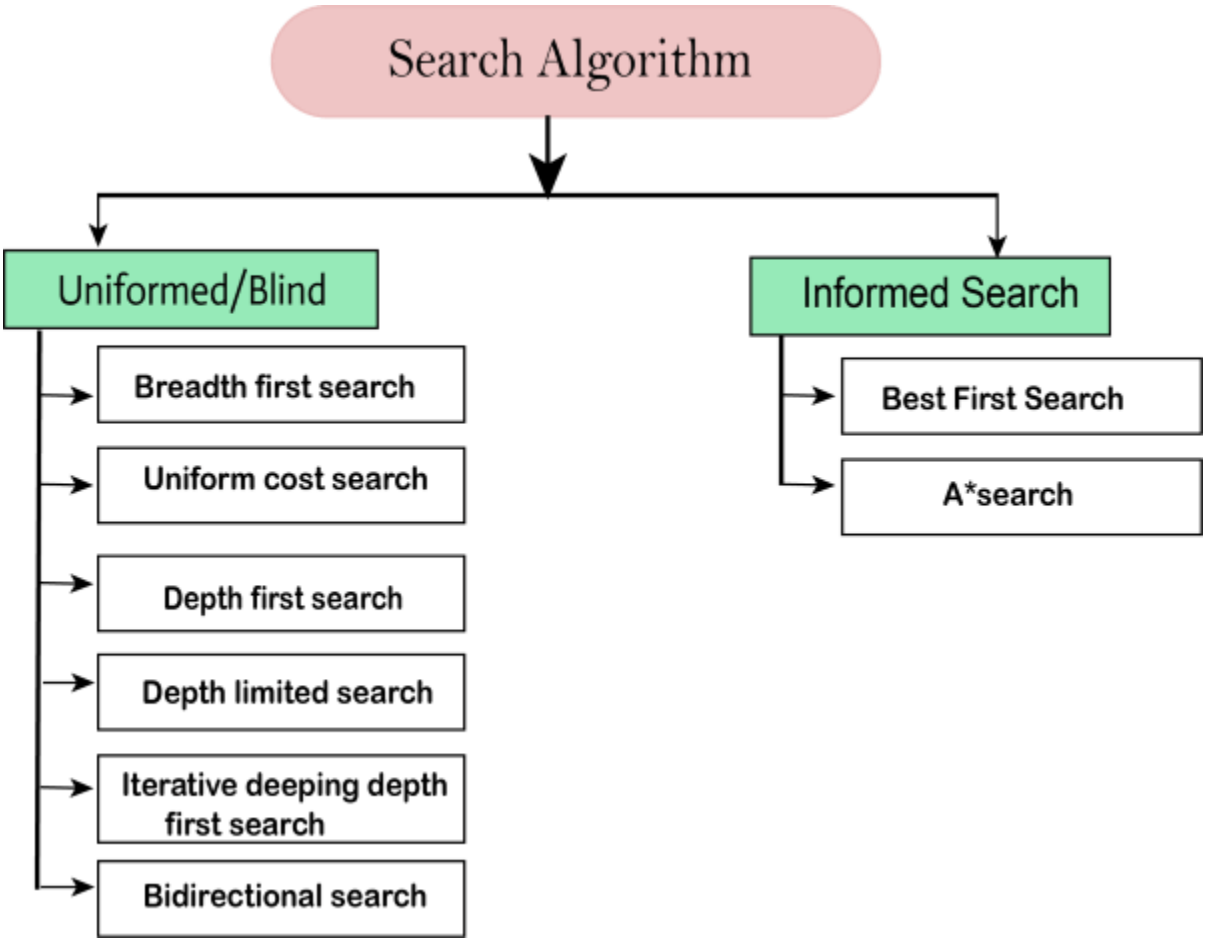
Unit 3

Searching in Ai

TYPES OF SEARCH ALGORITHM

Based on the search problem we can classify the search algorithm into :

- ➔ Uninformed search (Blind search)
- ➔ Informed search (Heuristic search)



Uninformed search: The uninformed search does not contain any domain knowledge such as closeness, the location of the goal.

- ➔ It operates in a brute force way, as it only includes information about how to traverse the tree and how to identify the leaf node and goal node.
- ➔ Uninformed search applies a way in which search tree is searched without any information about the search space. Like initial state operators and test for the

goal, so it is called blind search. Examine each node until it achieves the goal node.

Informed search:

- ➔ It uses domain knowledge; the problem information is available which can guide the search.
- ➔ Informed search strategies can find a solution more efficient than an uninformed, informed search is also called Heuristic search.
- ➔ A heuristic is a way which might not always be guarantee for the best solution but guarantee to find a good solution in reasonable time.
- ➔ It can solve many complex problems which could not be solved in another way.

Uninformed search:

1. **Breadth first search**
   a. It is the most common search strategy for traversing a tree or graph.
   b. This algorithm searches breadthwise in a tree or graph so it is called breadth first search.
   c. BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to node of next level.
   d. BFS algorithm is an example of a general graph search algorithm.
   e. BFS implemented using FIFO queue data structure.

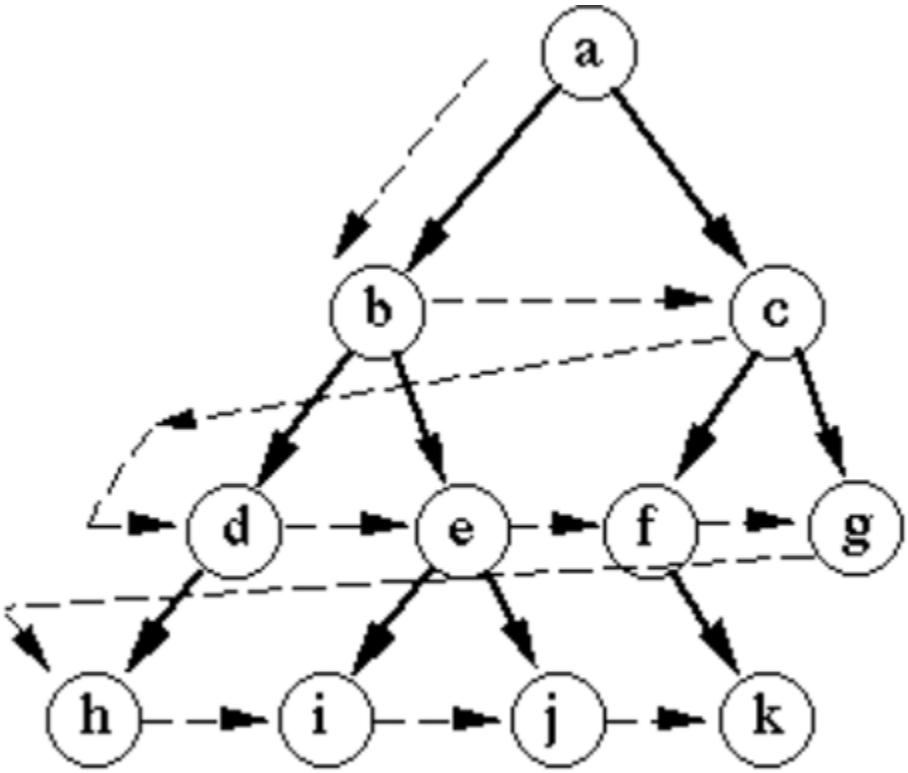
advantages:

a. BFS will provide a solution if any solution exists.
b. If there is more than one solution for a given problem then BFS will provide a minimal solution which requires the least number of steps .

Disadvantages

a. It requires a lot of memory since each level of tree must be saved into memory to expand to the next level.
b. BFS needs lots of time if the solution is far away from the root node.

Example:



Breadth-first search

Here let d= depth b is a node at every state so $T(b) = 1 + b^2 + b^3 + \ldots\ldots + b^d$

: -time complexity $T(b) = O(b^d)$

Space complexity $S(b) = O(b^d)$

Completeness: bfs is complete because we reach the goal node if solution exists.

Optimality: it is optimal

Depth first search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.

- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

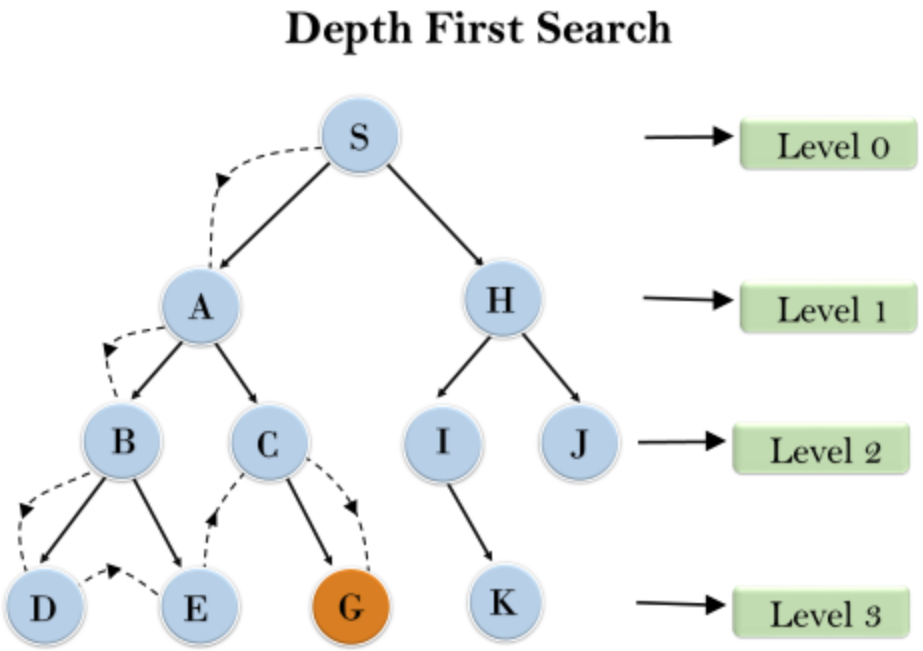
- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example :

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$(n)= 1+ n^2+ n^3 +.....+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **$O(b^m)$**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

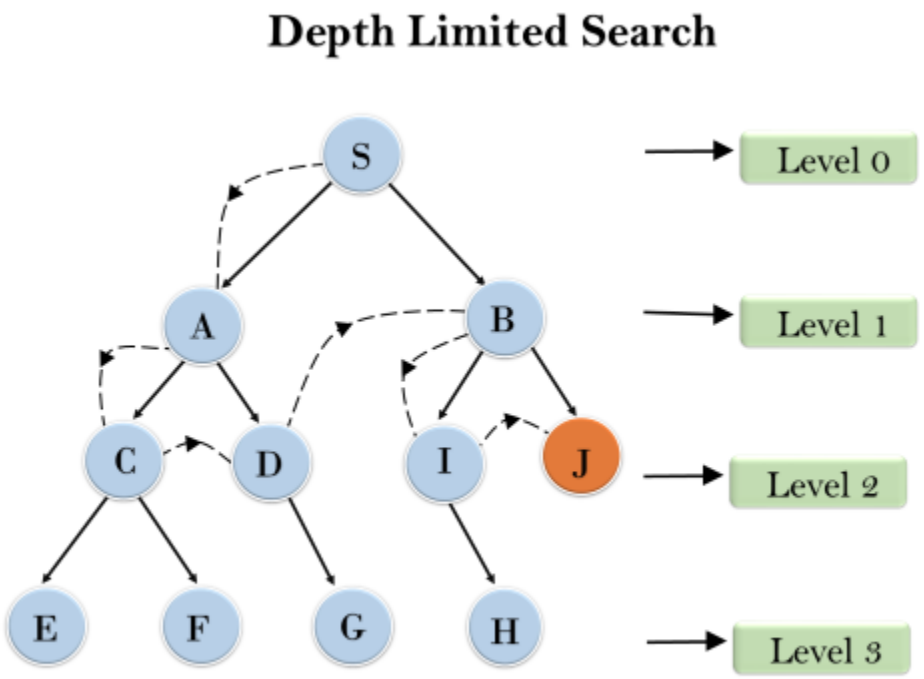
Depth limited search

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example :



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **O(b^{ℓ})**.

Space Complexity: Space complexity of DLS algorithm is O**(b×ℓ)**.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

4. uniform cost search algorithm

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It

gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

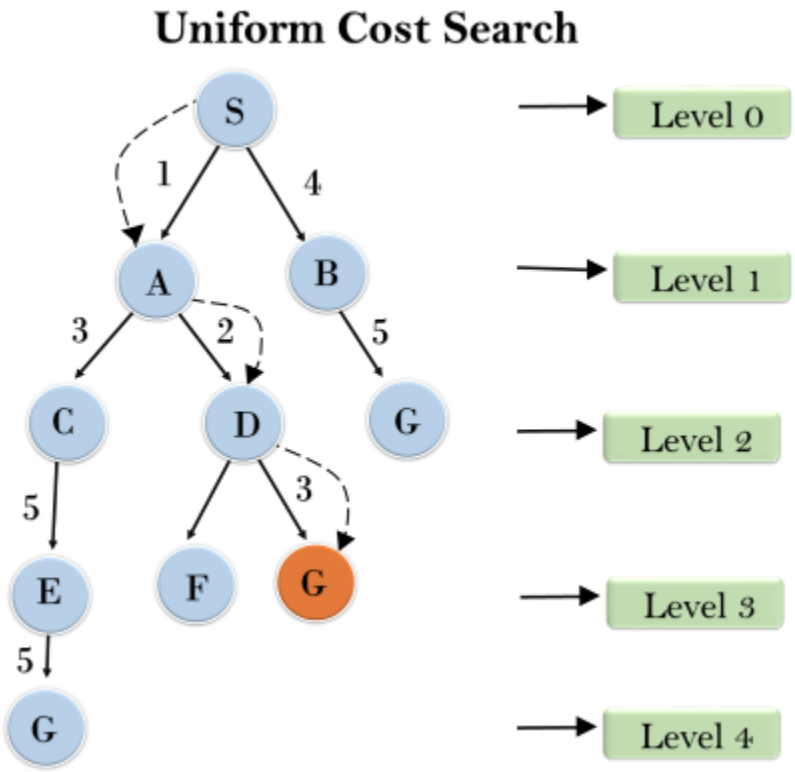
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Example:



Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is $\mathbf{O(b^{1 + [C^*/\varepsilon]})}$.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $\mathbf{O(b^{1 + [C^*/\varepsilon]})}$.

Optimal: Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

5. iterative deepening depth first search algorithm:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

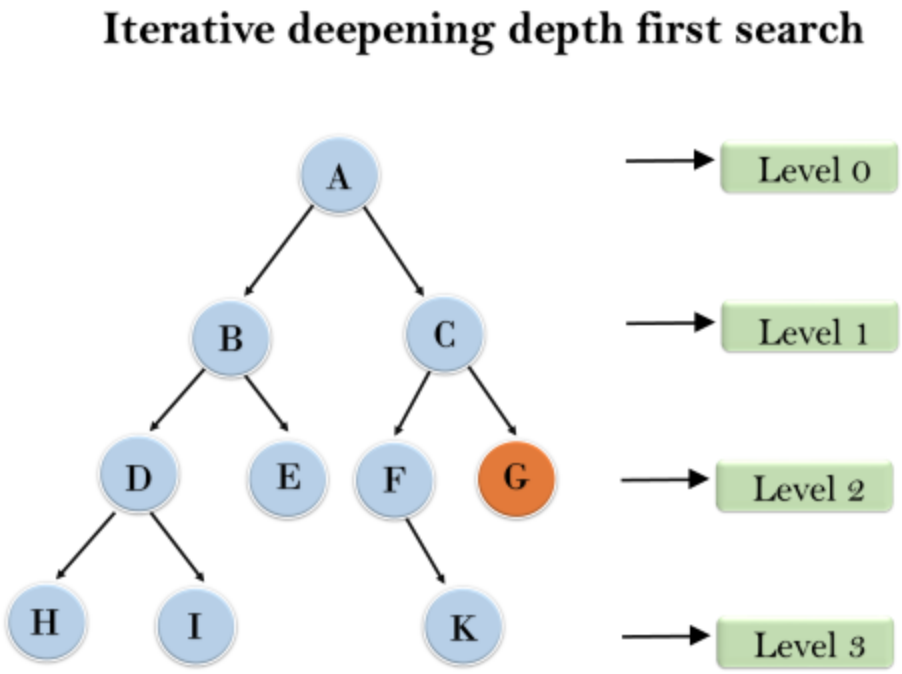
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G
4'th Iteration------>A, B, D, H, I, E, C, F, K, G
In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete is if the branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**.

