UNIT 2 PROBLEM SOLVING USING SEARCH

Structure **Page No**

2.0 INTRODUCTION

Many AI-based applications need to figure out how to solve problems. In the world, there are two types of problems. First, the problem which can be solved by using deterministic procedure and the success is guaranteed. But most real-world problems can be solved only by searching a solution. AI is concerned with these second types of problems solving.

To build a system to solve a problem, we need to

- ***Define the problem precisely***-find initial and final configuration for acceptable solution to the problem.
- ***Analyse the problem***-find few important features that may have impact on the appropriateness of various possible techniques for solving the problem

- ***Isolate and represent*** *task* knowledge necessary to solve the problem
- ***Choose the best problem-solving technique***(s) and apply it to the particular problem.

To provide a formal description of a problem, we need to do the following:

a. Define a **state space** that contains all the possible configurations of the relevant objects.

b. Specify one or more states that describe possible situations, from which the problem-solving process may start. These states are called ***initial states***.

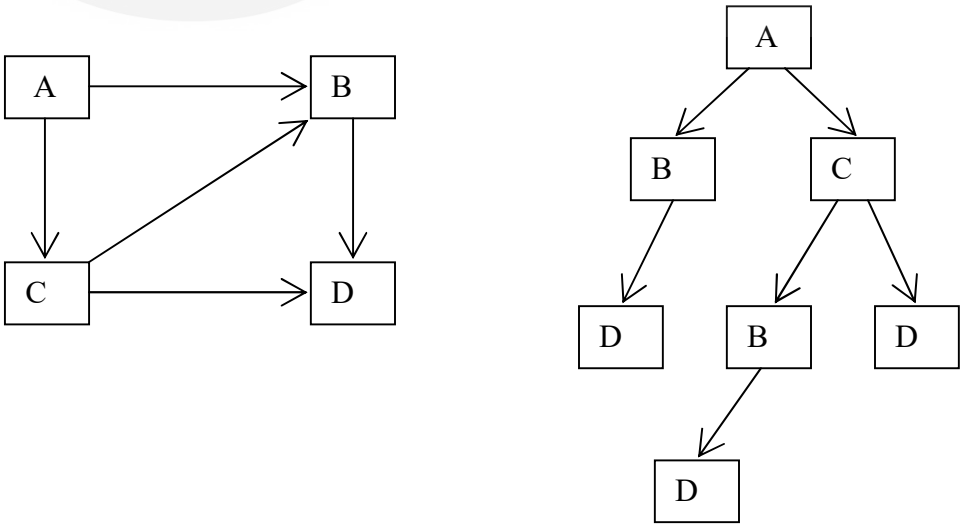
c. Specify one or more than one ***goal states***.

d. Defining a *set of rules* for the actions (operators) that can be taken.

The problem can then be solved by using the rules, in combination with an appropriate ***control strategy***, to move through the problem space until a path from an ***initial state*** to a ***goal state*** is found. This process is known as ***'search***'. Thus, search is fundamental to the problem-solving process. Search is a general mechanism that can be used when a more direct method is not known. Search provides the framework into which more direct methods for solving subparts of a problem can be embedded. **All Al problems are formulated as search problems**.

A **problem space** is represented by a directed graph, where *nodes* represent *search state* and *paths* represent the *operators* applied to change the state. To simplify search algorithms, it is often convenient to logically and programmatically represent a problem space as a ***tree***. A tree usually decreases the complexity of a search at a cost. Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph, e.g., node B and node D shown in example below.

Graph **Tree**



A **tree** is a graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph with no cycles is a tree.

Before an AI problem can be solved it must be represented as a **state space**. Here **state** means representation of elements at a given moment. Among all possible states, there are two special states called ***initial state*** (the start point) and ***final state*** (the goal state). A ***successor function (a set of operators)***is used to change the state. It is used to move from one state to another. A state space is the set of all states reachable from the initial state. A ***state space*** essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state, and a goal state. Each state space takes the form of a tree or a graph. In AI, a wide range of problems can be formulated as search problem. The process of searching means a sequence of action that take you from an initial state to a goal state as sown in the following figure 1.

Fig 1: A sequence of action in a search space from initial to goal state.

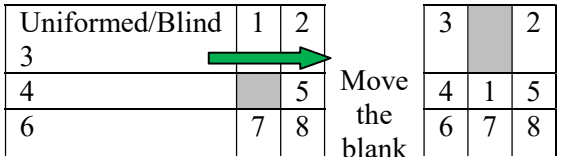
So, State space is the one of the methods to represent the problem in AI. A set of all possible states reachable from the initial state by taking some sequence of action (using some operator) for a given problem is known as the **state space** of the problem. A state space represents a problem in terms of ***states*** and ***operator***s that change states".

In this unit we examine the concept of a **state space** and the different **search process** that can be used to explore the search space in order to find a solution (Goal) state. In the worst case, search explores all possible paths between the *initial state* and the *goal state*.

For better understanding of these definitions describe above, consider the following 8-puzzle problem:

Eight-Puzzle problem Formulation from AI perspectives

initial state: some configuration of the 8-tiles on a 9-cell board.
operators (Action): it's easier if we focus on the blank. There are 4 operators that is, "Moving the blank" : UP, DOWN, LEFT and RIGHT.



Goal state: Tiles in a specific order

	1	2
3	4	5
6	7	8

Solution: Optimal sequence of operators (Actions)
Path costs:
cost of each action = 1
cost of a sequence of actions= the number of actions

A state space representation of 8-puzzle problem is shown in figure-2

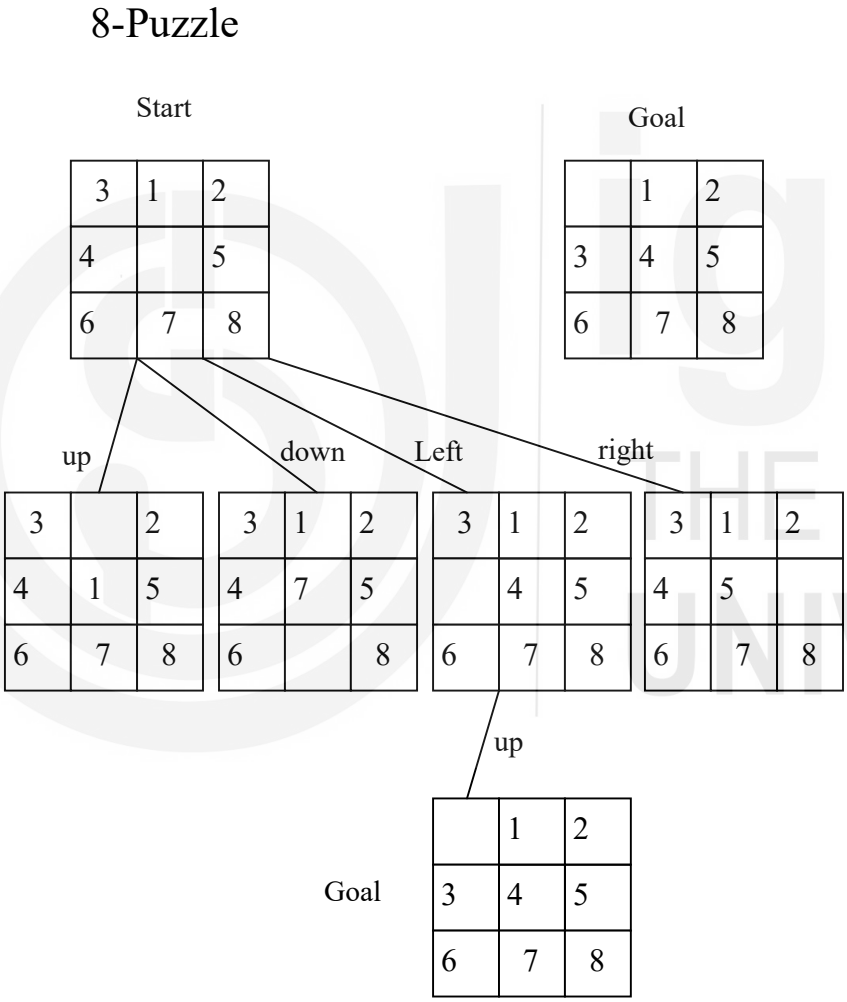


Fig2: A state space representation of 8-puzzle problem generated by "**Move Blank**" operator

2.1 OBJECTIVES

After studying this unit, you should be able to:

- Understand the state space search
- Formulate the problems in the form of state space
- Understand how implicit state space can be unfolded during search
- Explain State space search representation for Water-Jug, 8-puzzle and N-Queen's problem.
- Solve N Queen's Problem using Backtracking approach
- Understand adversarial search (two agent search)
- Differentiate between Minimax and Alpha-beta pruning search algorithm.

2.2 Introduction to State Space Search

It is necessary to represent an AI problem in the form of a state space before it can be solved. A state space is the set of all states reachable from the initial state. A state space forms a graph in which the nodes are states and the arcs between nodes are actions. In state space, a path is a sequence of states connected by a sequence of actions.

2.2.1 Structure of a state space:

The structures of state space are trees and graphs. A tree has one and only one path from any point to any other point. Graph consists of a set of nodes (vertices) and a set of edges (arcs). Arcs establish relationship (connections) between the nodes, i.e., a graph has several paths to a given node. **Operators** are directed arcs between nodes.
The method of solving problem through AI involves the process of defining the search space, deciding start and goal states and then finding the path from start state to goal state through search space.
Search process explores the state space. In the worst case, the search explores all possible paths between the initial state and the goal state.

2.2.2 Problem Solution:

In a state space, a ***solution is a path*** from the initial state to a goal state or sometime just a goal state. A numeric cost is assigned to each path. It also gives the cost of applying the operators to the states. A path cost function is used to measure the quality of solution and out of all possible solutions, an optimal solution has the lowest path cost. The importance of cost depends on the problem and the type of solution asked.

2.2.3 Problem formulation:

Many problems can be represented as ***state space***. The state space of a problem includes: an *initial state*, one or more *goal state*, set of *state transition operator* (or a ***set of production rules***), used to change the current state to another state. This is also known as ***actions.*** A ***control strategy*** is used that specifies the order in which the rules will be applied. For example, Depth-first search (DFS), Breath-first search (BFS) etc. It helps to find the goal state or a path to the goal state.

In general, a state space is represented by 4 tuples as follows:$\boldsymbol{S_s}$: $[\boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{O}, \boldsymbol{G}]$

Where **S**: Set of all possible states.
$\boldsymbol{s_0}$**:** start state (initial configuration) of the problem, $s_0 \in S$.
O: Set of production rules (or set of state transition operator) used to change the state from one state to another. It is the set of arcs (or links) between nodes.

The ***production rule*** is represented in the form of a pair. Each pair consists of a left side that determines the applicability of the rule and a right side that describes the action to be performed, if the rule is applied.

G: Set of Goal state, $G \in S$.

The sequence of actions (or operators) is called a solution path. It is a path from the initial state to a goal state. This sequence of actions leads to a number of states, starting from initial state to a goal state, as $\{s_0, s_1, s_2, \ldots \ldots, s_n \in G\}$.A sequence of state is called a path. The cost of a path is a positive number. In most of the cases the path cost is computed as the sum of the costs of each action.

The following figure 3 shows a search process in a given state space.

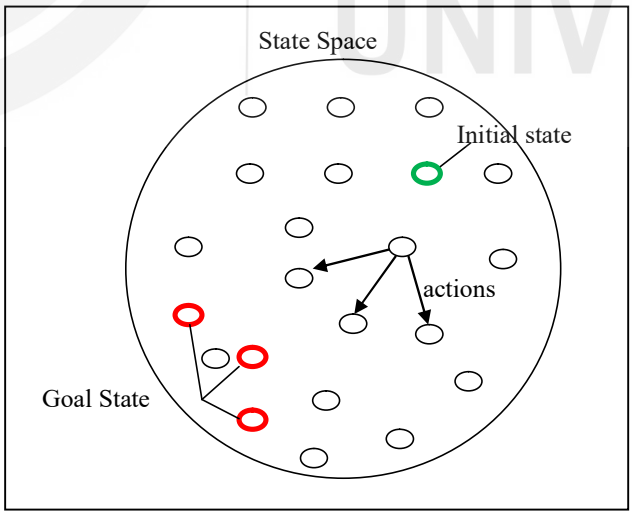


Fig 3 State space with initial and goal node

We need to identify a sequence of actions that will turn the initial state s_0 into the desired goal state G. State space is commonly defined as a directed graph or as a tree in which each node is a state and each arc represents the application of an operator transforming a state to a successor state.

2.2.4 SEARCHING FOR SOLUTIONS

After formulating our problem, we are ready to solve it, this can be done by searching through the state space for a solution, this search will be applied On a search tree or generally a graph that is generated using the initial state and the successor function.

Searching is applied to a search tree which is generated through state expansion, that is applying the successor function to the current state, note that here we mean by state a node in the search tree.

Generally, search is about selecting an option and putting the others aside for later in case the first option does not lead to a solution. The choice of which option to expand first is determined by the search strategy used.

Thus, the problem is solved by using the rules (operators), in combination with an appropriate control strategy, to move through the problem space until a path from **initial state to a goal state** is found. This process is known as **search**. A solution path is a path in state space from s_0(initial sate) to G (Goal state).

Example1: State space representation of Water-Jug problem (WJP):

Problem statement:

Given two jugs, a 4-gallon and a 3-gallon, both of which do not have measuring indicators on them. The jugs can be filled with water with the help of a pump that is available (as shown in figure 4)

The question is "how can you get exactly 2 gallons of water into 4-gallon jug".

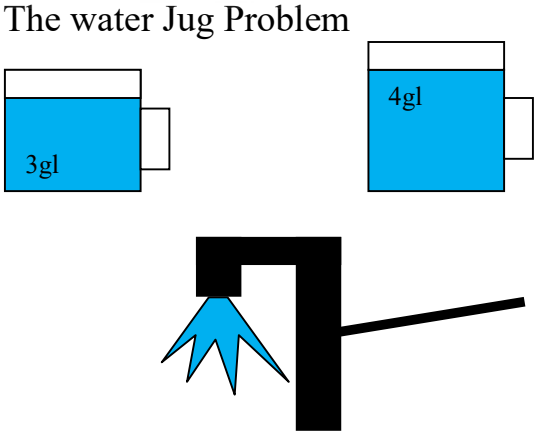


Fig 4 The Water Jug problem

The state space or production rule of this problem can be defined as a collection of ordered pairs of integers (x,y) where x=0,1,2,3,4and y= 0,1,2,or 3.
In the order pair (x,y), x is the amount of water in four-gallon jug and y is the amount of water in the three-gallon jug.
State space: all possible combination of (x,y)
The **start state** is (0,0) and
Goalstate is $(2, n), where\ n = 0,1,2,3$
The following table-1 shows the set of ***production rules*** (actions) that can be used to change one state to another.
Table-1 : Production rules for Water Jug problem

Production Rules		
Rule No	**Production**	**Meaning**
R1	$(x, y \mid x < 4) \rightarrow (4, y)$	Fill 4-gallon jug
R2	$(x, y \mid y < 3) \rightarrow (x, 3)$	Fill 3-gallon jug
R3	$(x, y \mid x > 0) \rightarrow (0, y)$	Empty 4-gallon jug
R4	$(x, y \mid y > 0) \rightarrow (x, 0)$	Empty 3-gallon jug
R5	$(x, y \mid x + y \geq 4\ and\ y > 0) \rightarrow (4, y - (4 - x))$	Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full
R6	$(x, y \mid x + y \geq 3\ and\ x > 0) \rightarrow (x - (3 - y), 3)$	Pour water from 4-gallon jug into 3-gallon jug until 3-gallon jug is full
R7	$(x, y \mid x + y \leq 4\ and\ y > 0) \rightarrow (x + y, 0)$	Pour all water from 3-gallon jug into 4-gallon jug.
R8	$(x, y \mid x + y \leq 3\ and\ x > 0) \rightarrow (0, x + y)$	Pour all water from 4-gallon jug into 3-gallon jug.
R9	$(x, y \mid x > 0) \rightarrow (x - d, y)$	Pure some water d out from 4-gallon jug
R10	$(x, y \mid y > 0) \rightarrow (x, y - d)$	Pure some water d out from 3-gallon jug

The following 2 solutions are found for the problem "how can you get exactly 2 gallons of water into 4-gallon jug", as shown in Table-2 and in Table-3.

Solution-1:
Table-2Getting exactly 2 gallons of water into 4-gallon jug (solution1)

4-gal jug	**3-gal jug**	**Rule applied**
0	0	Initial state
4	0	R1 {fill 4-gal jug}
1	3	R6 {Pure all water from 4-gal jug to 3-gal jug
1	0	R4 {empty 3-gal jug}
0	1	R8 {Pour water from 4 to 3-gal jug}
4	1	R1 {Fill 4-gal jug}
2	3	R6 {Pour water from 4-gal jug to 3-gal jug until it is full}
2	0	R4 {empty 3-gal jug}

Solution-2

Table-3Getting exactly 2 gallons of water into 4-gallon jug (solution2)

4-gal jug	3-gal jug	Rule applied
0	0	Initial state
0	3	R2 {fill 3-gal jug}
3	0	R7 {Pure all water from 3-gal jug to 4-gal jug
3	3	R2 {fill 3-gal jug}
4	2	R5 {Pour from 3 to 4-gal jug until it is full}
0	2	R3 {Empty 4-gal jug}
2	0	R7 {Pour all water from 3-gal jug to 4-gal jug}

A state space tree for WJP with all possible solution is shown in figure 7.

