MEASI
Institute *of*
Information Technology

Ms.Rafath Zahra

MEASI Institute of Information Technology

SEARCHING FOR SOLUTIONS

.

Consider a simple 5x5 maze grid:

(0, 0) (0, 1) (0, 2) (0, 3) (0, 4)
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4)
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4)
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4)
(4, 0) (4, 1) (4, 2) (4, 3) (4, 4)

State: Represents the agent's current position in the maze.

For example, if the agent is at (2, 3), it means the agent is in the 3rd row and the 4th column of the grid.
Expanded Example
Let's use the maze to demonstrate how states change as the agent moves:

Initial State: (0, 0) - The agent starts at the top-left corner.
Possible Actions: Move Right, Move Down.
Resulting States:
Move Right: (0, 1)
Move Down: (1, 0)
If the agent starts at (0, 0) and moves right: New State: (0, 1)
If the agent then moves down from (0, 1): New State: (1, 1)

MEASI Institute of Information Technology

SEARCHING FOR SOLUTIONS

.

Consider a simple 5x5 maze grid:

(0, 0) (0, 1) (0, 2) (0, 3) (0, 4)
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4)
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4)
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4)
(4, 0) (4, 1) (4, 2) (4, 3) (4, 4)

Tree-Search in the Maze
Initial State: (0, 0)
Goal State: (4, 4)
Fringe: Starts with the initial state [(0, 0)]
Expansion Process:
From (0, 0), the agent can move to (0, 1) or (1, 0).
From (0, 1), the agent can move to (0, 2), (1, 1), or back to (0, 0).
Tree-Search Steps
Start at (0, 0).
Check if (0, 0) is the goal (4, 4). It's not.
Expand (0, 0): add (0, 1) and (1, 0) to the fringe.
Choose (0, 1) from the fringe.
Check if (0, 1) is the goal. It's not.
Expand (0, 1): add (0, 2), (1, 1), and remove (0, 0) (since it has already been explored).
This process continues until the goal state (4, 4) is reached.

MEASI Institute of Information Technology

SEARCHING FOR SOLUTIONS

Search Techniques and State Space:

Use an explicit search tree generated by the initial state and the successor function.
A search graph may occur when the same state can be reached via multiple paths.

Search Tree Example:

Example of finding a route from Arad to Bucharest.
The root of the search tree is a search node corresponding to the initial state.
Test if the initial state is a goal state; if not, expand the current state to generate new states.

Expanding and Generating States:

Apply the successor function to the current state to generate new states.
Example: From Arad, generate states In(Sibiu), In(Timisoara), and In(Zerind).

Search Strategy:

Choose which state to expand next.
Continue choosing, testing, and expanding until a solution is found or no more states can be expanded.

MEASI Institute of Information Technology

SEARCHING FOR SOLUTIONS

Distinction Between State Space and Search Tree:

State space: finite number of states (e.g., 20 cities).
Search tree: infinite number of paths (e.g., Arad-Sibiu-Arad-Sibiu).

Node data structure includes:
STATE: The state in the state space.
PARENT-NODE: The node in the search tree that generated this node.
ACTION: The action applied to the parent to generate the node.
PATH-COST: Cost of the path from the initial state to the node.
DEPTH: Number of steps along the path from the initial state

Fringe:

Collection of nodes generated but not yet expanded.
Each element of the fringe is a leaf node.

MEASI Institute of Information Technology

GENERAL TREE SEARCH ALGORITHM

Initialize the search tree using the initial state of the problem.

Loop:

If no candidates for expansion, return failure.
Choose a leaf node for expansion according to the strategy.
If the node contains a goal state, return the solution.
Expand the node and add the resulting nodes to the search tree.

Ms.Rafath Zahra

Measuring problem-solving performance

The output of a problem-solving algorithm is either failure or a solution. (Some algorithms might get stuck in an infinite loop and never return an output.)

We will evaluate an algorithm's performance in four ways:

Completeness: Is the algorithm guaranteed to find a solution when there is one?
Optimality: Does the strategy find the optimal solution?
Time complexity: How long does it take to find a solution?
Space complexity: How much memory is needed to perform the search?

