Parul® University

ARTIFICIAL INTELLIGENCE II

Prof. Mehulkumar Dalwadi

Assistant Professor

IT & Computer Science

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-2

Game Playing and Planning

What is Game Playing in AI ?

- Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.

- Both players try to win the game. So, both of them try to make the best move possible at each turn.

- So, we need another search procedures that improve –
  - Generate procedure so that only good moves are generated.
  - Test procedure so that the best move can be explored first.

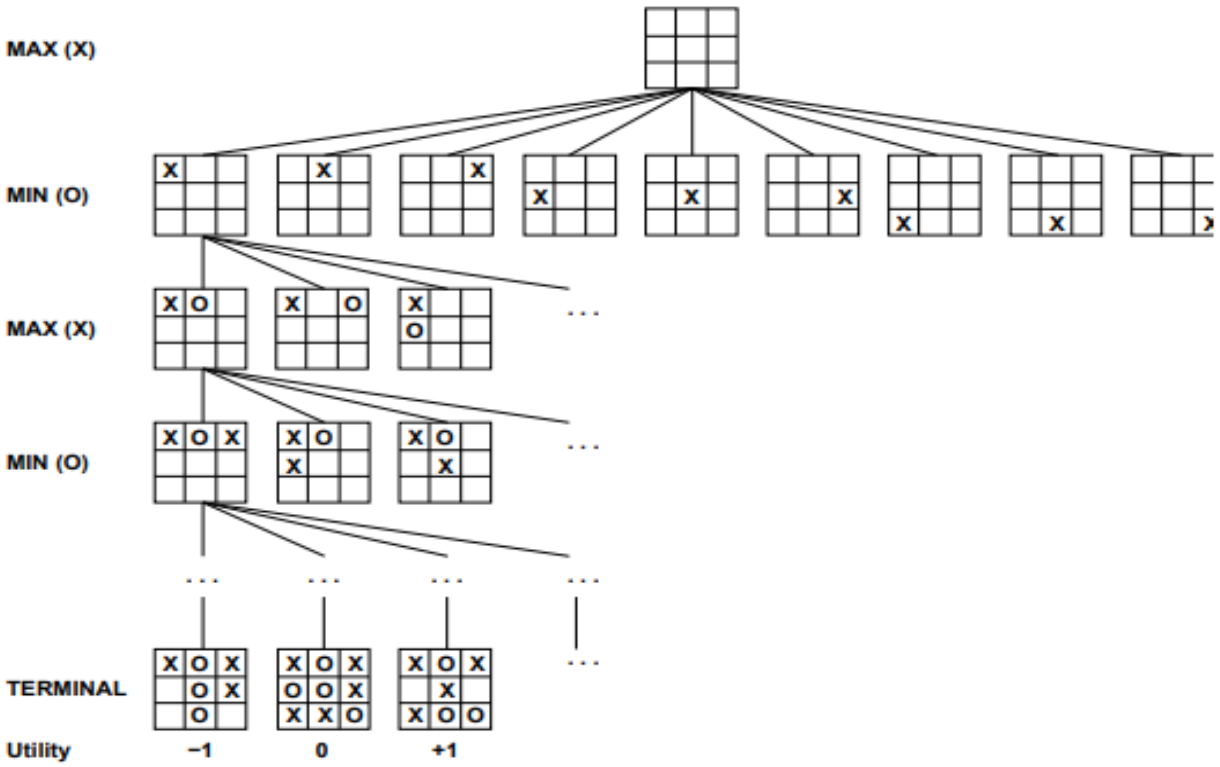
What is Game Playing in AI ?

- Games are well-defined problems that are generally interpreted as requiring intelligence to play well.
- Introduces uncertainty since opponents moves can not be determined in advance

Game Playing Problem

- Instance of the general search problem.
- States where the game has ended are called terminal states.
- A utility (payoff) function determines the value of terminal states, e.g. win=+1, draw=0, lose=-1.
- In two-player games, assume one is called MAX (tries to maximize utility) and one is called MIN (tries to minimize utility).
- In the search tree, first layer is move by MAX, next layer by MIN, and alternate to terminal states.
- Each layer in the search is called a ply

Sample Game Tree (Tic-Tac-Toe)



Parul®
University

DIGITAL LEARNING CONTENT

Mini-Max Algorithm in Artificial Intelligence

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.