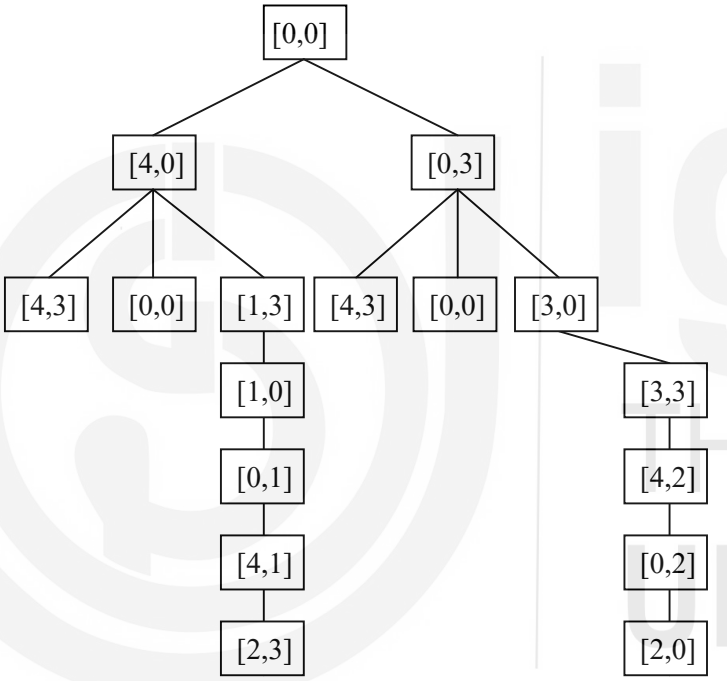


Fig 5 all possible Solution of WJP using state space tree

2.3 Formulation of 8 Puzzle problem

The eight-tile puzzle consists of a 3-by-3 (3×3) square frame board which holds eight (8) movable tiles numbered as 1 to 8. One square is empty, allowing the adjacent tiles to be shifted. The objective of the puzzle is to find a sequence of tile movements that leads from a starting configuration to a goal configuration

The Eight Puzzle Problem formulation:

Given a 3×3 grid with 8 sliding tiles and one "blank"
Initial state: some other configuration of the tiles, for example

3	1	2
4		5
6	7	8

Goal state:

	1	2
3	4	5
6	7	8

Operator: Slide tiles (Move Blank) to reach the goal (as shown below). There are 4 operators that is, "Moving the blank":
Move the blank UP,
Move the blank DOWN,
Move the blank LEFT and
Move the blank RIGHT.

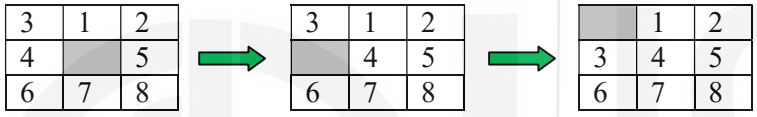


Fig6 Moving Blank LEFT and then UP

Path Cost: Sum of the cost of each path from initial state to goal state. Here cost of each action (blank move) = 1, so cost of a sequence of actions= the number of actions. A optimal solution is one which has a lowest cost path.

Performing State-Space Search: Basic idea:

If the initial state is a goal state, return it.
If not, apply the operators to generate all states that are one step from the initial state (its successors)

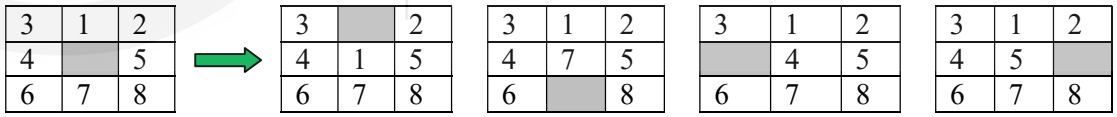


Fig 7 All possible successors for a given initial state

Consider the successor (and their successors…) until you find a goal state.
Different search strategies consider the state in different orders. They may use different data structures to store the states that have yet to be considered.

State-Space Search Tree for 8-Puzzle problem:

The predecessor reference connects the search nodes, creating a data structure known as a tree.

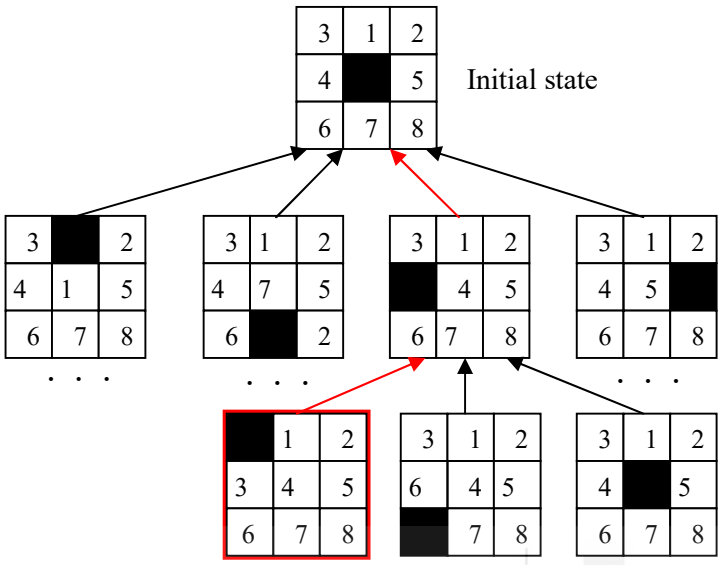


Fig 8 Tracing a tree bottom-up form Goal state to initial sate

When we reach a goal, we trace up the tree to get the solution i.e., the sequence of actions from the initial state to the goal.

Q.1 Find the minimum cost path for the 8-puzzle problem, where the start and goal state are given as follows:

Initial State: {(1,2,3),(4,8,-),(7,6,5)}

Successor State: {(1,2,3),(4,8,5),(7,6,-)}; Move 5 up or – to down

{(1,2,3),(4,8,5),(7,-,6)}; Move 6 right or – to left

{(1,2,3),(4,-,5),(7,8,6)}; Move 8 down or – to up

{(1,2,3),(4,5,-),(7,8,6)}; Move 5 to left or – to right

Goal State: {(1,2,3),(4,5,6),(7,8,-)}; Move 6 to up or – to down

PATH COST=5

2.4 N Queen's Problem-Formulation and Solution

The N-Queen problem is the problem of placing N Queen's (Q1,Q2,Q3,….Qn) on an $N \times N$ chessboard so that no two queens attack each other. The colour of the queens is meaningless in this puzzle, and any queen is assumed to be attack any other. So, a solution requires that no two queens share the same row, column, or diagonal.

The N-queen problem must follow the following rules:

1. There is at most one queen in each column.
2. There is at most one queen in each row.
3. There is at most one queen in each diagonal.

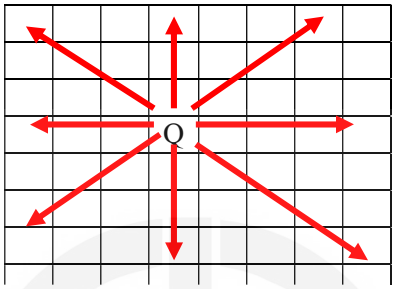


Fig-9 No two queens placed on same row, column or diagonal

The N Queen's problem was originally proposed in 1848 by the chess player Max Bazzel, and over the years, many mathematicians, including Gauss have worked on this puzzle. In 1874, S. Gunther proposed a method of finding solutions by using determinants, and J.W.L. Glaisher refined this approach.

The solutions that differ only by summary operations (rotations and reflections) of the board are counted as one. For 4 queen's problems, there are 16_{C_4} possible arrangements on a 4×4 chessboard and there are only 2 possible solutions for 4 Queen's problem. Note that,there are only 1 unique solution, out of 2 possible solutions as second solution is just a mirror image of the first solution

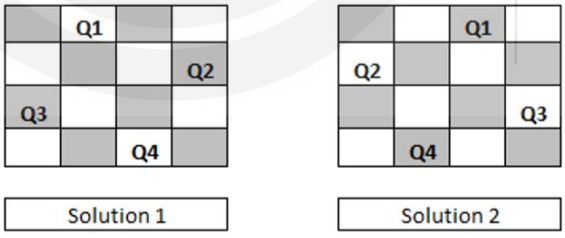


Fig 10 Two possible solutions of 4-Queen's problem

Similarly, the one possible solution for 8-queen's problem is shown in figure 11.

The 8-queen problem is computationally very expensive since the total number of possible arrangements of queen on a 8×8 chessboard is $64_{C_8} = 64!/(56! \text{ x } 8!) \approx 4.4 \times 10^9$. Note that, 8-Queens problem has 92 **distinct** solutions and 12 **unique** solutions, as shown in table-5

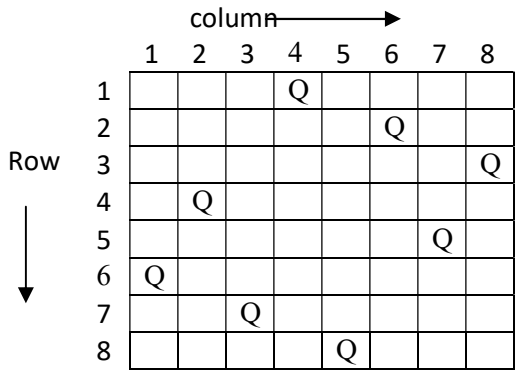


Fig 11 One possible solution of 8 Queen's problem

8-tuple = (4, 6, 8, 2, 7, 1, 3, 5)

The following table-4 summarizes the both distinct and unique solution for the problem of 1-Queen to 26 Queens problem. In general, there is no known formula to find the exact number of solutions for N queen's problem.

Table-4 Solution of N Queen's problem for N=1 to N=26, both Unique and Distinct

No. of Queen's	Unique Solution	Total distinct solutions
1	1	1
2	0	0
3	0	0
4	1	2
5	2	10
6	1	4
7	6	40
8	12	92
9	46	352
10	92	724
11	341	2,680
12	1787	14,200
13	9233	73,712
14	45,752	365,596
15	285,053	2,279,184
16	1,846,955	14,772,184
17	11,977939	95,815,104
18	83,263,591	666,090,624
19	621,012,754	4,968,057,848
20	4,878,666,808	39,029,188,884
21	39,333,324,973	314,666,222,712
22	336,376,244,042	2,691,008,701,644

23	3,029,242,658,210	24,233,937,684,440
24	28,439,272,956,934	227,514,171,973,736
25	275,986,683,743,434	2,207,893,435,808,352
26	2,789,712,466,510,289	22,317,699,616,364,044

2.4.1 Formulation of 4-Queen's problem:

States: any arrangement of 0 to 4 queens on the board

Initial state: 0 queens on the board

Successor function: Add queen in any square

Goal test: 4 queens on the board, none attacked

For the initial state, there are 16 successors. At the next level, each of the states has 15 successors, and so on down the line. This search tree can be restricted by considering only those successors where No queens are attacking each other. To do that, we have to check the new queen with all the other queens on the board. In this way, the answer is found at a depth 4.For the sake of simplicity, you can consider a problem of 4-Queen's and see how 4-queen's problem is solved using the concept of "Backtracking".

2.4.2 State space tree for 4 Queen's Problem

We place queen row-by-row (i.e.,Q_1in row 1, Q_2 in row 2 and so on).
Backtracking gives "all possible solution". If you want optimal solution, then go for Dynamic programming.
Let's see Backtracking method, there are ${}^{16}_{4}C$ ways to place a queen on a 4x4 chess board as shown in the following state space tree (figure 12). In a tree, the value (i, j) means in thei^{th} row, j^{th} queen is placed.

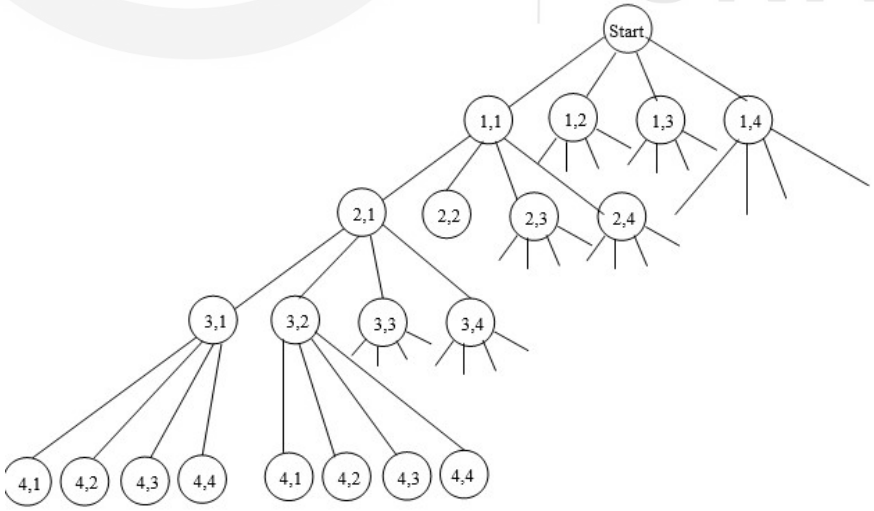


Fig 12 State-space tree showing all possible ways to place a queen on a 4x4 chess board

So, to reduce the size (not anywhere on chess board, since there are ${}^{16}_{4}C$ Possibilities), we place queen row-by-row, and no Queen in same column.This tree is called a permutation tree (**here we avoid same row or same columns but allowing diagonals**)

Total nodes=$1 + 4 + 4 \times 3 + 4 \times 3 \times 2 + 4 \times 3 \times 2 \times 1 = 65$

The edges are labeled by possible values of x_i. Edges from level 1 to level 2 nodes specify the values for x_1. Edges from level $\boldsymbol{i}$ to level $\boldsymbol{i+1}$ are labeled with the values of x_i.
The solution space is defined by all paths from root node to leaf node. There are 4! = 24 leaf nodes are in the tree Nodes are numbered as depth first Search. The state space tree for 4-Queen's problem (**avoid same row or same columns but allowing diagonals**)is shown in figure 13.

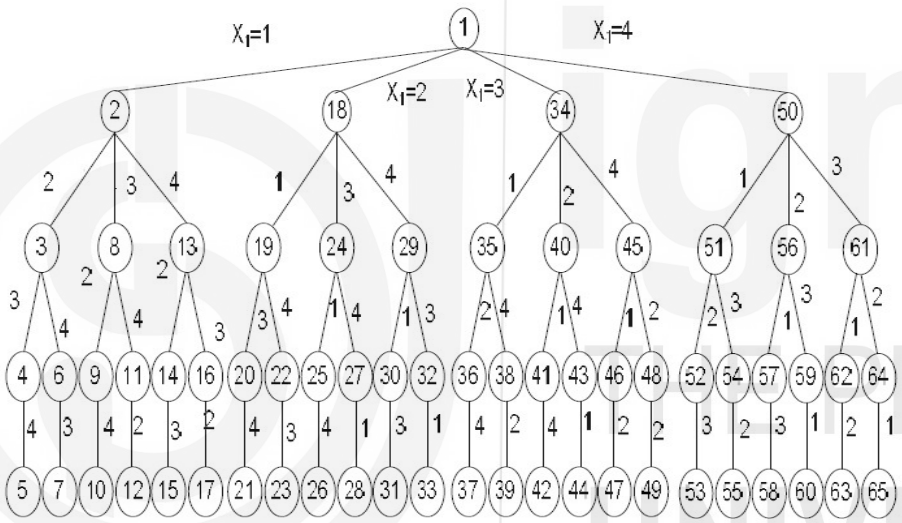


Fig13 State space tree for 4 queen's problems (allowing same diagonal but not same row and same column)

The two solutions are found in the tree, as
$(x_1, x_2, x_3, x_4) = (2,4,1,3)$ and
$(x_1, x_2, x_3, x_4) = (3,1,4,2)$, which is shown in figure-14

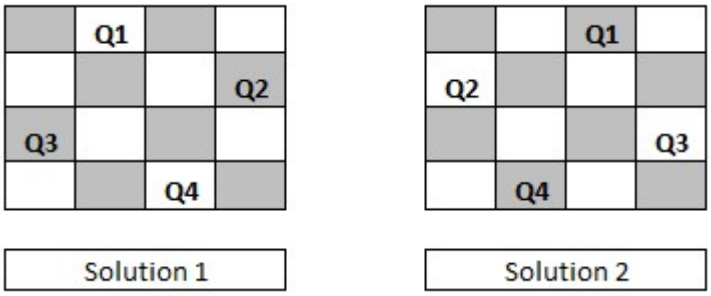


Fig 14 Two possible solutions of 4-Queen's problem

Note that the second solution is just a mirror image of the first solution.

