Single Agent environment

- Until now, we have discussed single agent environment
  - only one person or agent searching the solution space to find the goal or the solution.
  - often expressed in the form of a sequence of actions

- There might be some situations where more than one agent/person is searching for the solution in the same search space
  - usually occurs in game playing where two opponents (adversaries) are searching for a goal.
  - Multi-agent environment

Multi-agent environment

- The environment with more than one agent
- Each agent is an opponent of other agent and playing against each other
- Each agent needs to consider the action of other agent and effect of that action on their performance.

Adversarial Search

- Adversarial Search
  - Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution
  - often known as Games
- For example, in a game of tic-tac-toe player one might want that he should complete a line with crosses while at the same time player two wants to complete a line of zeros

Types of Games in AI

	Deterministic	**Chance Moves**
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

Contd..

- **Perfect information**
  - A game in which agents can look into the complete board.
  - Agents have all the information about the game, and they can see each other moves also.
  - Examples are Chess, Checkers, Go, etc.

- **Imperfect information**
  - A game in which agents do not have all information about the game and not aware with what's going on,
  - Examples are tic-tac-toe, Battleship, blind, Bridge, etc.

Contd..

- **Deterministic games**
  - Those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them.
  - Examples are chess, Checkers, Go, tic-tac-toe, etc.

- **Non-deterministic games**
  - Those games which have various unpredictable events and has a factor of chance or luck.
  - This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed.
  - Also called as stochastic games.
  - Example: Backgammon, Monopoly, Poker, etc.

Zero-Sum Game

- Adversarial search which involves pure competition.
- Each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent.
- One player of the game try to maximize one single value, while other player tries to minimize it.
- Each move by one player in the game is called as **ply**.
- Example:
  - Chess and tic-tac-toe.

Formalization of the problem

- A game can be defined as a type of search in AI which can be formalized of the following elements:
- **Initial state**
  - It specifies how the game is set up at the start.
- **Player(s)**
  - It specifies which player has moved in the state space.
- **Action(s)**
  - It returns the set of legal moves in state space.
- **Result(s, a)**
  - It is the transition model, which specifies the result of moves in the state space.

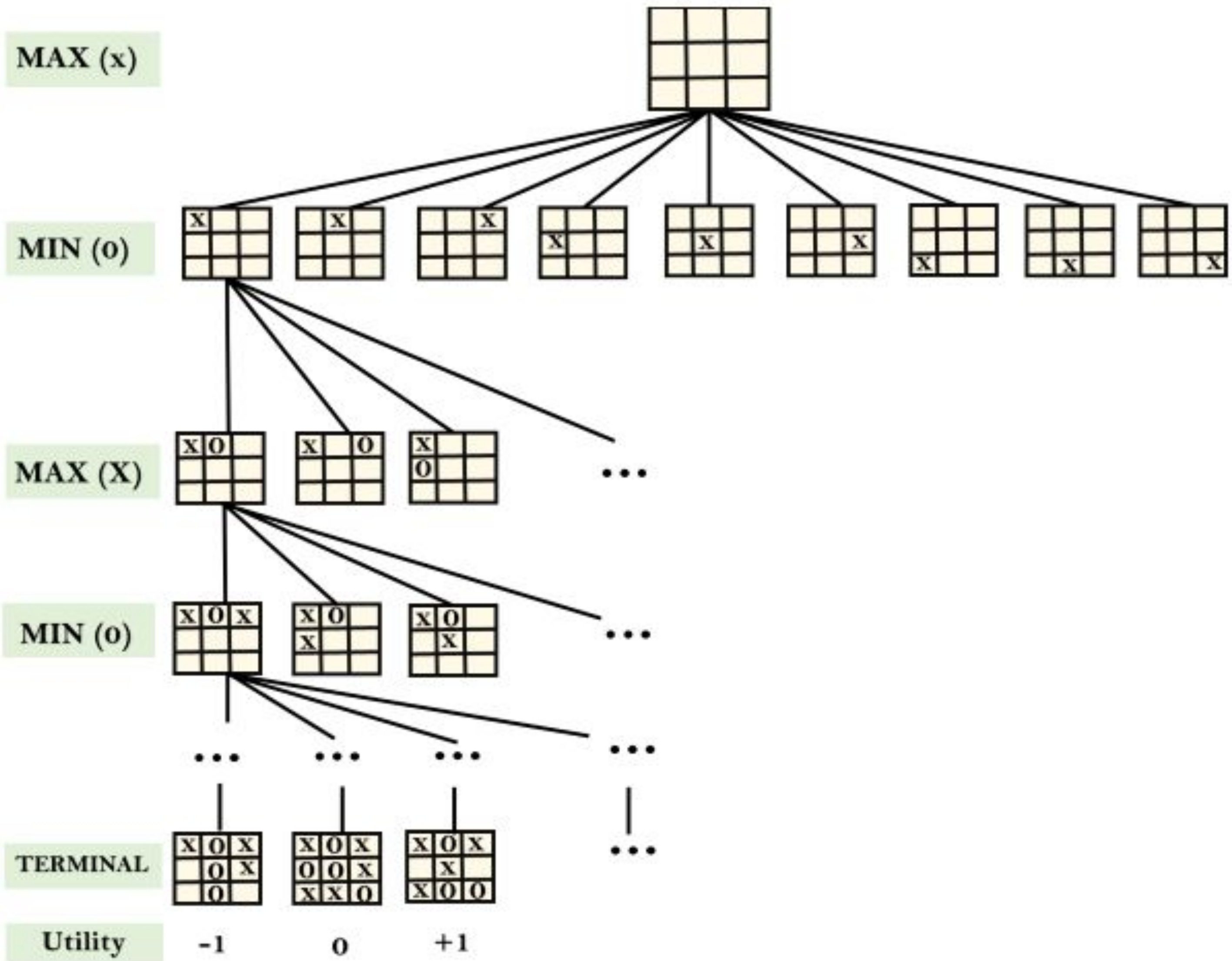
Contd..

