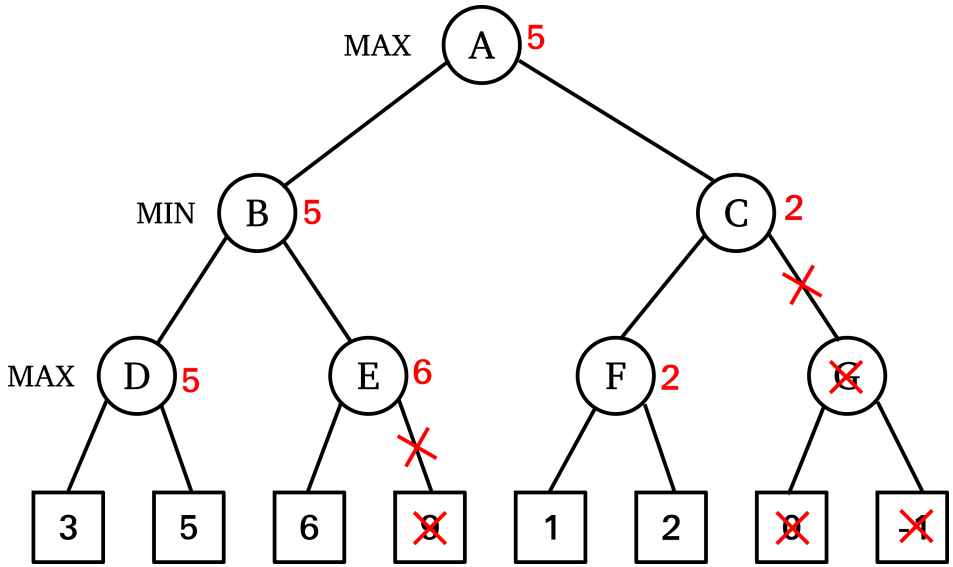


- The intuition behind this break off is that, at C the minimizer was guaranteed a value of 2 or lesser. But the maximizer was already guaranteed a value of 5 if he choose B. So why would the maximizer ever choose C and get a value less than 2? Again you can see that it did not matter what those last 2 values were. We also saved a lot of computation by skipping a whole sub tree.

CYPRUS INTERNATIONAL UNIVERSITY

Alpha-Beta Pruning VII

- C now returns a value of 2 to A. Therefore the best value at A is max(5,2) which is a 5.
- Hence the optimal value that the maximizer can get is 5.



What if the game tree is too deep? I

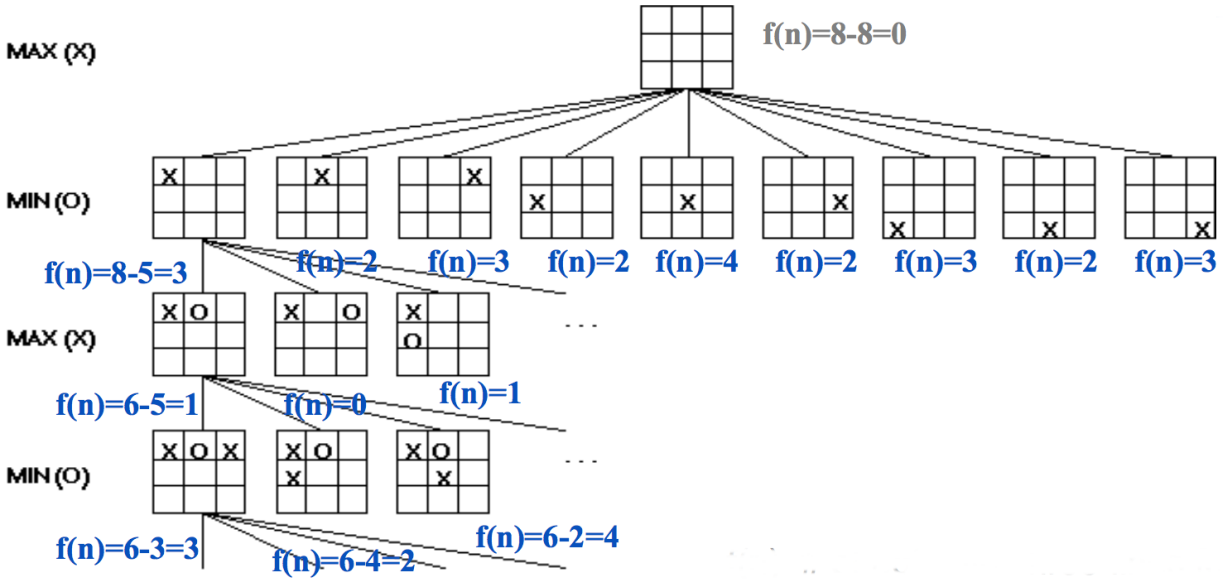
- For more complex games, we will not have the time/memory required to search the entire game tree. So we will need to cut off search at some chosen depth. In this case, we heuristically evaluate the game configurations in the bottom level, trying to guess how good they will prove. We then use minimax to propagate values up to the top of the tree and decide what move we will make.
- As play progresses, we will know exactly what moves have been taken near the top of the tree. Therefore, we will be able to expand nodes further and further down the game tree. When we see the final result of the game, we can use that information to refine our evaluations of the intermediate nodes, to improve performance in future games.

