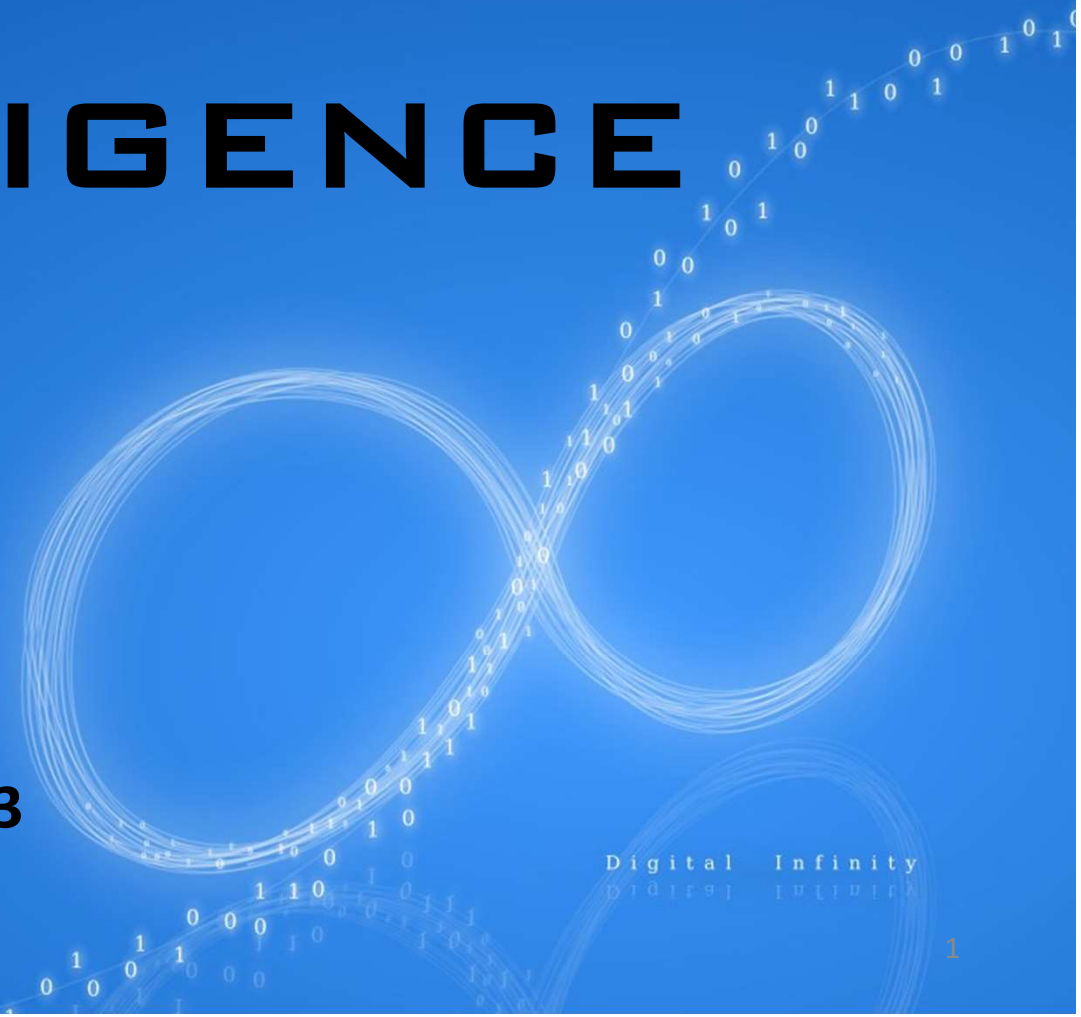
ARTIFICIAL INTELLIGENCE



2nd Year, 4th Semester, 2022-23

AI & ML

Uninformed Search Strategies

- ***Uninformed*** strategies use only the information available in the problem definition
  - **Also known as blind searching**

