Q3. A* Search algorithm

A* algorithm is one of the best and popular techniques used for path finding and graph traversals

A lot of games and web based maps use this algorithm for finding the shortest path efficiently

It is essentially a best first search algorithm

It uses heuristic function $h(n)$ and cost to reach node n from the start state $g(n)$

It has combined features of uniform cost search (UCS) and greedy best first search, by which it solves the problem efficiently.

It finds the shortest path through the search space using heuristic function. and provides optimal result faster

A* is similar to UCS, except that it uses $g(n) + h(n)$ instead of $g(n)$

$$f(n) = g(n) + h(n)$$

- $f(n)$: estimated cost of cheapest soln
- $g(n)$: cost to reach n from start state
- $h(n)$: cost to reach n from goal node.

Advantages →

- A* search algorithm is the best algorithm than other search algorithms.
- A* algorithm is optimal and complete
- This algorithm can solve very complex problems

Disadvantages →

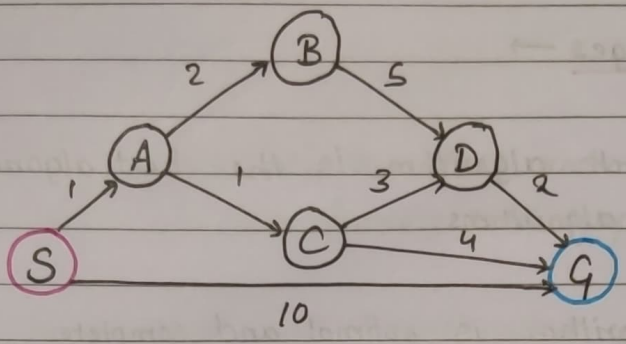
- It does not always produce shortest path as it is mostly based on heuristics and approximation.
- A* has some complexity issues
- Main drawback is memory requirement as it keeps all the generated nodes in the memory, so it is not very practical for large scale problems

Example →

h(n) is given for all states

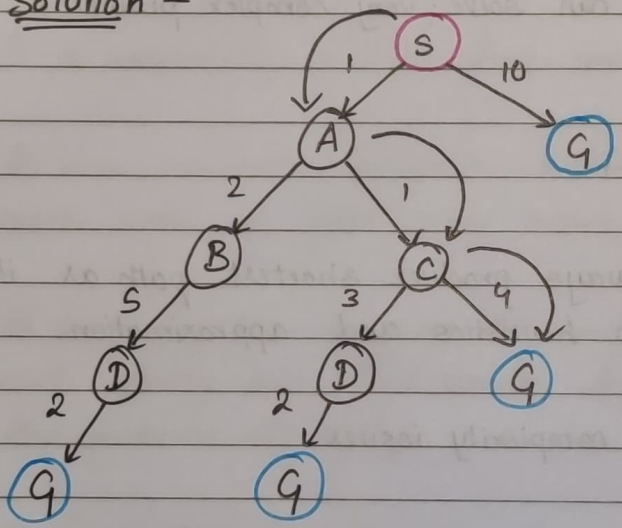
calculate f(n) using the formula $f(n) = g(n) + h(n)$

where g(n) is the cost to reach any node from start state.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution –



Initialization - {(S, 5)}

Iteration 1 - {(S→A, 4), (S→G, 10)}

Iteration 2 - {(S→A→B, 7), (S→A→C, 4), (S→G, 10)}

Iteration 3 - {(S→A→C→D, 11), (S→A→C→G, 6), (S→A→C, 4), (S→G, 10)}

Iteration 4 - S→A→C→G provides optimal path with cost 6

Properties →

A* algorithm is complete as long as
→ branching factor is finite
→ every action cost is fixed

A* algorithm is optimal if it follows the two conditions -
→ admissible: h(n) should be admissible heuristic for A* tree search. Admissible heuristic is optimistic in nature
→ consitency: second function required for optimal is consistency

If the heuristic function is admissible, A* tree search will always produce least cost path.

Time complexity → $O(b^d)$
where b = branching factor
d = solution depth

Space complexity → $O(b^d)$

Q7. Genetic algorithm

- It is a search heuristic inspired by Charle's Darwin's theory of natural evolution
- The algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation
- The genetic algorithm is a variation of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.
- In simple words, it simulates "survival of the fittest"
- Each generation consists of a population of individuals, and each individual represents a point in search space and a possible solution.
- Each individual is represented by a string of characters/integers/float/bits
- This string is ~~eal~~ analogous to the chromosome.

- The process of natural selection starts with the selection of the fittest individuals from a population.
They produce offspring which inherit the characteristics of the parents and will be added to the next generation
If parents have better fitness, their offspring will be better than parents and have a better chance of surviving
This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