- **Terminal-Test(s)**
  - The state where the game ends is called terminal states.
  - Terminal test is true if the game is over, else it is false at any case.
- **Utility(s, p)**
  - A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. (**Reward after winning the game**)
  - For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0

Game tree

- A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players.

Tic-Tac-Toe game tree

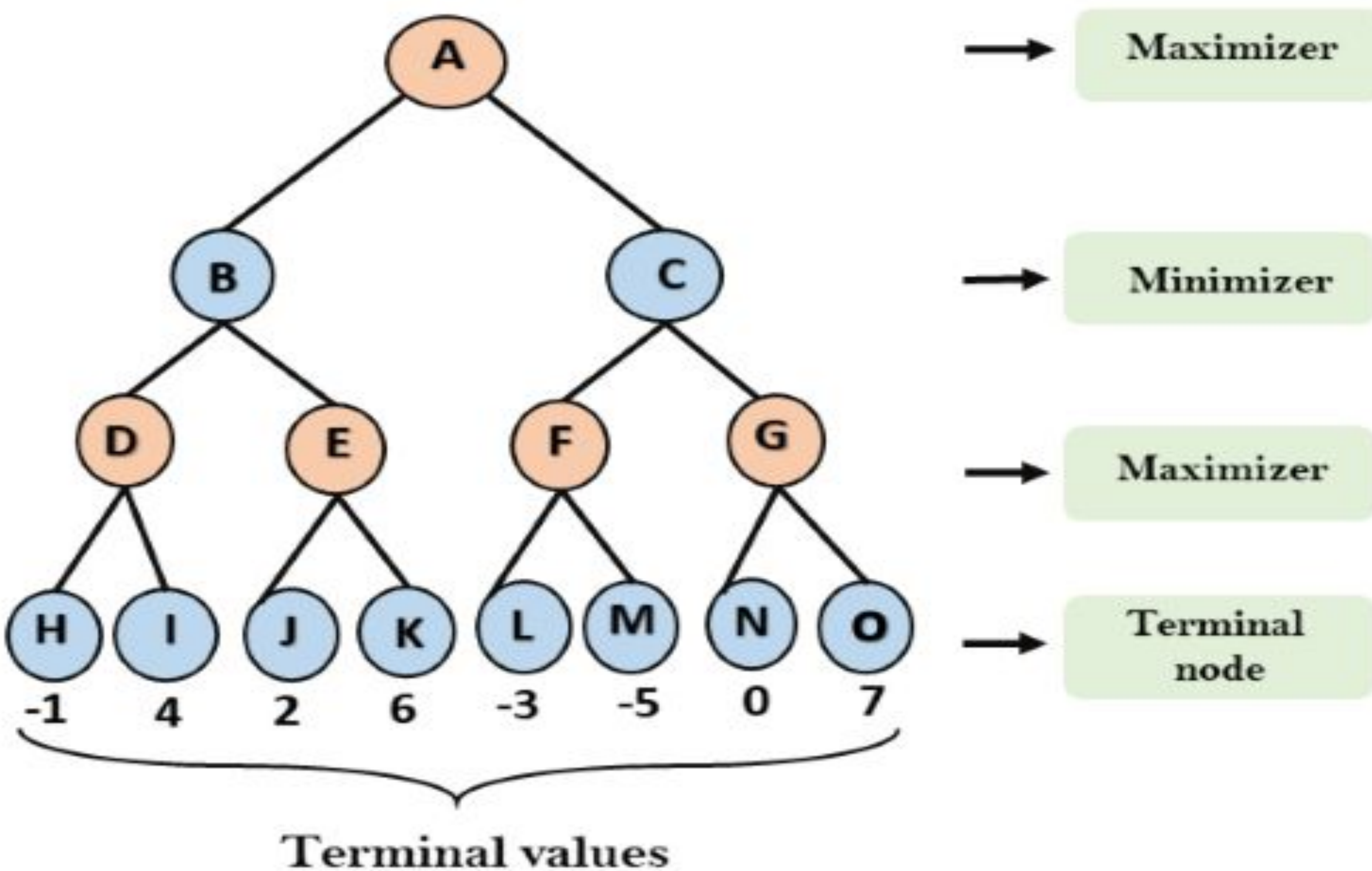


Mini-Max Algorithm

- A recursive or **backtracking** algorithm
  - The algorithm proceeds all the way down to the terminal node of the tree, compare the values and backtrack the tree to root node to decide the move.
- Performs a depth-first search algorithm for the exploration of the complete game tree.
  - Why not BFS??
- **Best Move strategy used**
  - Both players will adopt best move to not allow the opponent to win
- Computes the minimax decision for the current state
  - Max will try to maximize its utility (Best move)
  - Min will try to minimize the utility (Worst move)

Working of Min-Max Algorithm

- In the first step, the algorithm generates the entire game-tree and applies the utility function to get the utility values for the terminal states