We can further reduce the search space, as shown in figure 6 by avoiding diagonal also. Now you can avoid the **same row, avoid same columns and avoid same diagonals**, while placing any queen. In this case, the state space tree is look like as shown in figure 15.

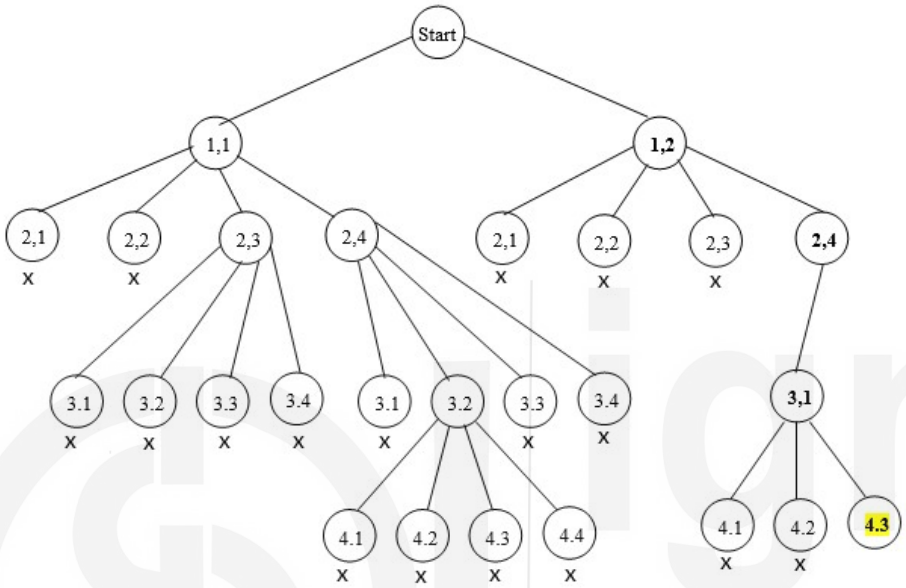


Fig 15 State space tree for 4 queen's problem (avoiding same row, columns, and diagonal)

Note that Queens are placed row-by-row, that is,Q_1in row 1, Q_2 in row 2 and so on. In a tree, node (1,1) is a promising node (no queen attack) as Q_1 is placed in 1st row and 1st column. Node (2,1) is non promising node, because we cannot place Q_2in the same column (as Q_1 is already placed in column 1). Note that nonpromising node is marked as ×. So, we try (2,2) again nonpromising (due to same column), next try (2,3), it's a promising node, so proceed and try to place 3rd queen on 3rd row. But in 3rd row, all positions (3,1),(3,2),(3,3) and (3,4) are non promising and we cannot place the Q_3 in any of this position. So, we backtrack to (1,1) and try for (2,4) and so on.Backtracking approach gives "all possible solution". Figure 7 shows one possible solution for 4-Queen's problem as {(1,2),(2,4),(3,1),(4,3)}. This can also be written as $(x_1, x_2, x_3, x_4) = (2,4,1,3)$. There are 2 possible solution of 4-Queen's problem. Another solution is $(x_1, x_2, x_3, x_4) = (3,1,4,2)$, which is a mirror image of 1st solution.

2.4.3 Backtracking Approach to solve N queen's Problem:

Consider the chess board squares indices of the 2-Dimentional array [1…n,1…n]. we observe that every element on the same diagonal that rows from the upper left to the right has same $(row - column)$ value. It is called ***left diaonals***. Similarly, every elemnet on the same diagonal that goes from the upper rigth to the lower left has same $(row + column)$ value. This is called

Right Diagonals. For example consider a 5×5 chessboard as [1…5,1…5] (as shown in figure 16).

Case1:(Left diagonal):- suppose queen's are palced in same diadinal in locations: (1,2),(2,3),(3,4),(4,5) or (1,4),(2,5) or any other same left diagonal value. Observe that every element on the same diagonal has the same $(row - column)$value. Similarly,

Case2:(right diagonal):- suppose queen's are palced in same diadinal in locations: (1,3),(2,2),(3,1) or (1,4),(2,3),(3,2),(4,1) or any other same right diagonal value. Observe that every element on the same diagonal has the same $(row + column)$value.

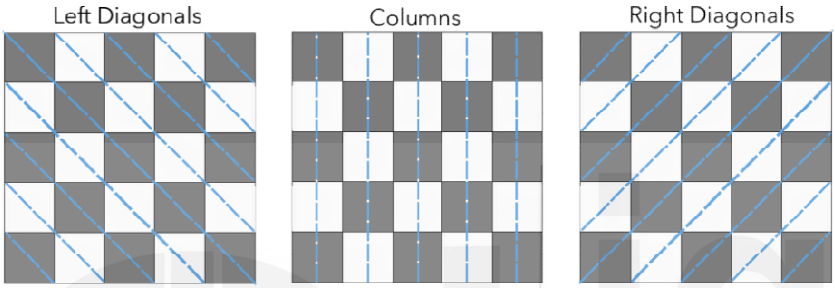


Fig 16 Left diagonal and right diagonal for 5×5 chessboard.

Suppose two queen's are placed at position (i, j) and (k, l) then they are on the same diagonal if and only if:

$(i - j) = (k - l)$ *or* $(j - l) = (i - k)$ ----------(1) [left diagonal]

$(i + j) = (k + l)$ *or* $(j - l) = (k - i)$ ---------- (2) [right diagonal]

From equation (1) and (2), we can combine and write a one condition to check diagonal as: $\boldsymbol{abs(j - l) = abs(i - k).}$

Algorithm NQueen(k,n)

// This procedure prints all possible placement of n queen's on $n \times n$
//chessboard so that they are non-attacking.
{
1. for i=1 to n do
2. {
3. if place(k,i) then
4. {
5. x[k]=i;
6. if (k==n) then print(x[1….n])
7. else
8. NQueen(k+1,n);
9. }

```
10.  }
11.  }

Algorithm place(k,i)
  // This algorithm return true, if a queen can be placed in kth row ith
  //column. Else it return false. X[] is a global array. Abs® returns
  //absolute value of r.

1. {
2.    for j=1 to k-1 do
3.     {
4.if(x[j]=i)      // in the same column
5.Or (abs(x[j]-i)==abs(j-k))// in the
                          //same diagonal
6.          Return false;
7.       }
8.      Return true;
9.     }
```

☞ **Check Your Progress 1**

Q.1 What are the various factors need to be taken into consideration when developing a statespace representation?

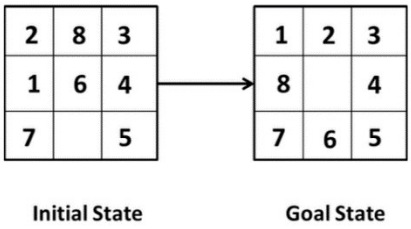
Q.2 Consider the following Missionaries and cannibal problem:

Three missionaries and three cannibals are side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side, without ever leaving a group of missionaries outnumbered by cannibals.

a) Formulate the missionaries and cannibal problem.
b) Solve the problem formulated in part (a)
c) Draw the state- space search graph for solving this problem.

Q.3 Draw a state space tree representation to solve Tower of Hanoi problem. (Hint: You can take number of disk n=2 or 3).

Q.4: Draw the state space tree for the following 8-puzzle problem, where the start and goal state are given below. Also Find the minimum cost path for this 8-puzzle problem. Each blank move is having cost=1.



Q.5 Discuss a Backtracking algorithm to solve a N-Queen's problem. Draw a state space tree to solve a 4-Queen's problem.

2.5 Adversarial Search-Two agent search

In computer science, a search algorithm is an algorithm for finding an item with specified properties among a collection of items. The items may be stored individually as records in a database; or may be elements of a search space defined by a mathematical formula or procedure.
Adversarial search is a **game-playing** technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search. Here, game-playing means discussing those games where **human intelligence** and **logic factor** is used, excluding other factors such as **luck factor**. **Tic-tac-toe, chess, checkers**, etc., are such type of games where no luck factor works, only mind works.
Mathematically, this search is based on the concept of **'Game Theory.'** ***According to game theory, a game is played between two players. To complete the game, one has to win the game and the other loses automatically.'***

Techniques required to get the best optimal solution
There is always a need to choose those algorithms which provide the best optimal solution in a limited time. So, we use the following techniques which could fulfil our requirements:

- **Pruning:** A technique which allows ignoring the unwanted portions of a search tree which make no difference in its final result.
- **Heuristic Evaluation Function:** It allows to approximate the cost value at each level of the search tree, before reaching the goal node.

2.5.1 Elements of Game Playing search

To play a game, we use a game tree to know all the possible choices and to pick the best one out. There are following elements of a game-playing:

- **S_0:** It is the **initial state** from where a game begins.
- **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
- **ACTIONS (s):** It defines the set of legal moves that a player can make to change the state.
- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s)**: It defines that the game has ended (or over) and returns true. States where the game has ended are called **terminal states.**
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**. This utility function gives a numeric value for the outcome of a game i.e.

For example, in **chess, tic-tac-toe,** we have two or three possible outcomes. Either win or lose, or draw the match, which we can represent by the values **+1,-1 or 0.** In other word we can say that **(-1):** if the PLAYER loses, **(+1),** if the PLAYER wins and **(0):** If there is a draw between the PLAYERS.

Let's understand the working of the elements with the help of a game tree designed for **tic-tac-toe**. Here, the node represents the game state and edges represent the moves taken by the players. The root of the tree is the initial state. Next level is all of MAX's moves, then next level is all of MIN's moves and so on. Note that root has 9 blank square (MAX), level 1 has 8 blank squares (MIN), level 2 has 7 blank square (MAX) and so on.

Objective:
Player1: Maximize outcome and **Player2**: Minimize outcome
Terminal (goal) state:
utility: -1, 0, +1 (that is win for X is +1 and win for O is -1 and 0 for draw. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX. High values are assumed to be good for MAX and bad for MIN (which is how the players get their

names). It is MAX's job to use the search tree to determine the best move. Note that if MAX win then Utility value is +1, if MIN wins then utility value is -1 and if DRAW then utility value is 0.

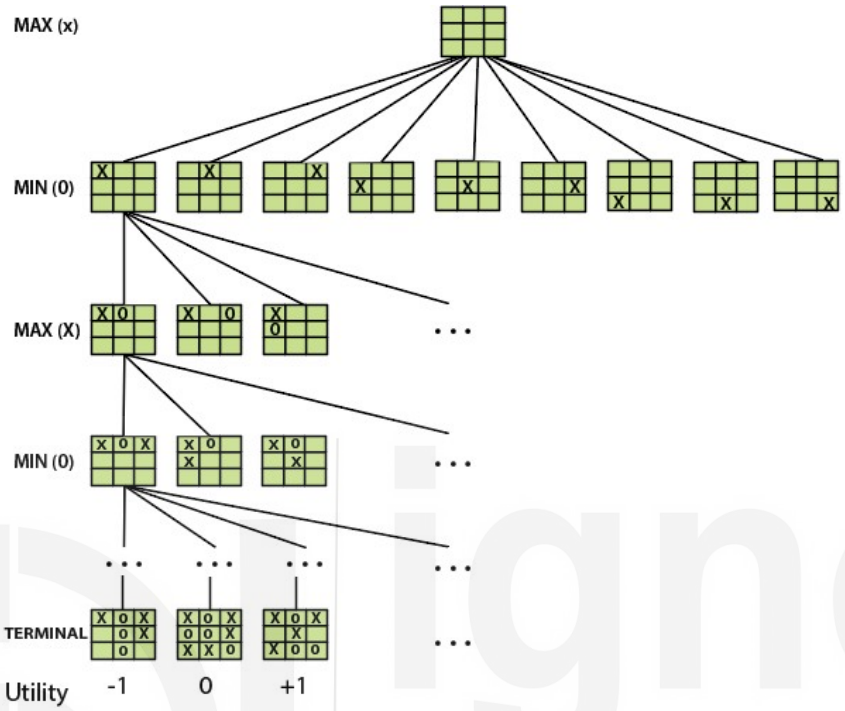


Fig 17**A game-tree for tic-tac-toe**

In a tic-tac-toe game playing, as shown in figure 17, we have the following elements:

- **INITIAL STATE (S_0):** The top node in the game-tree represents the initial state in the tree and shows all the possible choice to pick out one.
- **PLAYER (s):** There are two players, **MAX and MIN**. **MAX** begins the game by picking one best move and place **X** in the empty square box.
- **ACTIONS (s):** Both the players can make moves in the empty boxes chance by chance.
- **RESULT (s, a):** The moves made by **MIN** and **MAX** will decide the outcome of the game.
- **TERMINAL-TEST(s):** When all the empty boxes will be filled, it will be the terminating state of the game.
- **UTILITY:** At the end, we will get to know who wins: **MAX** or **MIN,** and accordingly, the price will be given to them**.** If MAX win then Utility value is +1, if MIN wins then utility value is -1 and if DRAW then utility value is 0.

2.5.2 Issues in Adversarial search

In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally. But in an **adversarial search**, the result depends on the players which will decide the result of the game. It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time. **Minimaxsearch Algorithm**is an example of adversarial search. **Alpha-beta Pruning** in this is used to reduce search space.

2.6 Min-Max search strategy

In artificial intelligence, minimax is a **decision-making** strategy under **game theory,** which is used to minimize the losing chances in a game and to maximize the winning chances. This strategy is also known as '**Min-Max,' 'MM,' or 'Saddle point.'** Basically, it is a two-player game strategy where *if one wins, the other lose the game*. This strategy simulates those games that we play in our day-to-day life. Like, if two persons are playing chess, the result will be in favour of one player and will againstthe other one. The person who will make his bes*t try, efforts as well as cleverness, will surely win.*
We can easily understand this strategy via **game tree**-where nodes are the states of the game, and edges are moves that were made *by the players in the game*. Players will be two namely:

- **MIN:** Decrease the chances of **MAX** to win the game.
- **MAX:** Increases his chances of winning the game.

They both play the game alternatively, i.e., turn by turn and following the above strategy, i.e., if one wins, the other will definitely lose it. Both players look at one another as competitors and will try to defeat one-another, giving their best.
In minimax strategy, the result of the game or the utility value is generated by a **heuristic function** by propagating from the initial node to the root node. It follows the **backtracking technique** and backtracks to find the best choice. MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.

2.6.1 MINIMAX Algorithm:

MINIMAX algorithm is a backtracking algorithm where it backtracks to pick the best move out of several choices. MINIMAX strategy follows the **DFS (Depth-first search)** concept. Here, we have two players **MIN and MAX,** and the game is played alternatively between them, i.e., when **MAX** made a move, then the next turn is of **MIN.** It means the move made by MAX is fixed and, he cannot change it. The same concept is followed in DFS strategy, i.e., we follow the same path and cannot change in the middle. That's why in MINIMAX algorithm, instead of BFS, we follow DFS. The following steps are used in MINMAX algorithm:

- Generate the whole game tree, all the way down to the terminal states.

- Apply the utility function to each terminal state to get its value.
- Use the utility of the terminal states to determine the utility of the nodes one level higher up in the search tree.
- Continue backing up the values from the leaf nodes toward the root, one layer at a time.
- Eventually, the backed-up values reached the top of the tree; at that point, MAX chooses the move that leads to the highest value.

This is called a minimax decision, because it maximizes the utility under the assumption that the opponent will play perfectly to minimize it. To better understand the concept, consider the following game tree or search tree as shown in figure 18.

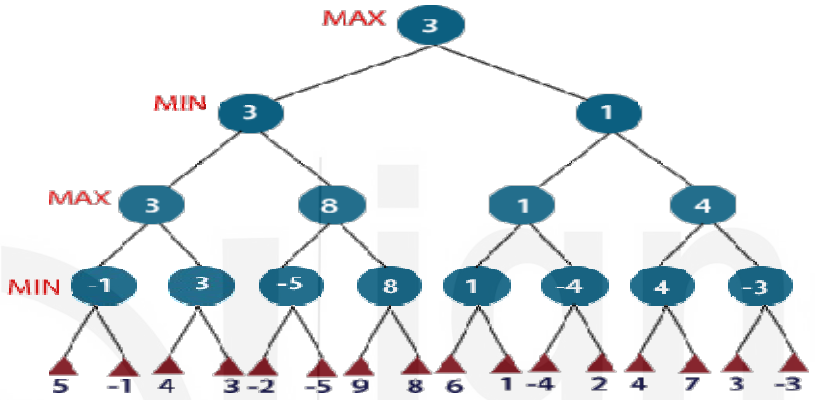


Fig 18 Two player game tree

In the above figure 2, the two players **MAX** and **MIN** are there. **MAX** starts the game by choosing one path and propagating all the nodes of that path. Now, **MAX** will backtrack to the initial node and choose the best path where his utility value will be the maximum. After this, its **MIN** chance. **MIN** will also propagate through a path and again will backtrack, but **MIN** will choose the path which could minimize **MAX** winning chances or the utility value.
So, if the level is minimizing, the node will accept the minimum value from the successor nodes. If the level is maximizing, the node will accept the maximum value from the successor.