Space Complexity:

The space complexity of IDDFS will be **$O(b^d)$**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

Informed search algorithm

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm

contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

1. h(n) <= h*(n)

Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search:

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

What is heuristic function?

➔ the information about the problem which can sometime help us to guide search more efficiently.

a. the nature of the states.
b. The cost of transforming from one place to another.
c. The promise of taking a certain path.
d. The characteristics of goal.

→ this information is often expressed in form of a heuristic evaluation function f(n,g) where n= nodes and g= goal .

→ hence , heuristic function have additional knowledge of the problem which is important to search algorithm.

2.) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently.

A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster.

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

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

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

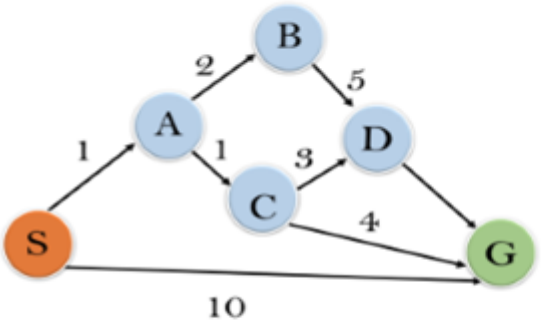
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.

- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

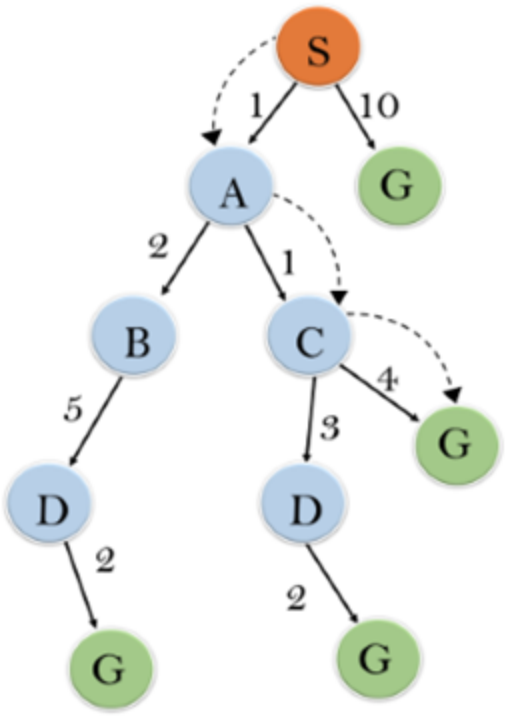
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and CLOSED list.



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

Solution:



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

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

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

Hill Climbing Algorithm in Artificial Intelligence

This search technique can be used to solve problems that have many solutions, some of which are better than others. It starts with a random solution, and iteratively makes small changes to the solution, each time improving a little. When the algorithm cannot see any improvement anymore, it terminates. It can be described as follows:

a. Evaluate the initial state, if it is goal state then terminate. Otherwise, current state is initial state.

b. Select a new operator for this state and generate a new state.

c. Evaluate new state:

    # if it is closer to goal , make it current state.

    # if it is not closer to goal , ignore.

d. If the current state is goal state than terminate. Otherwise repat from step b.

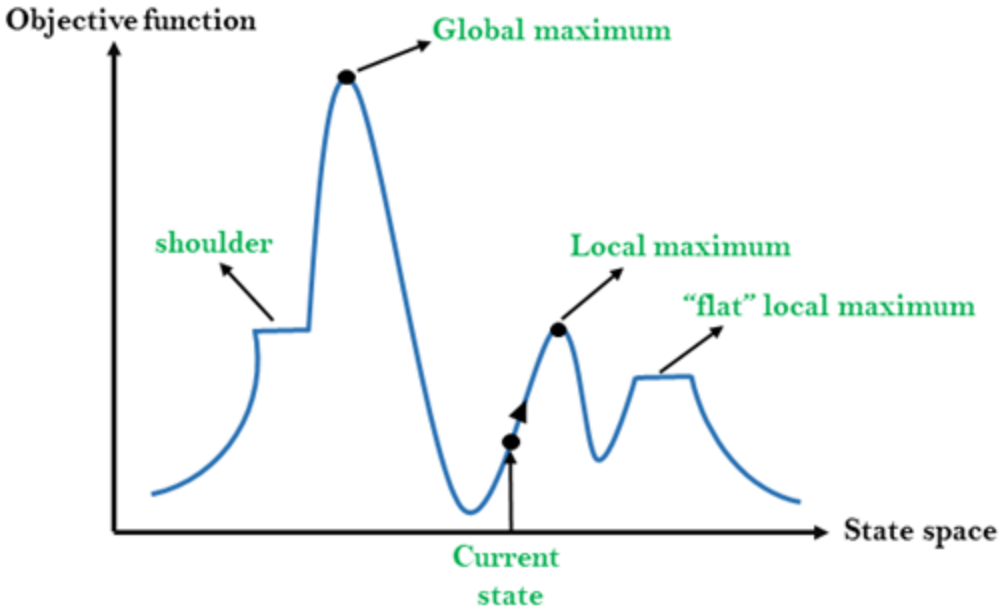
Hill climbing terminates when there are no successor of the current state which are better than the current state itself.

Features of Hill Climbing:

Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to 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.

State-space Diagram for Hill Climbing:



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

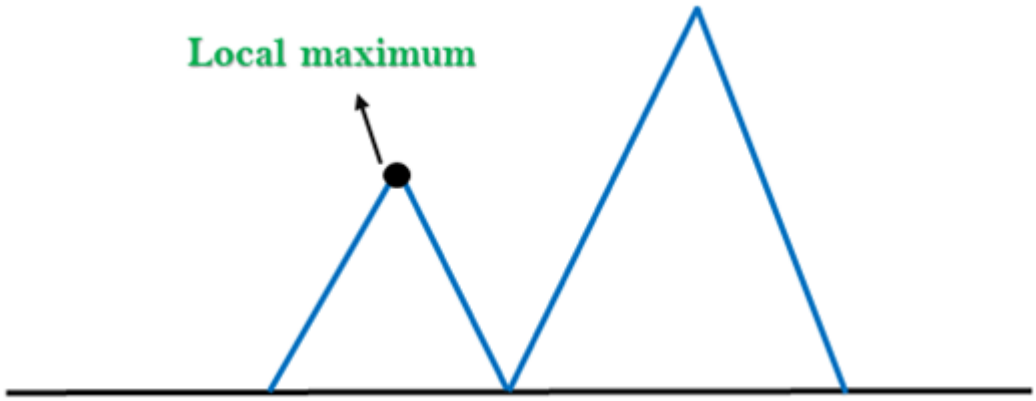
Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Problems in Hill Climbing Algorithm:

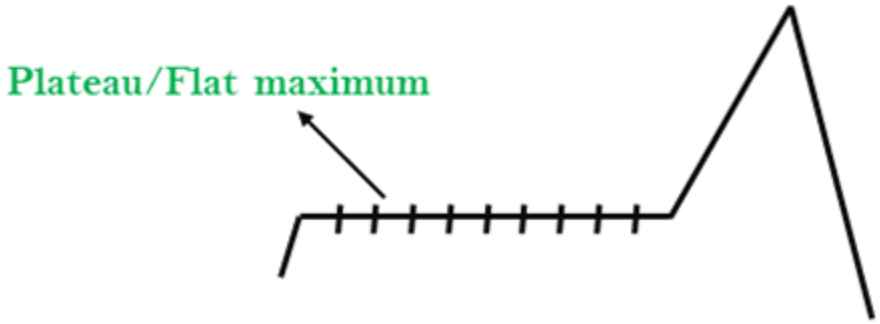
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



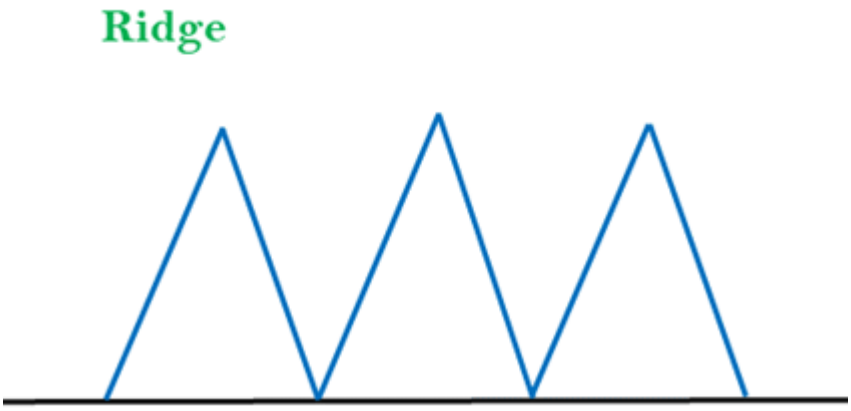
2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.