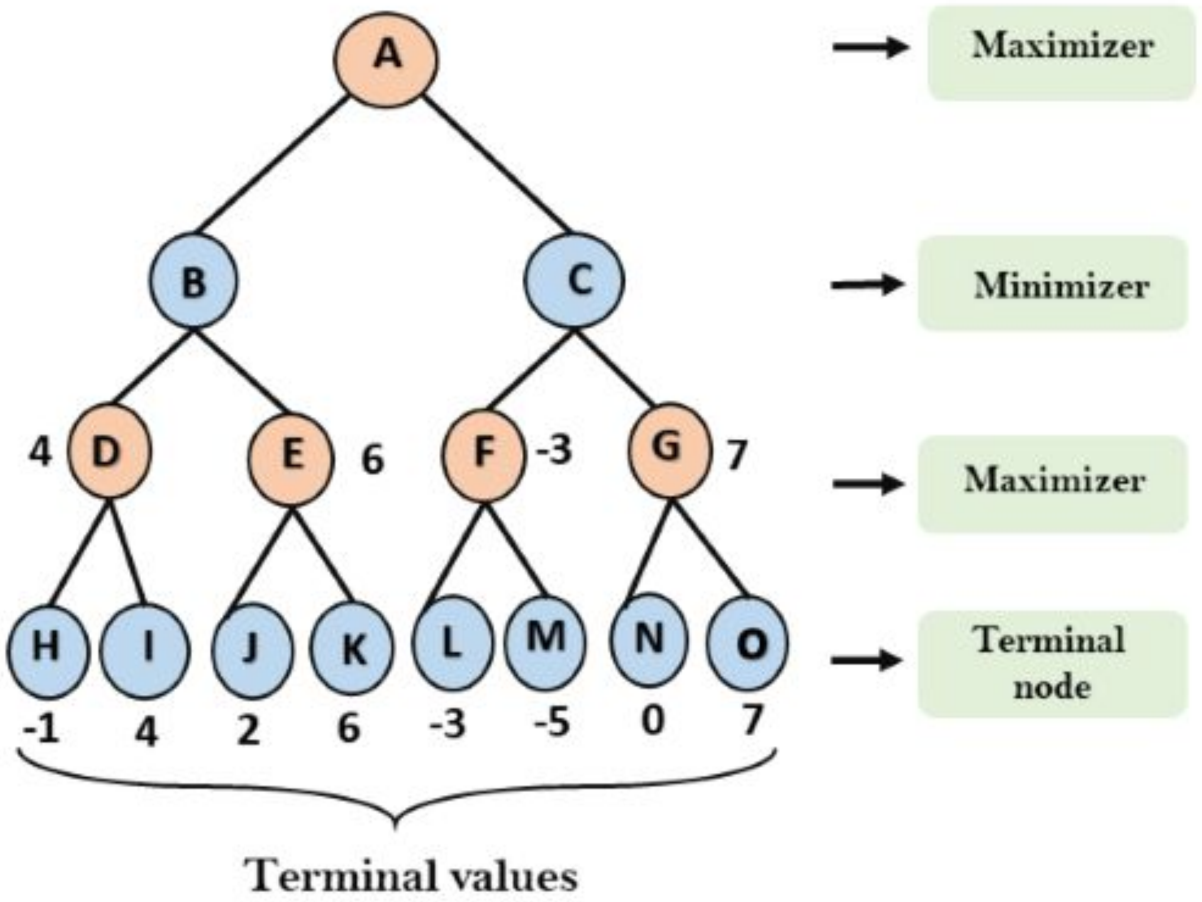


Contd..

- Step 1
  - let's take A is the initial state of the tree. Suppose maximizer takes first turn (normally max takes first move)
  - Then min will take second move and then again max, so on
- Step2
  - Start from terminal values, and find utilities for the maximizer by comparing the values of terminal nodes

For node D	max(-1,4)= 4
For Node E	max(2, 6)= 6
For Node F	max(-3,-5) = -3
For node G	max(0, 7) = 7

Contd..