In other word we can say that - Minimax is a decision rule algorithm, which is represented as a game-tree. It has applications in decision theory, game theory, statistics and philosophy. Minimax is applied in two player games. The one is the MIN and the other is the MAX player. By agreement the root of the game-tree represents the MAX player. It is assumed that each player aims to do the best move for himself and therefore the worst move for his opponent in order to win the game. The question may arise "How to deal with the contingency problem?" The answer is:

- Assuming that the opponent is rational and always optimizes its behaviour (opposite to us) we consider the best response. opponent's
- Then the minimax algorithm determines the best move

2.6.2 Working of Minimax Algorithm:

Minimax is applicable for decision making for two agent systems participating in competitive environment. These two players P1 and P2, also known as MIN and MAX player, maximizes and minimizes utility value of heuristics function. Algorithm uses recursion to search through game tree and compute minimax decision for current state. We traverse the complete game tree in a depth-first search (DFS) manner to explore the node. MAX player always select the maximum value and MIN always select the minimum value from its successor's node. The initial value of MAX and MIN is set to as $MAX = -\infty$ and $MIN = +\infty$. This is a worst value assigned initially and as the algorithm progress these values are changes and finally, we get the optimal value**.**

Example1: Let's take an example of two-player game tree search (shown in figure 19a) to understand the working of Minimax algorithm.

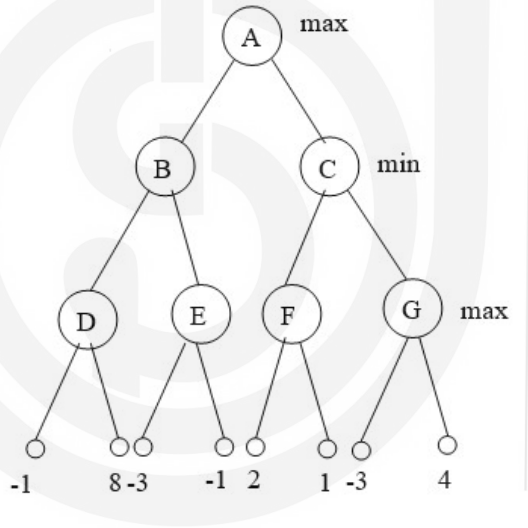


Fig 19a Two player game tree

The initial value of MAX and MIN is set to as $MAX = -\infty$ and $MIN = +\infty$. The tree is traversed in a DFS manner. So, we start from node A, then move to node B and then D.

Now at node D[$\boldsymbol{MAX = -\infty}$**].** Now, at D, it first checks the left child (which is a terminal node) with value-1. This node returns a value of $MAX = (-\infty, \mathbf{-1}) = \mathbf{-1}$. So,modified value at node D is $[MAX = -1]$. Next, we proceedfor right child of Node D (which has terminal value 8) and compare this value (8) with previous value at node D. that is $MAX = max(\mathbf{-1}, \mathbf{8}) = \mathbf{8}$. So final value at node D is 8.

Similarly,

the value at node E (which is a Max node) is $MAX = max(-\infty, \mathbf{-3}) = \mathbf{-3}$, then $\boldsymbol{max(-3, -1) = -1.}$

So, at node B, which is at MIN level, select the minimum value from its successor node D and E as $MIN = min(8, -1) = -1$

Similarly, the value at node F (which is also Max node) is $MAX = max(-\infty, \mathbf{2}) = \mathbf{2}, \boldsymbol{then}\ \boldsymbol{max}(\mathbf{2}, \mathbf{1}) = \mathbf{2},$ and
The value at node G (which is also MAX node) is
$MAX = max(-\infty, -\mathbf{3}) = -\mathbf{3},$ and then$max(-3,4) = 4$**.**
Thus, at node C, which is also at MIN level, select the minimum value from its successor node F and G as $MIN = min(2,4) = 2.$

Now, the value at node B and C is -1 and 2 respectively.

Thus, finally, the value at node A, which is at MAX level, is

$MAX = max(-1,2) = 2.$

The final game tree with max or min value at each node and optimal path, with shaded line A→C→F→2, is shown in the following figure 19(b).

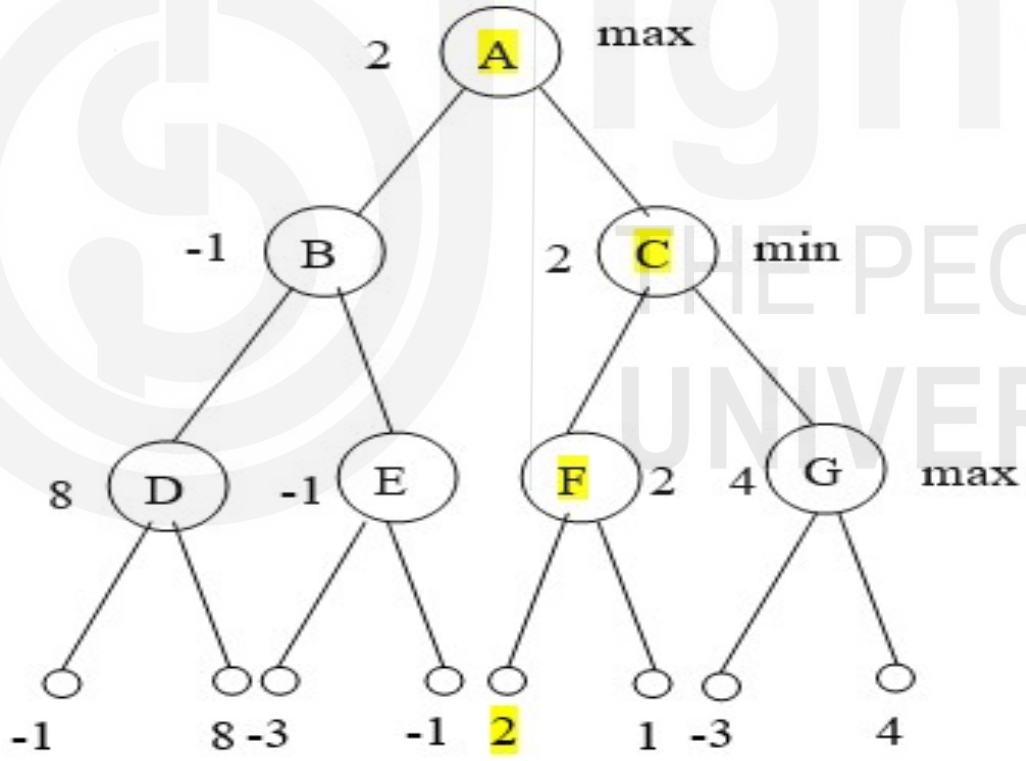


Fig 19(b) Game tree with final value at each node with optimal path

Example2 Consider the following two-player game tree search. The working of Minimax algorithmis illustrated from fig (a)-fig(k)

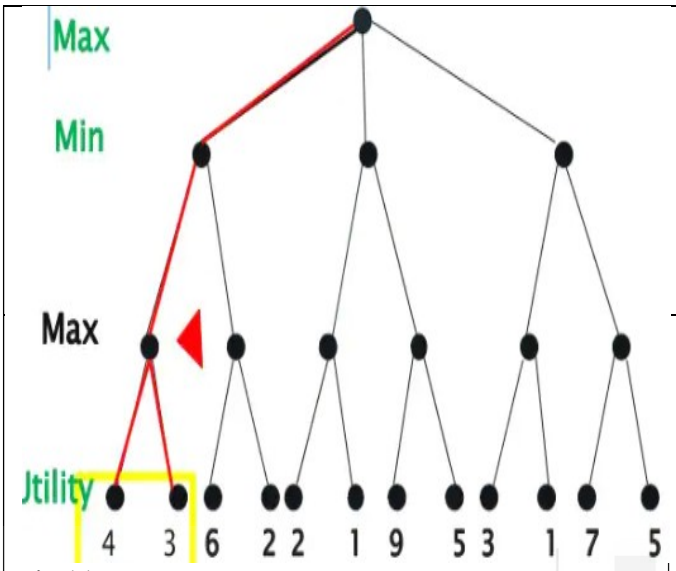


Fig (a)

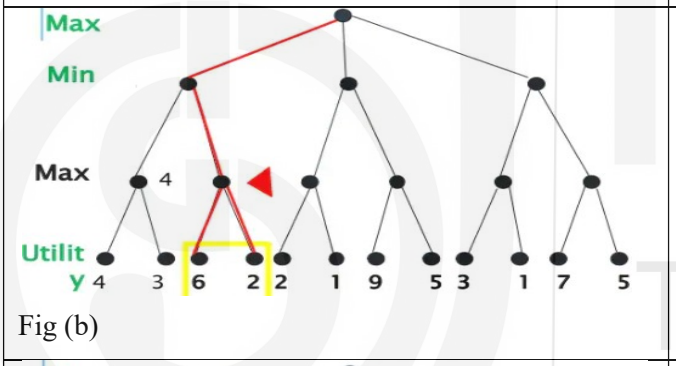


Fig (b)

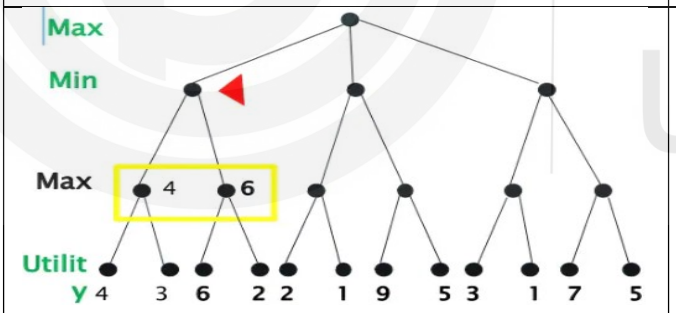


Fig (c)

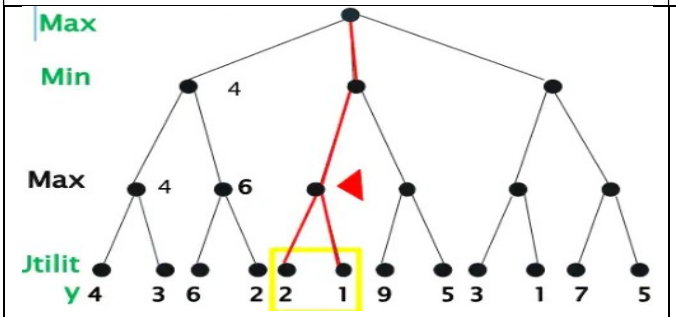


Fig (d)

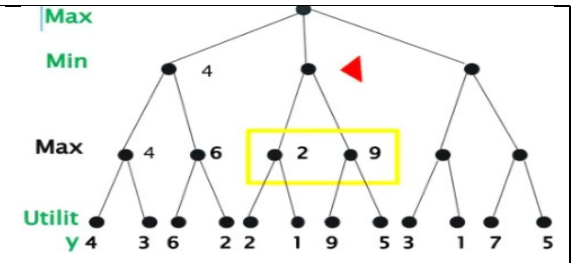


Fig (f)

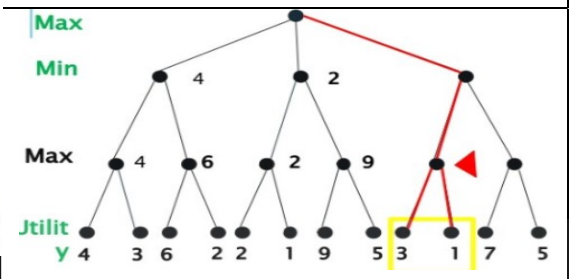


Fig (g)

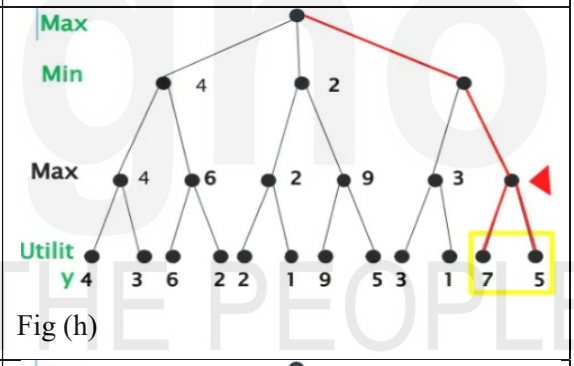


Fig (h)

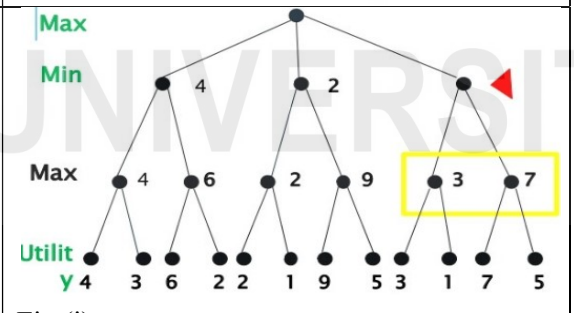


Fig (i)

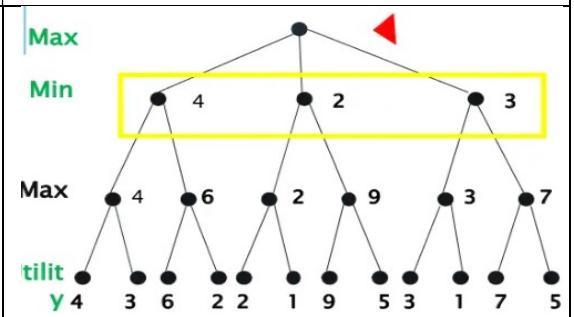


Fig (j)

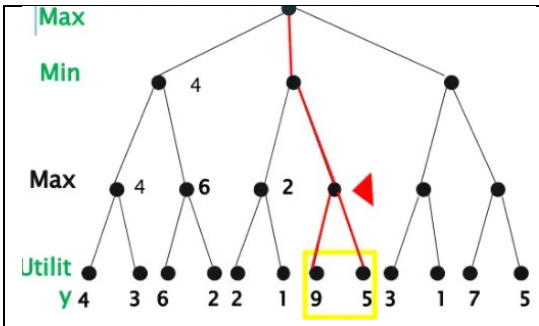


Fig (e)

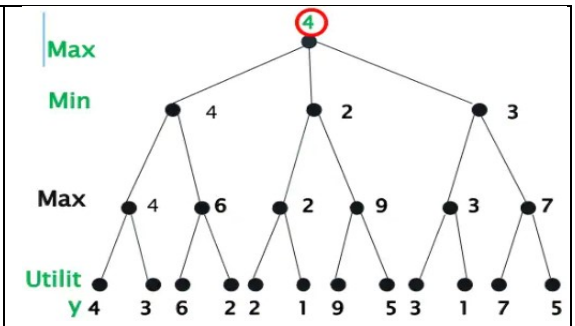


Fig (k)

2.6.3 Properties of Minimax Algorithm:

1. **Complete:** Minimax algorithm is complete, if the tree is finite.
2. Optimal: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution is said to be an optimal solution. Minimax is Optimal.
3. **Time complexity:** $O(b^m)$, where b: Branching Factor and m is the maximum depth of the game tree.
4. **Space complexity:** $O(bm)$

For example, in chess playing game b = 35, m ≈100 for "reasonable" games. In this case exact solution completely infeasible

2.6.4 Advantages and disadvantages of Minimax search

Advantages:

- Returns an optimal action, assuming perfect opponent play.
- Minimax is the simplest possible (reasonable) game search algorithm.

Disadvantages:

- It's completely infeasible in practice.
- When the search tree is too large, we need to limit the search depth and apply an evaluation function to the cut-off states.

2.7 Alpha-beta Pruning

The drawback of Minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity. If b is the branching factor and

d is the depth of the tree, then time complexity of MINIMAX algorithm is $O(b^d)$ that is exponential. But as we know, the performance measure is the first consideration for any optimal algorithm. Alpha-beta pruning is a method to reduce (prone) search space. Using Alpha-Beta pruning, the Minimax algorithm is modified. Therefore, alpha-beta pruning reduces this drawback of minimax strategy by less exploring the nodes of the search tree.

The method used in alpha-beta pruning is that its **cut-off the search** by exploring a smaller number of nodes. It makes the same moves as a minimax algorithm does, but it prunes the unwanted branches using the pruning technique (discussed in adversarial search). Alpha-beta pruning works on two threshold values, i.e., α **(alpha)** and β **(beta).**

- $\boldsymbol{\alpha}$ **:** It is the best highest value; a **MAX** player can have. The initial value of $\boldsymbol{\alpha}$ is set to negative infinity value, that is

  $\boldsymbol{\alpha} = -\infty$. As the algorithm progress its value may change and finally get the best (highest) value**.**
- $\boldsymbol{\beta}$ **:** It is the best lowest value; a **MIN** player can have. The initial value of $\boldsymbol{\beta}$ is set to positive infinity value, that is

  $\boldsymbol{\beta} = +\infty$. As the algorithm progress its value may change and finally get the best (lowest) value**.**

So, each MAX node has $\boldsymbol{\alpha}$-value, which never decreases, and each MIN node has $\boldsymbol{\beta}$-value, which never increases. The main condition which required for alpha-beta pruning is $\boldsymbol{\alpha \geq \beta}$**,** that is if$\boldsymbol{\alpha \geq \beta}$**,** then prune (cut) the branches otherwise proceed.