Mini-Max Algorithm in Artificial Intelligence

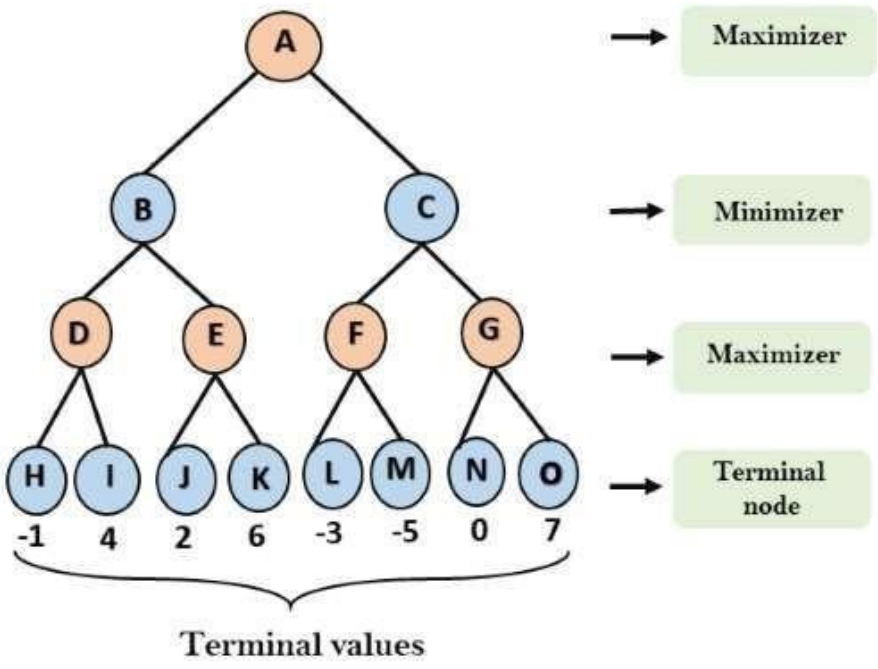
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Working of Min-Max Algorithm:

- We have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

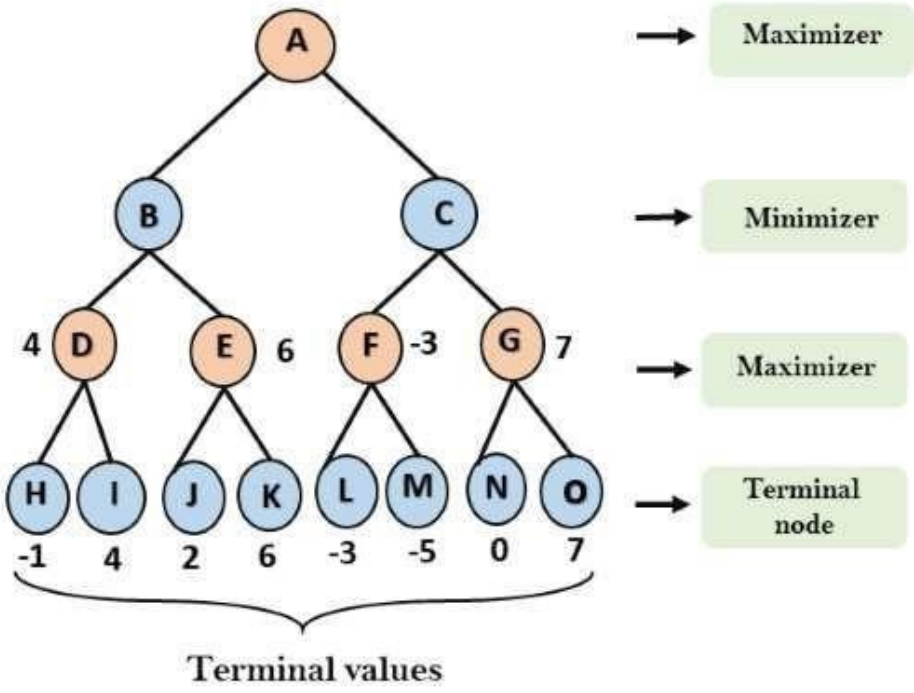
Working of Min-Max Algorithm:

- STEP 1 : In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states.
- Suppose maximizer takes first turn which has worst-case initial value =-infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



Working of Min-Max Algorithm:

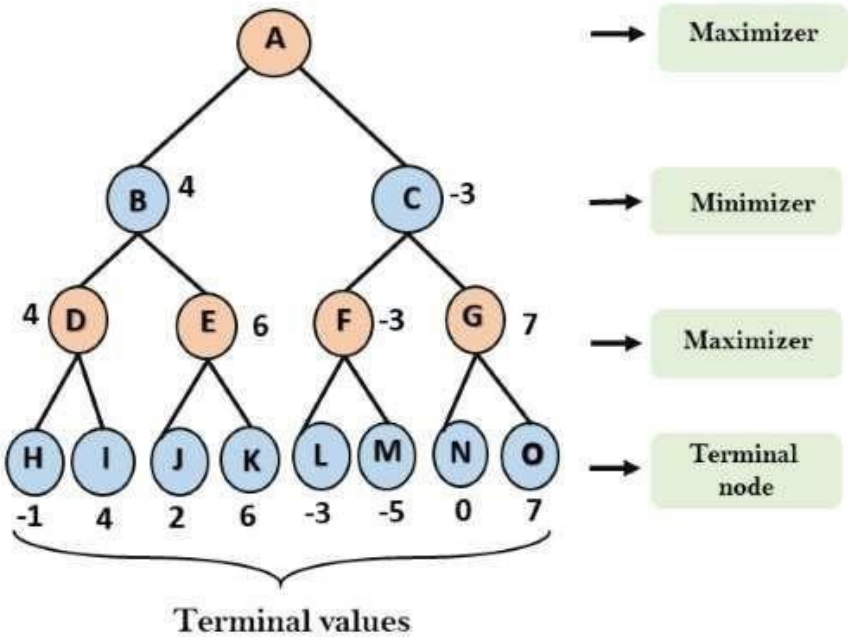
- Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.