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure

  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if EMPTY?(fringe) then return failure
    node ← REMOVE-FIRST(fringe)
    if GOAL-TEST[problem] applied to STATE[node] succeeds
      then return SOLUTION(node)
    fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

```
function EXPAND(node, problem) returns a set of nodes

  successors ← the empty set
  for each (action, result) in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    STATE[s] ← result
    PARENT-NODE[s] ← node
    ACTION[s] ← action
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(STATE[node], action, result)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```

Figure 3.9 The general tree-search algorithm. (Note that the *fringe* argument must be an empty queue, and the type of the queue will affect the order of the search.) The *SOLUTION* function returns the sequence of actions obtained by following parent pointers back to the root.

Here are the key points regarding time and space complexity, and the assessment of search algorithms:

Ms.Rafath Zahra

1. Measure of Problem Difficulty:
   - Typically measured by the size of the state space graph in theoretical computer science.
   - In AI, complexity is expressed in terms of:
     - b: Branching factor (maximum number of successors of any node).
     - d: Depth of the shallowest goal node.
     - m: Maximum length of any path in the state space.

2. Time and Space Complexity:
   - Time Complexity: Measured by the number of nodes generated during the search.
   - Space Complexity: Measured by the maximum number of nodes stored in memory.

3. Search Cost:
   - Search cost includes time complexity and can also consider memory usage.
   - Total cost combines search cost and the path cost of the solution found.

4. Example:
   - For finding a route from Arad to Bucharest:
     -Search Cost: Time taken by the search.
     - Solution Cost: Total length of the path in kilometers.
   - Total cost requires combining search cost (time) and solution cost (distance).

Ms.Rafath Zahra

INFORMED AND UNINFORMED SEARCH

Parameters	Informed Search	Uninformed Search
Known as	It is also known as Heuristic Search.	It is also known as Blind Search.
Using Knowledge	It uses knowledge for the searching process.	It doesn't use knowledge for the searching process.
Performance	It finds a solution more quickly.	It finds solution slow as compared to an informed search.
Completion	It may or may not be complete.	It is always complete.
Cost Factor	Cost is low.	Cost is high.
Time	It consumes less time because of quick searching.	It consumes moderate time because of slow searching.
Direction	There is a direction given about the solution.	No suggestion is given regarding the solution in it.
Implementation	It is less lengthy while implemented.	It is more lengthy while implemented.

INFORMED AND UNINFORMED SEARCH

Parameters	Informed Search	Uninformed Search