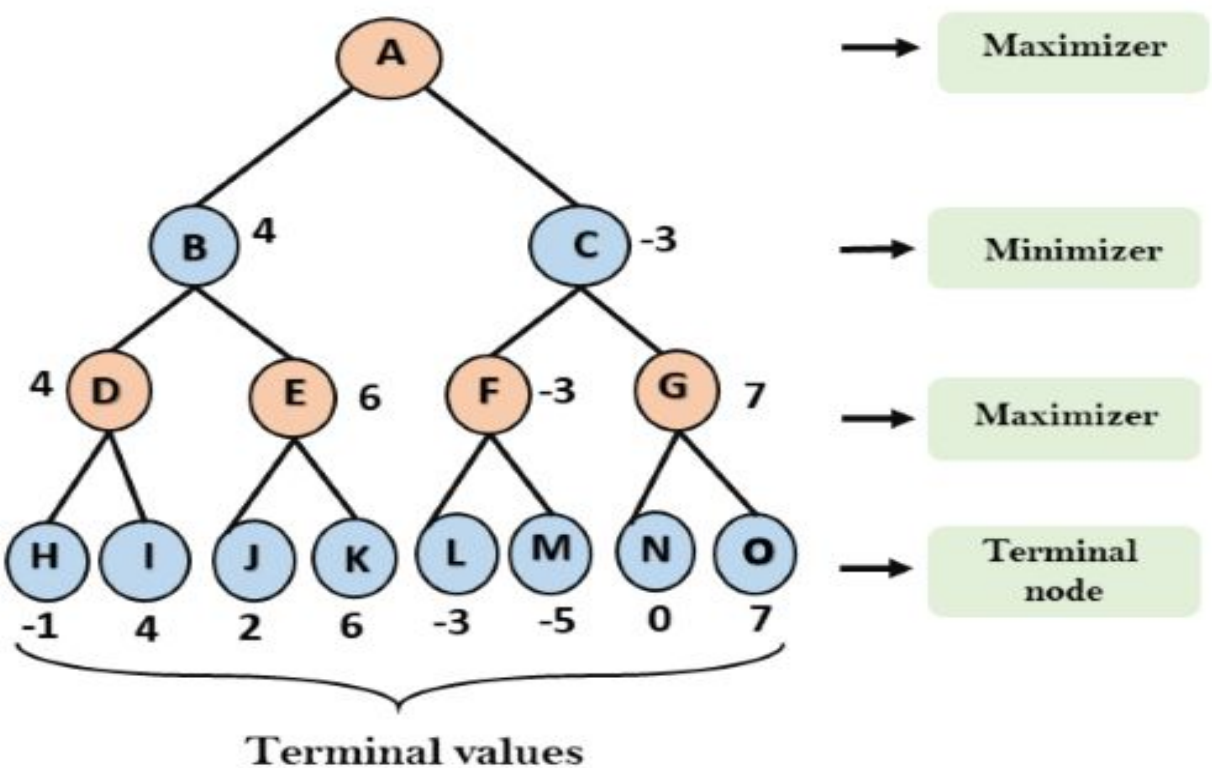


A
B
C
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

Contd..

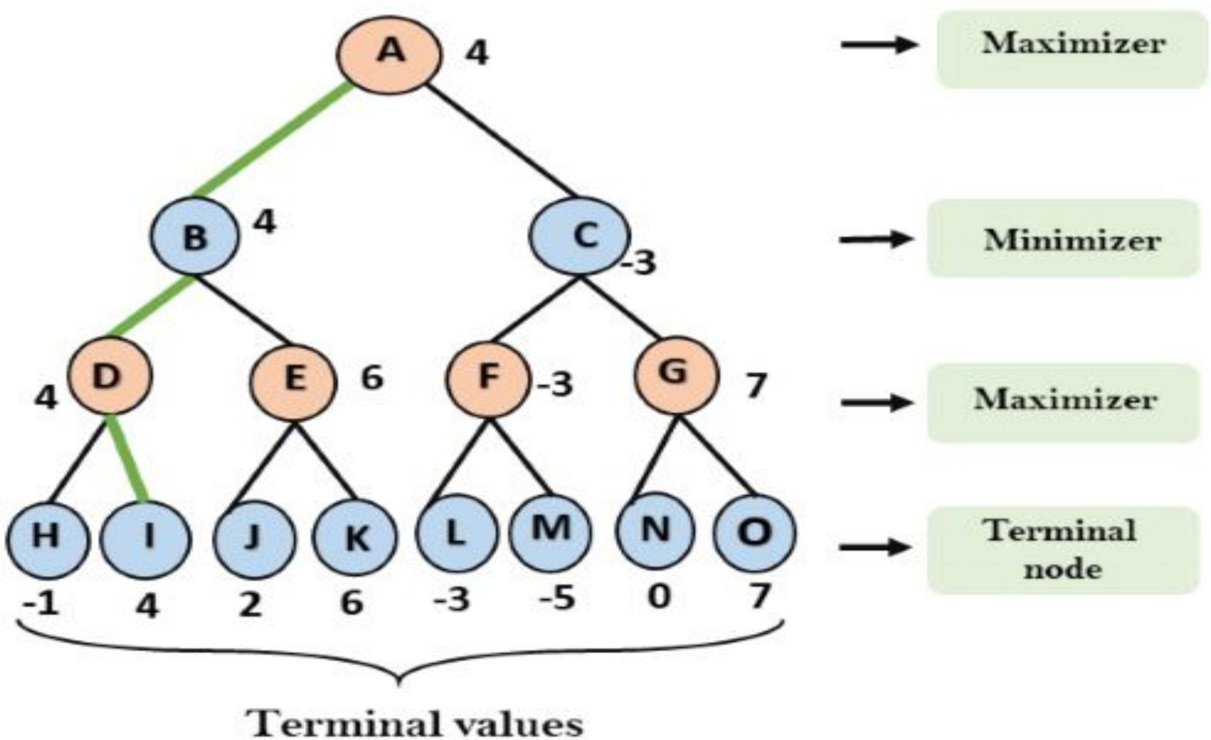
- Step 3
  - In the next step, it's a turn for minimizer
  - it will compare all nodes value, and will find the min values.

    For node B= min(4,6) = 4   For node C= min (-3, 7) = -3



Contd..