- This notion can be applied for a search problem

- Five phases are considered in genetic algorithm →

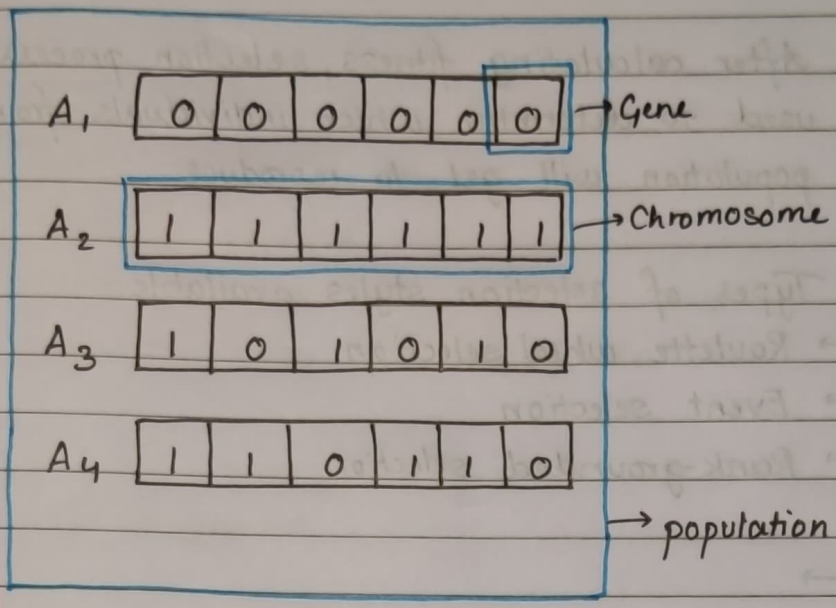
  → Initial population
  → Fitness function
  → Selection
  → Crossover
  → Mutation

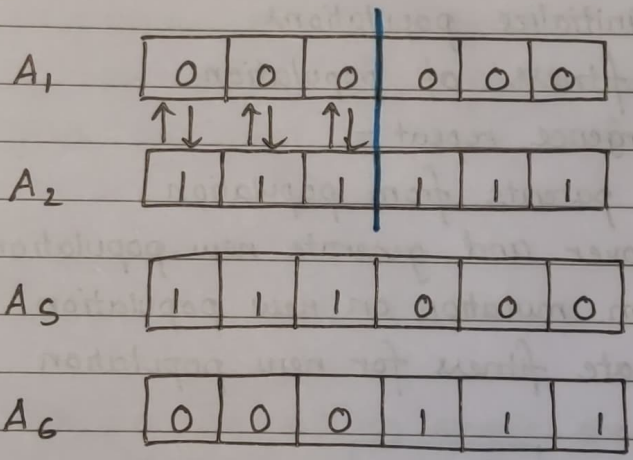
- Population: set of all possible or probable solutions which can solve the given problem

- Chromosome: one of the solutions in the ~~problem~~ population for the given problem

- Gene: A gene is an element of a chromosome. A chromosome is divided into genes.

- Allele: A value provided to a gene within a particular chromosome.





- Fitness function - used to determine the individual's fitness level in the population. It refers to the ability of the individual to compete with other individuals.

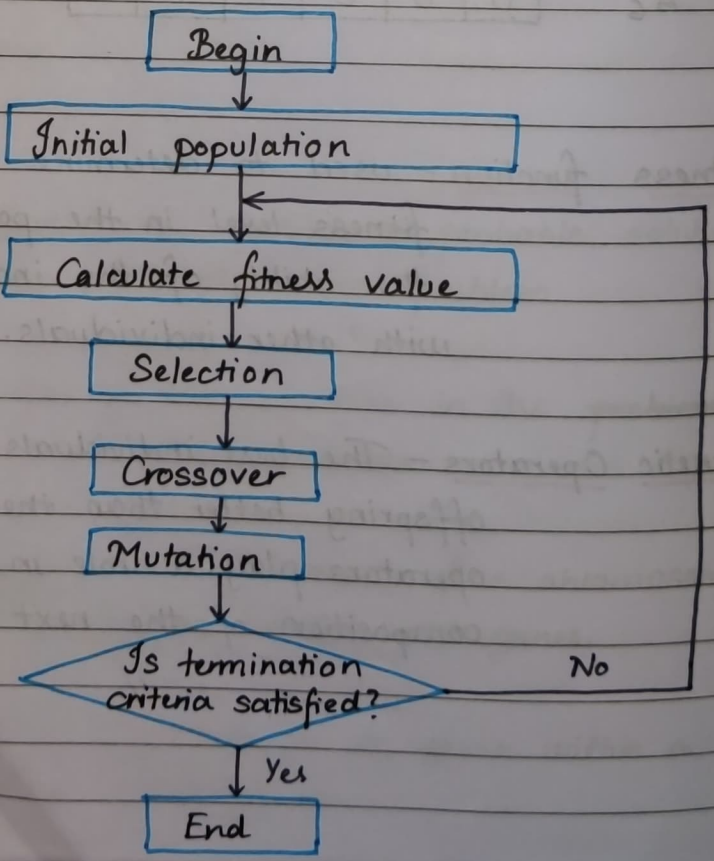
- Genetic Operators - The best individuals mate to regenerate offspring better than the parents, here genetic operators play a role in changing the genetic composition of the next generation.

- **Selection** - After calculating fitness, selection process is used to determine which individuals from the population will get to reproduce.

  Types of selection styles available -
  → Roulette wheel selection
  → Event selection
  → Rank-grounded selection

- **Algorithm** →

1. Randomly initialize populations
2. Determine fitness of population
3. Until convergence repeat -
   a) Select parents from population
   b) Crossover and generate new population
   c) Perform mutation on new population
   d) Calculate fitness for new population



- **Applications →**

- Recurrent neural network
- Mutation testing
- Code breaking
- Learning fuzzy rule base, etc.