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

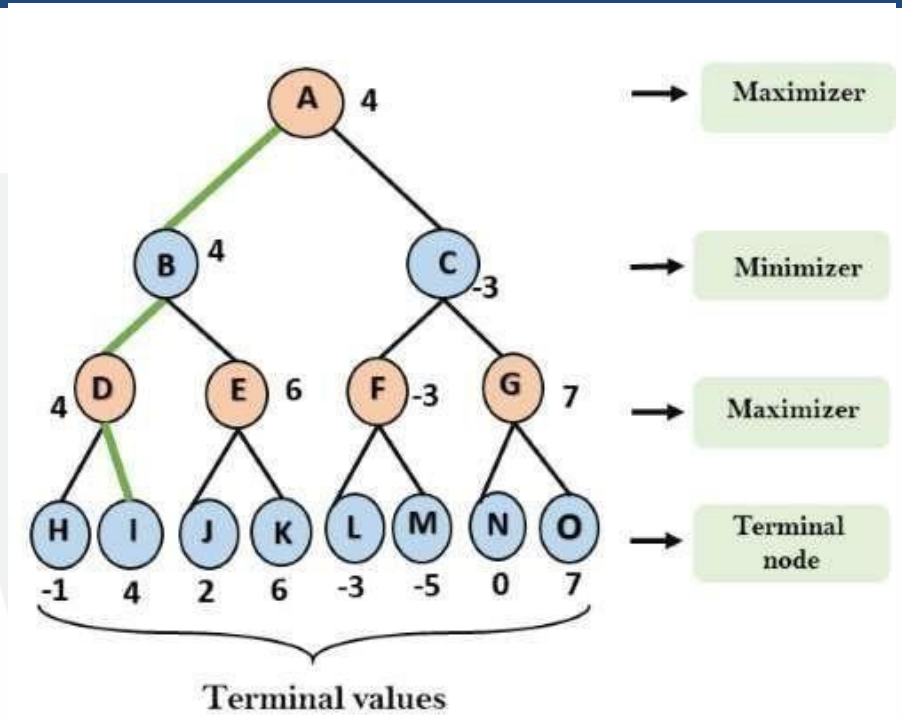
Working of Min-Max Algorithm:

- Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.
- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



Working of Min-Max Algorithm:

- 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.
- In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.
- For node A max(4, -3)= 4



This is the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **$O(b^m)$**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is O(bm).

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half.
- Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta 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.

Alpha-Beta Pruning

- 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 Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow.
- Hence by pruning these nodes, it makes the algorithm fast.