Efficiency	It is more efficient as efficiency takes into account cost and performance. The incurred cost is less and speed of finding solutions is quick.	It is comparatively less efficient as incurred cost is more and the speed of finding the Breadth-Firstsolution is slow.
Computational requirements	Computational requirements are lessened.	Comparatively higher computational requirements.
Size of search problems	Having a wide scope in terms of handling large search problems.	Solving a massive search task is challenging.
Examples of Algorithms	• Greedy Search • A* Search • AO* Search • Hill Climbing Algorithm	• Depth First Search (DFS) • Breadth First Search (BFS) • Branch and Bound

MEASI Institute of Information Technology

Ms.Rafath Zahra

Breadth First Search (BFS) for Artificial Intelligence

The Breadth-First Search is a traversing algorithm used to satisfy a given property by searching the tree or graph data structure. It belongs to uninformed or blind search AI algorithms

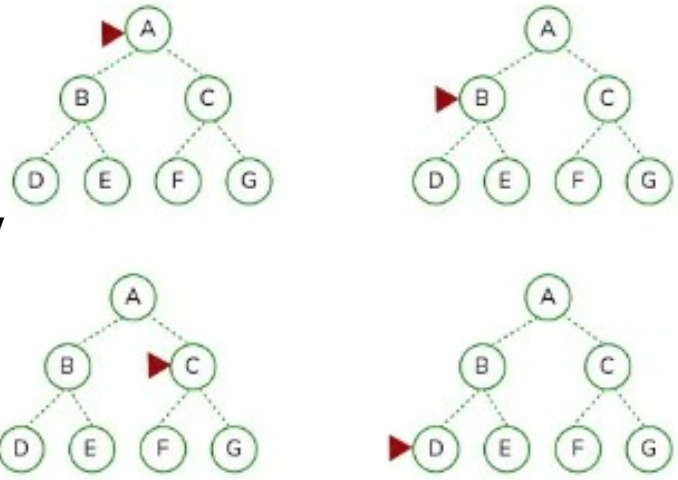
It is optimal for unweighted graphs and is particularly suitable when all actions have the same cost.

Originally it starts at the root node, then it expands all of its successors, it systematically explores all its Neighbouring nodes before moving to the next level of nodes.

First-in-First-Out (FIFO): The FIFO queue is typically preferred in BFS

Cost-optimal: BFS always aims to find a solution with a minimum cost prioritizing the shortest path, when BFS generates nodes at a certain depth d, it has already explored and generated all the nodes at the previous depth d-1.