- **Advantages →**

→ The parallel capabilities of genetic algos are best
→ helps optimize various problems
→ It provides solution for a problem that improves over time
→ It does not need derivative information.

- **Limitations →**

→ Not efficient for solving simple problems.
→ It does not guarantee the quality of the final solution to the problem
→ Repetitive calculation of fitness values may generate some computational challenges

Q5. Greedy best first search

- The simplest best first search strategy is to minimize the estimated cost to reach the goal, ie. always expand the node that appears to be closest to the goal.

- The closest cost is estimated by the heuristic function.

  $h(n)$ = estimated cost from a node n to the goal.

- A best first search that uses h to select the next node is called greedy search.

- Greedy best first always selects the path which appears best at the moment.

- It is the combination of depth first search (DFS) and breadth first search (BFS).

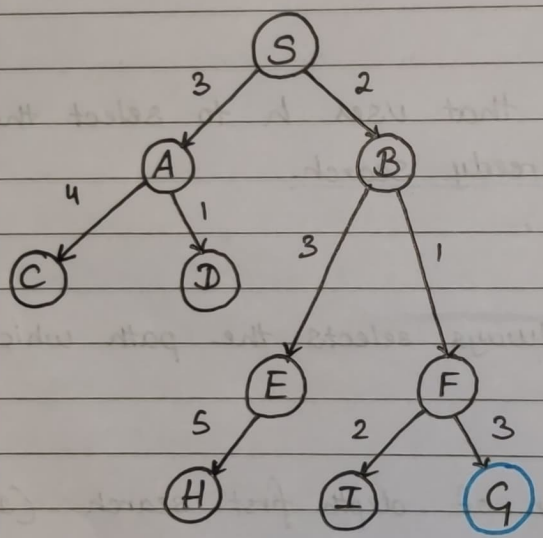
$$f(n) = h(n)$$

- The greedy best first search algorithm is implemented by the priority queue.

- At each step we must check for the node with the lowest $h(n)$ in the OPEN list and then move it to the CLOSED list to traverse it.

Example →

Consider the following search problem
At each stage, expand the node using $f(n) = h(n)$



node	h(n)
A	12
B	4
C	7
D	3
E	8
F	2
G	0
H	4
I	9

Two lists, OPEN and CLOSED are used.

Initialization : Open [A, B], Closed [S]
Iteration 1 : Open [A], Closed [S, B]
Iteration 2 : Open [A, E, F], Closed [S, B]
Open [E, A], Closed [S, B, F]
Iteration 3 : Open [E, A, I, G], Closed [S, B, F]
Open [E, A, I], Closed [S, B, F, G]

∴ Final solution path will be ⇒

$S \rightarrow B \rightarrow F \rightarrow G$

Properties →

- Time complexity : $O(b^m)$
- Space complexity : $O(b^m)$

  where m = maximum depth of search space.
- Complete : X (incomplete even if search space is finite)
- Optimal : X (not optimal)

Advantages →

→ Simple and easy to implement, since Greedy ~~breadth~~ best first search is relatively straightforward

→ Fast and efficient, making it ideal for applications where speed is essential.

→ Low memory requirements, suitable for application with limited memory

→ Flexible, it can switch between BFS and DFS by gaining advantage of both algorithms.

Disadvantages →

→ Inaccurate results, it is not always guaranteed to find the optimal solution as it is only concerned with choosing the path that appears most promising.

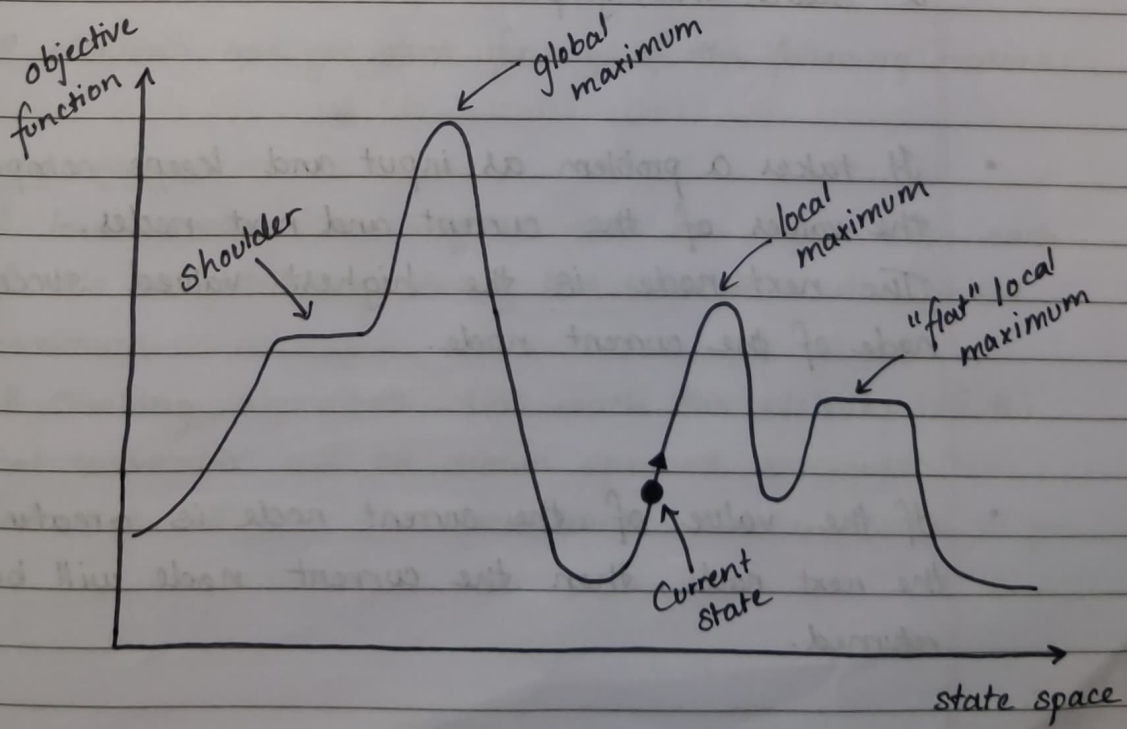
→ It may get stuck in a Local Optima, and the found solution may not be the best possible path.