Alpha-Beta Pruning

- **Condition for Alpha-beta pruning:**

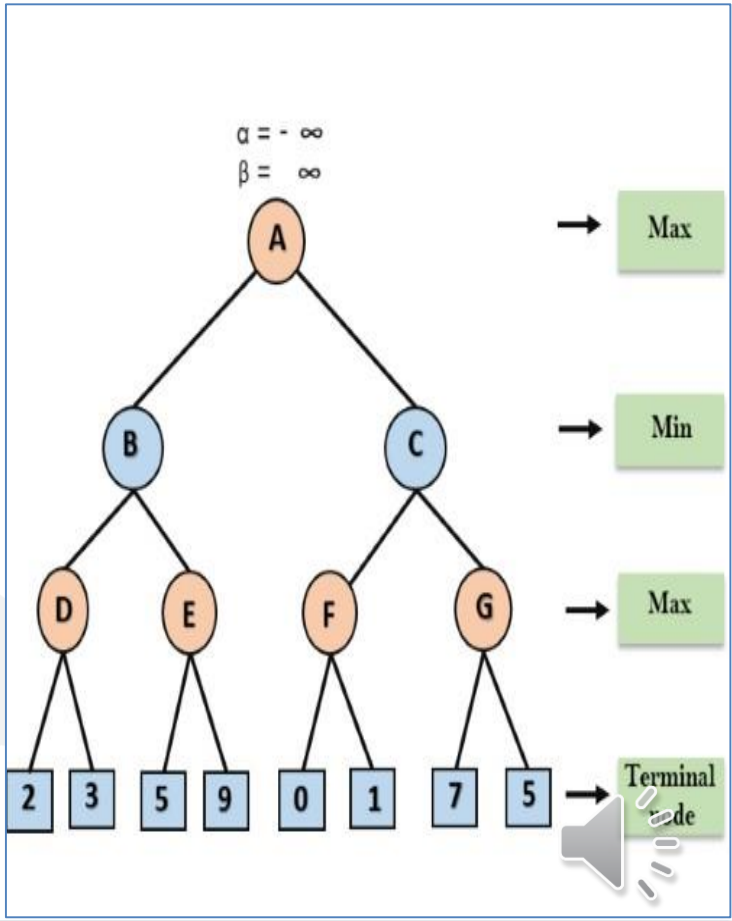
✓ $\alpha >= \beta$

- **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.

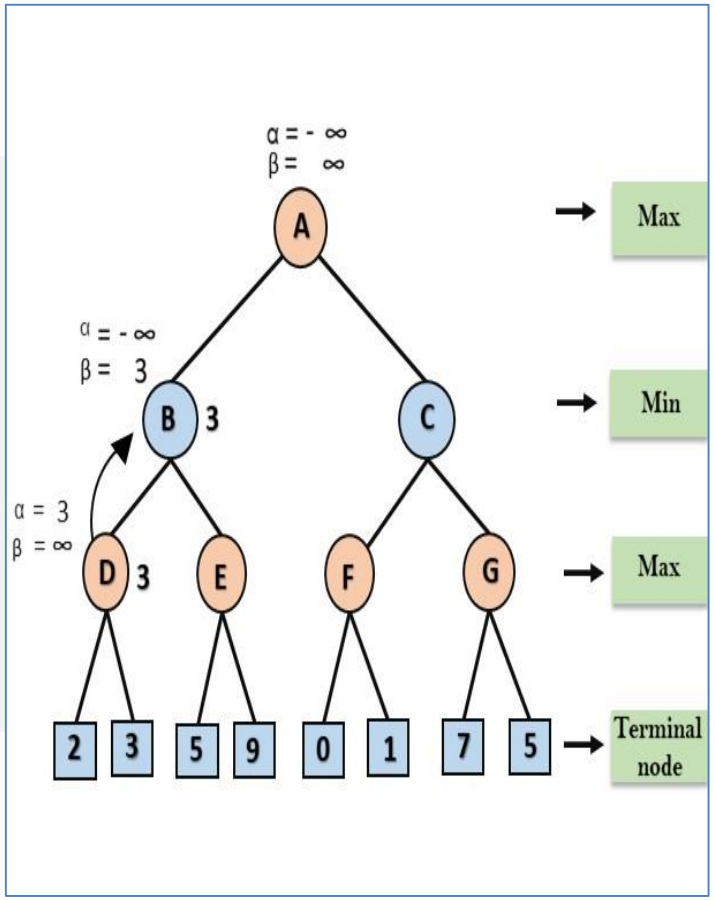
Working of Alpha-Beta Pruning:

- Let's take an example of two-player search tree to understand the working of Alpha-beta pruning
- **Step 1:** At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



Working of Alpha-Beta Pruning:

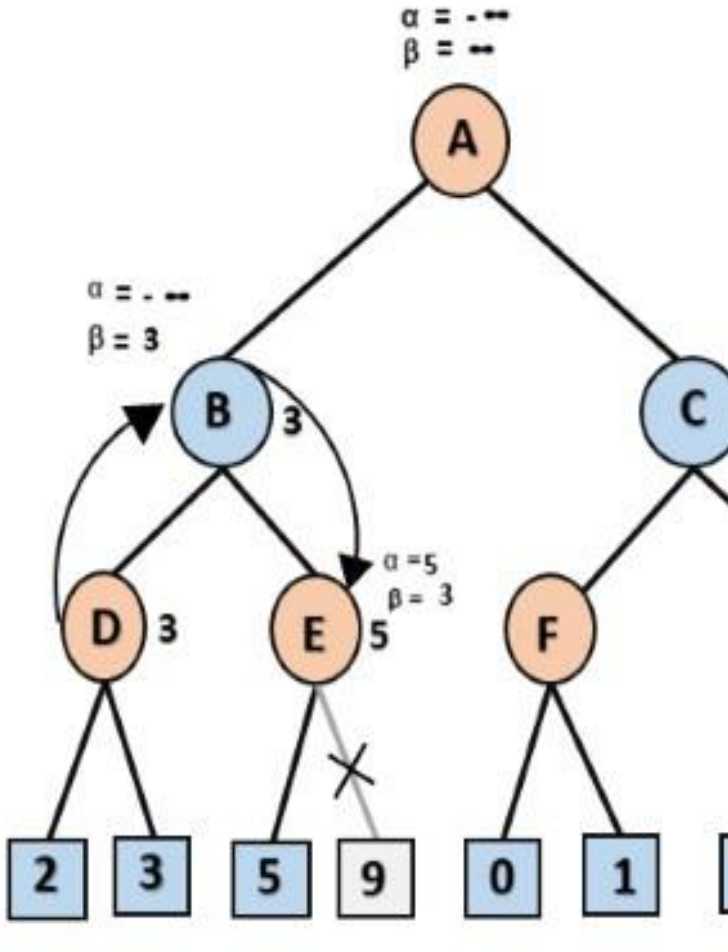
- **Step 2:** At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.
- **Step 3:** Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