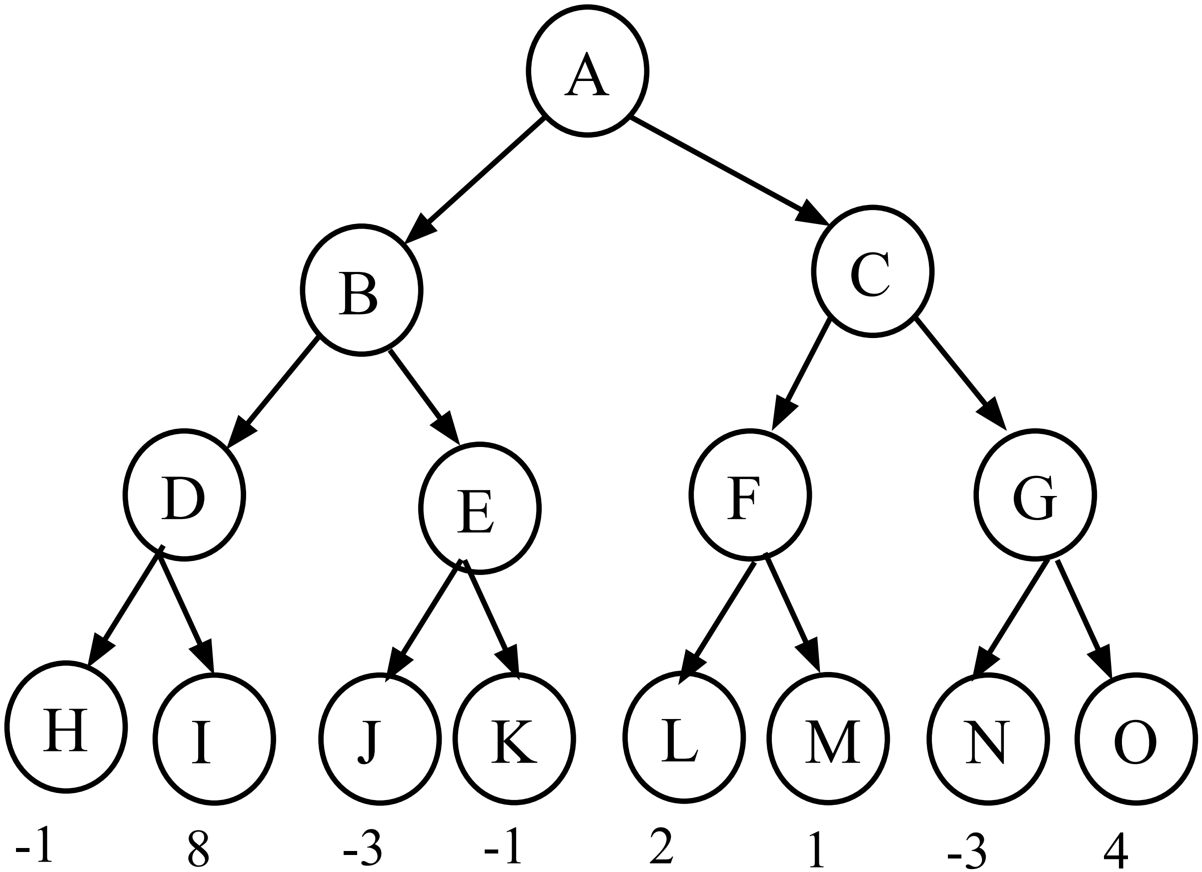
- Step 4
- Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. i.e. For node A max(4, -3)= 4



Contd..

- So, if max follows A->B->D->I, it will definitely win with the utility of 4.

Task



Performance Measurement

- **Complete ?**
  - YES, It will definitely find a solution (if exist), in the finite search tree.
- **Optimal ?**
  - YES, if both opponents are playing optimally
- **Time ?**
  - The worst case time complexity is $O(b^d)$. Simply O(n)
- **Space ?**
  - The worst case space complexity is $O(b^d)$.

Limitation of the minimax Algorithm

- The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc.
- This type of games has a huge branching factor, and the player has lots of choices to decide
  - Traversing becomes complex