→ It requires Heuristic function to work, which adds more complexity to the algorithm

Applications → Path finding
Machine learning
Optimization.

Q6. Hill Climbing search

- Each point in a state-space landscape is called a state
- Each state has an elevation which is defined by the heuristic cost function or objective function.
- If elevation corresponds to cost then the aim is to find the lowest valley - global minimum
- This is called gradient descent
- If the elevation corresponds to objective function then the aim is to find the highest peak - global maximum.
- This process is called hill climbing.



- The above diagram shows a one dimensional state-space landscape in which elevation corresponds to objective function and the aim is to find global maxima.

HILL CLIMBING SEARCH →

- It is a local search algorithm which continually moves towards the direction of increasing elevation/ value. (direction that provides steepest ascent)

- It terminates when it reaches a peak where no neighbour has a higher value.

- It is also called as greedy local search as it only looks for the best immediate neighbour of the current state and not beyond that

- It only keeps the current state instead of maintaining a search tree/graph

- It takes a problem as input and keeps comparing the values of the current and next nodes. The next node is the highest valued successor node of the current node.

- If the value of the current node is greater than the next node, then the current node will be returned.

- Hill climbing is the variant of the generate and test method
    1. Generate possible solution
    2. Test to see if this is the expected solution.
    3. If soln. has been found, quit
       else go to step 1.

The generate and test method produces feedback which helps decide which direction to move in the search space.

- Greedy approach: hill climbing algorithm search moves in the direction which optimizes the cost.

- No backtracking: It does not backtrack the search space, as it does not remember the previous states.

Limitations →

Hill climbing can get stuck for any of the following reasons:

- Local maxima -
  A local maxima is a peak that is higher than each of it's neighbouring states, but lower than the global maximum.
  Hill climbing algorithms that reach the vicinity of the local maximum will be drawn upward towards the peak, but will then be stuck with nowhere else to go.

- Ridges -
It results in a sequence of local maxima that is very difficult for greedy algorithms to navigate.

- Plateaus -
It is a flat area of the state space landscape
It can be a flat local maximum from which no uphill exists, or it can be a shoulder from which progress is possible.
A hill climbing search can get lost wandering on the plateau.

Types of hill climbing algorithms →

- Simple hill climbing -
It examines the neighbouring nodes one by one and selects the first node which optimizes the current cost, as the next node.

- Steepest - ascent hill climbing -
Examines all the neighbouring nodes and then selects the node closest to the solution state as the next node.

- Stochastic hill climbing -
It does not examine all neighbour states before moving. Rather it selects one neighbour at random and ~~chooses~~ decides whether to choose it as current state or examine another state

Applications of hill climbing algo → marketing
→ robotics
→ job scheduling

Q2. MinMax Algorithm, Alpha Beta pruning (Adverserial Search)

MinMax algorithm is a recursive or backtracking algorithm which is used in decision making and game theory

It uses recursion to search through game-tree

MinMax algorithm is mostly used for game playing in AI, such as chess, checkers, tictactoe etc.

In this algorithm two players play the game, one is called MAX and the other is called MIN, who are opponents

MAX will select the maximized value and MIN will select the minimized value.

The algorithm performs depth first search for exploration of complete game tree. It proceeds all the way down to the terminal node of the tree and then backtracks the tree until the initial state occurs.

MAX will try to get the maximum possible score and MIN will try to get the minimum possible score

initial state → includes board position, identifies the player to move

successor function → returns a list of legal moves and resulting state

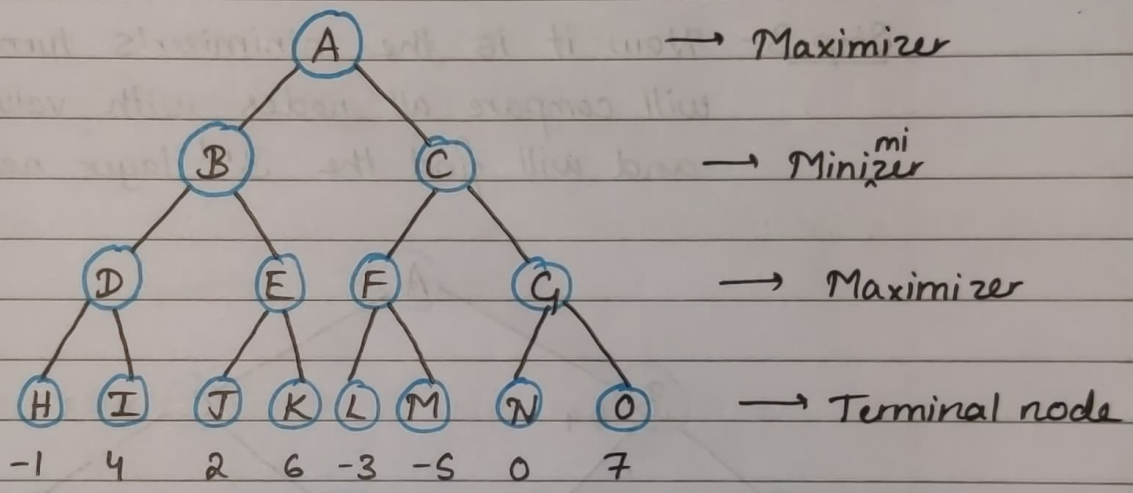
terminal test → describes the states where game is over