Mean end analysis : House hold robot / moneky banana problem

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

e. First, evaluate the difference between Initial State and final State.

a. Select the various operators which can be applied for each d ifference.

b. Apply the operator at each difference, which reduces the difference between the current state and goal state

Algorithm for Means-Ends Analysis:

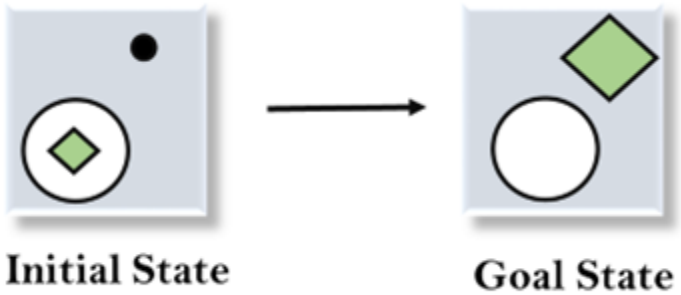
Let's we take Current state as CURRENT and Goal State as GOAL, then following are the steps for the MEA algorithm.

- **Step 1:** Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.
- **Step 2:** Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.

    1. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.
    2. Attempt to apply operator O to CURRENT. Make a description of two states.
       i) O-Start, a state in which O?s preconditions are satisfied.
       ii) O-Result, the state that would result if O were applied In O-start.
    3. If
       (First-Part <------ MEA (CURRENT, O-START)
       And
       (LAST-Part <----- MEA (O-Result, GOAL), are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART

Example of mean end analysis (house hold robot)

Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.



Solution:

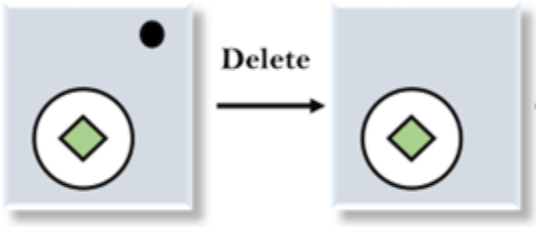
To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

- **Move**
- **Delete**
- **Expand**

1. Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.

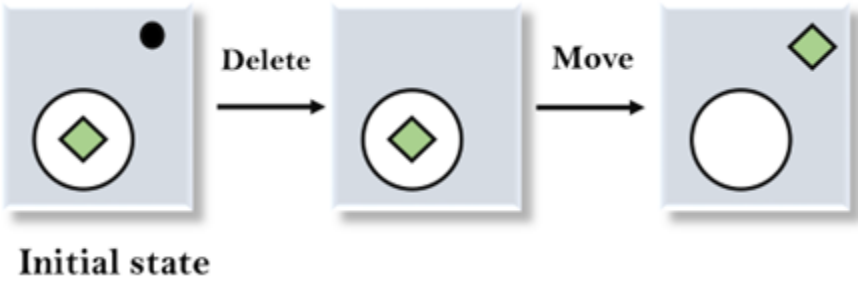
Initial state

2. Applying Delete operator: As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.

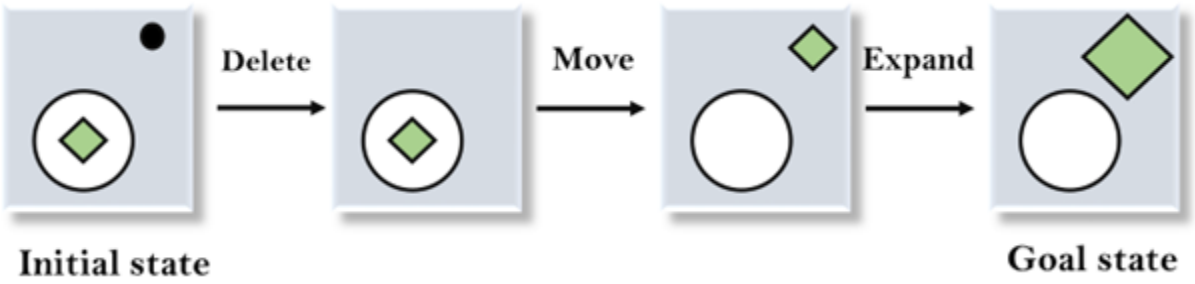


Initial state

3. Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



4. Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.

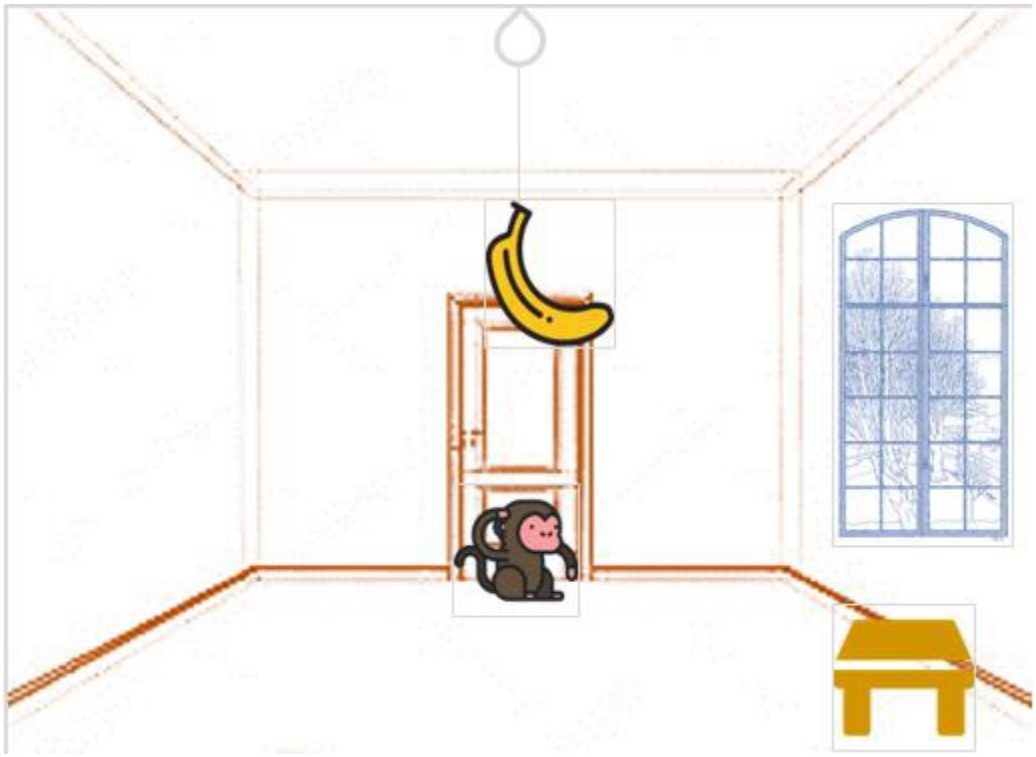


Monkey banana problem

Problem Statement

Suppose the problem is as given below −

- A hungry monkey is in a room, and he is near the door.
- The monkey is on the floor.
- Bananas have been hung from the center of the ceiling of the room.
- There is a block (or chair) present in the room near the window.
- The monkey wants the banana, but cannot reach it.