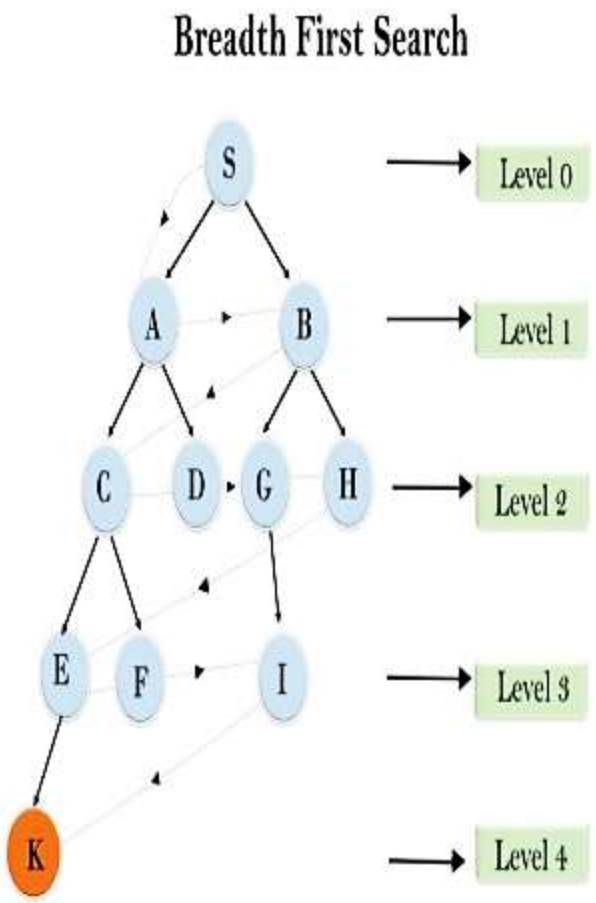
- ***Breadth-first search***
- ***Depth-first search***
- ***Iterative Depending search***
- ***Uniform-cost search***

Completeness : The search guarantees a solution for any random input.

Optimality : If the solution guarantees to be the best solution among all the solutions.

Breadth-First Search

- BFS is the most common search for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- In BFS search starts from root node and then before moving to next level all successor node from current level is expand.
- It is an example of a general-graph search algorithm.
- It uses queue data structure.



Breadth First Search

Begin

i) Place the starting node in the queue

ii) **Repeat**

Delete the queue to get the front element;

If the front element of the queue = goal,

return success and stop;

Else do

Begin

insert the children of the front element,

if exist, in any order at the rear end of the queue;

End

Until the queue is empty;

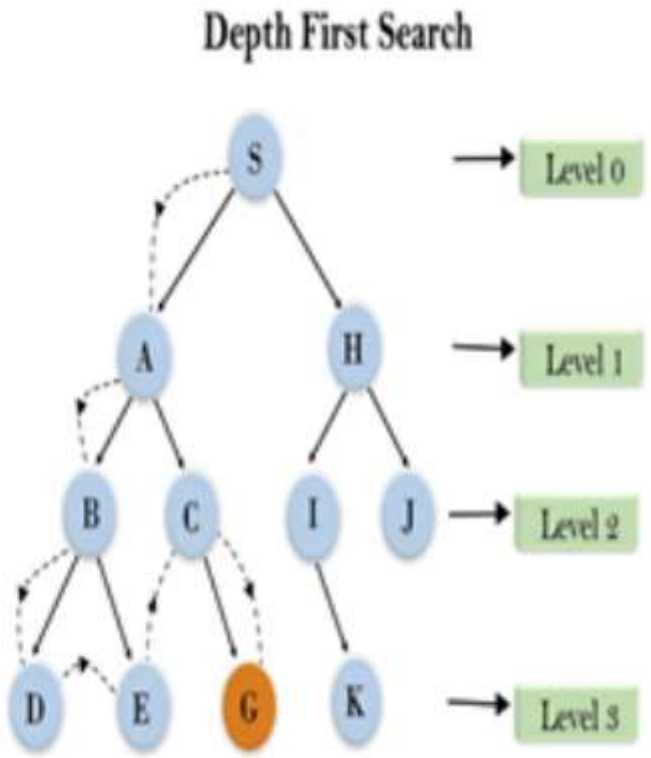
End

Lessons From Breadth First Search

- BFS is a **complete search**.
- BFS finds a solution with the **shortest path** length.
- It has **exponential time** and **space complexity**.
- A complete search tree of depth ‘d’ where each non-leaf node has ‘b’ children, has total of **(b^d)/(b-1)** nodes.
- The **memory requirements are a bigger** problem for breadth-first search than is execution time
- **Exponential-complexity search problems** cannot be solved by uniformed methods for any but the smallest instances

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.
- Backtracking is an algorithm technique for finding all possible solutions using recursion.