CYPRUS INTERNATIONAL UNIVERSITY

What if the game tree is too deep? II

- Here's a simple way to evaluate positions for tic-tac-toe. At the start of the game, there are 8 ways to place a line of three tokens. As play progresses, some of these are no longer available to one or both players. So we can evaluate the state by:

```
f(state) = [number of lines open to us] - [number of lines
open to opponent]
```



CYPRUS INTERNATIONAL UNIVERSITY

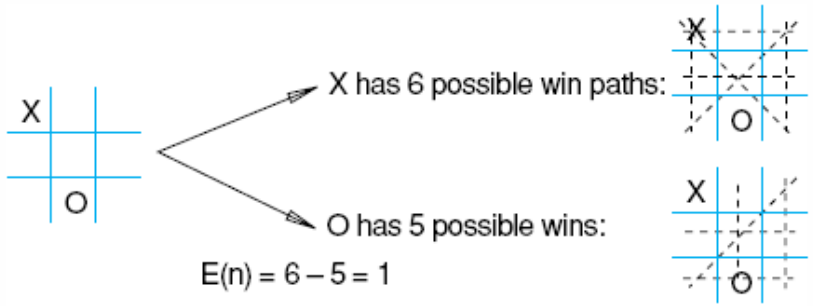
Static (Heuristic) Evaluation Functions I

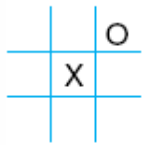
- An Evaluation Function:
  - Estimates how good the current board configuration is for a player.
  - Typically, one figures how good it is for the player, and how good it is for the opponent, and subtracts the opponents score from the player.
  - Othello: Number of white pieces - Number of black pieces
  - Chess: Value of all white pieces - Value of all black pieces

CYPRUS INTERNATIONAL UNIVERSITY

Static (Heuristic) Evaluation Functions II

- Heuristic measuring conflict applied to states of tic-tac-toe.