So how can the monkey get the bananas?

So if the monkey is clever enough, he can come to the block, drag the block to the center, climb on it, and get the banana. Below are few observations in this case –

- Monkey can reach the block, if both of them are at the same level. From the above image, we can see that both the monkey and the block are on the floor.
- If the block position is not at the center, then monkey can drag it to the center.
- If monkey and the block both are on the floor, and block is at the center, then the monkey can climb up on the block. So the vertical position of the monkey will be changed.
- When the monkey is on the block, and block is at the center, then the monkey can get the bananas.

Prolog program

```
On(floor, monkey).
On(floor,box).
In(room,monkey).
In(room,banana).
At(ceiling,banana).
Strong(monkey).
Grapsp(monkey).
Climb(monkey,box).
Push(monkey,box):-
        Strong(monkey).
```

```
Under(banana, box):-
        Push(monkey,box).
Canreach(banana,monkey):-
        At(floor,banana);
        At(ceiling,banana).
Under(banana,box),
Climb(monkey,box).
Canget(banana,monkey):-
        Canreach(banana,monkey), grasp(monkey).

?- canget(banana, monkey).
Ture
?-
```

General problem solver (GPS)

- GPS stands for general problem solver.
- General problem solver (GPS) is a computer program created in 1959.
- It is intended to work as an universal problem solver machine.
- GPS was the first program that was intended to solve any general problem.
- GPS was supposed to solve all the problems using the same base algorithm for every problem.
- In contrast to the former logic Theorist project, the general problem solver is working with means-ends analysis.

  How GPS works?

- The basic premise is to **express** any problem with a set of well formed formulas.
- These formulas would be a part of a **directed graph** with multiple **sources** and **sinks**.
- In a graph, the source refers to the **starting node** and the sink refers to the **ending node.**
- in the case of GPS, the source refers to **axioms** and the sink refers to the **conclusions.**
- Even though GPS was intended to be a general purpose, it could only solve **well-defined problems,** such as proving mathematical theorems in geometry and logic.
- It could also solve **word puzzles** and play **chess.**

  Solving a problem with GPS?

  Let's see how to structure a given problem to solve it using GPS:

1. The first step is to **define the goals.**
    - lets say our goal is to get some milk from the grocery store.
2. The next step is to **define the preconditions.**
    - These preconditions are in reference to the goals.
    - To get milk from the grocery store, we need to have a mode of transportation and the grocery store should have milk available.
3. After this, we need to **define the operators.**
    - if my mode of transportation is car and if the car is low on fuel, then we need to ensure that we can pay the fueling station.
    - We need to ensure that you can pay for the milk at the store.

- An operator takes care of the conditions and everything that affects them.
- It consists of actions, preconditions, and the changes resulting from taking actions.
- In this case , the action is giving money to the grocery store.
- Of course, this is contingent upon you having the money in the first place, which is the precondition.
- By giving them the money, you care changing your money condition , which will result in you getting the milk.

Constraint satisfaction problem

Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue. Finding values for a group of variables that fulfill a set of restrictions or rules is the aim of constraint satisfaction problems. For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

There are mainly three basic components in the constraint satisfaction problem:

Variables: The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables Variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

Domains: The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, **in Sudoku, the set of numbers from 1 to 9 can serve as the domain** of a variable representing a problem cell.
Constraints: The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, **in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.**

Constraint Satisfaction Problems (CSP) representation:

- The finite set of variables V_1, V_2, V_3 ……..V_n.
- Non-empty domain for every single variable D_1, D_2, D_3 ……..D_n.
- The finite set of constraints C_1, C_2 ……, Cm.
    - where each constraint C_i restricts the possible values for variables,
        - e.g., $V_1 \neq V_2$
    - Each constraint C_i is a pair <scope, relation>
        - Example: <(V_1, V_2), V_1 not equal to V_2>
    - Scope = set of variables that participate in constraint.
    - Relation = list of valid variable value combinations.
        - There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.

	6	1			7			3
	9	2			3			
		8	5	3				
						5		4
5					8			
	4							1
			1	6		8		
6								

<u>In above sudoku</u>

Variable are the **cells**

Domains are **each variables i.e {1,2,3,4,5,6,7,8,9}**

Constraints are **row, column , boxes contain all different numbers.**

AO* algorithm:

Problem Reduction: AND/ OR Tree

• The basic concept behind the method of problem reduction is:

– Reduce a hard problem to a number of simple problems and, when each of the simple problems is solved, then the hard problem has been solved.

• To represent problem reduction we can use an AND-OR tree. An AND–OR tree is a graphical representation of the reduction of problems to conjunctions and disjunctions of sub-problems.

represents the search space for solving the problem P, using the problem-reduction methods

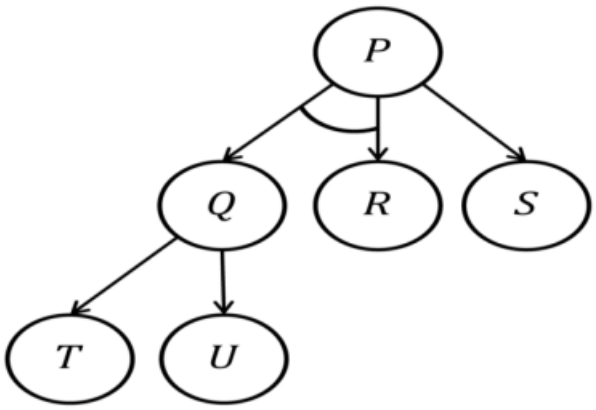
P if Q and R

P if S

Q if T

Q if U

• Example:



Searching AND/OR Tree:

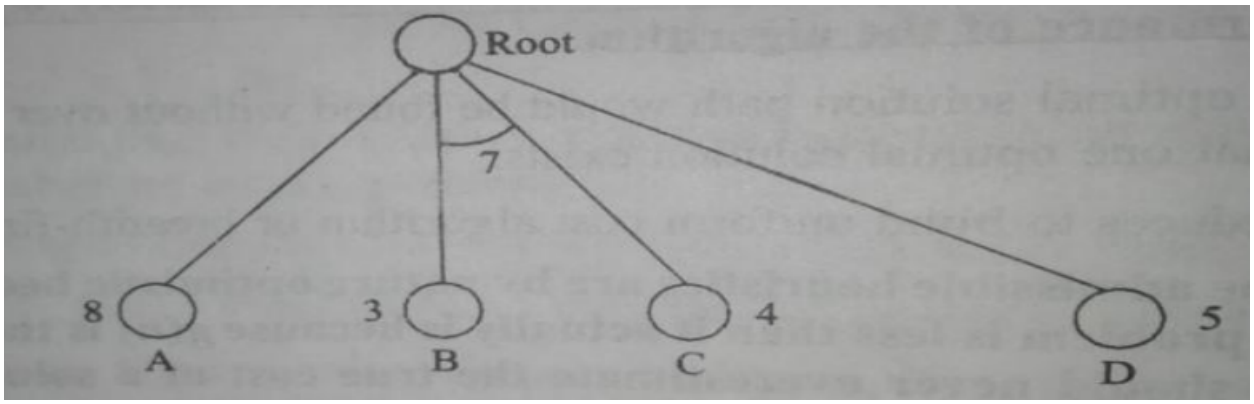
• To find a solution in AND–OR tree, an algorithm similar to Best first search algorithm is used but with the ability to handle AND arc appropriately.

• This algorithm evaluates the nodes based on the following evaluation function.

• If the node is OR node then cost function f(n)= h(n)

• If the node is AND node then cost function is the sum of costs in AND node. – f(n) = f(n1) + f(n2) + …. + $f(n_k)$= $h(n_1)$ + $h(n_2)$ + …. + $h(n_k)$ – here, n_1 , n_2 ,…. Nk are AND nodes.

Here, the number show the value of the heuristic function at that node.