Consequently, if a solution exists within a search space, BFS can discover it as soon as it reaches that depth level. Therefore, BFS is said to be a cost-optimal solution.



Ms.Rafath Zahra

MIN MAX GAME THEORY

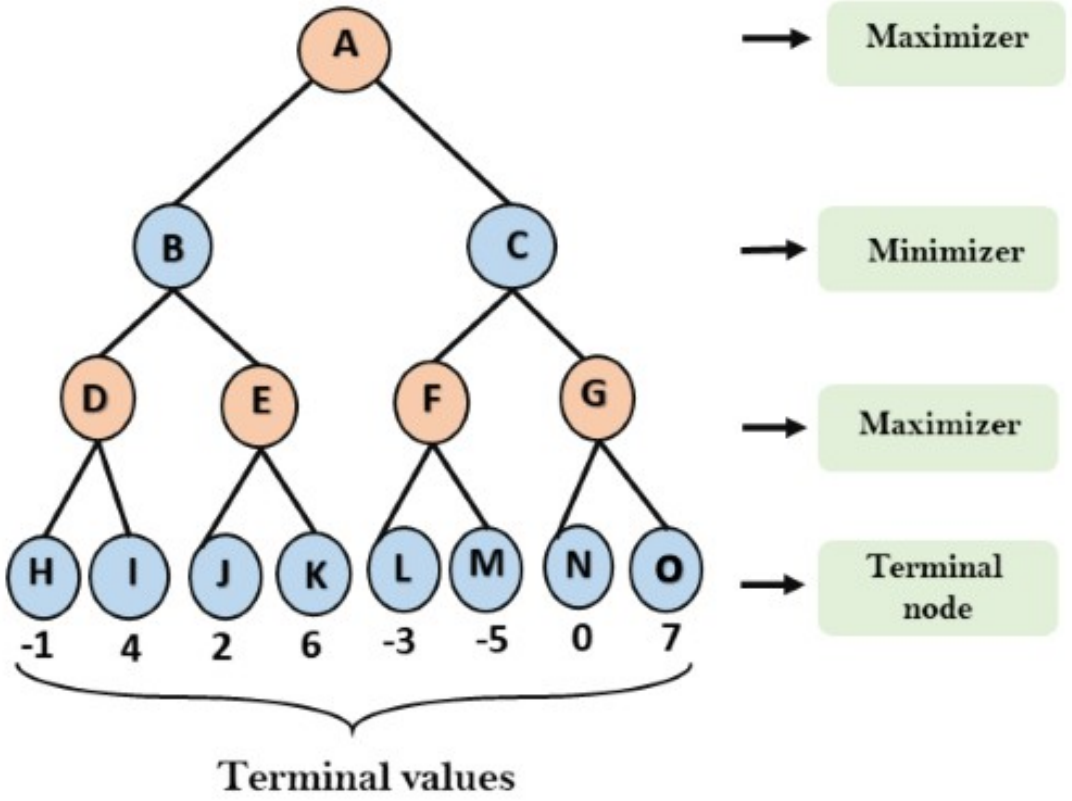
Strategy: The minimizer's strategy is to block the maximizer from achieving high scores, making it harder for the maximizer to win.
Game Theory: This approach ensures that both players (the maximizer and the minimizer) are playing optimally, leading to a fair and balanced game

Minimizer's Goal: The minimizer aims to minimize the score that the maximizer can achieve. This means the minimizer will choose the move that leaves the maximizer with the smallest possible score.

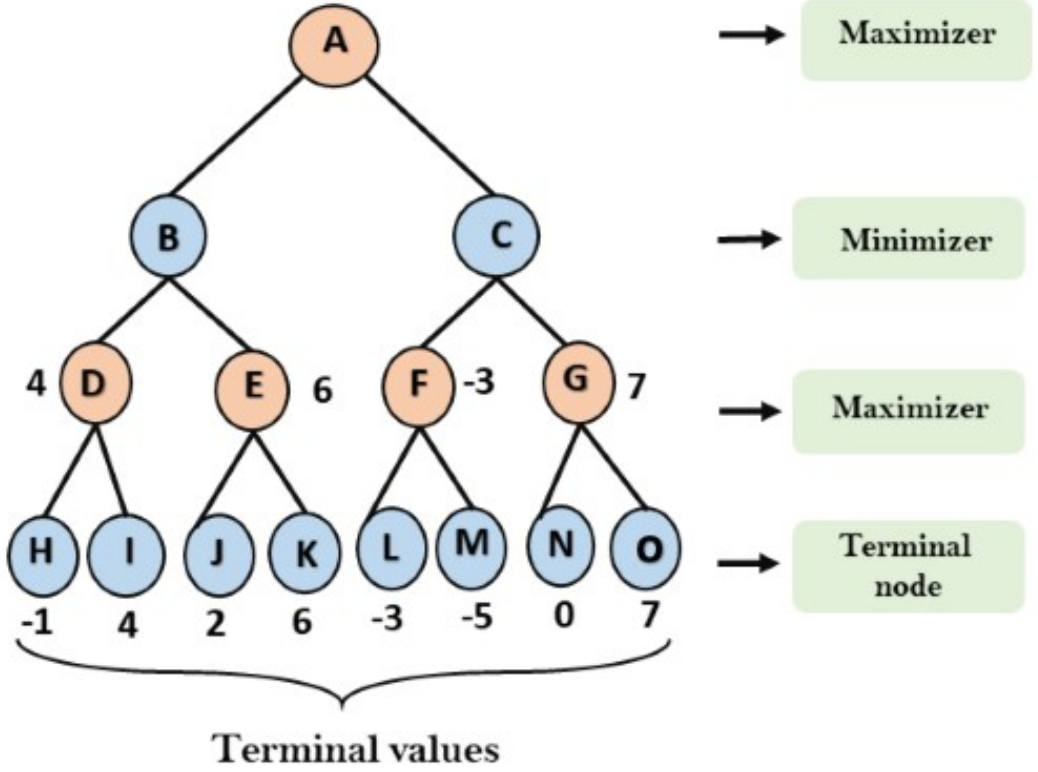
Effect on the Maximizer: By choosing moves that lead to lower scores, the minimizer forces the maximizer into less favorable positions, thus reducing the maximizer's overall score.

MIN MAX GAME THEORY

Ms.Rafath Zahra



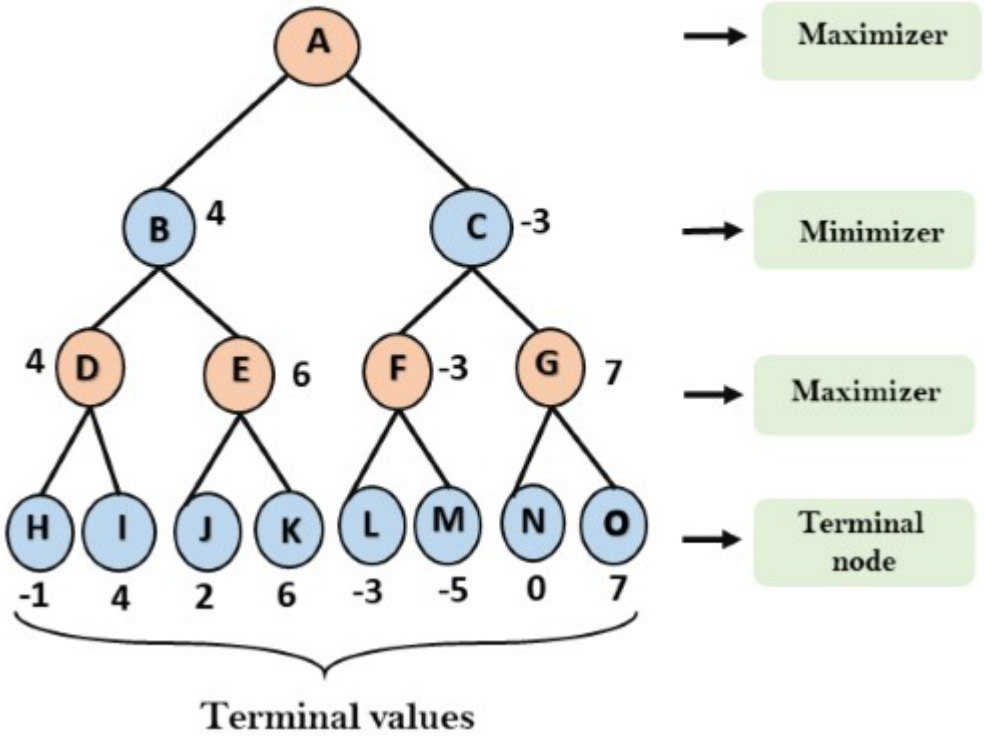
- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



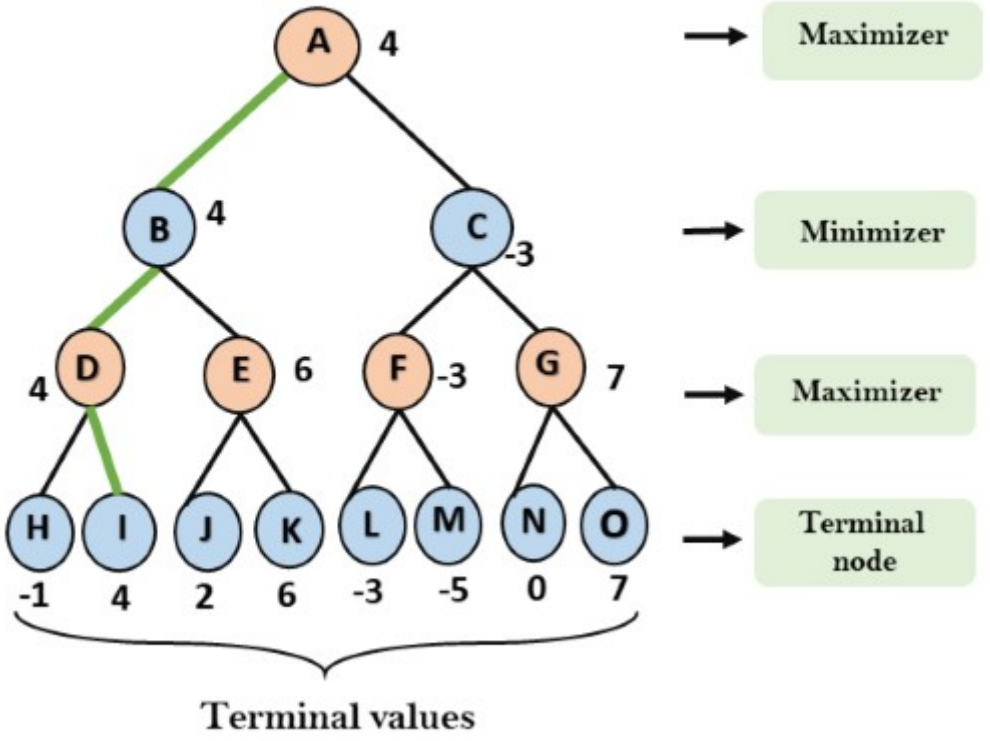
MIN MAX GAME THEORY

Ms.Rafath Zahra

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



- For node A max(4, -3)= 4



Alpha Beta Pruning

Ms.Rafath Zahra

Alpha-beta pruning is a modified version of the minimax algorithm.

Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
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 -∞.
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 +∞.