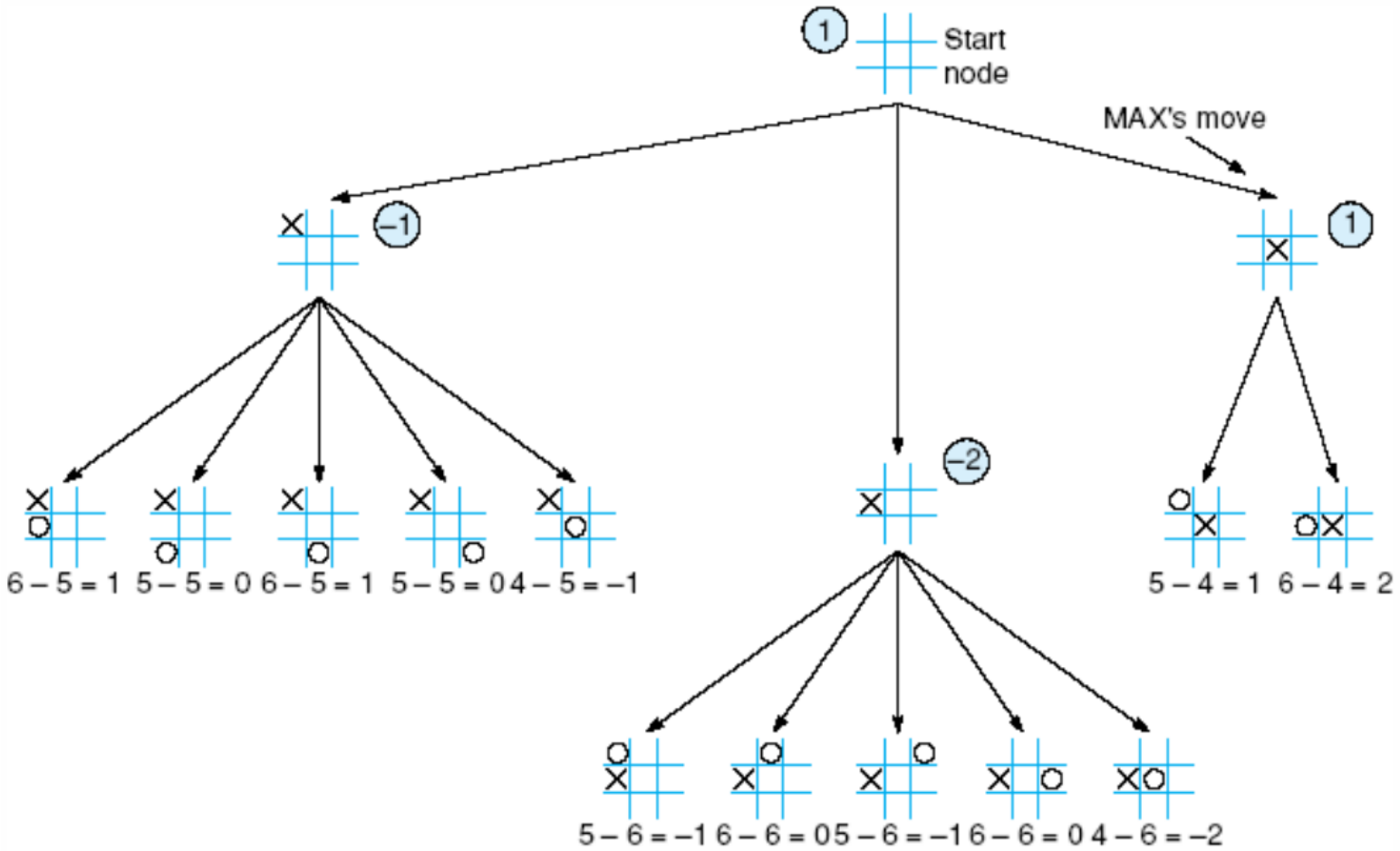




Heuristic is E(n) = M(n) – O(n)
where M(n) is the total of My possible winning lines
O(n) is total of Opponent's possible winning lines
E(n) is the total Evaluation for state n