utility function → gives numerical value for terminal states.

In chess, outcome is win, loss or draw, with values +1, -1, 0.

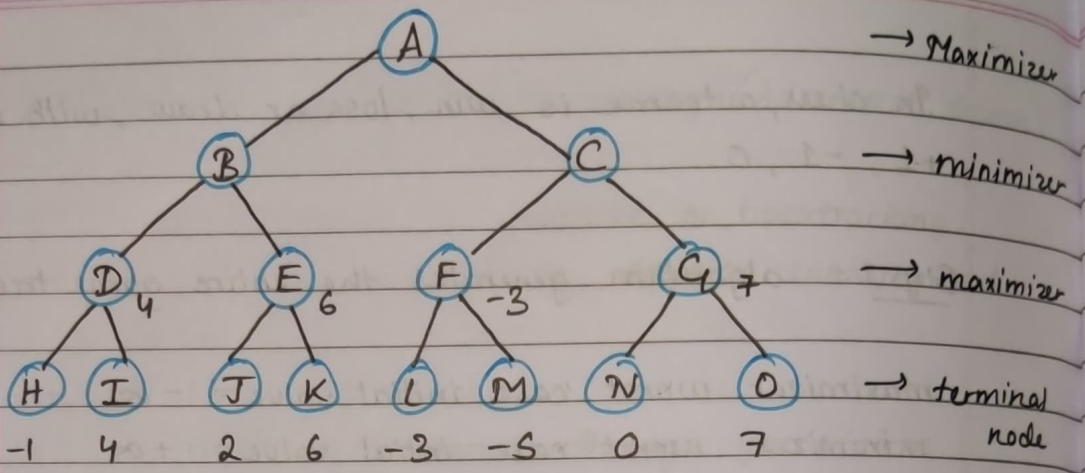
Step 1 - algorithm generates the entire game tree

maximizer worst case initial value = $-\infty$
minimizer worst case initial value = $+\infty$

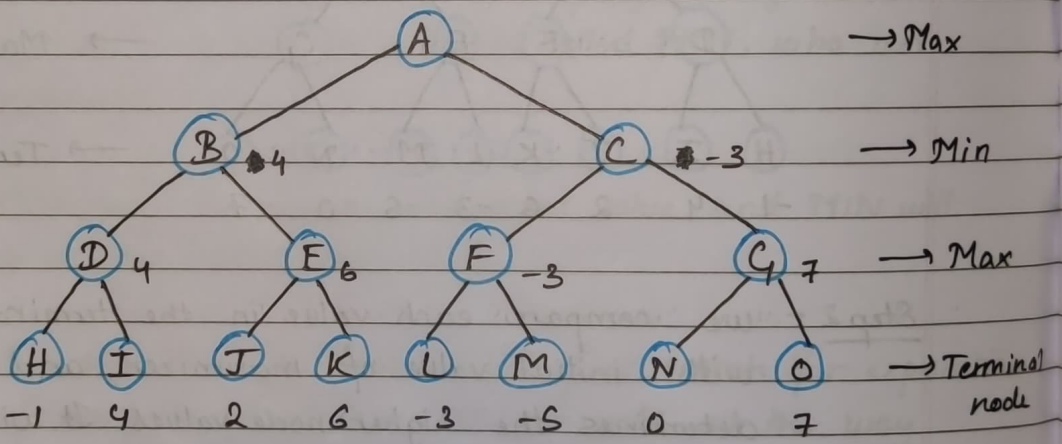


Step 2 - we compare each value in the terminal state with initial value of maximizer, and it determines the higher node values. It will find the maximum among all

for Node D → max (-1, $-\infty$) ⇒ max (-1, 4) = 4
for Node E → max (2, $-\infty$) ⇒ max (2, 6) = 6
for Node F → max (-3, $-\infty$) ⇒ max (-3, -5) = -3
for Node G → max (0, $-\infty$) ⇒ max (0, 7) = 7