Note: Alpha-beta pruning technique can be applied to trees of any depth, and it is possible to prune the entire sub-trees easily.

2.7.1 Working of Alpha-beta Pruning

As we know there are two-parameter is defined for Alpha-beta pruning, namely alpha ($\boldsymbol{\alpha}$)and beta($\boldsymbol{\beta}$)**.** The initial value of alpha and beta is set to as $\boldsymbol{\alpha = -\infty}$ **and** $\boldsymbol{\beta = +\infty}$.As the algorithm progresses its values are changes accordingly. Note that in Alpha-beta pruning (cut), at any node in a tree, if $\boldsymbol{\alpha \geq \beta}$**,** then prune (cut) the next branch else search is continued. Note the following point for alpha-beta pruning:

- The MAX player will only update the value of$\boldsymbol{\alpha}$(on MAX level).
- The MIN player will only update the value of $\boldsymbol{\beta}$(on MIN level).
- We will only pass the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$value from top to bottom (that is from any parent to child node, but never from child to parent node).
- While backtracking the tree, the node values will be passed to upper node instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.**

- Before going to next branch of the node in a tree, we check the value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$. If the value of$\boldsymbol{\alpha \geq \beta}$, then prune (cut) the next (unnecessary) branches (i.e., no need to search the remaining branches where the condition $\boldsymbol{\alpha \geq \beta}$is satisfied) else search continued.

Consider the below example of a game tree where **P** and **Q** are two players. The game will be played alternatively, i.e., chance by chance. Let, **P** be the player who will try to win the game by maximizing its winning chances. **Q** is the player who will try to minimize **P'**s winning chances. Here, $\boldsymbol{\alpha}$ will represent the maximum value of the nodes, which will be the value for **P** as well. $\boldsymbol{\beta}$ will represent the minimum value of the nodes, which will be the value of **Q**.

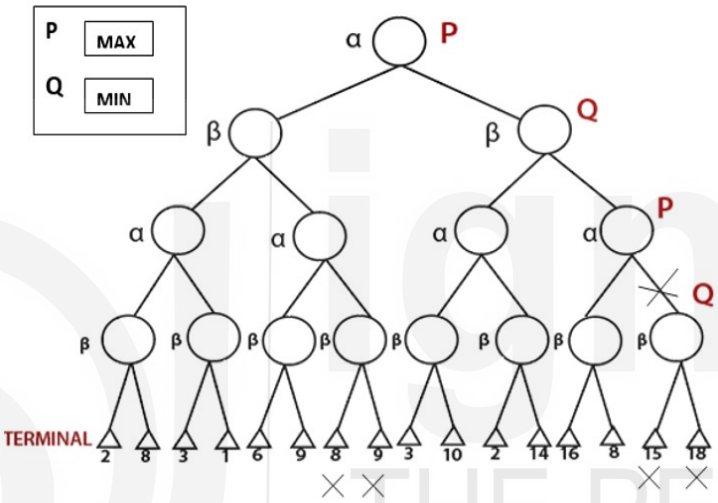


Fig 20 Alpha-beta pruning

- Any one player will start the game. Following the DFS order, the player will choose one path and will reach to its depth, i.e., where he will find the **TERMINAL** value.
- If the game is started by player P, he will choose the maximum value in order to increase its winning chances with maximum utility value.
- If the game is started by player Q, he will choose the minimum value in order to decrease the winning chances of A with the best possible minimum utility value.
- Both will play the game alternatively.
- The game will be started from the last level of the game tree, and the value will be chosen accordingly.
- Like in the figure 5, the game is started by player Q. He will pick the leftmost value of the TERMINAL and fix it for beta ($\boldsymbol{\beta}$). Now, the next TERMINAL value will be compared with the $\boldsymbol{\beta}$-value. If the value will be smaller than or equal to the $\boldsymbol{\beta}$-value, replace it with the current $\boldsymbol{\beta}$-value otherwise no need to replace the value.

- After completing one part, move the achieved $\boldsymbol{\beta}$-value to its upper node and fix it for the other threshold value, i.e.,$\boldsymbol{\alpha}$.
- Now, its P turn, he will pick the best maximum value. P will move to explore the next part only after comparing the values with the current $\boldsymbol{\alpha}$-value. If the value is equal or greater than the current $\boldsymbol{\alpha}$-value, then only it will be replaced otherwise we will prune the values.
- The steps will be repeated unless the result is not obtained.
- So, number of pruned nodes in the above example are **four** and MAX wins the game with the maximum **UTILITY** value, i.e.,**3.**

The rule which will be followed is: **"Explore nodes, if necessary, otherwise prune the unnecessary nodes."**
Note: It is obvious that the result will have the same **UTILITY** value that we may get from the MINIMAX strategy.

Alpha beta cut-off (or pruning):

1. for each node store limit $[\alpha, \beta]$.

2. Update $[\alpha, \beta]$,

where α is the lower bound at max node; it can't decrease.

β is the upper bound at min node; it can't increase.

3. If α value of a max node is greater than β value of its parent ($\alpha \geq \beta$), thesubtree of that max node need not be evaluated (i.e., pruned).

4. If β value of a min node is lesser than α value of its parent ($\beta \leq \alpha$), the subtree of that min node need not be evaluated (i.e., pruned).

Example1: Let's take an example of two-player search tree (Figure 21) to understand the working of alpha-beta pruning.

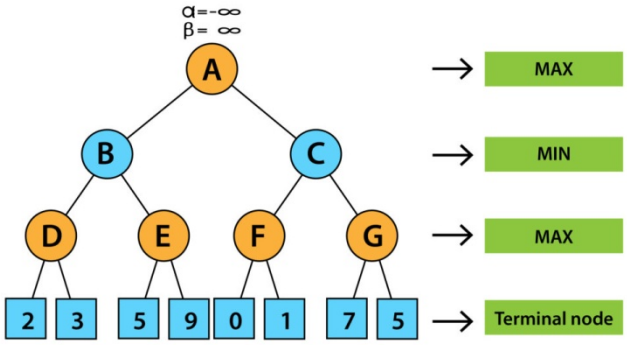


Fig 21 Two player search tree

We initially start the search by setting the initial value of $\boldsymbol{\alpha} = -\infty$and$\boldsymbol{\beta} = +\infty$to root **node A**.

Note the following important point to apply the Alpha-beta pruning:

- We will only pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top to bottom (that is from any parent to child node), but never from child to parent node.
- While backtracking the tree (from bottom to top node), the node values will be passed to upper node instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.**
- Before exploring the next branch in a tree, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**.** If **YES,** then prune (cut) the next (unnecessary) branches (i.e., no need to search the remaining branches where the condition $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$is satisfied) else search continued.
- The MAX player will only update the value of$\boldsymbol{\alpha}$(on MAX level) and the MIN player will only update the value of $\boldsymbol{\beta}$(on MIN level).

Step1: We traverse the tree in a depth-first search (DFS) manner and assign (pass) this value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$down to subsequent nodes**B** and then to node**D**as $[\boldsymbol{\alpha} = -\infty; \boldsymbol{\beta} = +\infty]$**.**
Now at node D $[\boldsymbol{\alpha} = -\infty, \boldsymbol{\beta} = +\infty]$**,** Since node D is at MAX level, so only $\boldsymbol{\alpha}$value will be changed. Now, at D, it first checks the left child (which is a terminal node) with value 2. This node returns a value of 2. Now, the value of $\boldsymbol{\alpha}$ **at** node D is calculated as$\alpha = max(-\infty, 2) = 2$. So modified value at node D is $[\alpha = 2, \beta = +\infty]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer in NO since $2 \ngeq +\infty$. So, proceed and search is continued for right child of Node D.
The value of right child (terminal node with value=3) of D returns a value 3. Now at D, the value of $\alpha$is compared with terminal node value 3, that is, $\alpha = max(2,3) = 3$.Now the value of

Node(D)=3, and the final values of α and $\beta$is updated at node D as$[\alpha = 3, \beta = +\infty]$ as shown in figure 21(a).

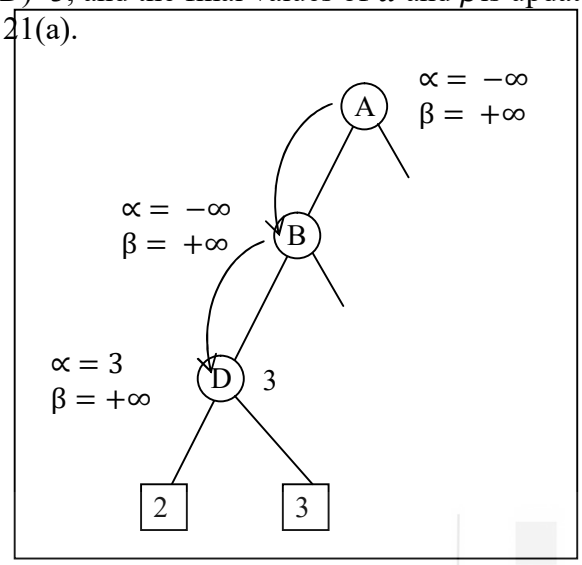


Fig 21(a)

Step 2. We backtrack from node D to B. Note that, while backtracking the node in a tree, the node values of D(=3) will be passed to upper nodeB instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.**Now the value of node(B)=node(D)=3. Since B is at MIN level, so only$\boldsymbol{\beta}$value will be changed. Now at node **B** $\boldsymbol{[\alpha = -\infty, \beta = 3]}$ (note that β is change from $+\infty$ to 3). Here we again check $\alpha \geq \beta$. It is False, so search is continued on right side of B, as shown in figure (b).

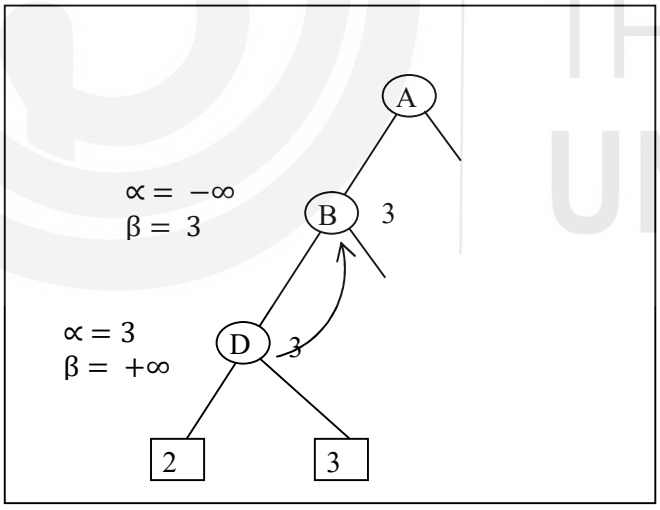


Fig21(b)

Step 3.B now calls E, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node B to bottom node E as $[\boldsymbol{\alpha = -\infty, \beta = 3}]$**. Since,** node E is at MAX level, so only $\boldsymbol{\alpha}$value will be change.Now, at E, it first checks the left child (which is a terminal node) with value5. This node returns a value of 5.

Now, the value of $\boldsymbol{\alpha}$ **at** node E is calculated as$\alpha = max(-\infty, 5) = 5$, so value of Node(E)=5 and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$ at node Eis $[\alpha = 5, \beta = 3]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer is**YES**, since $5 \geq 3$. So,we prune (cut) the right branch of E, as shown in figure 21(c).

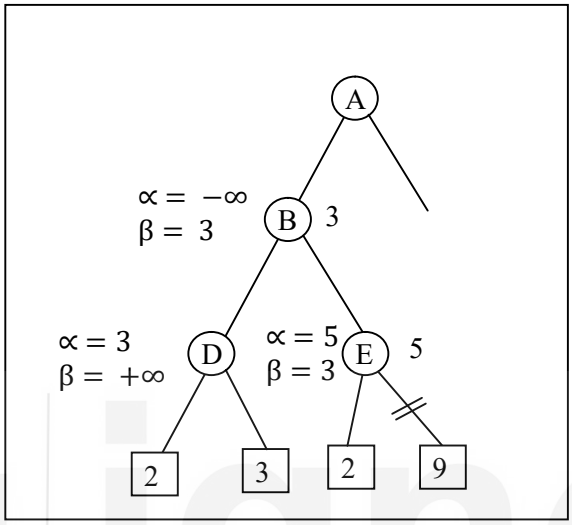


Fig21(c)

Step4. We backtrack from node E to B. Note that, while backtracking, the node values of E(=5) will be passed to upper node B, instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**. E return a value 5 to B**.Since B is at MIN level, so only$\boldsymbol{\beta}$value will be changed. Previously, at node **B** $[\boldsymbol{\alpha = -\infty, \beta = 3}]$**, but now**$\boldsymbol{\beta} = min(3,5) = 3$, so, there is no change in $\boldsymbol{\beta}$ value and value of node(B) is still 3. Thus finally, modified value at node **B is**$[\boldsymbol{\alpha = -\infty, \beta = 3}]$**.**
We backtrack from node B to A. Again, note that, while backtracking the tree, the value of node(B)=3 will be passed to upper node A, instead of values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$**.Now value of Node(A)=3.**

Since A is at MAX level, so only$\boldsymbol{\alpha}$value will be changed. Previously, at node **A** $[\alpha = -\infty, \beta = +\infty]$ and after comparing value of node(B)=3 with old value of α at node A, that is $\alpha = max(-\infty, 3) = 3$. Thus finally, at node A $[\alpha = 3, \beta = +\infty]$ and value of Node(A)=3. we check $\alpha \geq \beta$, it is False, so proceed on right side. Now, we completed the left sub tree of A and proceed towards right subtree, as shown in figure 21(d).

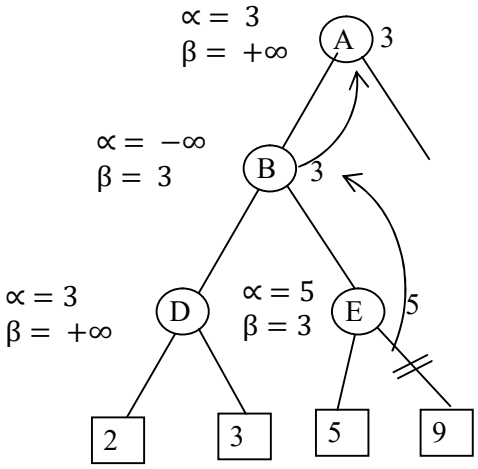


Fig21(d)

Step 5.

Now at node C, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node A to bottom node C as $[\boldsymbol{\alpha = 3, \beta} = +\infty]$**. Check,**$\alpha \geq \beta$. The answer in NO. So, search is continued. Now pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node C to bottom node F as $[\boldsymbol{\alpha = 3, \beta} = +\infty]$**.** Since F is at MAX level, so only$\boldsymbol{\alpha}$value will be changed.

Now, at F, it first checks the left child (which is a terminal node) with value0. This node returns a value of 0. Now, the value of $\boldsymbol{\alpha}$ **at** node F is calculated as$\alpha = max(3,0) = 3$. So modified value at node Fis $[\alpha = 3, \beta = +\infty]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha \geq \beta}$. The answer in NO since $3 \ngeq +\infty$. So, proceed and search is continued for right child of Node F.

The value of right child (terminal node with value=1) of F returns a value 1, so finally, value of node(F)=1. Now at F, the value of α is compared with terminal node value 1, that is, $\alpha = max(3,1) = 3$, and the final values of α and β is updated at node F as$[\alpha = 3, \beta = +\infty]$ as shown in figure 21(e).

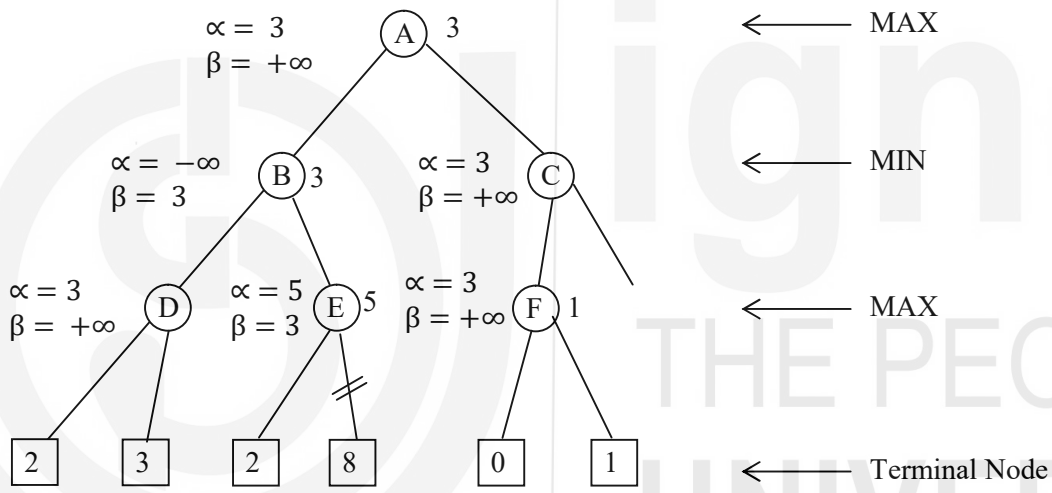


Fig21(e)

6. We backtrack from node F to C. Note that, while backtracking the tree, the node values of F(=3) will be passed to upper node C**.** Now the value of **node(C)=node(F)=1**.