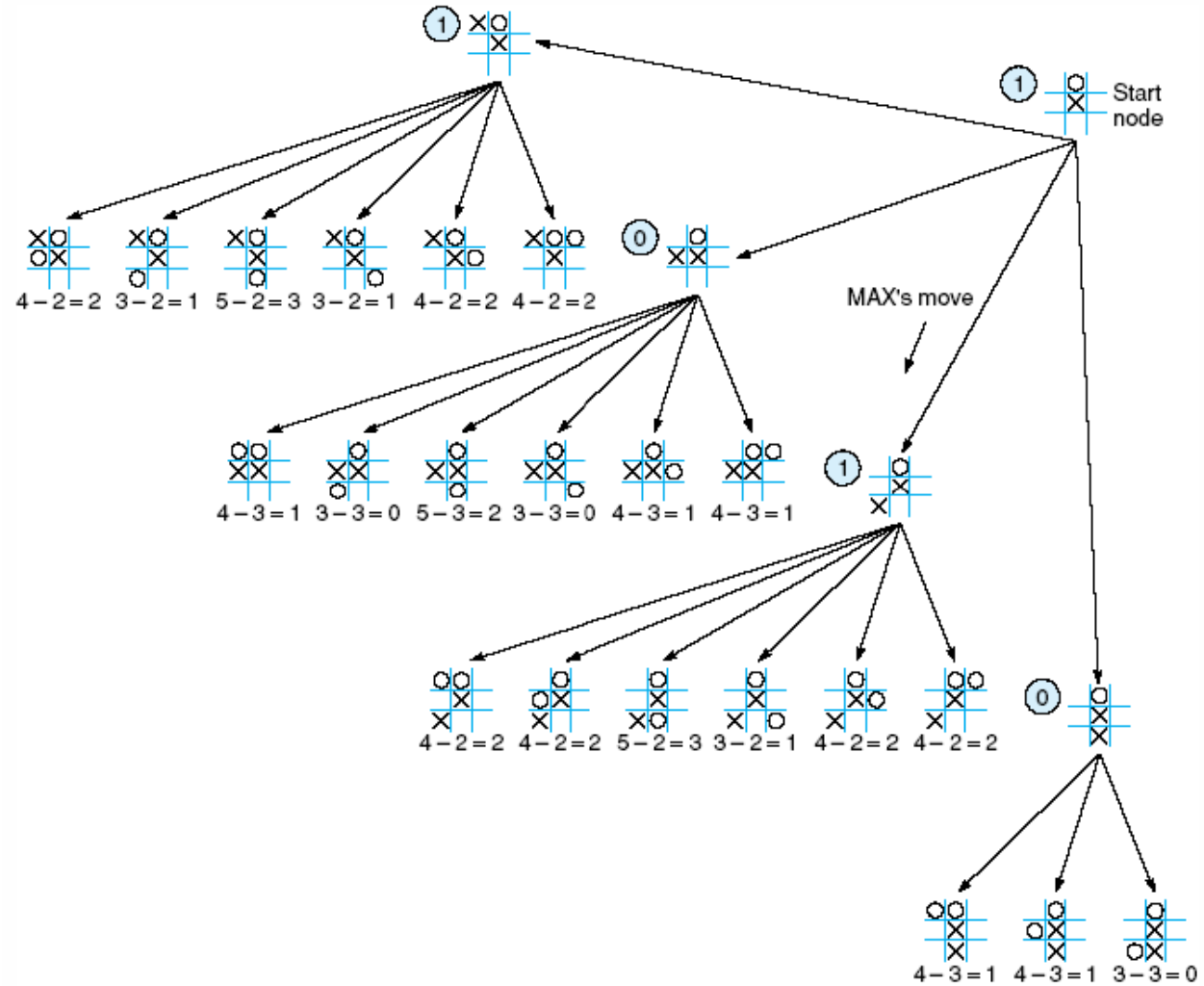
Static (Heuristic) Evaluation Functions III

- Two-ply minimax applied to the opening move of tic-tac-toe.