- Solution
  - **alpha-beta pruning to reduce the tree and increase efficiency**

Alpha-Beta Pruning

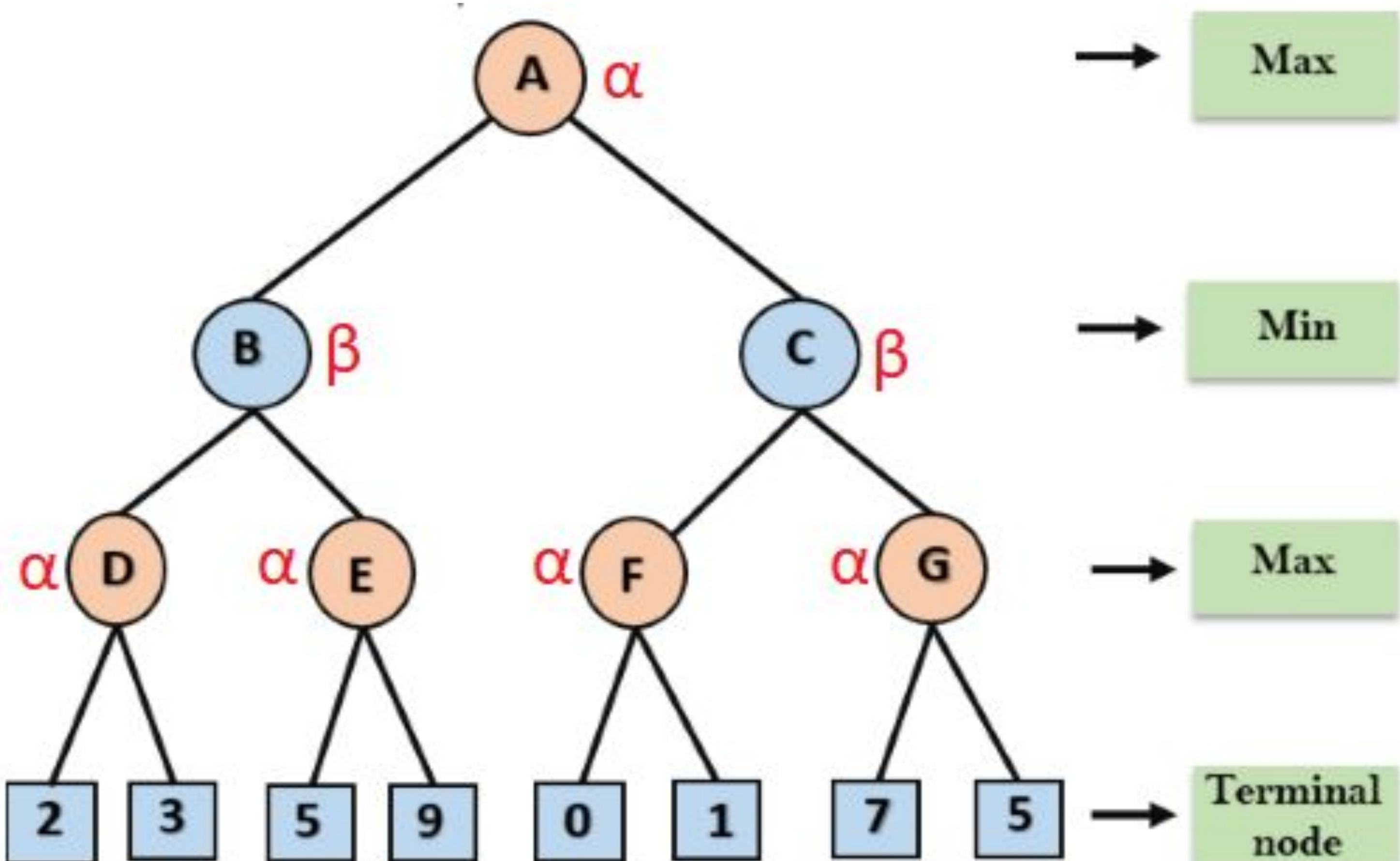
- An optimization technique/ modified version of the minimax algorithm.
- We cannot eliminate the exponential expansion of minimax, but we can cut it to half using **pruning** technique
  - A technique by which without checking each node of the game tree we can compute the correct minimax decision
  - Pruning makes the algorithm fast, as sometimes it not only prunes the leaves but also entire sub-tree that is not really affecting the final decision but making algorithm slow

Contd..

- Involves two threshold parameter Alpha and beta for future expansion, so name **alpha-beta pruning/Algo.**
- The two-parameter can be defined as:
  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is $-\infty$.
  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is $+\infty$.
- Cut off search by exploring less no of nodes
  - If a best path is found, remaining paths are not explored instead we cut off them

Example

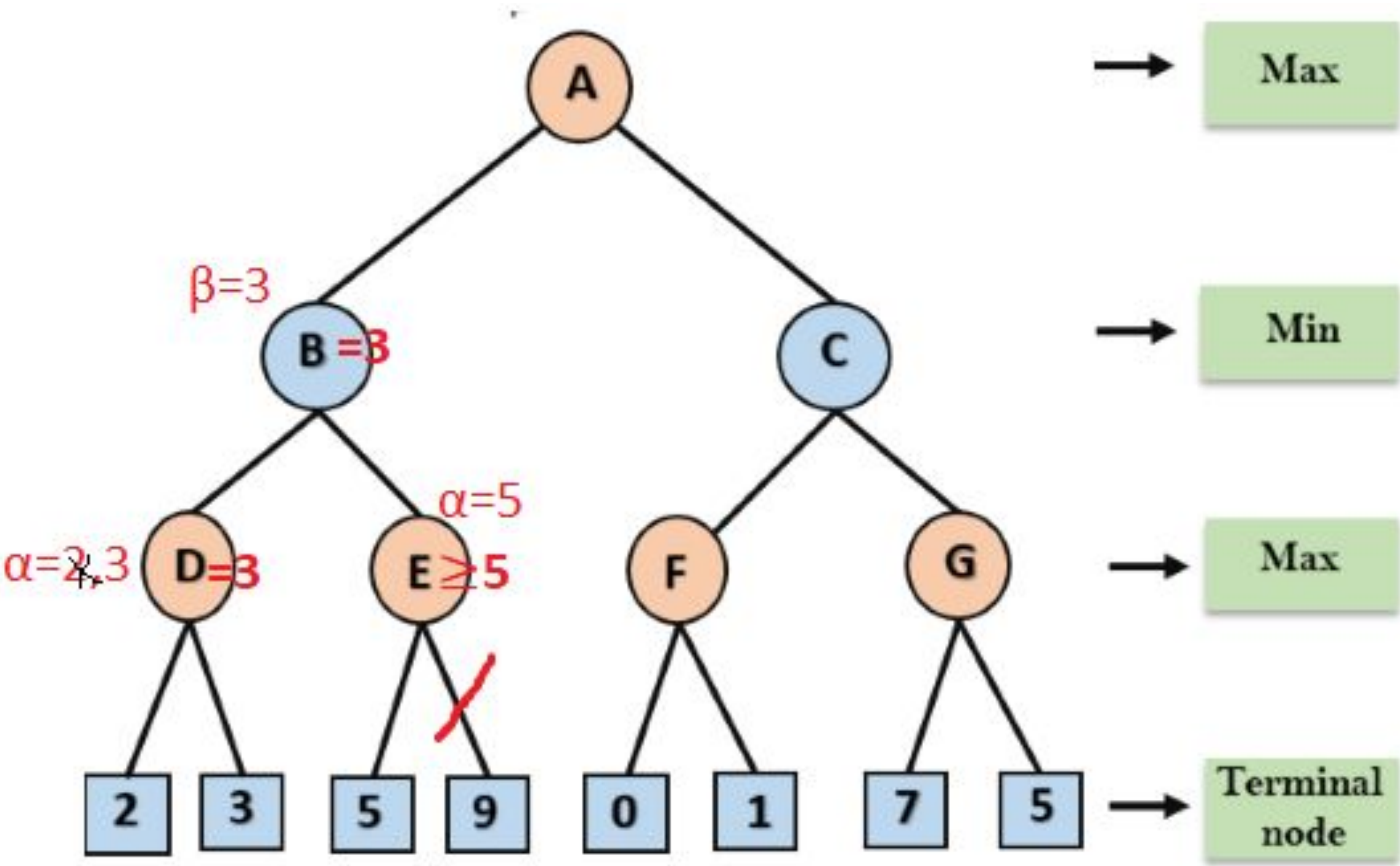
- **Step 1**: At the first step the, Max player will start first move from node A (Max), will move downwards to B->D and terminal node and find **α and** β values using utility values.