Working of Alpha-Beta Pruning:

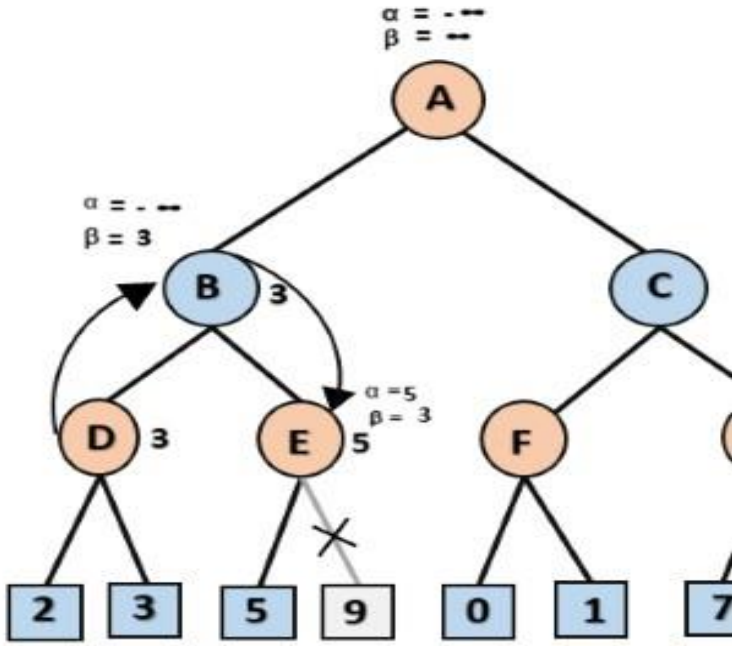
- In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

- **Step 4:** At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



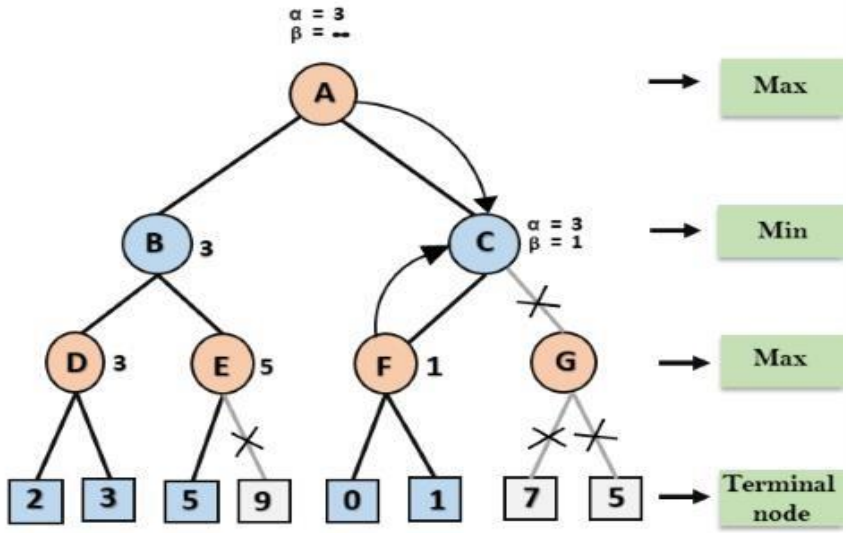
Working of Alpha-Beta Pruning:

- Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.
- At node C, α=3 and β= +∞, and the same values will be passed on to node F.
- Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