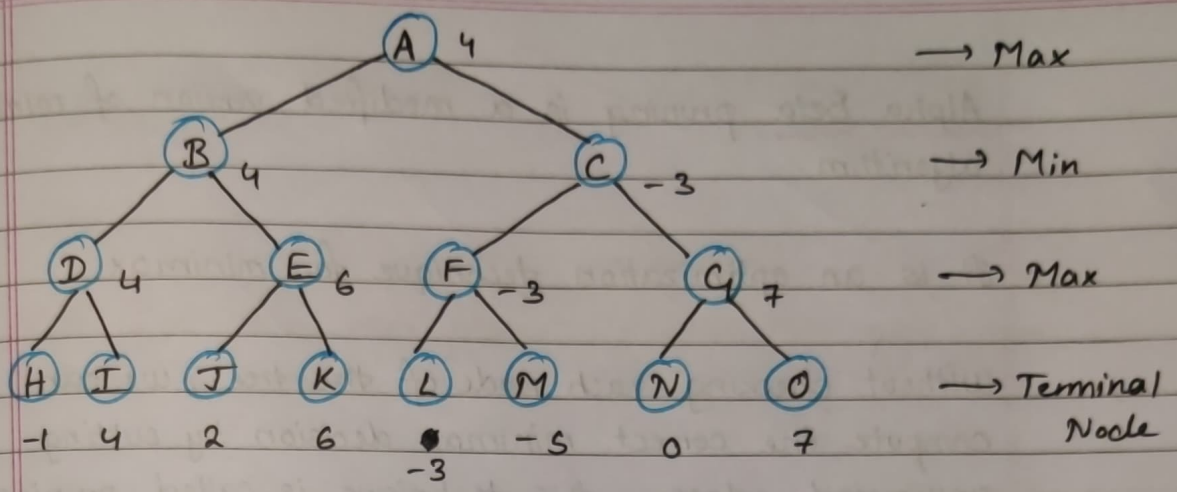
Step 3 - Now it is the minimizer's turn, it will compare all nodes with value $+\infty$ and will find the 3rd layer node values



for Node B => min (4,6) = 4
for Node C => min (-3,7) = -3

Step 4 - Now it is MAX turn, it will now find maximum value for root node
In this game tree there are only 4 layers hence we reach root node immediately, but in real games there will be more than 4 layers.

For node A → max(4,-3) => 4



Properties of MiniMax Algorithm →

- Complete – It is complete, and it will definitely find the solution (if exists)
- Optimal – It is optimal if both opponents are playing optimally
- Time complexity – As it performs DFS, time complexity is $O(b^m)$ where b = branching factor and m is maximum depth of tree
- Space complexity – Also similar to DFS ⇒ $O(bm)$

Limitations –

- Slow for complex games like chess.
- Huge branching factor, and player has lot of choices to decide
- This limitation of minimax algo can be improved from alpha beta pruning.

Alpha Beta pruning is a modified version of minimax algorithm.

It is an optimization technique for minimax

Without checking each node of the tree, we can compute the correct minimax decision by cutting unwanted edges, this technique is called pruning, it involves removing nodes which are not participating in final decision

2 parameters →

- Alpha : best ~~or~~ highest value choice found so far at any point along the path of maximizer.
  Initial value = $-\infty$

- Beta : best lowest value choice we found so far at any point along the path of the minimizer
  Initial value = $+\infty$

Main condition required for alpha beta pruning is →

$$\alpha >= \beta$$

Max player will only update value of α, and Min player will only update value of β.

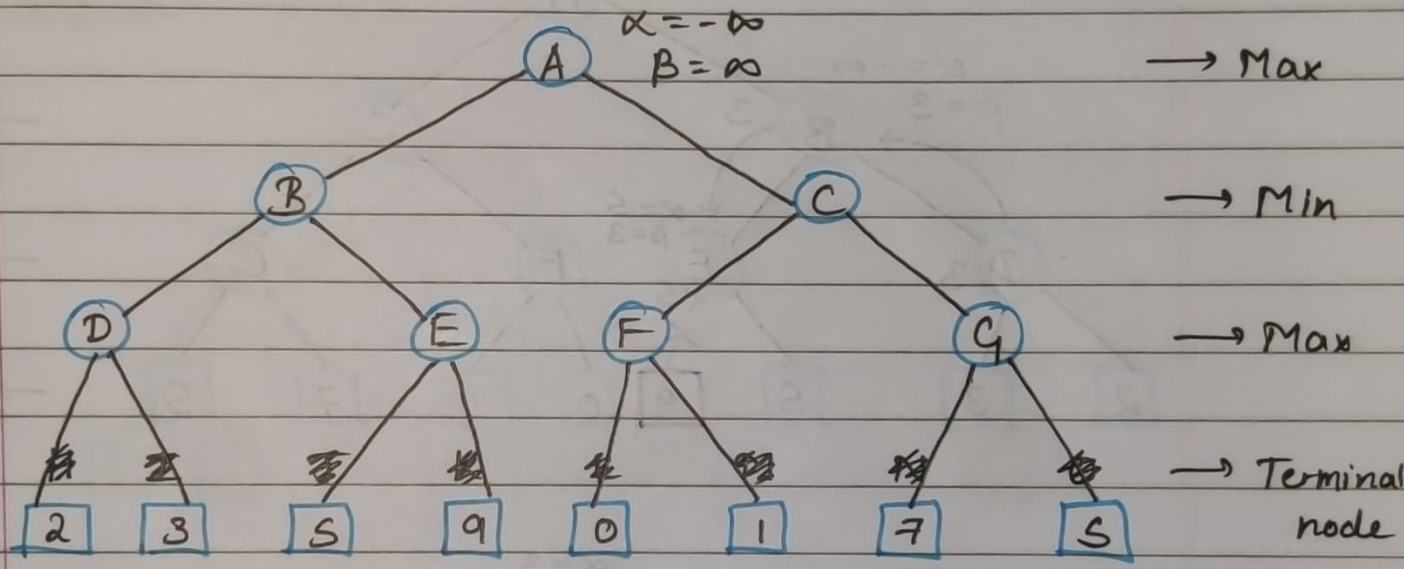
alpha beta values will only be passed to child nodes