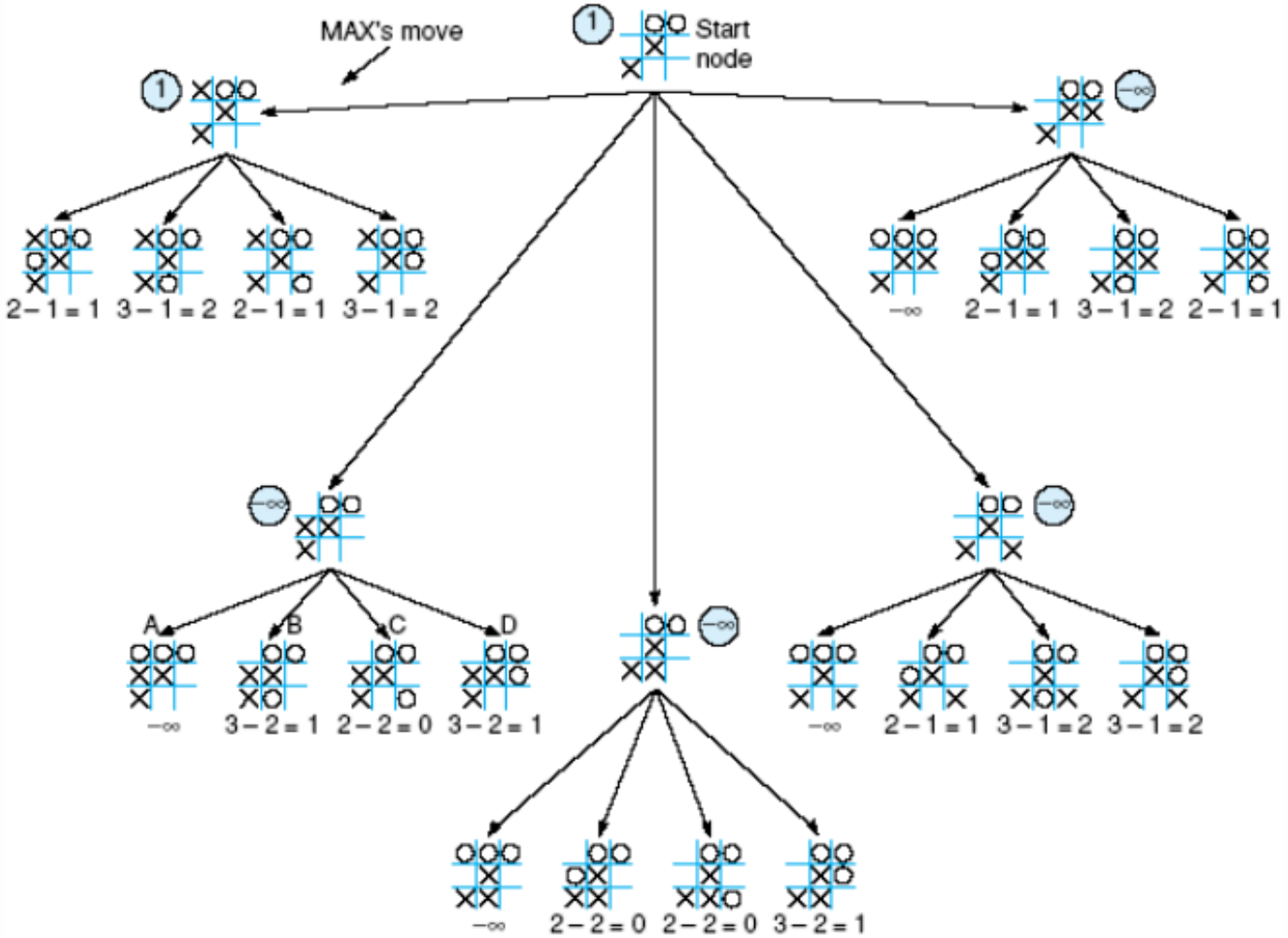
Static (Heuristic) Evaluation Functions IV

- Two ply minimax, and one of two possible MAX second moves.



Static (Heuristic) Evaluation Functions V

- Two-ply minimax applied to X's move near the end of the game.



CYPRUS INTERNATIONAL UNIVERSITY

Pseudocode for Minimax I

- ✓ *move(n,a)* is the child of n that results from taking action a
- ✓ the "value" of a node is either its actual value (if known) or a heuristic estimate (if we've reached the depth limit)
- ✓ Minimax is usually implemented using two mutually recursive functions, min-value and max-value.

max-value (node):

- if node is a leaf, return its value
- else
  - rv = -∞
  - for each action a
    - childval = min-value(move(node,a))
    - rv = max(rv, childval)
  - return rv

CYPRUS INTERNATIONAL UNIVERSITY

Pseudocode for Minimax II

min-value (node):