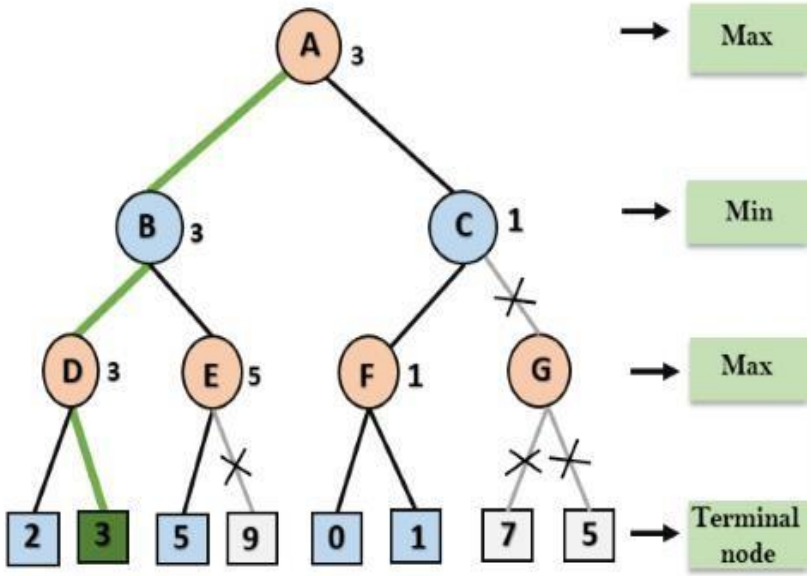
Working of Alpha-Beta Pruning:

- **Step 7:** Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Working of Alpha-Beta Pruning:

- **Step 8:** C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Planning

- The planning in Artificial Intelligence is about the decision making tasks performed by the robots or computer programs to achieve a specific goal.
- The execution of planning is about choosing a sequence of actions with a high likelihood to complete the specific task.

Components of Planning System

The planning consists of following important steps:

- Choose the best rule for applying the next rule based on the best available heuristics.
- Apply the chosen rule for computing the new problem state.
- Detect when a solution has been found.
- Detect dead ends so that they can be abandoned and the system's effort is directed in more fruitful directions.
- Detect when an almost correct solution has been found.

Blocks-World planning problem

- The blocks-world problem is known as Sussman Anomaly.
- Noninterleaved planners of the early 1970s were unable to solve this problem, hence it is considered as anomalous.
- When two subgoals G1 and G2 are given, a noninterleaved planner produces either a plan for G1 concatenated with a plan for G2, or vice-versa.
- In blocks-world problem, three blocks labeled as 'A', 'B', 'C' are allowed to rest on the flat surface. The given condition is that only one block can be moved at a time to achieve the goal.

Blocks-World planning problem

- The start state and goal state are shown in the following diagram.

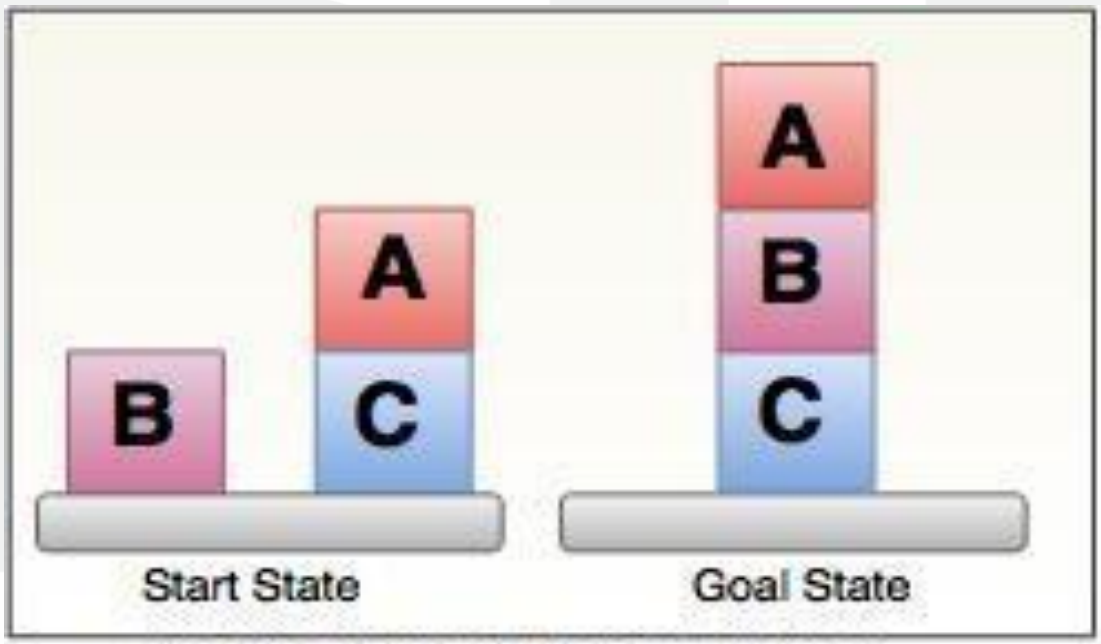
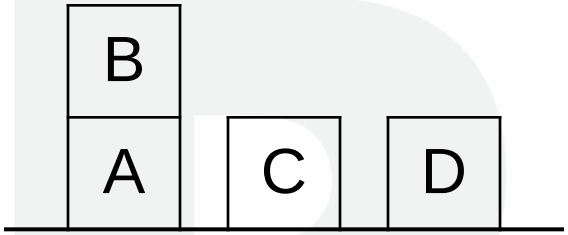