Since C is at MIN level, so only βvalue will be changed. Previously, at node **C** $[\alpha = 3, \beta = +\infty]$. Now, old value of $\boldsymbol{\beta} = +\infty$ is compared with value of node(F)=node(C)=1. That is, $\boldsymbol{\beta} = min(+\infty, 1) = 1$. Thus finally, at node **B** $[\boldsymbol{\alpha = 3, \beta = 1}]$**. Now we check,**$\alpha \geq \beta$. It is TRUE, so we prune (cut) the right branch of node C. That is node G will be pruned and algorithm stop searching on right subtree of node C.

Thus finally, **we** backtrack from node C to A and node C return the value 1 to node A. Since A is a MAX node, so only$\boldsymbol{\alpha}$value will be changed.

Previously, at node **A** $[\alpha = 3, \beta = +\infty]$ and after comparing value of node(C)=1 with old value of α at node A, that is $\alpha = max(3,1) = 3$. Thus finally, at node A $[\alpha = 3, \beta = +\infty]$ and value of Node(A)=3. Now, we completed the right sub tree of A also.

Following is the final game tree, showing the nodes which are computed and nodes which are pruned (cut) during search process of Alpha-beta pruning. Here the optimal value for the maximizer is 3 and there are 3 terminal nodes are pruned (9, 7 and 5). The optimal search path is A→B→D→3.

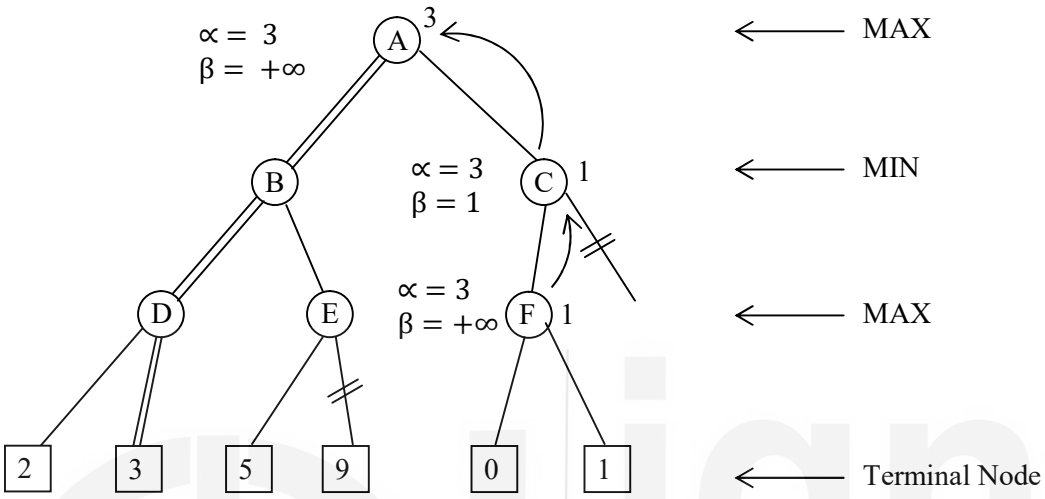


Fig 21(f)

Example2: Consider the following game tree (figure 22) in which root is maximizing node and children are visited from left to right. Find which nodes are pruned by the Alpha-beta pruning.

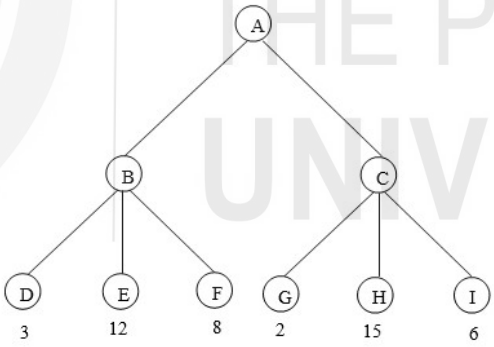


Fig 22 Two Player search tree

Solution:

Step1: We start the search by setting the initial value for node A as$[\boldsymbol{\alpha = -\infty, \beta = +\infty}]$. We traverse the node in depth-first search (DFS) manner so assign the same value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$tonode **B**$[\boldsymbol{\alpha = -\infty, \beta = +\infty}]$**,** Since node B is at MIN level, so only $\boldsymbol{\beta}$value will be changed at node B. Now, at D, it looks at it left child (terminal node), which returns a value 3 to D. So, we compare the old value of β at node B with this terminal node value 3, that is $\beta = \boldsymbol{min}(+\infty, \mathbf{3}) = \mathbf{3}$.So modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$tonode **B** is$[\boldsymbol{\alpha = -\infty}, \beta = 3]$.

To decide whether it worth looking at right subtree of B, we check

$\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer in NO, since $-\infty \not\geq 3$. So, proceed and search is continued for right child of Node B, that is E.

The terminal value of E=12. Now, the value of right child terminal E(=12) is compared with previous old value of $\beta = 3$, that is, $\beta = min(3,12) = 3$.So no change in β value. So, at present, current modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$atnode **B** is same$[\boldsymbol{\alpha} = -\infty, \beta = 3]$. Again check,$\alpha \geq \beta$. The answer in NO. So, proceed and search is continued for right child of Node B, that is F. The terminal value of F=8. The value of right child terminal at F(=8) is compared with previous old value of $\beta = 3$, that is, $\beta = min(8,3) = 3$. So no change in β value. So, finally value of **Node(B)=3** and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$atnode **B**$[\boldsymbol{\alpha} = -\infty, \beta = 3]$ as shown in figure 22(a)

Step2: We backtrack from node B to A. Note that, while backtracking the tree, the node values of B(=3) will be passed to upper node A, instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**. Now the value of node(A)=node(B)=3.** Since A is at MAX level, so only$\boldsymbol{\alpha}$value will be changed. Previously, at node **A** $[\alpha = -\infty, \beta = +\infty]$ and after comparing value of node(B)=3 with old value of α at node A, that is $\alpha = max(-\infty, 3) = 3$. Thus finally, at node **A** $[\boldsymbol{\alpha = 3, \beta = +\infty}]$ and value of **Node(A)=3**.

To decide whether it's worth looking at its right node of A or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer in NO since $3 \not\geq +\infty$. So, proceed and search is continued for right child of Node A. Now, we completed the left sub tree of A and proceed towards right subtree.

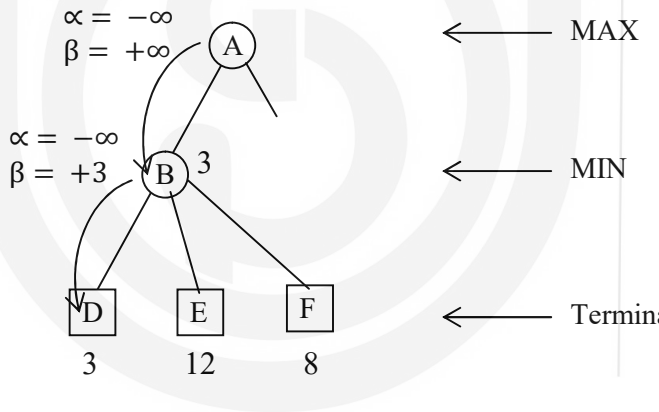


Fig22(a)

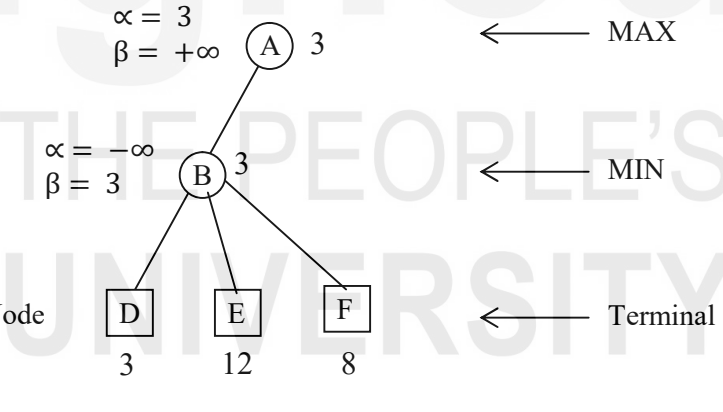


Fig 22(b)

Step 3: Now at node C, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node A to bottom node C as $[\boldsymbol{\alpha = 3, \beta = +\infty}]$**. Check,**$\alpha \geq \beta$. The answer in NO. So, continue the search on right side. Since C is at MIN level, so only$\boldsymbol{\beta}$value will be changed.

Now, we first check the left child (terminal) of node C, that is G=2. So, we compare the old value of $\boldsymbol{\beta}$ **at node C with this terminal node value 2, that is** $\boldsymbol{\beta = min(+\infty, 2) = 2}$**. So value of Node(C)=2** and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$atnode**C** is$[\boldsymbol{\alpha = 3}, \beta = 2]$. Now, before proceed next, we again check$\alpha \geq \beta$. The answer is YES. So,we prune (cut) the right branch of node C. That is **node H and I** will be pruned (cut) and algorithm stop searching on right subtree of node C, as shown in figure 22(c).

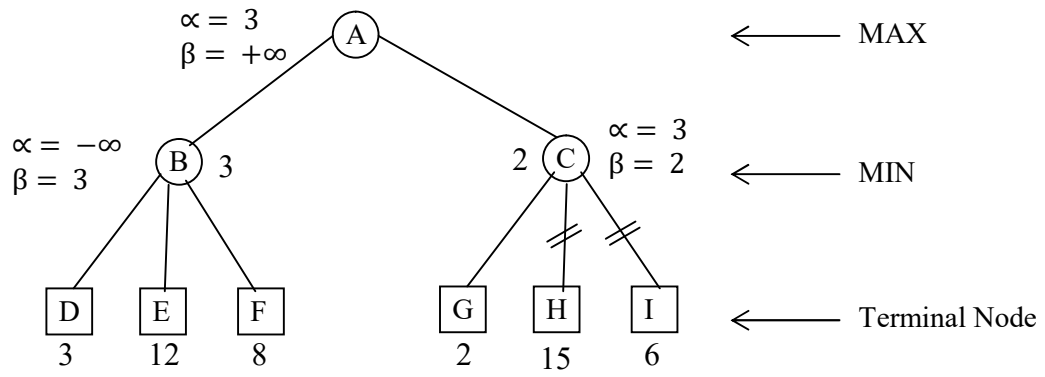


Fig22(c)

Step4: Finally, we backtrack from node C to A. Note that, while backtracking the node in a tree, the node values of C(=2) will be passed to upper node A, instead of values of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$**.The previous node(A)=3 value is compared with this new node(C)=2 value. The best value at node(A)=**$\alpha = max(3,2) = 3$.

The previous α and β value at node A$[\alpha = 3, \beta = +\infty]$. Since A is at **MAX** level so only α value is change. So, we compare old $\alpha = 3$ value with value at node(C)=2. That is $\alpha = max(3,2) = 3$. Thus, there is no change in α value as well.

Thus finally, $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value at node A is$[\alpha = 3, \beta = +\infty]$ and value of Node(A)=3. So, optimal value for the maximizer is 3 and there are 2 terminal nodes are pruned (H and I). The optimal search path is A→B→D (as shown in figure 22(d)).

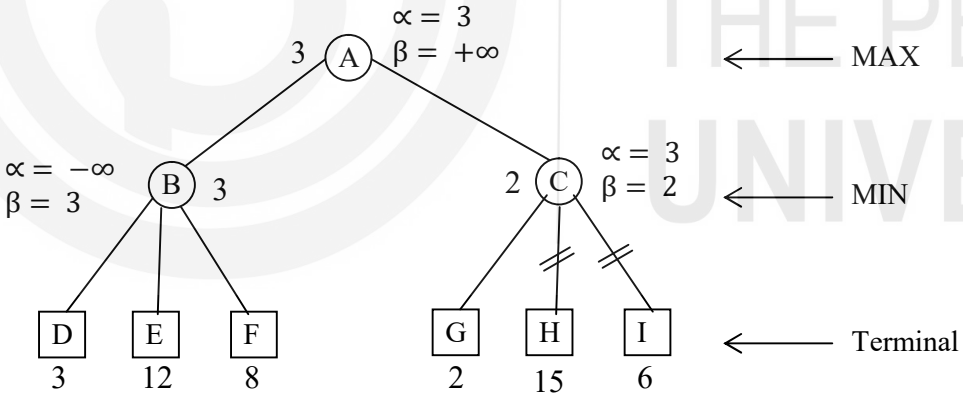


Fig 22(d)

2.7.2 Move ordering of Alpha-beta pruning

The effectiveness of Alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning. We have two types of move ordering:

Worst case 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, it consumes more time because of alpha-

beta factors, such a move of pruning is called a worst ordering. The time complexity for such an order is $O(b^m)$where b: Branching Factor and m is the depth of the tree.

Best (ideal) case ordering: The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best move 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. The time complexity for best case order is $O(b^{m/2})$ (since we search only left sub tree, not a right subtree).

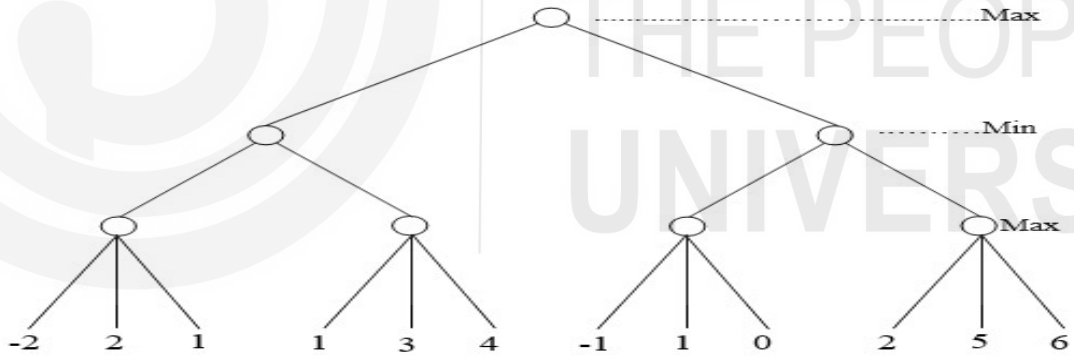
Note that pruning does not affect the final result. Good move ordering improves the effectiveness of pruning. With ideal case ordering, time complexity is $O(b^{\frac{m}{2}})$

In Alpha-beta pruning:
- α value can never decrease and β value can never increase. Search can be discontinued at anode if:
-It is a Max node and $\alpha \geq \beta$ it is beta cutoff
-It is a Min node and $\beta \leq \alpha$ it is a alpha cutoff.

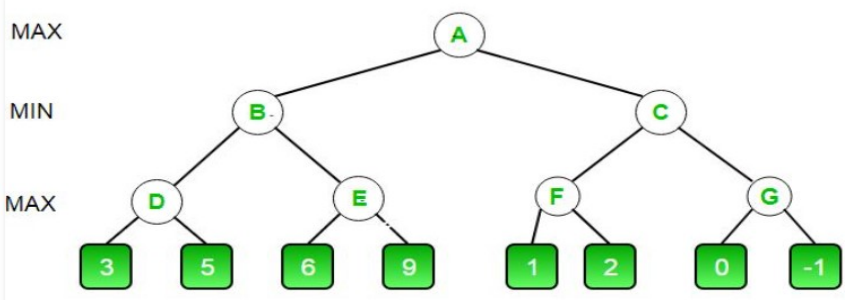
☞ **Check Your Progress 2**

Q.1: Compare the MINIMAX and Alpha-Beta Pruning algorithm with respect to Time complexity.

Q.2 Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find the value of the root node of the game tree?



Q.3 Apply Alpha-Beta pruning algorithm on the following graph and find which node(s) are pruned?



Q.4: Consider the following Minimax game tree search (figure1) in which root is maximizing node and children are visited from left to right.