Contd..

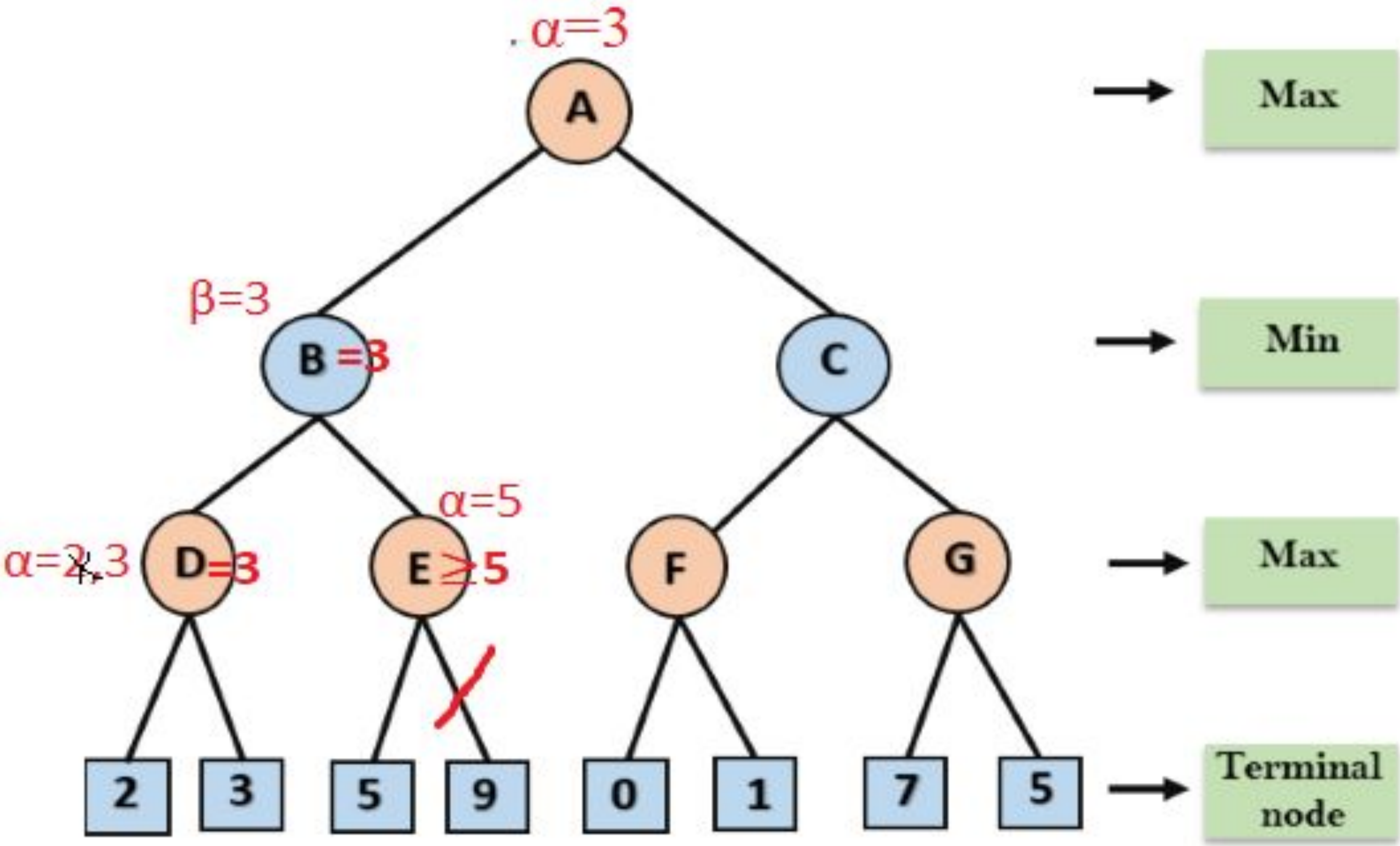
- **Step 2**: at node D, left value of 2 will be assigned to **α.** Now it's mean that value of D will be ≥ 2 as its max turn. Now check other node that has value of 3 and it is turn of Max, so α value be updated with 3 and D node will have value of 3.
- **Step 3:** now at node B, assign value of node D to β at node B. as its min turn its mean node B will have value ≤ 3.
- After that, next successor E will be traversed which will get value of α=5 from left leaf of E. as it's max turn, its mean that node E will have value of ≥ 5, where as node B already have a minimum value of 3. So, there is no need to traverse the other successor of E and hence pruned.