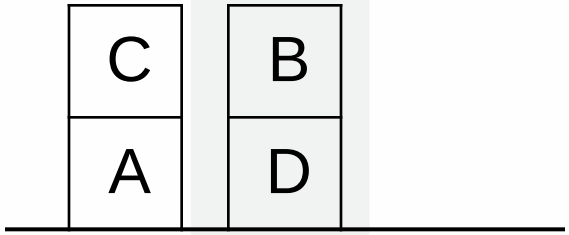
Fig: Blocks-World Planning Problem

Planning

The blocks world



start



goal

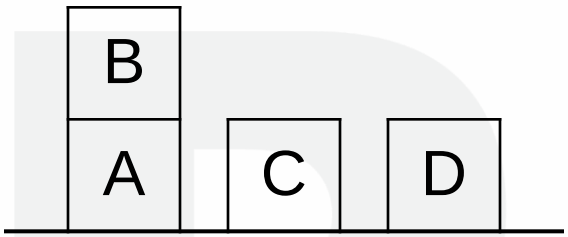
How to achieve the goal from the start?

- Problem-solving is searching and moving through a state space.

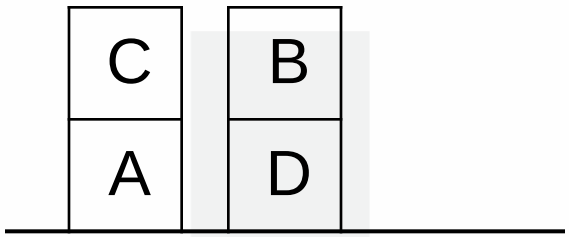
Planning

- Problem-solving is searching and moving through a state space.
- Planning is searching for successful paths through a state space.
- Planning is important if solutions cannot be undone.
- If the universe is not predictable, then a plan can fail
- ⇒ dynamic plan revision.