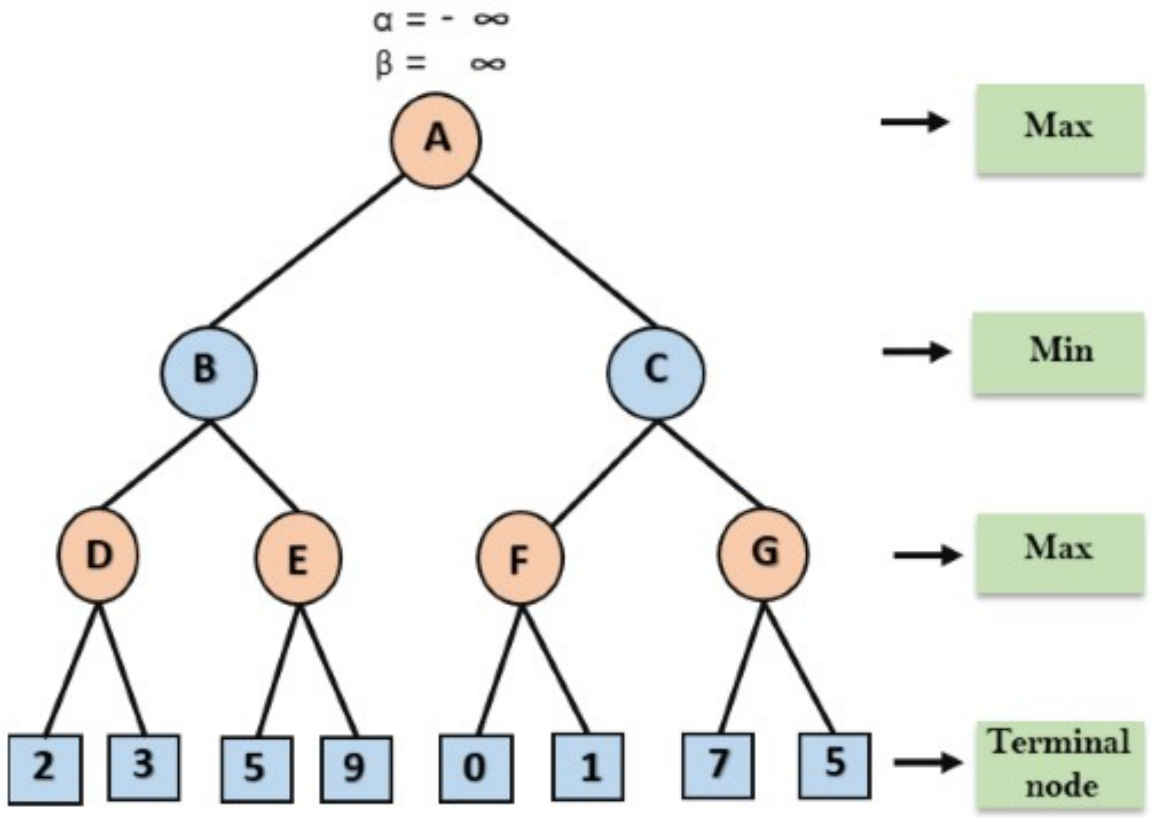
The main condition which required for alpha-beta pruning is: **α >= β**

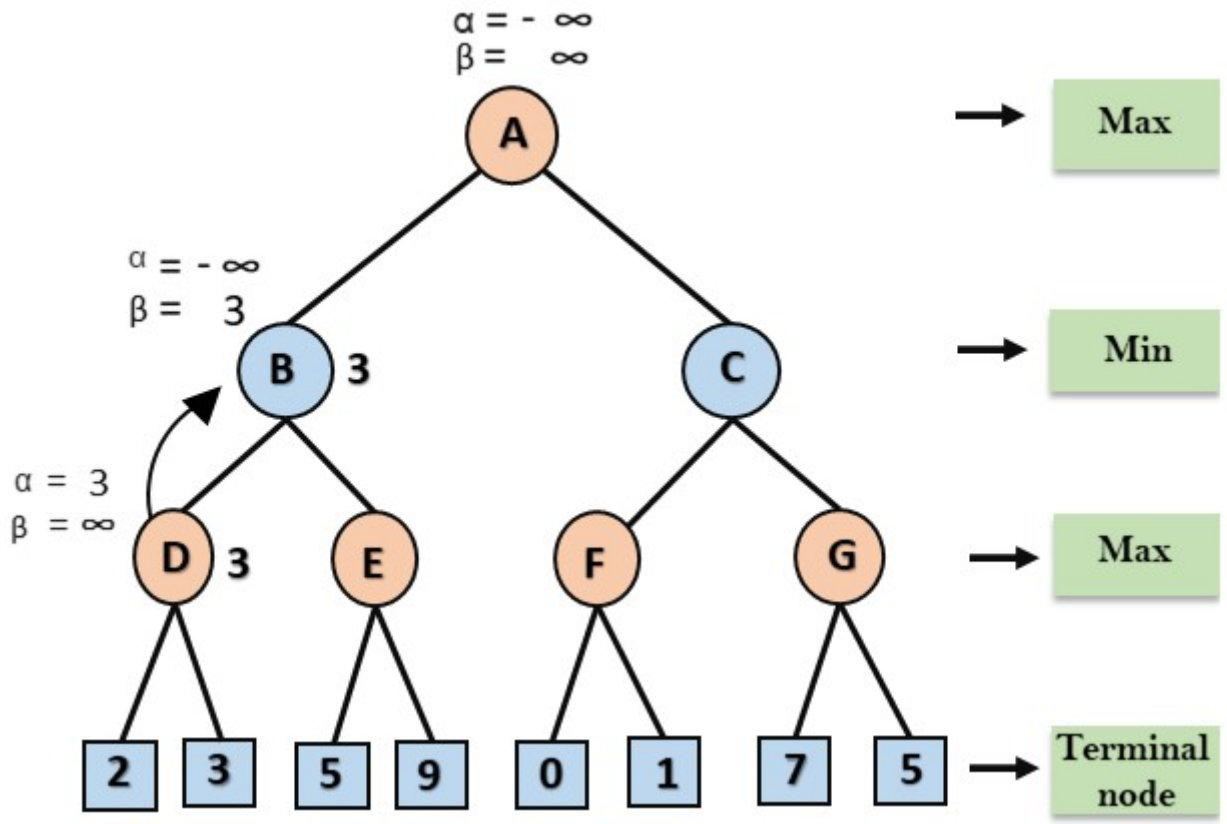
Key points about alpha-beta pruning:

The Max player will only update the value of alpha.
The Min player will only update the value of beta.
While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
We will only pass the alpha, beta values to the child nodes.

Alpha Beta Pruning

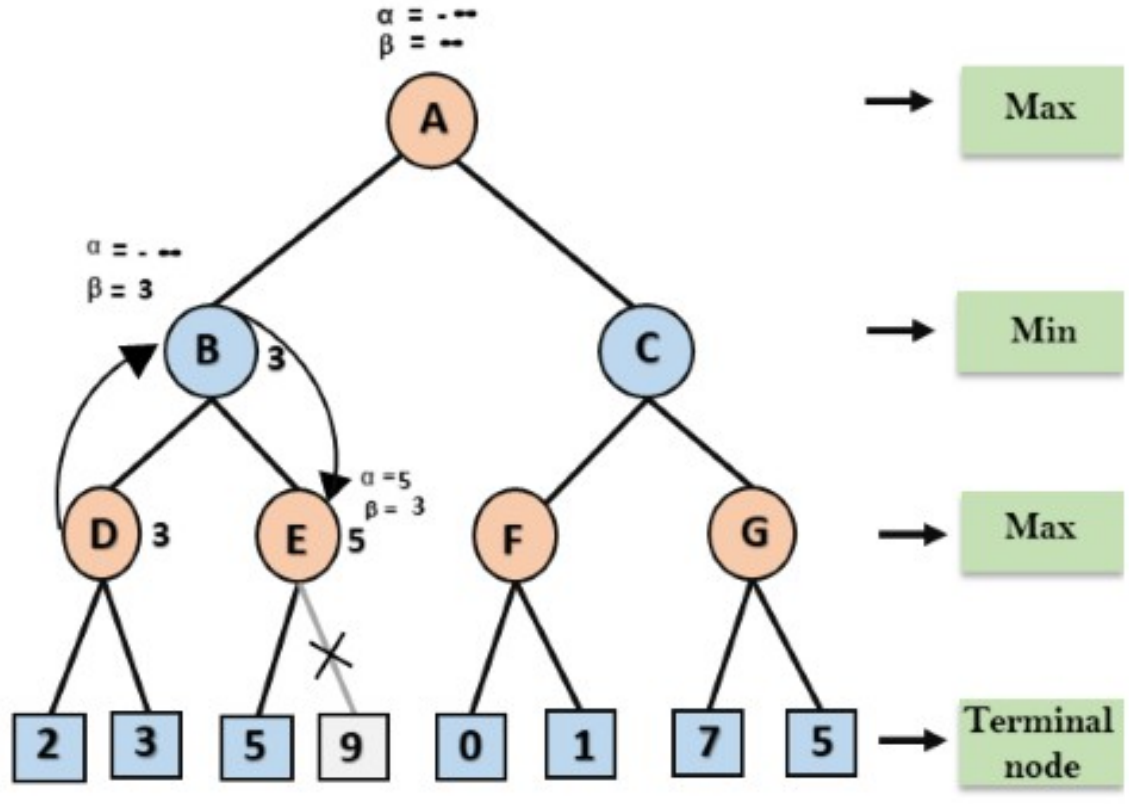
Ms.Rafath Zahra

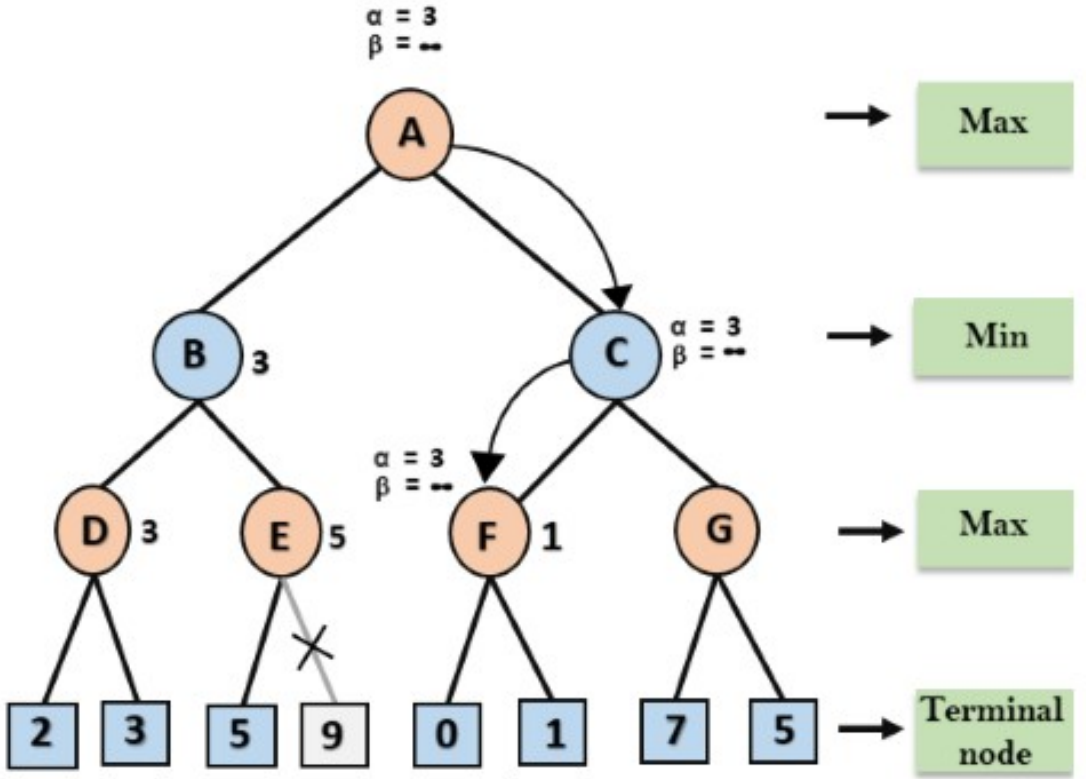




Alpha Beta Pruning

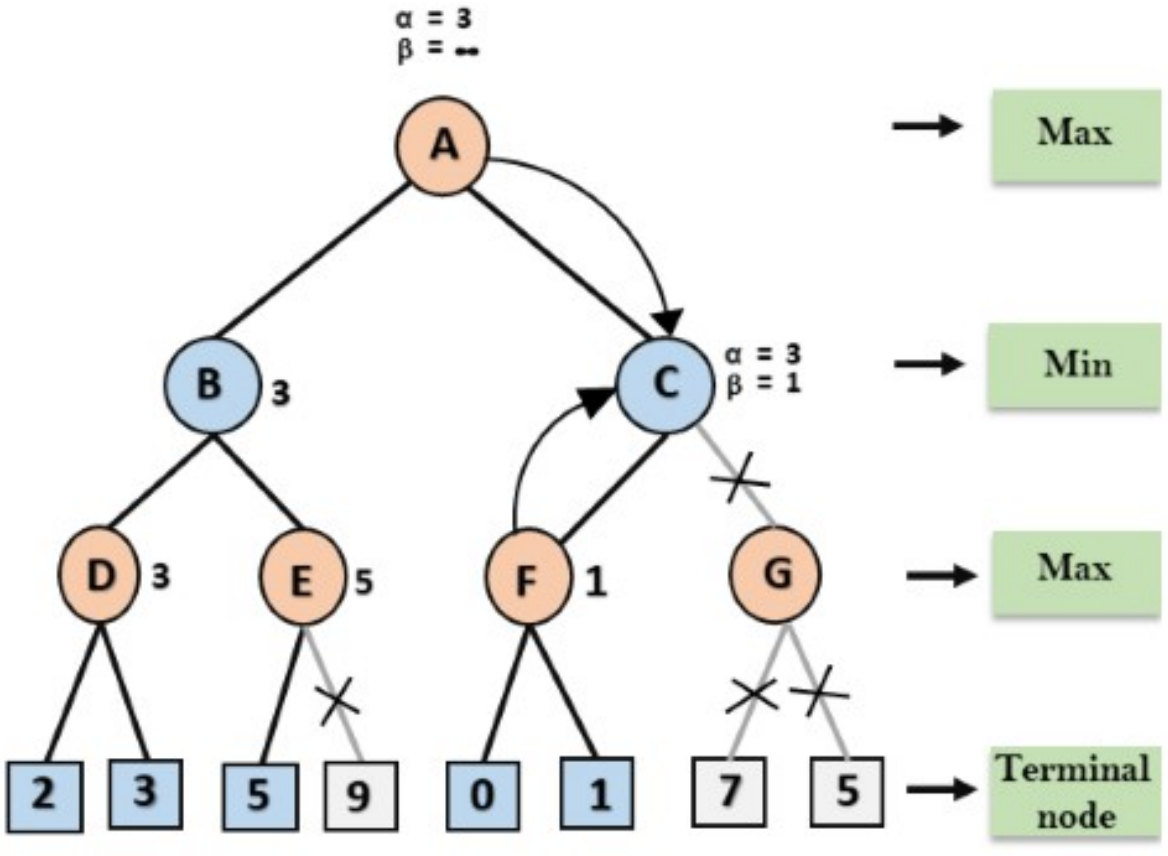
Ms.Rafath Zahra

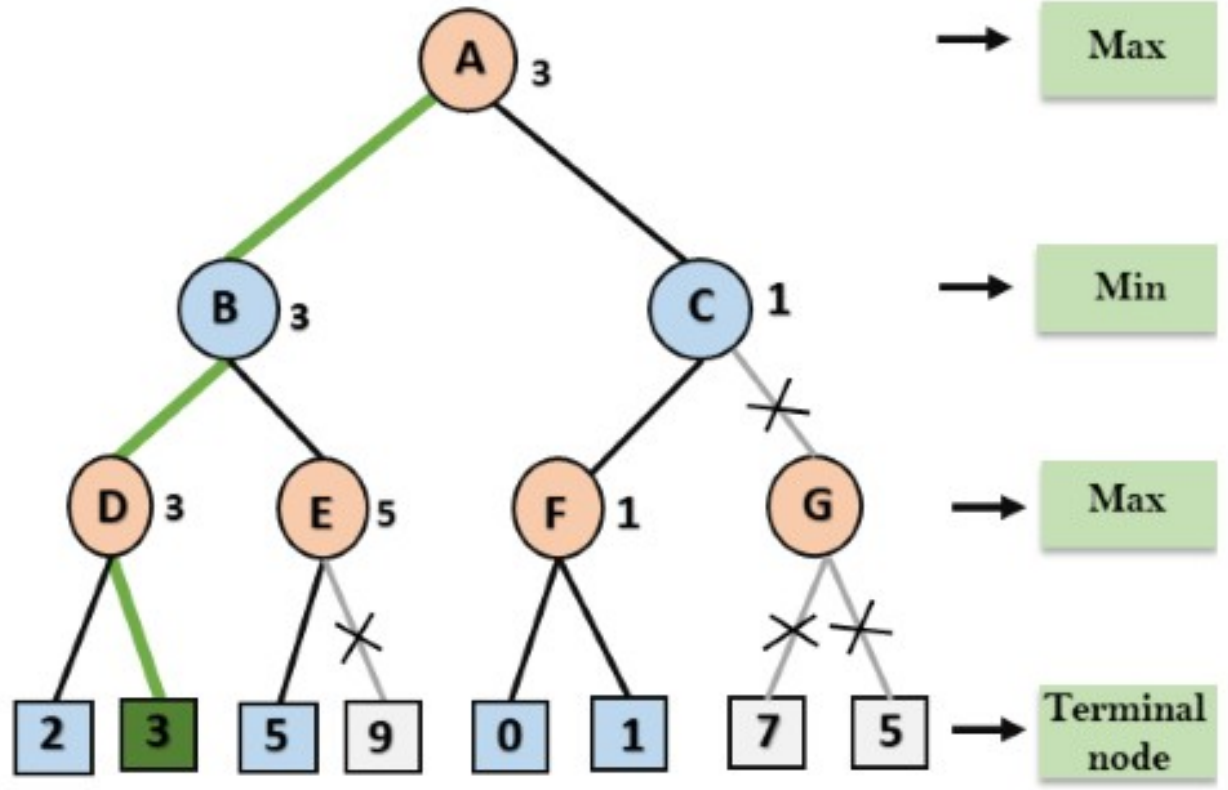




Alpha Beta Pruning

Ms.Rafath Zahra





As the algorithm progresses, it evaluates nodes and updates α and β. For instance, at Node D, Max updates α to 3. When backtracking to Node B, Min updates β to 3, and this value is passed down to Node E. At Node E, α is updated to 5, and since α ≥ β, further exploration of E's right child is pruned. The process continues, with similar pruning happening at Node C when α is 3 and β becomes 1. Ultimately, the algorithm efficiently finds the optimal move for Max, with a final value of 3, while skipping unnecessary evaluations.

Ms.Rafath Zahra

Evaluation Function

An evaluation function in a game-playing program estimates the likely outcome (win, loss, or draw) from a given position. It helps the program decide its next move without having to look ahead at every possible outcome, which saves time and computational resources. Here's a simpler breakdown:

Purpose of Evaluation Functions: Just like humans use intuition to judge their position in a game, evaluation functions help computer programs estimate the value of a position in a game like chess. This estimation is crucial because computers, despite their speed, can't always calculate every possible move to the end of the game.

Accuracy :

The effectiveness of a game-playing program heavily depends on the accuracy of its evaluation function.
If the function is inaccurate, the program might make poor decisions, leading to losses.

Ms.Rafath Zahra

Evaluation Function

Designing Good Evaluation Functions:

Order Terminal States Correctly: The evaluation function must rank endgame positions (like checkmate or stalemate in chess) in the same order as their actual outcome. This ensures that the program chooses moves leading to the best possible result.
Quick Computation: The function should quickly provide an estimate without extensive calculations, as taking too long would defeat its purpose.
Correlate with Winning Chances: For non-endgame positions, the evaluation should correlate well with the chances of winning.

Dealing with Uncertainty: In games without randomness (like chess), the evaluation function deals with uncertainty due to the computational limits. It can't always predict the exact outcome, so it makes an educated guess based on the position.

Using Features: Evaluation functions often rely on various "features" of the game state. For example, in chess:

Material Value: Assign values to pieces (pawns, knights, bishops, rooks, queens). For instance, a pawn might be worth 1 point, a knight or bishop 3 points, a rook 5 points, and a queen 9 points.
Other Features: Consider aspects like pawn structure and king safety.

Evaluation Function

Calculating Values: The evaluation function combines these feature values to estimate the position's overall value. For example, if a position has a material advantage of 3 points (e.g., an extra bishop and a pawn), it suggests a strong likelihood of winning.

Weighted Linear Functions: The function can be mathematically represented as a sum of feature values, each multiplied by a weight:

Example: EVAL(position) = (weight1 * feature1) + (weight2 * feature2) + ... + (weightN * featureN)
For chess, feature1 could be the number of pawns, feature2 the number of knights, and so on, with weights representing their respective values.

Knight = 2: 1 knight weight= 3
Eval Function(W*N)=(3*2)=6

Soldiers= 8(weight =1) and Bishops=2 ((weight =3)
Eval Func = (8*1)+(2*3)=14

Beyond Linear Combinations: Simple addition assumes each feature's contribution is independent of others, which isn't always true. For instance, bishops are more valuable in the endgame when there's more space to move.

Learning from Experience: The values and weights used in evaluation functions come from extensive human experience. In less-explored games, machine learning can help determine these weights by analyzing large amounts of