☆ Example →

Step 1 -

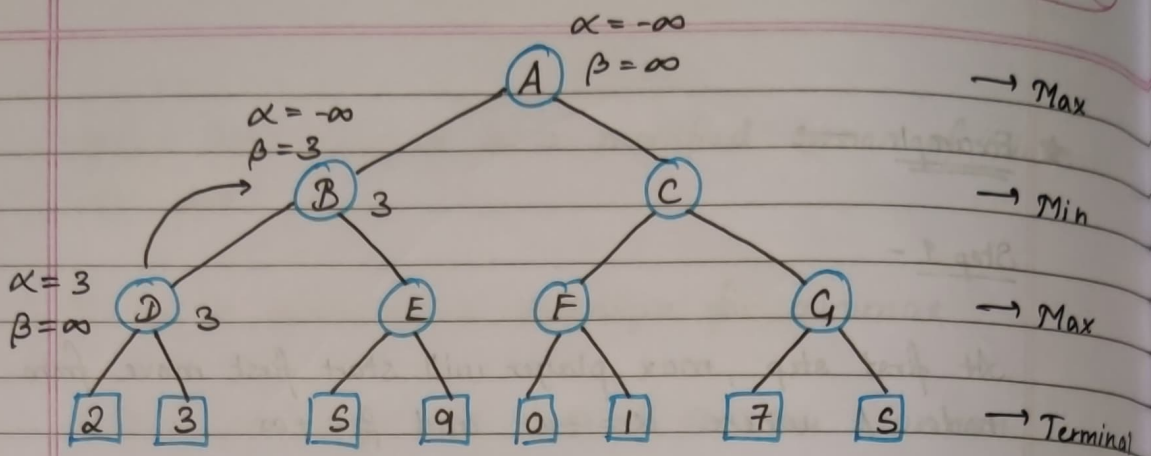
At first step, max player will start first move from node A where $\alpha = -\infty$ and $\beta = +\infty$

these values will be passed down to node B, who passes the same down to node D (it's child)

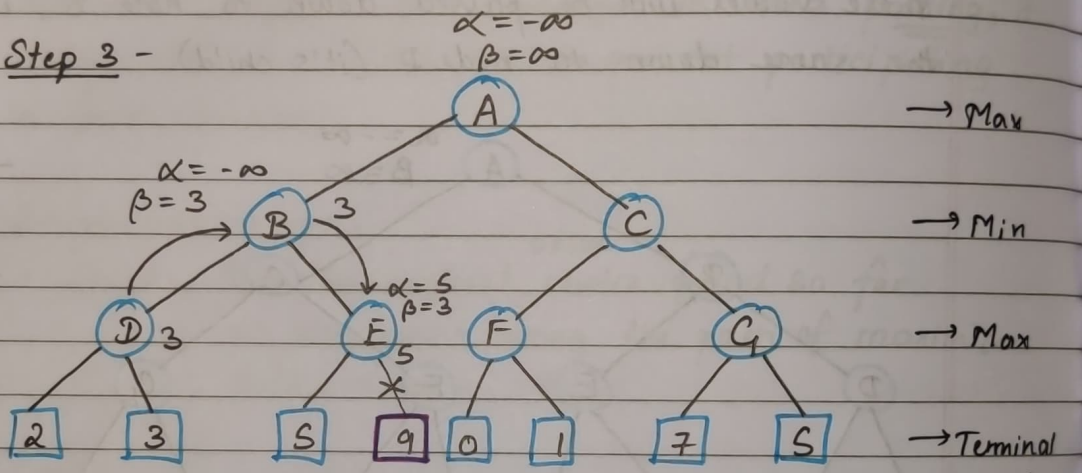


Step 2 -

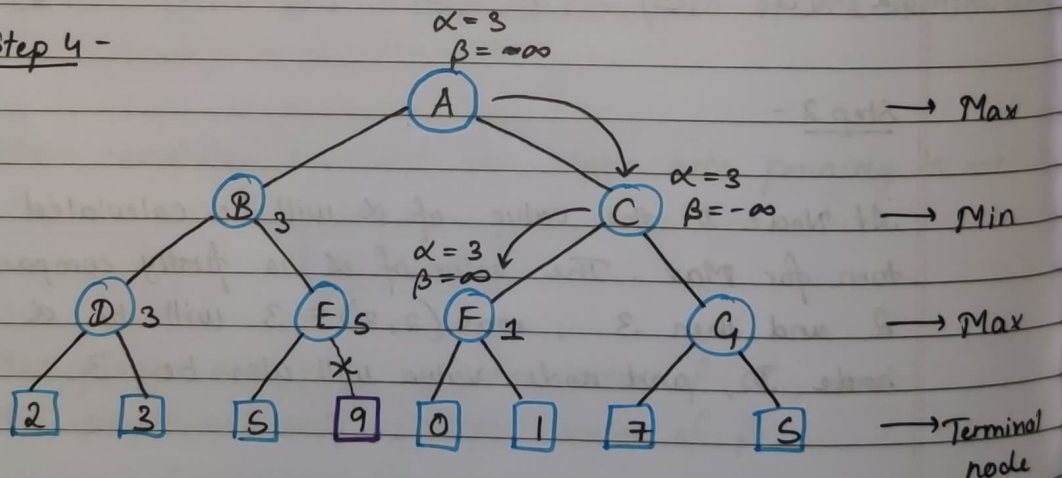
At Node D the value of α will be calculated as it is turn for Max. The value of α is firstly compared with 2 and then 3, $\max(2,3) = 3$ will be α value at node D, and node value will also be 3.