- DFS takes exponential time.
- If ‘N’ is maximum depth of node in the search space, in the worst case algorithm takes time ‘b^d’.
- The space taken is linear in the depth of the search tree ‘bN’

Depth First Search

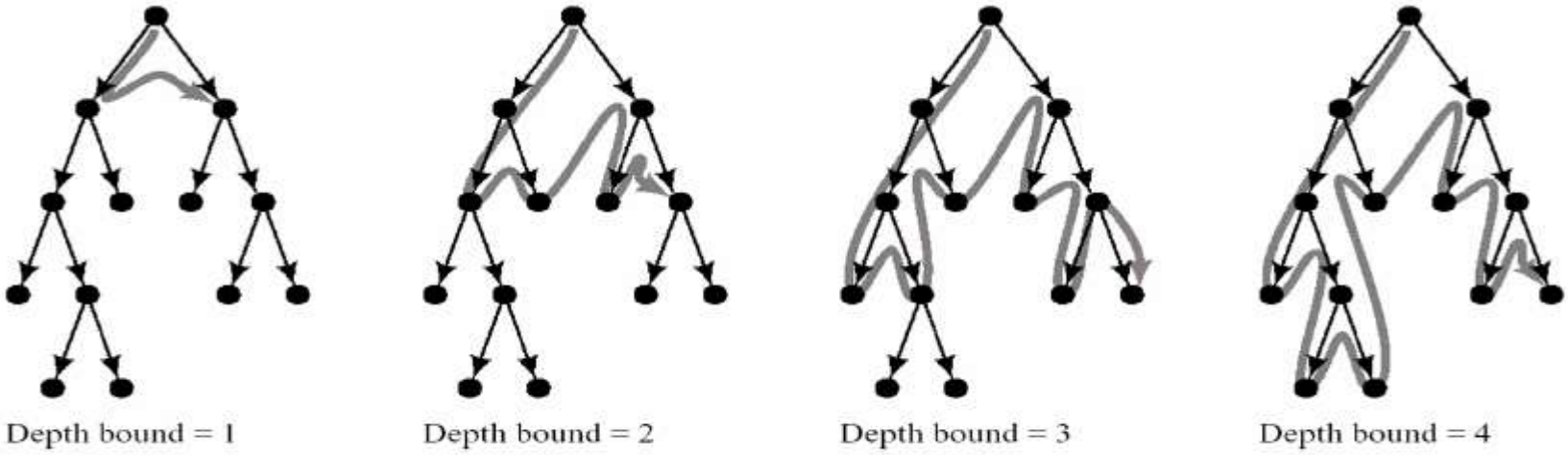
Begin

1. Push the starting node at the stack,
   pointed to by the stack-top;
2. **While** stack is not empty do
   Begin
   Pop stack to get stack-top element;
   If stack-top element = goal, return success and stop
   Else push the children of the stack-top element in any order into the stack;
   End while;

End

Iterative Deepening Search

- Iterative deepening depth-first search
  - First do depth 0, then, if no solution found, do DFS to depth 1 and gradually increases the depth limit; 0, 1, 2, … until a goal is found
  - For large ‘d’ the ratio of the number of nodes expanded is **b/(b-1)**.
  - ID searching moves along the increasing depth of the nodes, where as BFS moves level by level.
  - In DFS the search propagates at the maximum depth of the tree, whereas in ID searching doesn't go below the current depth.