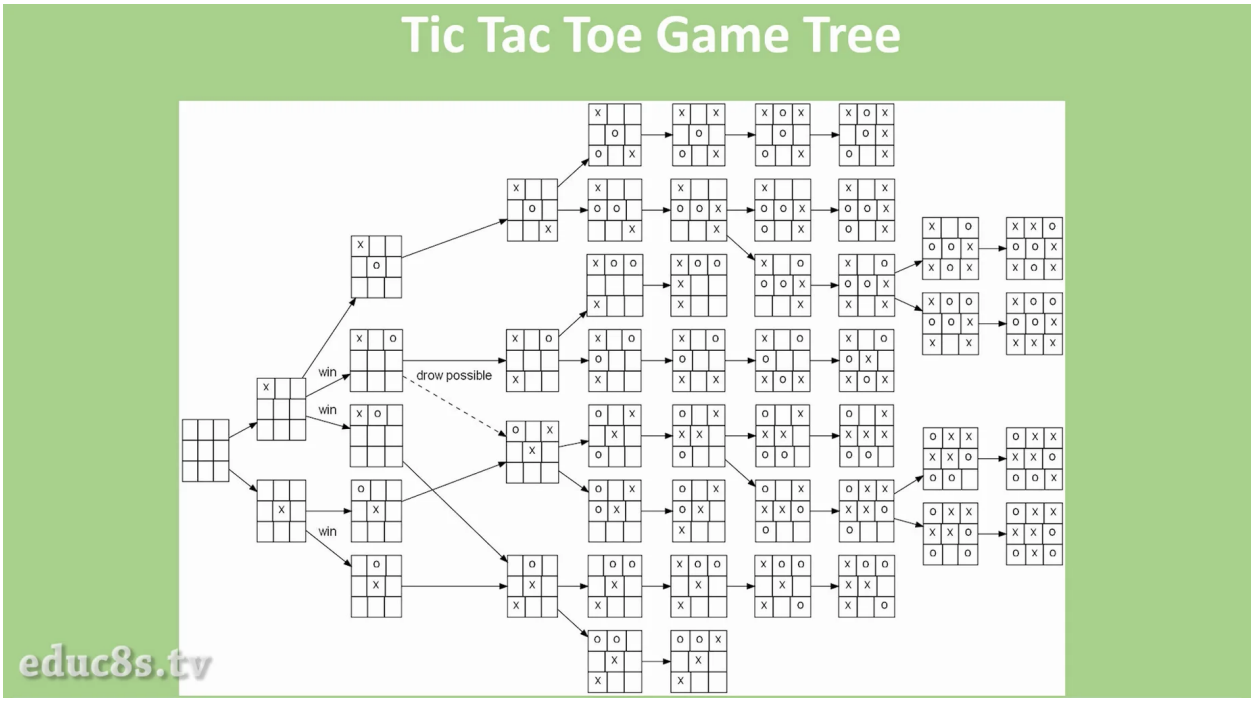
• In the figure the minimal is B which is the value of 3. But B forms a part of the AND tree so we need to consider the other branch also of this AND tree i.e., C which has a weight of 4. So, our estimate now is (3+4) = 7.

• Now this estimate is more costlier than that of branch D i.e., 5. So we explore node D instead of B as it has the lowest value.

• This process continues until either a solution is found or all paths led to dead ends, indicating that there is no solution.

Adversarial Search(Game Playing)

Competitive environments in which the agents goals are in conflict, give rise to adversarial search, often known as games.



• In AI, games means deterministic, fully observable environments in which there are two agents whose actions must alternate and in which utility values at the end of the game are always equal and opposite. **– E.g., If first player wins, the other player necessarily loses**

• This Opposition between the agent's utility functions make the situation adversarial.

• **Games as Adversarial Search**: – A Game can be formally defined as a kind of search problem with the following elements:

• **States**: board configurations.

• **Initial state**: the board position and which player will move.

• **Successor function**: returns list of (move, state) pairs, each indicating a legal move and the resulting state.

• **Terminal test**: determines when the game is over.

• **Utility function**: gives a numeric value in terminal states – (e.g., -1, 0, +1 for loss, tie, win)

• **Game Trees:**

Problem spaces for typical games represented as trees, in which:

– **Root node:** Represents the state before any moves have been made.

– **Nodes**: Represents possible states of the games. Each level of the tree has nodes that are all **MAX** or all **MIN**;

nodes at level i are of the opposite kind from those at level i+1. And

– **Arcs:** Represents the possible legal moves for a player.

Moves are represented on alternate levels of the game tree so that all edges leading from root node to the first level represent moves for the first(MAX) player and edges from the first level to second represents moves for the second(MIN) player and so on.

– **Terminal nodes** represent end-game configurations.

• **Evaluation Function**: – An evaluation function is used to evaluate the "goodness" of a game position. – i.e., estimate of the expected utility of the game position

– The performance of a game playing program depends strongly on the quality of its evaluation function.

– An inaccurate evaluation function will guide an agent toward positions that turn out to be lost. …

• A good evaluation function should:

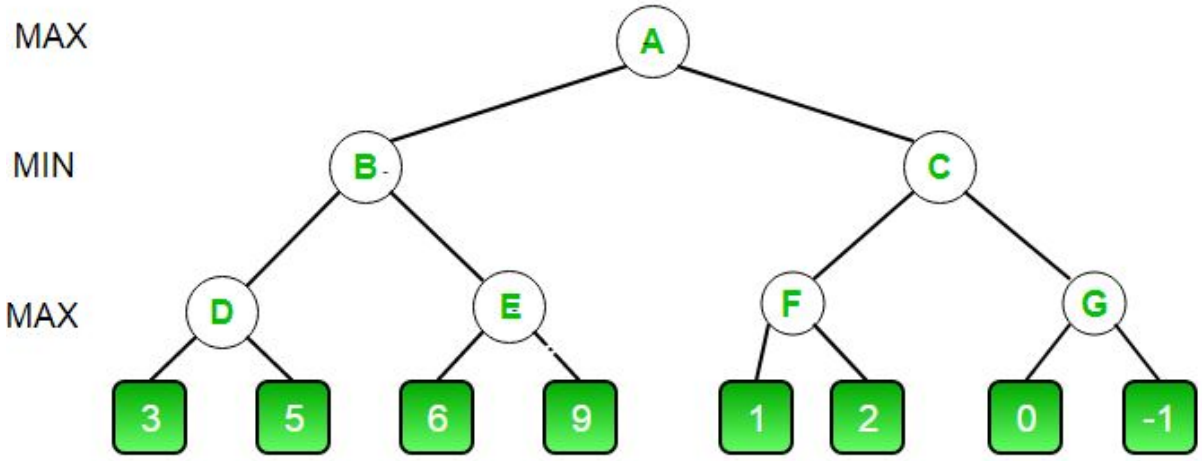
• Order the terminal states in the same way as the true utility function:

• i.e., States that are wins must evaluate better than draws, which in turn must be better than losses. Otherwise, an agent using the evaluation function might err even if it can see ahead all the way to the end of the game.

• For non-terminal states, the evaluation function should be strongly correlated with the actual chances of winning.

• The computation must not take too long i.e., evaluate faster.

Min max algorithm



- ➔ It is recursive on backtracking algorithm which is used in game theory and decision making.
- ➔ It is mostly used for game playing in AI- Such as chess, tic-tac-toc, go, checker etc.
- ➔ There are 2 players MAX and MIN.
- ➔ Max for maximized value and MIN for minimized value.
- ➔ Minmax performs a DFS algorithm.
    - ⇨ Consider the following two players game tree in which the static scores are given from the first players point of view.
    - ⇨ Apply the min max search algorithm and compute the value of the tree .
    - ⇨ Also find the most convenient path for MAX Node.

Steps to solve problem in min max algorithm

1. Generate the whole game tree to leaves.
2. Apply utility (payoff) function to leaves.
3. Uses DFS for expanding the tree.
4. Back-up values from leaves toward the root:
   - A max node computes the maximum value from its child value.
   - A min node computes the minimum value from its child values.
5. When values reaches the root : optimal move is determined.

Properties:

i. Complete – yes

ii. Optimal – yes

iii. Time complexity – $O(b^m)$

iv. Space complexity – $O(b^m)$

Limitation:

i. It is slow for complex games such as chess because chess have branching factor so the limitation of minmax can be improved from **alpha beta pruning**.