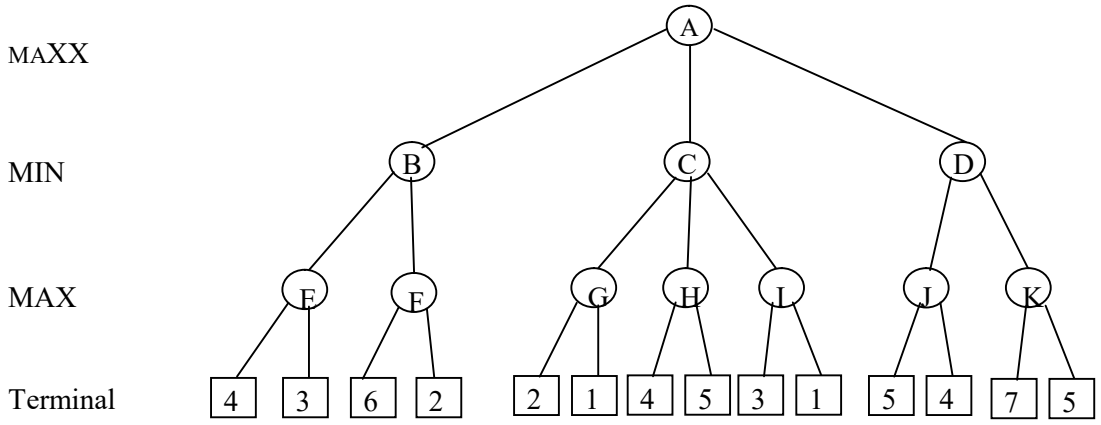
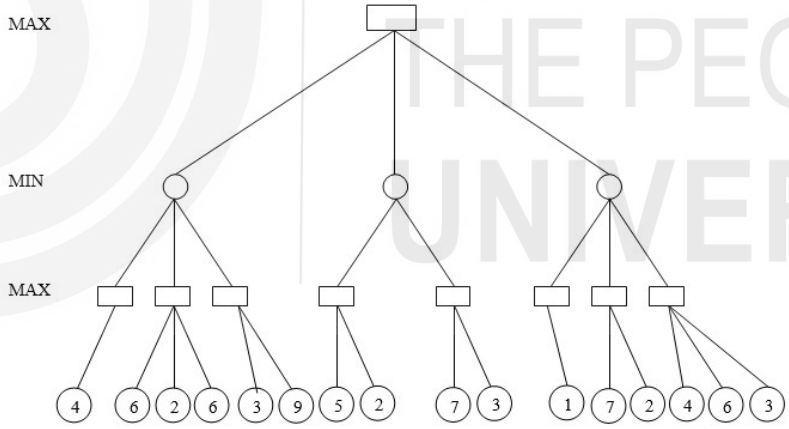


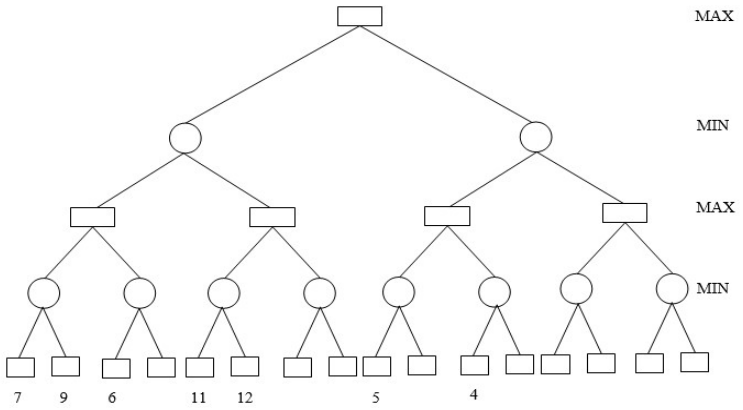
Figure1(a)

(a) Find the value of the root node of the game tree?
(b) Find all the nodes pruned in the tree?
(c) Find the optimal path for the maximizer in a tree?

Q.5: Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find what will be the value propagated at the root?



Q.6: Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find the value of the root node of the game tree?



Multiple choice Question

Q.7: Consider the following Minimax game tree search in which root is maximizing node and children are visited from left to right. Find the value of the root node of the game tree?

A. 14 B. 17 C. 111 D. 112

2.8 Summary

- Before an AI problem can be solved it must be represented as a **state space**. Among all possible states, there are two special states called ***initial state*** (the start point) and ***final state*** (the goal state).
- A ***successor function (a set of operators)***is used to change the state. It is used to move from one state to another.
- A state space is set of all possible states of a problem.
- A ***state space*** essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state, and a goal state. Each state space takes the form of a tree or a graph.
- The process of searching means a sequence of action that take you from an initial state to a goal state.

- search is fundamental to the problem-solving process. Search means the problem is solved by using the rules, in combination with an appropriate***control strategy***, to move through the problem space until a path from an ***initial state*** to a ***goal state*** is found.
- A **problem space** is represented by a directed graph, where *nodes* represent *search state*and *paths*represent the *operators* applied to change the state.
- In general, a state space is represented by 4 tuples as follows:$\boldsymbol{S_s}$: $[\boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{O}, \boldsymbol{G}]$**,** Where **S**: Set of all possible states (possibly infinite)**,** $\boldsymbol{s_0}$**:** start state (initial configuration) of the problem, $s_0 \in S$. **O**: Set of production rules (or set of state transition operator) used to change the state from one state to another. It is the set of arcs (or links) between nodes.
- Adversarial search is a game-playing technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search.
- In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally. But in an **adversarial search**, the result depends on the players which will decide the result of the game. It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time.
- There are 2 types of adversarial search: **Minimax Algorithm** and **Alpha-beta Pruning.**
- Minimax is a two-player (namely MAX and MIN) game strategy where *if one wins, the other lose the game*. This strategy simulates those games that we play in our day-to-day life. Like, if two persons are playing chess, the result will be in favour of one player and will go against the other one. **MIN:** Decrease the chances of **MAX** to win the game and **MAX:** Increases his chances of winning the game. They both play the game alternatively, i.e., turn by turn and following the above strategy, i.e., if one wins, the other will definitely lose it. Both players look at one another as competitors and will try to defeat one-another, giving their best.
- In minimax strategy, the result of the game or the utility value is generated by a **heuristic function** by propagating from the initial node to the root node. It follows the **backtracking technique** and backtracks to find the best choice. MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.
- The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity.
- If b is the branching factor and d is the depth of the tree, then time complexity of MINIMAX algorithm is $O(b^d)$ that is exponential.
- Alpha-beta pruning is an advance version of MINIMAX algorithm. Therefore, alpha-beta pruning reduces the drawback of minimax strategy by less exploring the nodes of the search tree.

- The alpha-beta pruning method **cut-off the search** by exploring a smaller number of nodes. It makes the same moves as a minimax algorithm does, but it prunes the unwanted branches using the pruning technique.
- Alpha-beta pruning works on two threshold values, i.e., α **(alpha)** and β **(beta).$\boldsymbol{\alpha}$:** It is the best highest value; a **MAX** player can have. The initial value of $\boldsymbol{\alpha}$ is set to negative infinity value, that is $\boldsymbol{\alpha} = -\infty$. As the algorithm progress its value may change and finally get the best (highest) value**. $\boldsymbol{\beta}$:** It is the best lowest value; a **MIN** player can have. The initial value of $\boldsymbol{\beta}$ is set to positive infinity value, that is $\boldsymbol{\beta} = +\infty$. As the algorithm progress its value may change and finally get the best (lowest) value**.**
- So, each MAX node has $\boldsymbol{\alpha}$ value, which never decreases, and each MIN node has $\boldsymbol{\beta}$ value, which never increases. The main condition which required for alpha-beta pruning is $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**,** that is if$\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**,** then prune (cut) the branches otherwise search is continued.
- As we know there are two-parameter is defined for Alpha-beta pruning, namely alpha ($\boldsymbol{\alpha}$)and beta($\boldsymbol{\beta}$)**.** The initial value of alpha and beta is set to as $\boldsymbol{\alpha} = -\infty$ **and** $\boldsymbol{\beta} = +\infty$.As the algorithm progresses its values are changes accordingly. Note that in Alpha-beta pruning (cut), at any node in a tree, if $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$**,** then prune (cut) the next branch else search is continued.
- The effectiveness of Alpha-beta pruning is highly dependent on the order in which each node is examined.
- **Worst case 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 time complexity is $O(b^m)$where b: Branching Factor and m is the depth of the tree.
- **Best (ideal) case ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best move occur at the left side of the tree. The time complexity for best case order is $O(b^{m/2})$ (since we search only left sub tree, not a right subtree).

2.9 Solutions/Answers

Check your progress 1:

Answer1: A number of factors need to be taken into consideration when developing a statespace representation. Factors that must be addressed are:

- What is the goal to be achieved?
- What are the legal moves or actions?
- What knowledge needs to be represented in the state description?

- Type of problem - There are basically three types of problems. Some problems only need a representation, e.g., crossword puzzles. Other problems require a yes or no response indicating whether a solution can be found or not. Finally, the last type problem are those that require a solution path as an output e.g., mathematical theorems Towers of Hanoi. In these cases we know the goal state and we need to know how to attain this state

- Best solution vs. Good enough solution - For some problems a good enough solutionis sufficient. For example: theorem proving eight squares. However, some problems require a best or optimal solution, e.g., the traveling salesman problem.

Answer 2

(a) Formulation of Missionaries and Cannibal problem:

State: (#M,#C,0/1)

Where #M represents Number of missionaries in the left side bank (i.e., left side of the river)

#C : represents the number of cannibals in the left side bank (i.e., left side of the river)

0/1 : indicate the boat position of the boat. 0 indicates the boat is on the left side of the river and 1 indicate the boat is on the right side.

Start state:(3,3,0)

Goal State: (0,0,1)

Operator: Sate will be changed by moving missionaries and (or) cannibals from one side to another using boat. So, it can be represented as number of persons on the either side of the river. Note that the boat can carries maximum 2 persons.

Boat carries: (1,0) or (0,1) or (1,1) or (2,0) or (0,2).Here in (i,j), i represents number of missionaries and j means number of cannibals.

(b) Solution of Missionaries and Cannibal problem:

Start state:(3,3,0)

Goal State: (0,0,1)

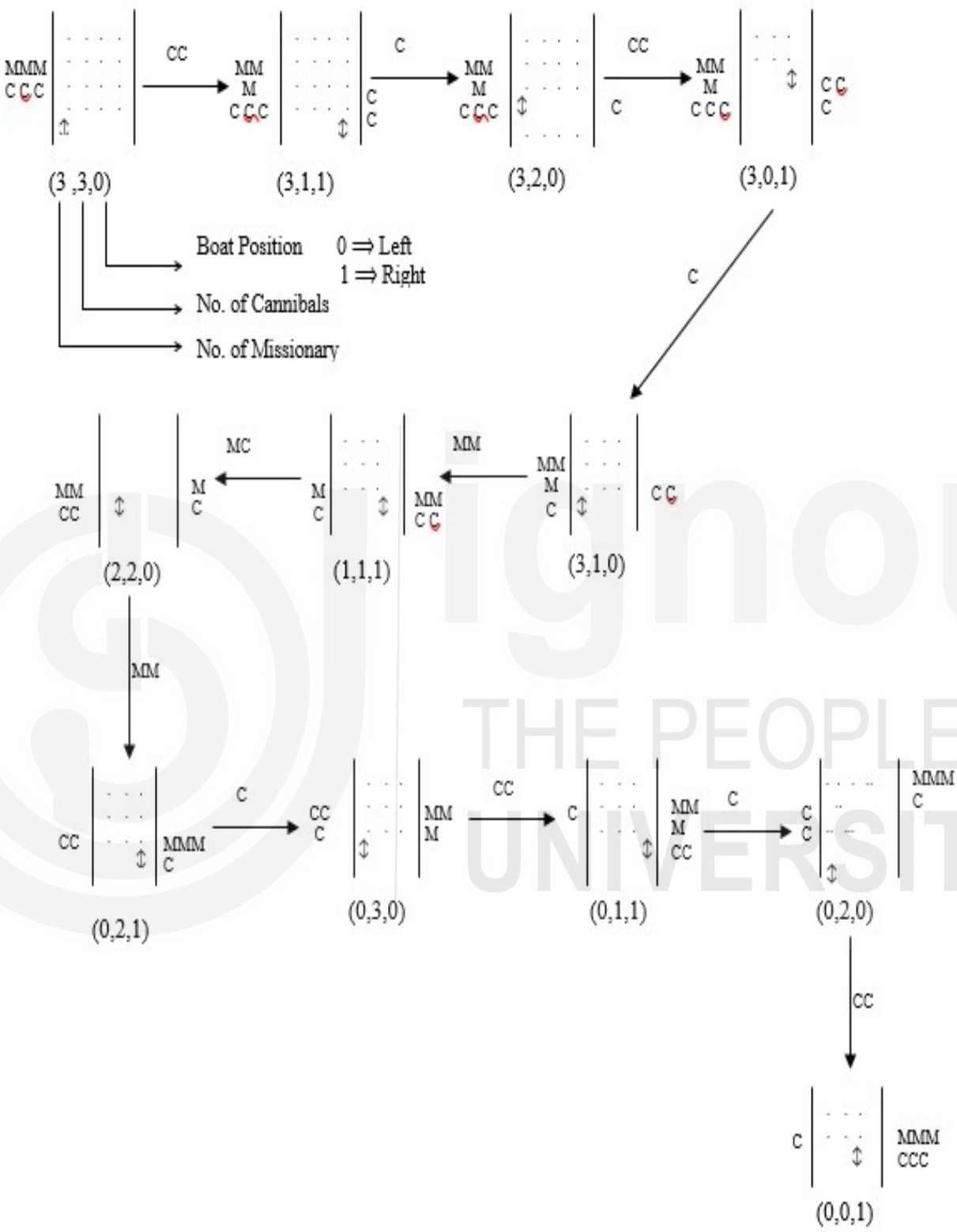


Figure: Solution of Missionaries and Cannibal problem

A state space tree for this problem is shown below:

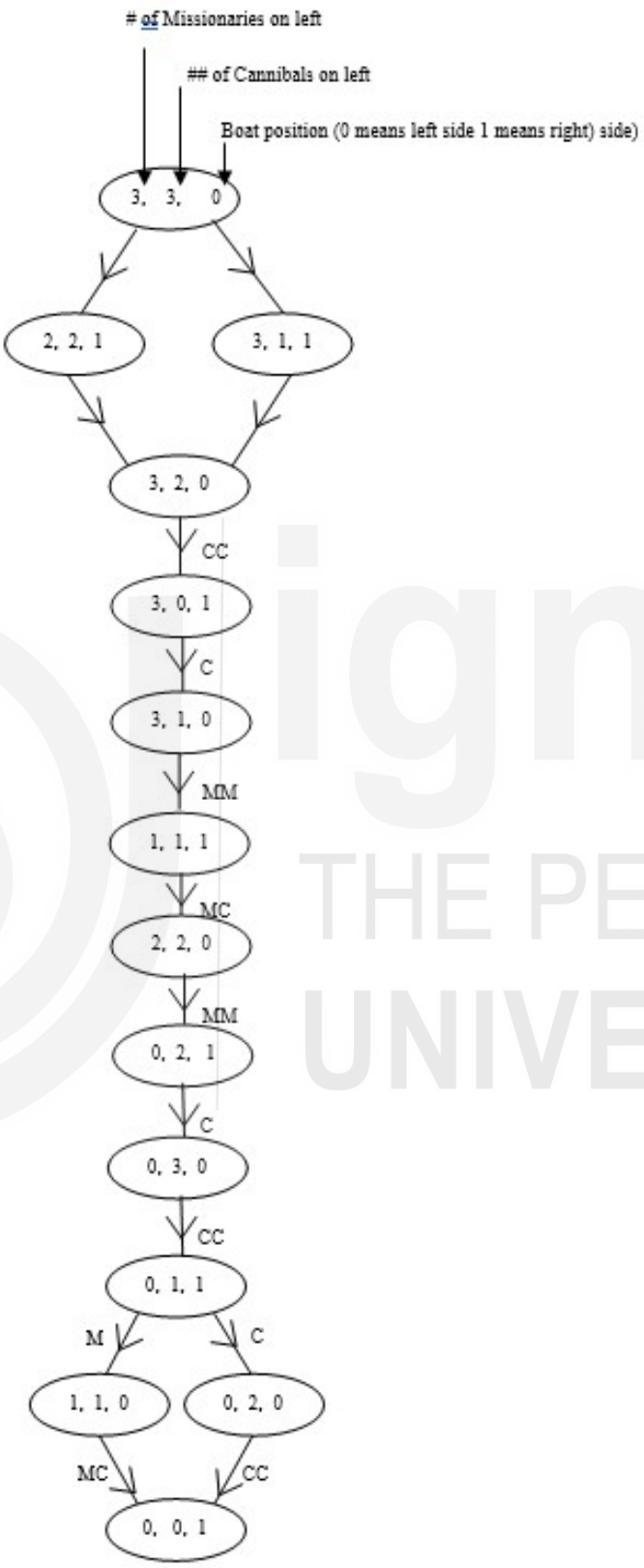


Figure: A state space tree showing all possible solution of missionaries and cannibal problem.