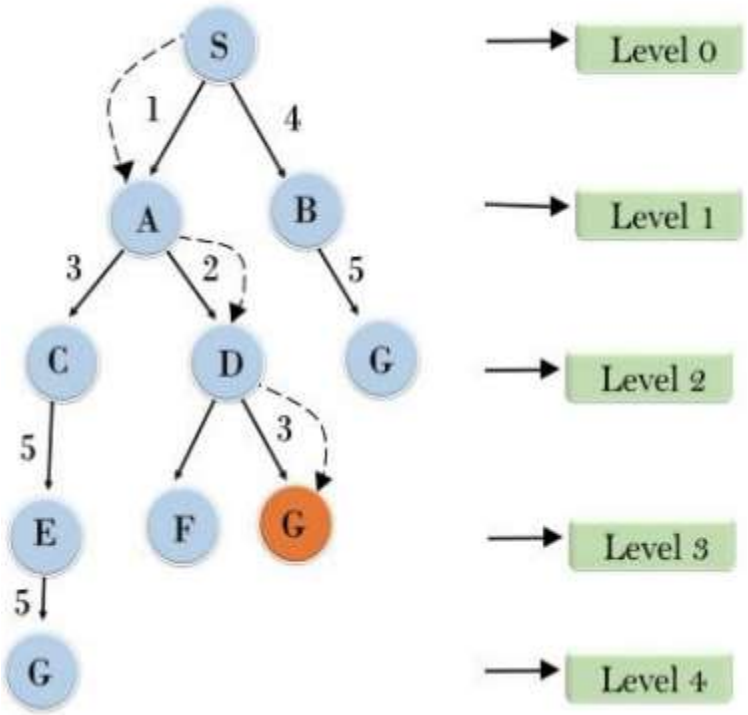


Depth First Iterative Deepening

	Breadth first	Depth first	Iterative deepening
Time	b^d	b^d	b^d
Space	b^d	bm	bd
Optimum?	Yes	No	Yes
Complete?	Yes	No	Yes

Uniform-Cost Search

- Same idea as the algorithm for breadth-first search…but…
  - Nodes are expanded in the order of their cost from the source.
  - FIFO queue is ordered by cost.
  - This search is **complete** and **optimal**.
  - Exponential space and time complexity of b^d.
  - the value of the function ‘f’ for a given node ‘n’, f(n), for ***uninformed search algorithms***, takes into consideration g(n), the total action cost **from the root node to the node n**, that is, the path cost.