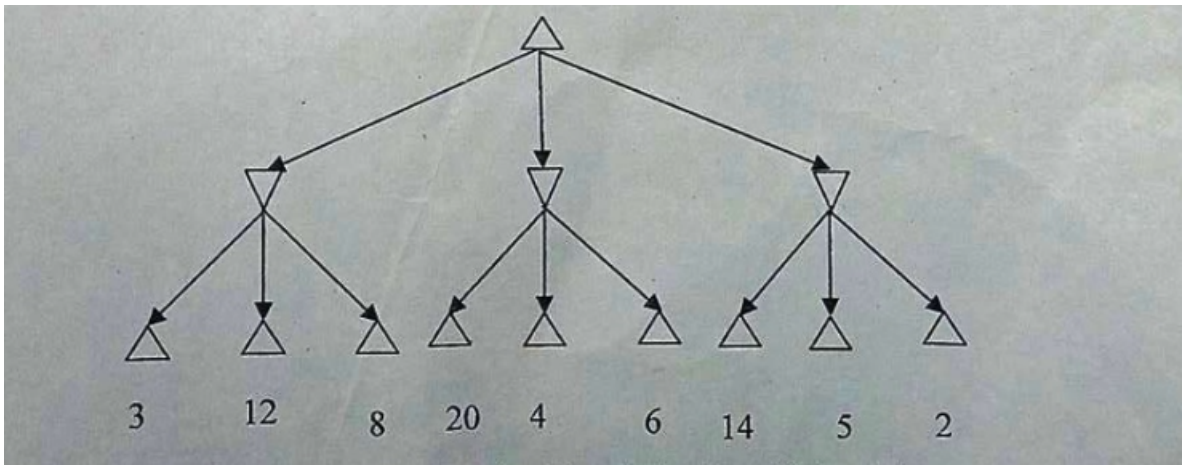
[note : symbolic notation for max player : Δ and min player is : ▼]

Previous year questions from min max algorithm:

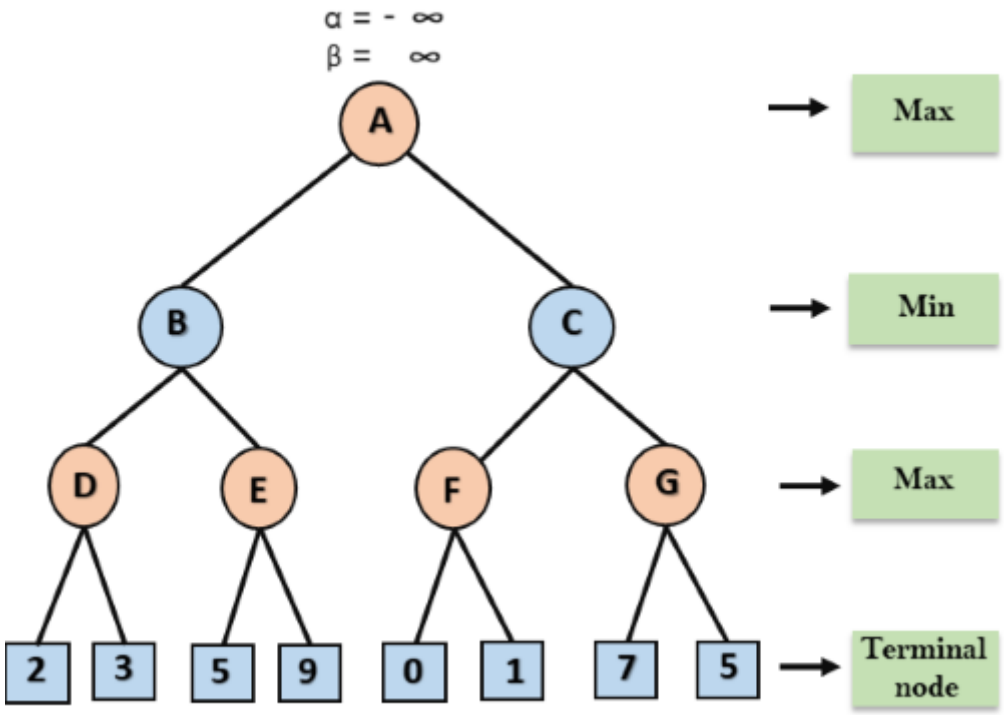
Define min max search procedure with suitable figure. [5 marks/ 2013/2015]

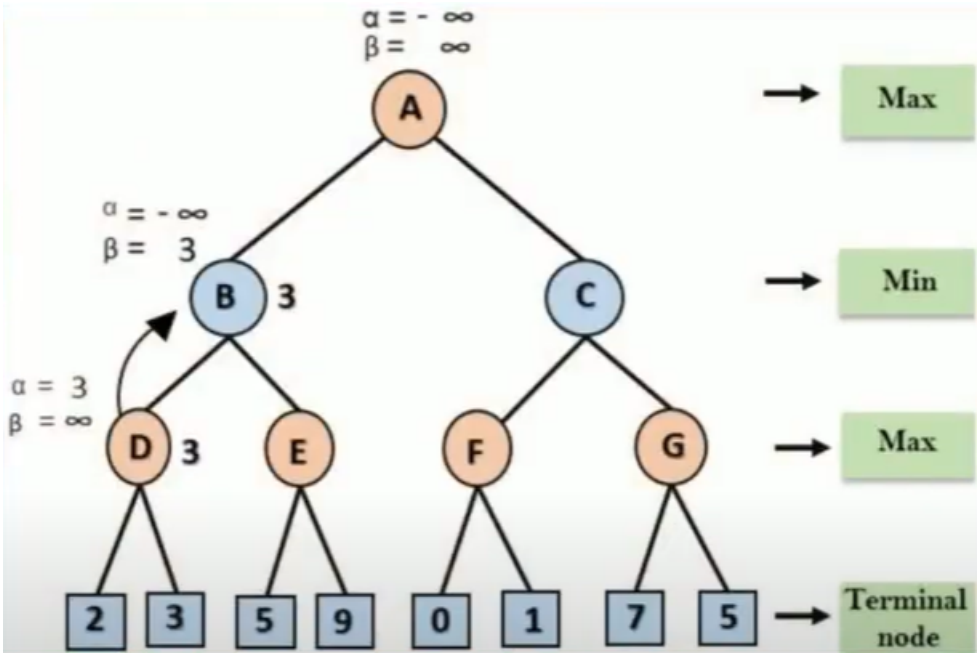
Describe the concept of alpha beta pruning in min max search. [5 marks/ 2022]

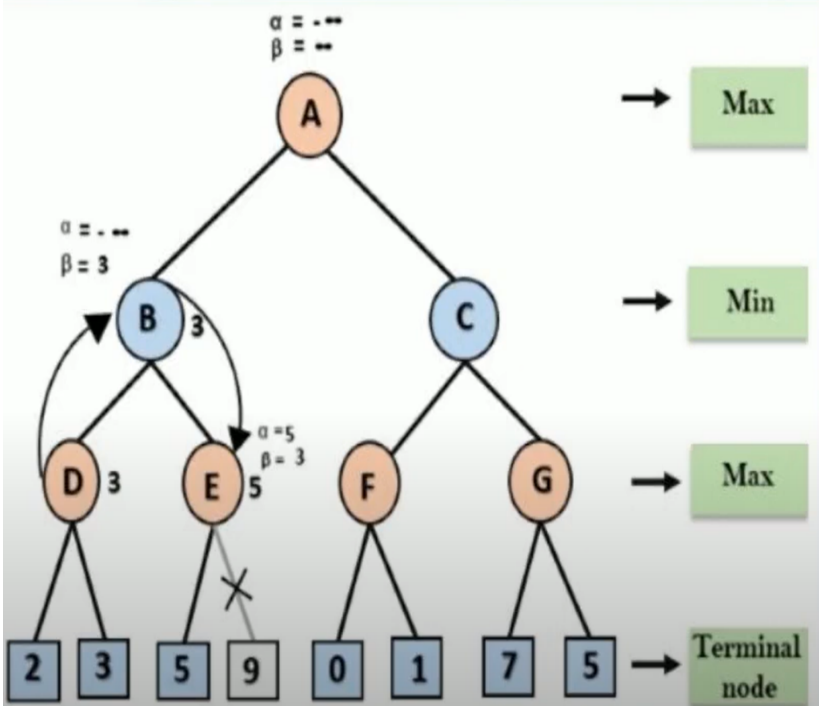
Write the steps used in searching. justify the statement " the alpha beta pruning game playing search is better than the min max game playing search with the help of game tree. [5 marks / 2019]