- if node is a leaf, return its value
- else
  - rv $= +\infty$
  - for each action a
    - childval = max-value(move(node,a))
    - rv = min(rv, childval)
  - return rv

Pseudocode for Alpha-Beta I

✓ The pseudo-code for minimax with alpha-beta pruning. The changes from normal minimax are shown in red.

max-value (node, alpha, beta):

- if node is a leaf, return its value
- else
  - rv = -∞
  - for each action a
    - childval = min-value(move(node,a), alpha, beta)
    - rv = max(rv, childval)
    - if rv$\geq$ beta, return rv
    - else alpha=max(alpha,rv)
  - return rv

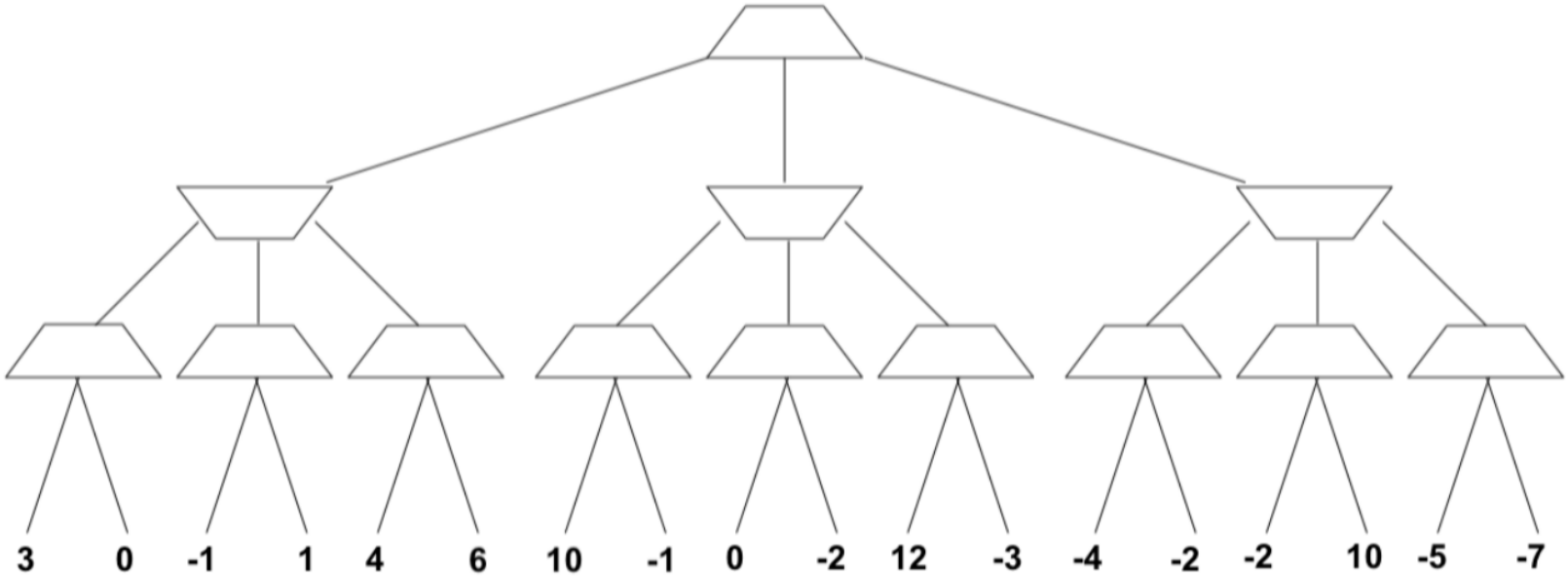
Pseudocode for Alpha-Beta II

min-value (node, alpha, beta):

- if node is a leaf, return its value
- else
  - rv $= +\infty$
  - for each action a
    - childval = max-value(move(node,a), alpha, beta)
    - rv = min(rv, childval)
    - if rv$\leq$ alpha, return rv
    - else beta=min(beta,rv)
  - return rv