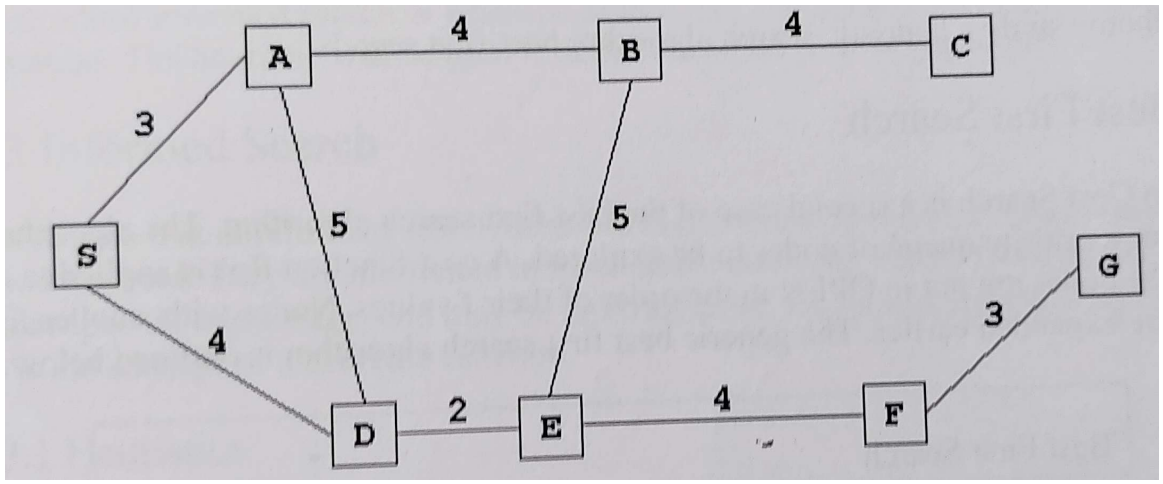
Informed Search

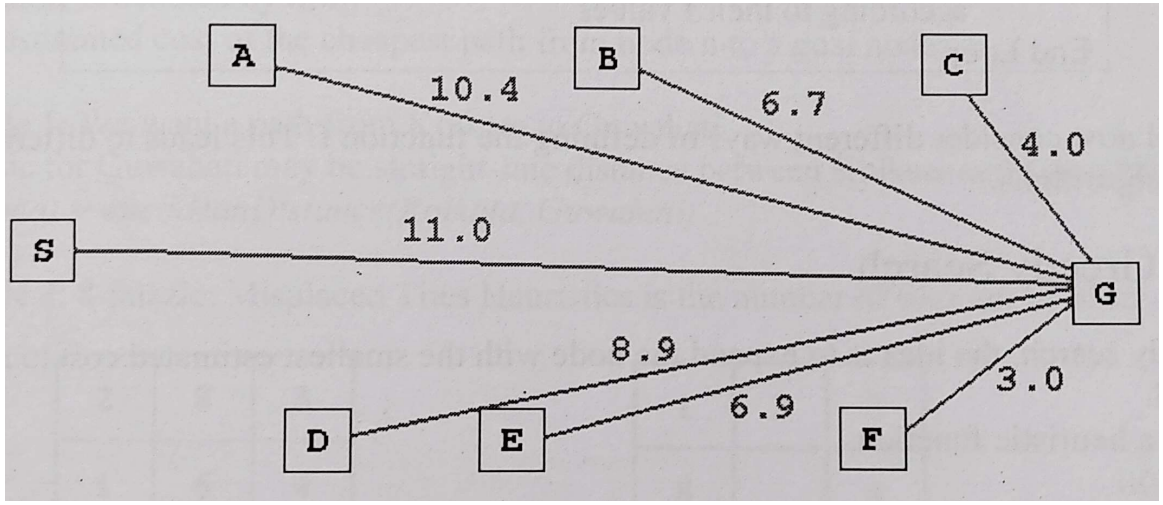
Best-first Search Algorithm (Greedy Search)

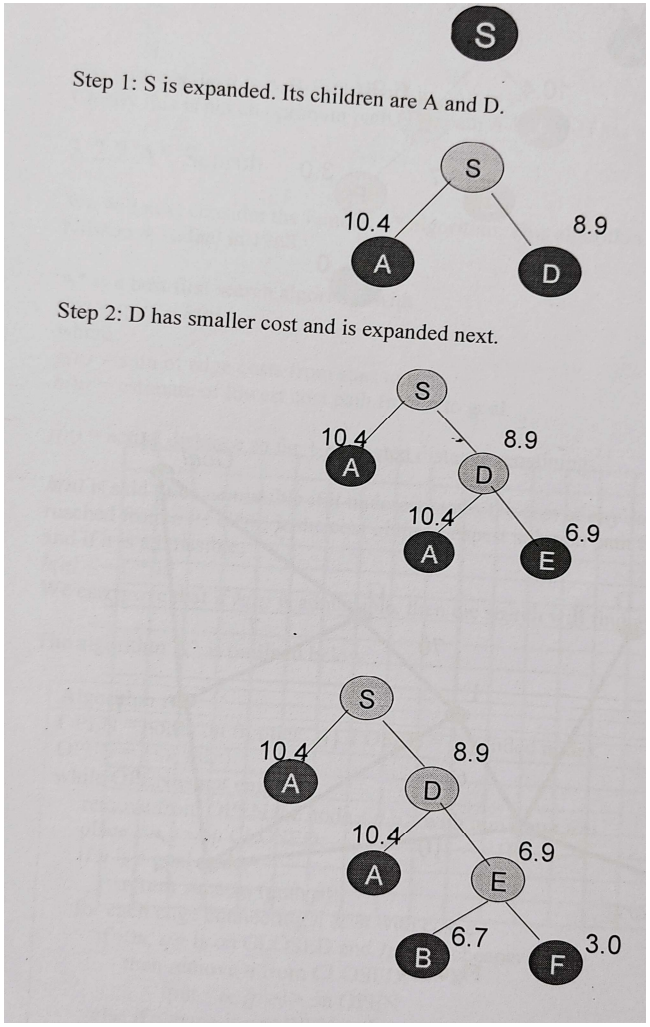
- Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

  $$f(n) = g(n).$$

- Were, h(n)= estimated cost from node n to the goal.
- The greedy best first algorithm is implemented by the priority queue.



