Minimax algorithm

The Minimax algorithm is claimed to be a recursive or backtracking algorithm that is responsible for choosing the best optimal move in the conflicting environment. The Minimax algorithm operates on a tree structure known as the game tree, which is the collection of all the possible moves in the corresponding game states in a given game. The game tree's leaf node accommodates all the possible moves. The game state denotes the current board condition. With every single move, the game state changes and the game tree gets updated height-wise. When visualized, this game tree often resembles an inverted tree, with the root representing the current game state and the branches representing possible moves.

This algorithm simply works by proceeding down to the tree until it reaches the leaf node and then it backs up the minimax values through the tree as the recursion unwinds. The prime motive of the Minimax algorithm is to maximize the chance of winning for the maximizer. In the game tree, every node is assigned weights that influence the chances of winning, the higher the weights, the higher the chances of winning.

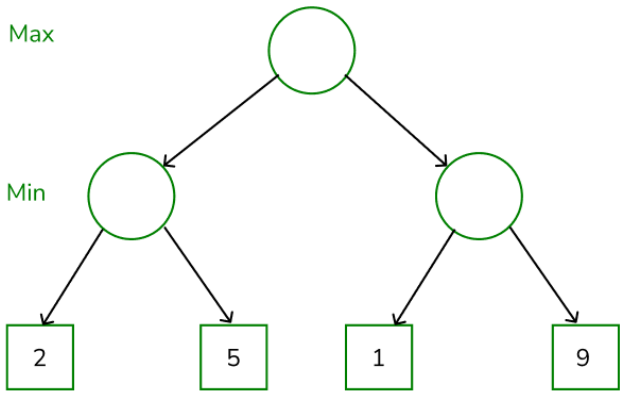
Key terminologies in the Minimax algorithm

- **Minimax Tree:** A tree structure shows all possible moves and game states in a game, used by Minimax to find the best move.
- **MAX (Maximizer)** – Maximizer seeks favorable outcomes in games by maximizing chances of winning for themselves as players.
- **MIN (Minimizer)** – Minimizer reduces winning chances, making moves leading to least favorable outcome for the Maximizer in games.
- **Initial state** – Starting state of game board, representing the configuration at the beginning of the game.
- **Terminal state** -Terminal states are the final outcomes of a game board, resulting in either a win, loss, or draw.
- **Heuristic Evaluation Function:** Function evaluates game states for maximizer, assigning numerical values to each game state based on piece positions, material advantage, and board control.

Understanding the Minimax algorithm: Decision-making in game trees

The minimax algorithm is a **backtracking algorithm** that **recursively** tries different possibilities and backtracks when the solution is not found. In the context of the Minimax algorithm, this involves exploring all possible moves in the game tree that evaluate each potential outcome and making decisions based on the evaluations.

The diagram below represents the binary tree that holds the possible game states and moves in a game, with each leaf node representing a final game state.



figure(a) Example of Minimax algorithm that holds the possible game states

Maximizer and Minimizer

In the Minimax algorithm, there are two players such as the maximizer and the minimizer. The maximizer aims to maximize its score (e.g., by winning the game), conversely, the minimizer aims to minimize the maximizer's score (e.g., by preventing the maximizer from winning).

Example of decision-making

- **Initial decision-making:** An example scenario in Figure (a) within the perfect binary tree causes the maximizer to make the first move at the root. The maximizer has the option to choose to traverse through the tree, in this case, it needs to choose the path between left or right.
- **Minimizer's choice:** Assuming that maximizer traverses to the left where it reaches the minimizer node, the minimizer has the responsibility of selecting the optimal value and it chooses between two values (2 and 5). The objective of the minimizer is to minimize the maximizer's score, so it chooses the lowest value 2. The value 2 is then passed to the root node so that it can be compared later when the player earns an optimal value when he traverses to the right side of the game tree.
    - **Traversal to the Right:** The maximizer then traverses to the right side of the game tree and encounters another set of decision nodes. Eventually, it reaches another minimizer node where the minimizer again selects the optimal minimal value 1 among its options.