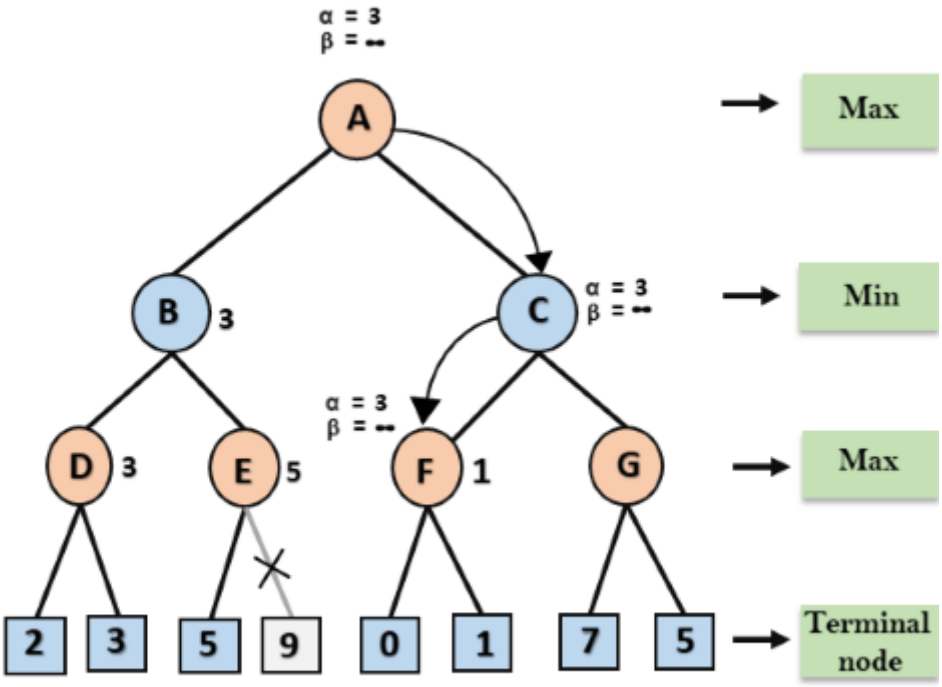


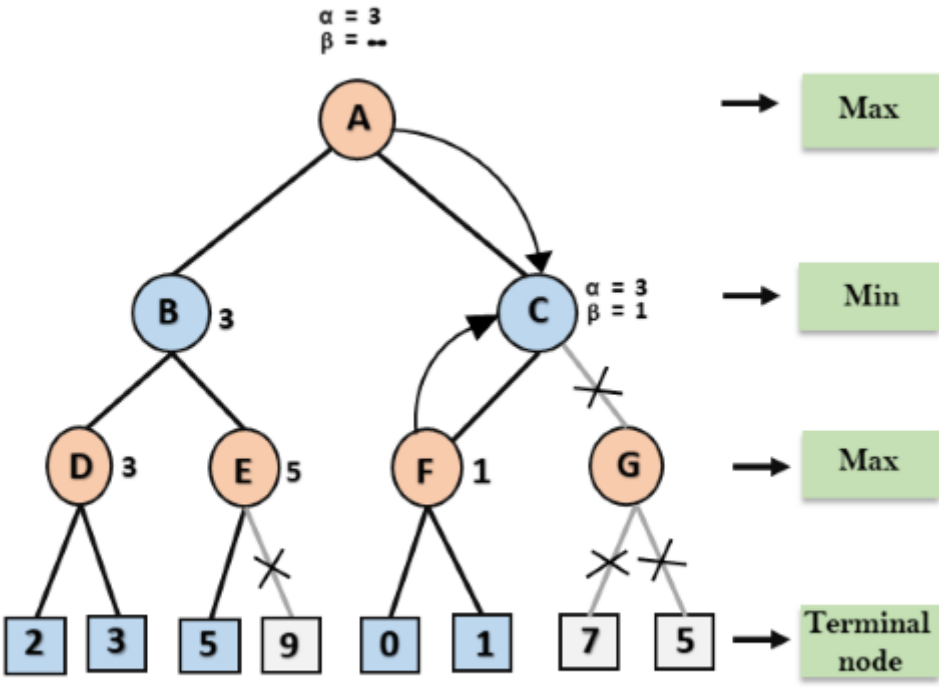
Example problem for alpha beta pruning

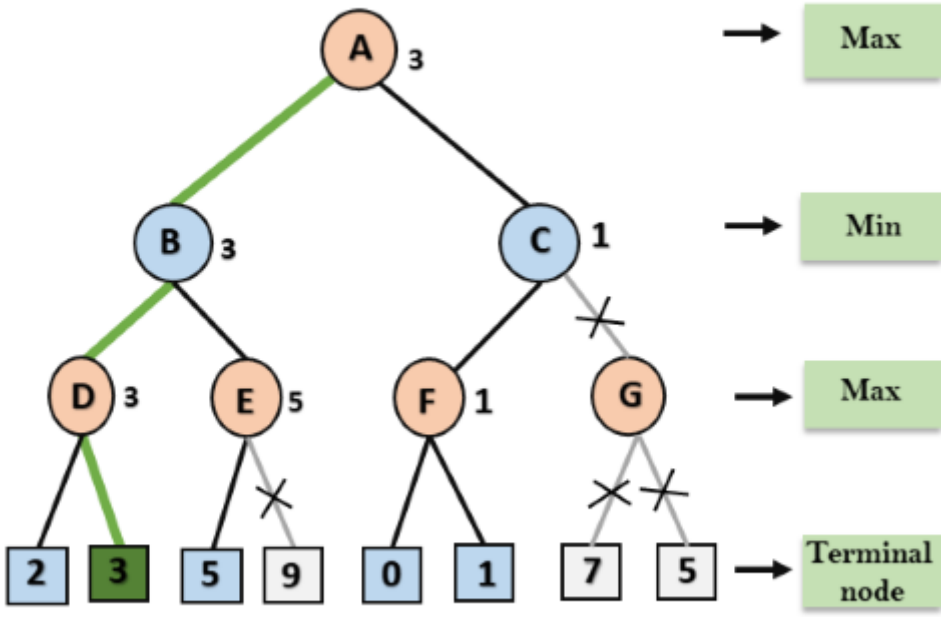

α = - ∞
β = ∞
A
B
C
D
E
F
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal node


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


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


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


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



Games of Chance

• In the game with uncertainty Players include a random element(roll dice, flip a coin, etc.) to determine what moves to make

• i.e., Dice are rolled at the beginning of a player's turn to determine the legal moves. Such games are called game of chance.

• Chance games are good for exploring decision making in adversarial problems involving skill and luck.

What is Game Theory?

• Game theory is a study of how to mathematically determine the best strategy for given conditions in order to optimize the outcome

• Finding acceptable, if not optimal, strategies in conflict situations.

• Abstraction of real complex situation

• Game theory is highly mathematical

• Game theory assumes all human interactions can be understood and navigated by presumptions.

Previous year questions from unit 3

1. **What are the various components of problem definition in searching ? [2013/1]**
2. **What is heuristic search? Write A* search algorithm. [2013/3]**
3. **What are the problems in hill climbing search? [2014/1]**
4. **How is the performance of a search algorithm measured? [2015/1]**
5. **Make a comparison between BFS and DFS. [2015/3]**
6. **In what situations hill climbing search is appropriate? Explain [2018/4]**

7. **Write an algorithm of depth first search. [2018/4]**
8. **Compare informed search with uninformed search. How A* searches state space explain with suitable example. [2020/ 5]**
9. **What are the four ways of evaluating performance of searching? [2022/1]**
10. **How hill climbing search works? What is its problems? [2022/5]**
11. **What does a heuristic mean? Differentiate between greedy best first search and A* search. [2022/5]**