Contd..



Contd..

- **Step 4:** now at node A, α will get value of 3. as its turn of max, so value of A will be ≥ 3.
- Here any other path that will give value less than 3 will be pruned.



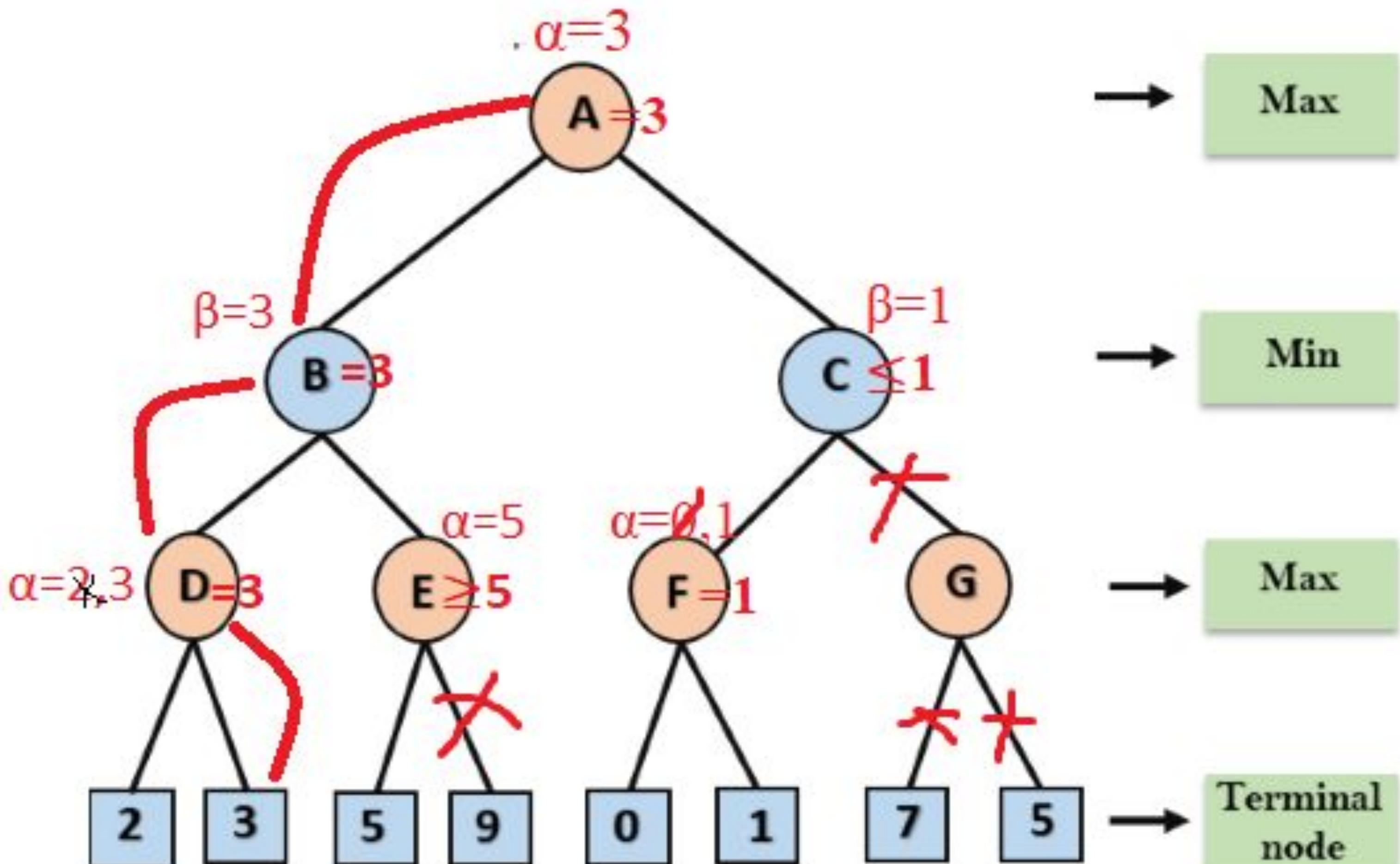
Contd..

- **Step 5:**Now, node C from right side will be traversed, which will go downward to F and F will get value of 0 for α. As it s turn of Max so value for F will be ≥ 0. right successor of F has value 1, so α value be updated with 1 and F will get max value of 1.
- **Step 6:** F will return value of 1 to β at node C. as its turn of Min and value of C will be ≤ 1, whereas node A already has max value of 3, so another value generated after traversing nodes further will return value ≤ 1. Therefore, no need to explore further and right complete branch of C will be pruned.

- A will retain utility value of 3 i.e. optimal value for maximizer.

Contd..

- So, So, if max follows A->B->D->I, it will definitely win with the utility of 3.



Complexity

- Complexity is dependent on traversal ordering of nodes. It can be of two types:
- **Worst ordering**
  - In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm.
  - In this case, the best move occurs on the right side of the tree and it also consumes more time because of alpha-beta factors.
  - The time complexity for such an order is $O(b^d)$.
- **Ideal ordering:**
  - The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree.
  - We apply DFS hence, it first search left of the tree and go deep twice as minimax algorithm in the same amount of time.
  - Complexity in ideal ordering is $O(b^{d/2})$.

Task

- Find the best path and optimal value for Max using alpha-beta pruning.