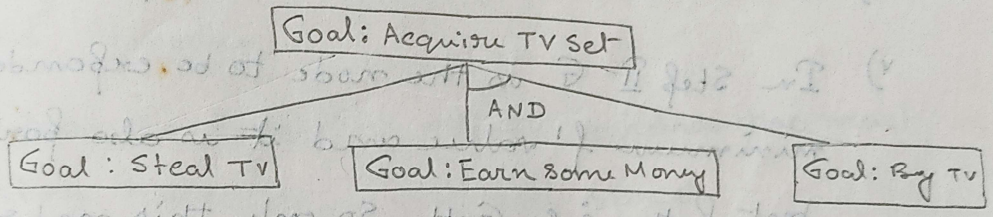




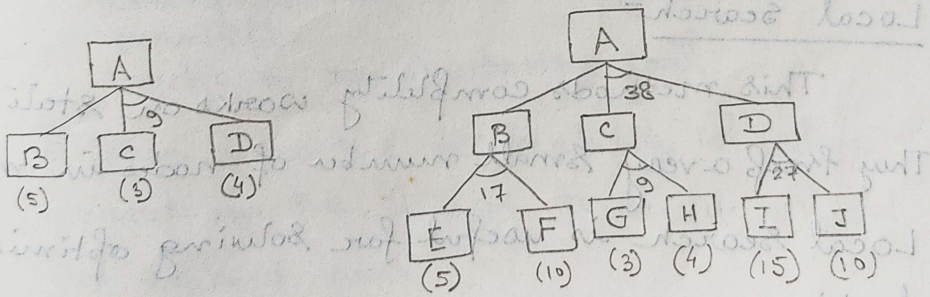
- g(n) is the essential distance remaining to a goal.
- The search is **not an optimal** one **but has completeness**.
- the function that is evaluated to choose which node to expand has the form of f(n) = h(n), where h is the heuristic function for a given node n that returns the estimated value **from this node n to a goal state**.

Best - First - Search : AND OR Graph / AO* Algo. –

This method of searching is useful when problems can be decomposed into a set of small problems It means, may be two set of nodes leads to the goal point rather than only one goal node. Hence due to this decomposition, arcs are generated which are known as AND arc.



Hence in the above example, the Goal can be achieved by decomposition the problem into two nodes i.e., Earn some money & Buy TV. So it is the AND of these two nodes which can solve the problem.
Lets, consider another example –



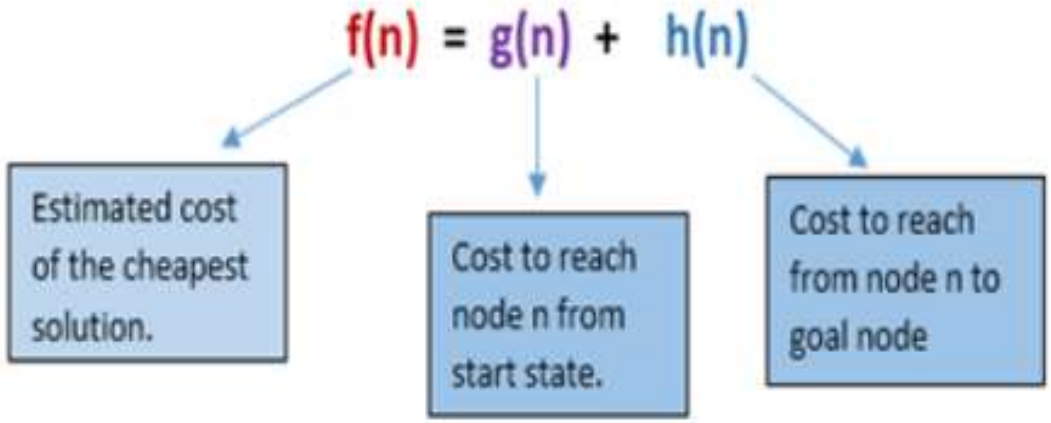
- It is considered to have uniform cost function of every expansion = 1
- Suppose 'B' node is to be selected for expansion but
  (B+C)=5+3+1+1=10
  (C+D)=3+4+1+1=9

So best path for expansion is (C+D)

- It is not only about the value of 'f' but also whether it is part of best node or not.
- This method of algorithm is known as AO* algorithm.

A* Search Algorithm

- 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**.