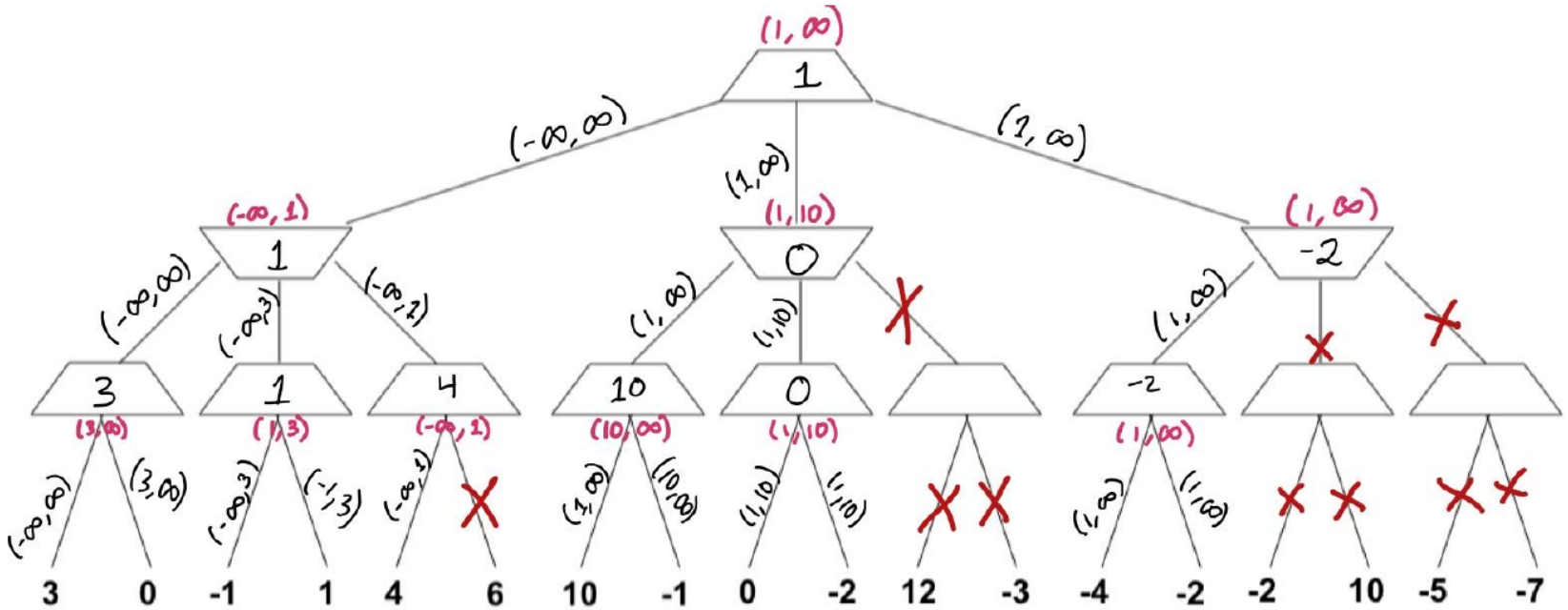
CYPRUS INTERNATIONAL UNIVERSITY

Exercise



CYPRUS INTERNATIONAL UNIVERSITY

Solution


(1, ∞)
1
(-∞, ∞)
(1, ∞)
(1, ∞)
(-∞, 1)
1
(1, 10)
0
(1, ∞)
-2
(-∞, ∞)
(-∞, 3)
(-∞, 1)
(1, ∞)
(1, 10)
(1, ∞)
3
1
4
10
0
-2
(3, ∞)
(1, 3)
(-∞, 1)
(10, ∞)
(1, 10)
(1, ∞)
(-∞, ∞)
(3, ∞)
(-∞, 3)
(-1, 3)
(-∞, 1)
(1, ∞)
(10, ∞)
(1, 10)
(1, 10)
(1, ∞)
(1, ∞)
3
0
-1
1
4
6
10
-1
0
-2
12
-3
-4
-2
-2
10
-5
-7