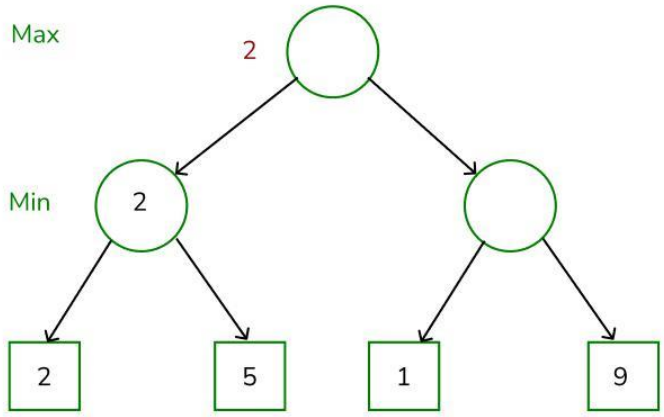


Figure (b) Minimizer's choice

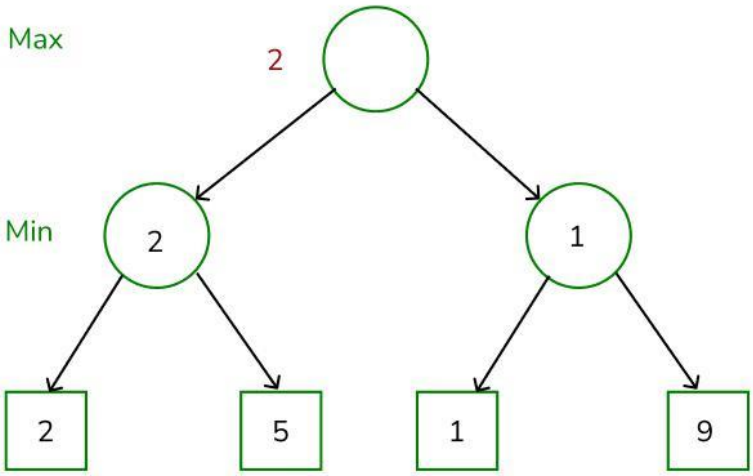
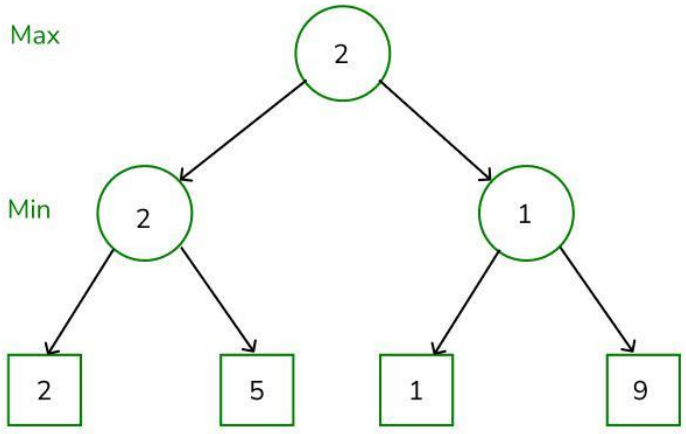


Figure (c) The maximizer traversing through the right

- **Recursive process:** The process continues recursively where each player makes decisions based on whether they are maximizer or minimizer and the objective of maximizing or minimizing the score respectively.
- **Comparison and outcome:** The maximizer node (root node) already holds the value of 2, when the player earns another optimal value that is 1 while traversing in the right path, it is then compared to get the final optimal value. The maximizer earns the optimal value 2 based on the decisions made by both players.



Final outcome of binary game tree

Time and Space complexity of Minimax algorithm

Time complexity

The minimax algorithm explores the complete game tree using a depth-first search approach that considers all possible moves at each level. The time complexity of the minimax algorithm can be written as,

$$O(b^m)$$

b – denotes the legal moves available at each point,
m – denotes the maximum depth of the tree.

This complexity arises because, at each level of the tree, the minimax algorithm must consider all b legal moves and this process repeats recursively for m levels until a terminal state is reached.

Space complexity

The space complexity of the minimax algorithm relies on how the algorithm generates and stores the actions. In the case, where if all the possible actions are generated and stored simultaneously, the space complexity can be written as

$$O(bm)$$

b – denotes the legal moves available at each point,
m – denotes the maximum depth of the tree.

This arises because the algorithm needs to store all the possible actions at each level of the game tree which leads to exponential growth in space requirements with the depth of the tree.
However, if the actions are generated and processed one at a time, the space complexity reduces to O(m).