α = -∞
β = ∞
A
α = -∞
β = 3
B 3
C
α = 3
β = ∞
D 3
E
F
G
2
3
5
9
0
1
7
5
→ Max
→ Min
→ Max
→ Terminal node

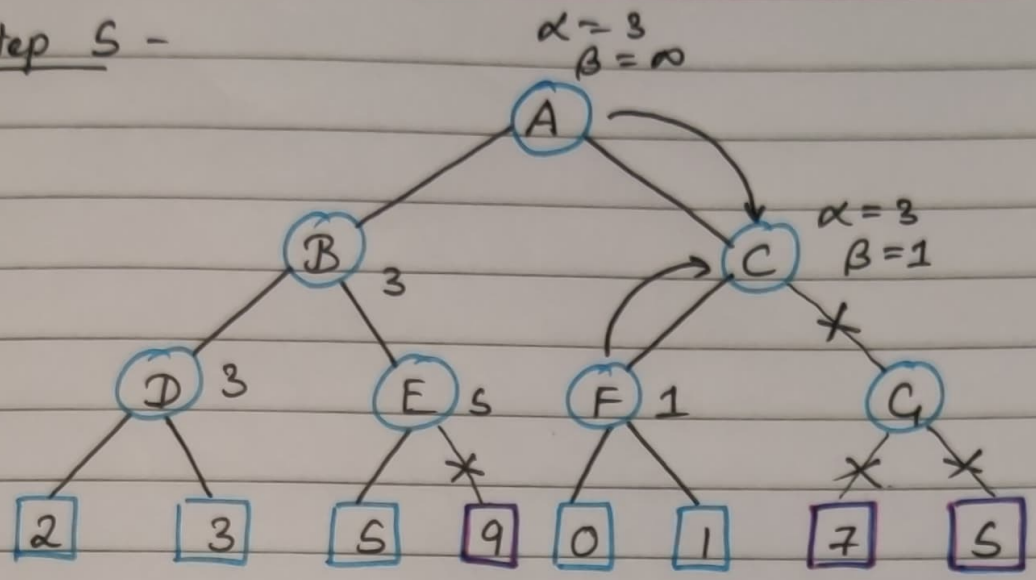
Step 3 -


α = -∞
β = ∞
A
α = -∞
β = 3
B 3
C
α = 5
β = 3
D 3
E 5
F
G
2
3
5
9
0
1
7
5
→ Max
→ Min
→ Max
→ Terminal node

Step 4 -

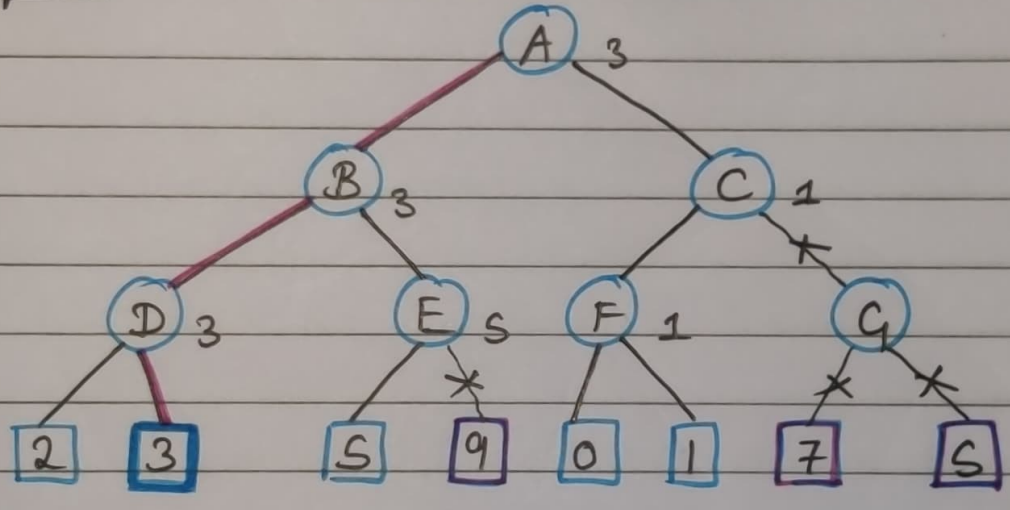

α = 3
β = ∞
A
B 3
α = 3
β = -∞
C
α = 3
β = ∞
D 3
E 5
F 1
G
2
3
5
9
0
1
7
5
→ Max
→ Min
→ Max
→ Terminal node

Step 5 -



→ Max
→ Min
→ Max
→ Terminal node

Step 6 -



→ Max
→ Min
→ Max
→ Terminal node