Answer 3: Towers Hanoi A possible state space representation of the Towers Hanoi problem using a graph is indicated in **Figure 1.**

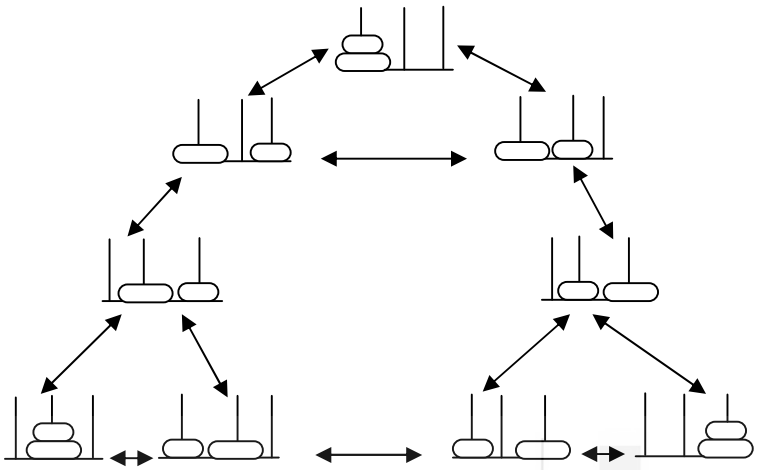
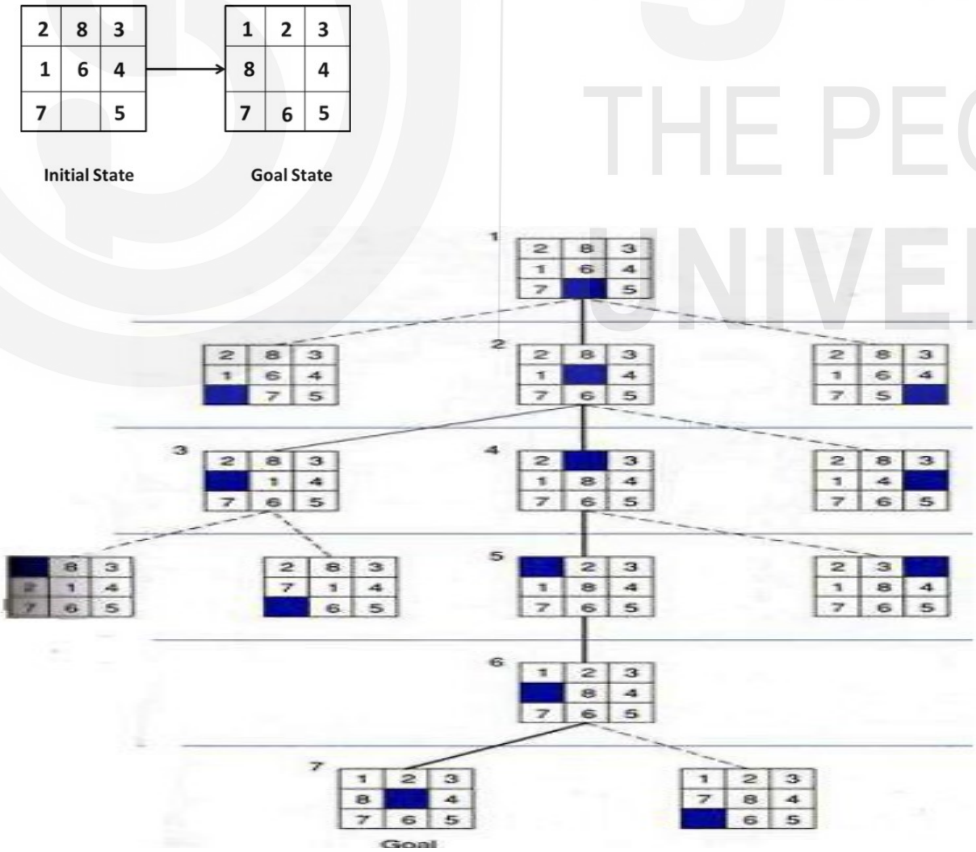


Figure 1: Towers of Hanoi slate space representation for n=2

The legal moves in this state space involve moving one ring from one pole to another, moving one ring at a time, and ensuring that a larger ring is not placed on a smaller ring.

Answer 4:



Minimum Cost path for solution= 6

Answer 5: Backtracking Algorithm: The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.

1) Start in the leftmost column

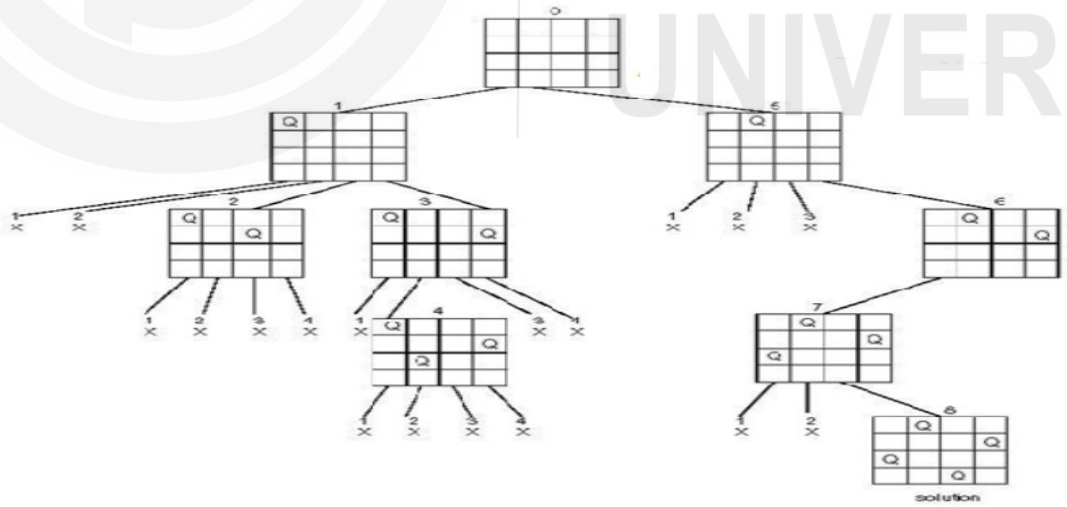
2) If all queens are placed
return true

3) Try all rows in the current column.
Do following for every tried row.

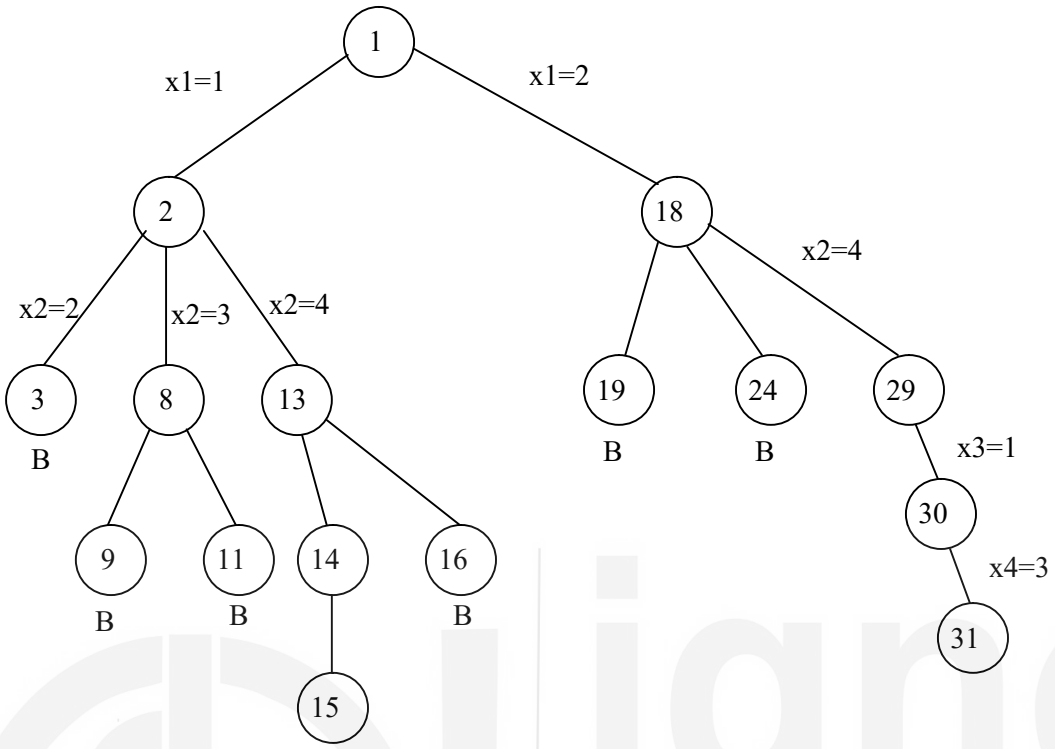
a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
b) If placing the queen in [row, column] leads toa solution then return true.
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
4) If all rows have been tried and nothing worked, return false to trigger backtracking.

State space tree for 4 Queen's problem

A state-space tee (SST) can be constructed to show the solution to this problem.The following SST (figure 1) shows one possible solution$\{x1, x2, x3, x4\} = \{2,4,3,1\}$for the 4 Queen's Problem.



Or we can also denote the State space tree as follows:



B denotes the Dead Node (nonpromising node). The figure 1 shows the Implicit tree for 4 queen problem for solution <2,4,1,3>. The Root represents an initial state. The Nodes reflect the specific choices made for the components of a solution. Explore The state space tree using **depth-first search.** "Prune" non-promising nodesdfs stops exploring subtree rooted at nodes leading to no solutions and then backtracks to its parent node

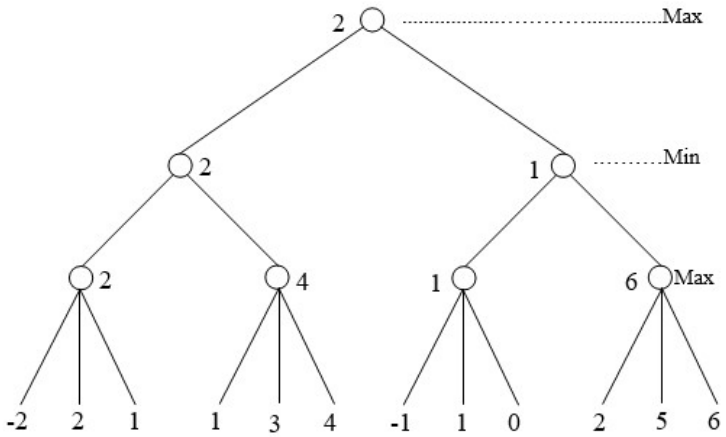
Check your progress 2

Answer1:

Solution: Alpha-beta pruning is an advance version of MINIMAX algorithm. The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity. If b is the branching factor and d is the depth of the tree, then time complexity of MINIMAX algorithm is $O(b^d)$ that is exponential.

Alpha-Beta pruning is a way of finding the optimal Minimax solution while avoiding searching subtrees of moves which won't be selected. The effectiveness of Alpha-beta pruning is highly dependent on the order in which each node is examined. **The** ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best move 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. The time complexity for best case order is $O(b^{m/2})$ (since we search only left sub tree, not a right subtree).

Answer 2: The final game tree with max and min value at each node is shown in the following figure.



Answer3:

Solution: Solve the question as sown in Example1

The initial call starts from A. We initially start the search by setting the initial value of $\boldsymbol{\alpha} = -\infty$and$\boldsymbol{\beta} = +\infty$for root node A. 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. To decide whether it's worth looking at its right node or not, it checks the condition $\alpha \geq \beta$. This is false since $\boldsymbol{\beta} = +\infty$ and ∞= 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. Value at node D=5, move up to node B(=5). Now at node B, β value will be modified as

$\beta = min(+\infty, 5) = 5.$

B now calls E, we pass the $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value from top node B to bottom node E as $[\boldsymbol{\alpha} = -\infty, \boldsymbol{\beta} = \mathbf{5}]$. **Since,** node E is at MAX level, so only $\boldsymbol{\alpha}$value will be change. Now, at E, it first checks the left child (which is a terminal node) with value 6. This node returns a value of 6.

Now, the value of $\boldsymbol{\alpha}$ **at** node E is calculated as $\alpha = max(-\infty, 6) = 6$, so value of Node(E)=6 and modified value of $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$ at node E is $[\alpha = 6, \beta = 5]$. To decide whether it's worth looking at its right node or not, we check $\boldsymbol{\alpha} \geq \boldsymbol{\beta}$. The answer is **YES**, since $6 \geq 5$. So, we prune (cut) the right branch of E, as shown in figure (a).

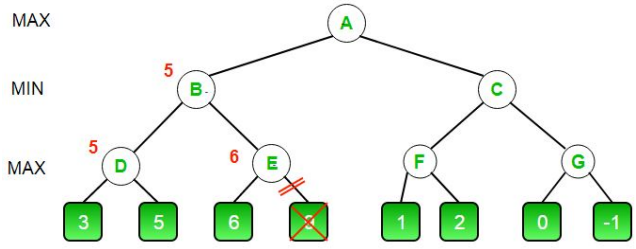


Figure (a) game tree after applying alpha-beta pruning on left side of node A

Similarly, we solve for right sub tree for Node A [refer the example 1 and solve for right sub tree part]. The final tree with node value at every node is shown in the figure(b).

Thus finally, $\boldsymbol{\alpha}$and$\boldsymbol{\beta}$value at node A is$[\alpha = 5, \beta = +\infty]$ and best value at Node(A)=max (5,2)=5. So, optimal value for the maximizer is 5 and there are 3 terminal nodes are pruned (9, 0 and -1). The optimal search path is A→B→D→5 , as shown in figure (b).

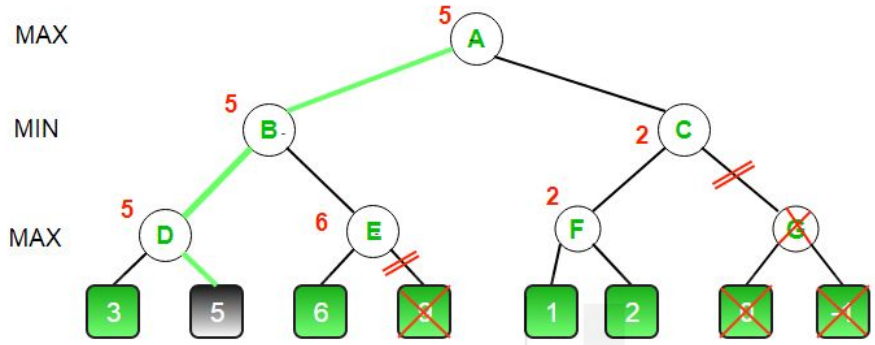
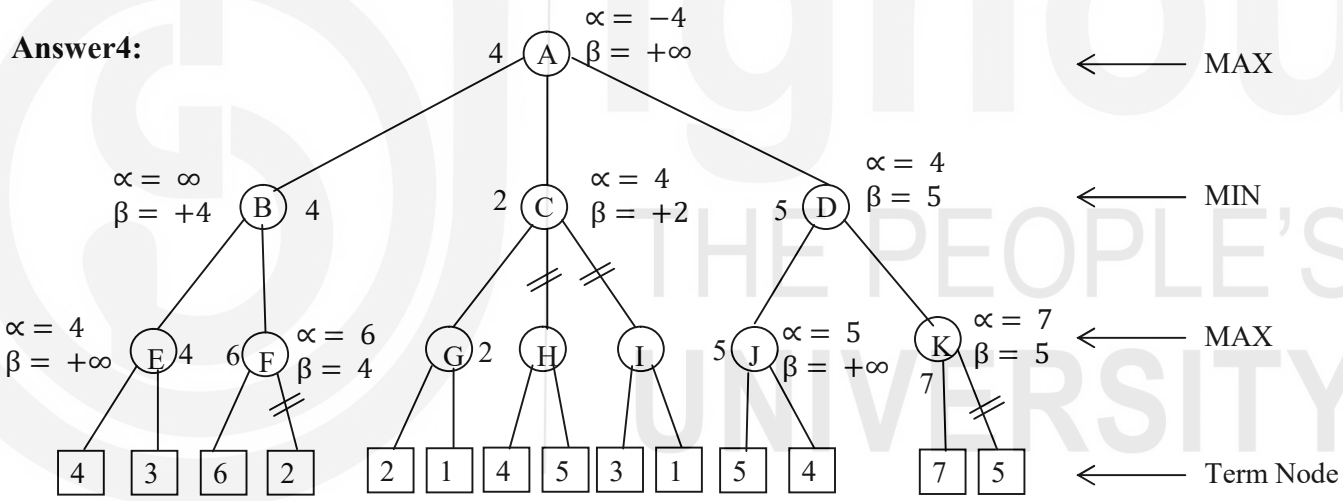


Figure (b) Final tree with node value on every node with prune branches

Answer4:



Answer5: 5

Answer6: 7

Answer 7: Option (B) 17

2.11 FURTHER READINGS

1. Ela Kumar, " Artificial Intelligence", IK International Publications
2. E. Rich and K. Knight, "Artificial intelligence", Tata Mc Graw Hill Publications
3. N.J. Nilsson, "Principles of AI", Narosa Publ. House Publications
4. John J. Craig, "Introduction to Robotics", Addison Wesley publication
5. D.W. Patterson, "Introduction to AI and Expert Systems" Pearson publication