The Blocks World



start



goal

Planning = generating a sequence of actions to achieve the goal from the start

The Blocks World

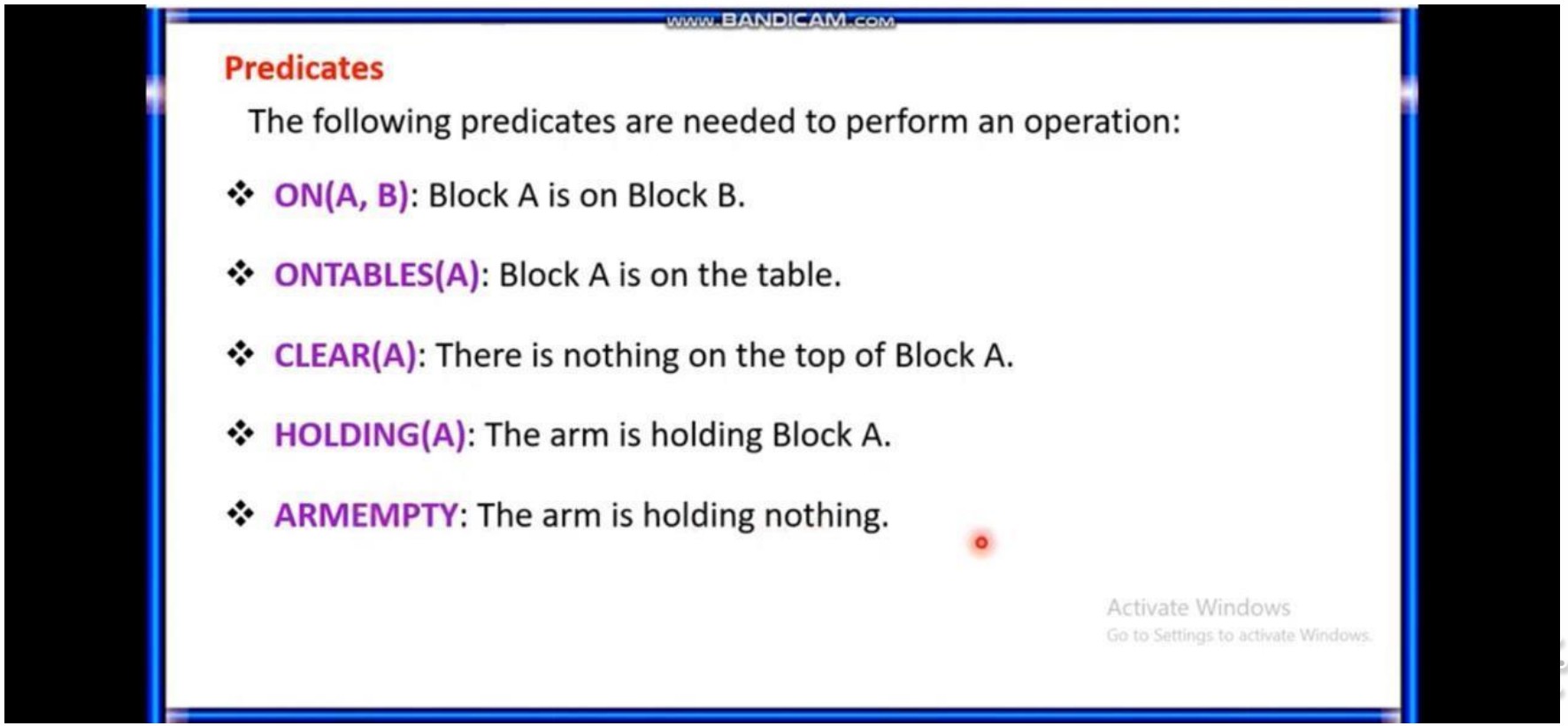
- **Actions:**

The actions it can perform include

- UNSTACK(A,B):
  - remove block A from block B.
  - The arm must be empty
  - block A must have no blocks on top of it.
- STACK(A,B):
  - put block A on block B.
  - The arm must already be holding A
  - the surface of B must be clear.
- PICKUP(A):
  - pickup block A from the table.
  - The arm must be empty and there must be nothing on top of A.
- PUTDOWN(A):
  - put block A on the table.
  - The arm must have been holding block A.

The Blocks World

- **Conditions and results:**



The Blocks World

- **Specification of actions:**

- **PRECONDITION**: list of predicates that must be true for an operator to be applied.

- **ADD**: list of new predicates that an operator causes to become true.

- **DELETE**: list of old predicates that an operator causes to become false.
- Predicates not in ADD nor DELETE are unaffacted.

The Blocks World

Specification of actions:

STACK(x, y):

P: CLEAR(y) ∧ HOLDING(x)

A: ARMEMPTY ∧ ON(x, y) ^ CLEAR(x)

UNSTACK(x, y):

P: ON(x, y) ∧ CLEAR(x) ∧ ARMEMPTY

A: HOLDING(x) ∧ CLEAR(y)

The Blocks World

Specification of actions:

PICKUP(x):

P: CLEAR(x) ∧ ONTABLE(x) ∧ ARMEMPTY

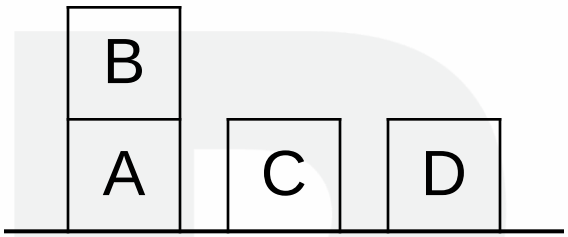
A: HOLDING(x)

PUTDOWN(x):

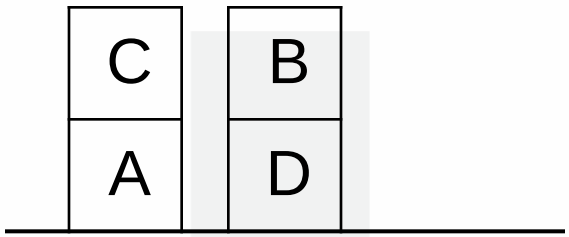
P: HOLDING(x)

A: ONTABLE(x) ∧ ARMEMPTY

The Blocks World



start: ON(B, A) ∧
ONTABLE(A) ∧
ONTABLE(C) ∧
ONTABLE(D) ∧
ARMEMPTY



goal: ON(C, A) ∧
ON(B, D) ∧
ONTABLE(A) ∧
ONTABLE(D) ∧

Goal Stack Planning

Stack		Database
Goals		Current situation
Operators to satisfy the Goals	+	Specification of Operators/Actions

Goal Stack Planning

Push the original goal to the stack. Repeat until the stack is empty:

- If stack top is a compound goal, push its unsatisfied subgoals to the stack.
- If stack top is a single unsatisfied goal, replace it by an operator that makes it satisfied and push the operator's precondition to the stack.
- If stack top is an operator, pop it from the stack, execute it and change the database by the operation's affects.
- If stack top is a satisfied goal, pop it from the stack.

Goal Stack Planning

Push the original goal to the stack. Repeat until the stack is empty:

- If stack top is a compound goal, push its unsatisfied subgoals to the stack.
- If stack top is a single unsatisfied goal, replace it by an operator that makes it satisfied and push the operator's precondition to the stack.
- If stack top is an operator, pop it from the stack, execute it and change the database by the operation's affects.
- If stack top is a satisfied goal, pop it from the stack.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in