- At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.
- A* search algorithm is optimal and complete.

Example: Water Pouring

- Given a 4 gallon bucket and a 3 gallon bucket, how can we measure exactly 2 gallons into one bucket?
  - There are no markings on the bucket
  - You must fill each bucket completely
- **Initial state**:
  - The buckets are empty
  - Represented by the tuple (0 0)
- **Goal state**:
  - One of the buckets has two gallons of water in it
  - Represented by either (x 2) or (2 x)
- **Path cost**:
  - 1 per unit step

Example: Water Pouring

List of PRs for the water-jug problem

PR 1. $(u, v : u < 4) \rightarrow (4, v)$

PR 2. $(u, v : v < 3) \rightarrow (u, 3)$

PR 3. $(u, v : u > 0) \rightarrow (u - D, v)$, where D is a fraction of the previous content of u.

PR 4. $(u, v : v > 0) \rightarrow (u, v - D)$, where D is a fraction of the previous content of v.

PR 5. $(u, v : u > 0) \rightarrow (0, v)$

PR 6. $(u, v : v > 0) \rightarrow (u, 0)$

PR 7. $(u, v : u + v \geq 4 \wedge v > 0) \rightarrow (4, v - (4 - u))$

PR 8. $(u, v : u + v \geq 3 \wedge u > 0) \rightarrow (u - (3 - v), 3)$

PR 9. $(u, v : u + v \leq 4 \wedge v > 0) \rightarrow (u + v, 0)$

PR 10. $(u, v : u + v \leq 3 \wedge u > 0) \rightarrow (0, u + v)$

Example: Water Pouring

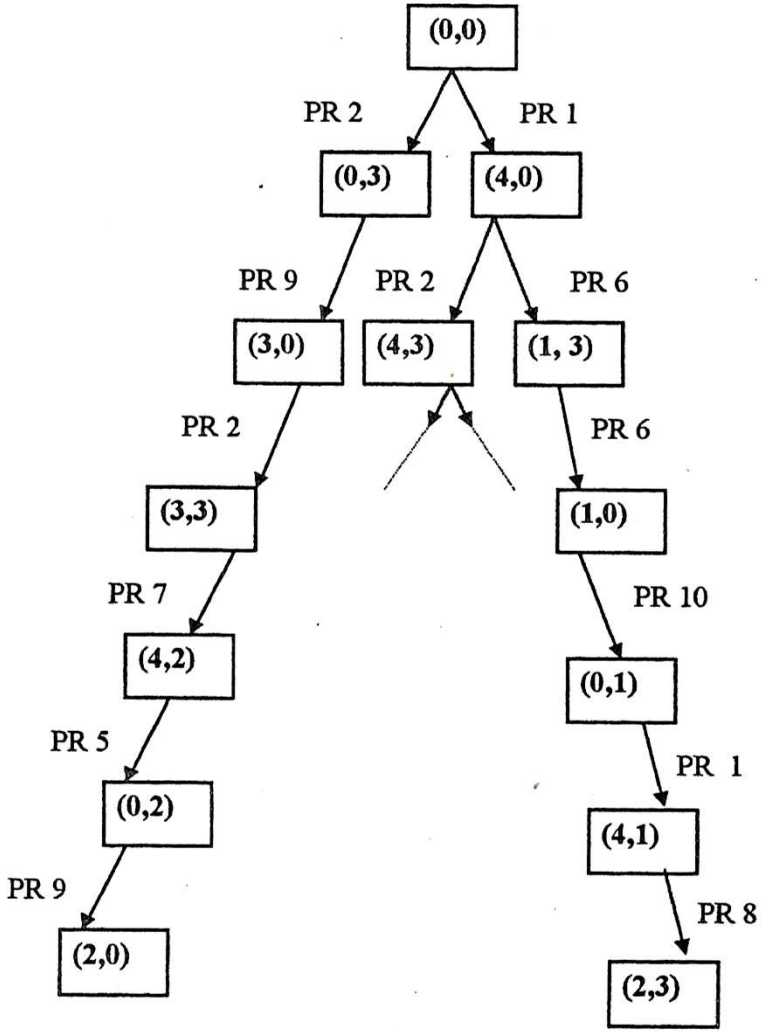
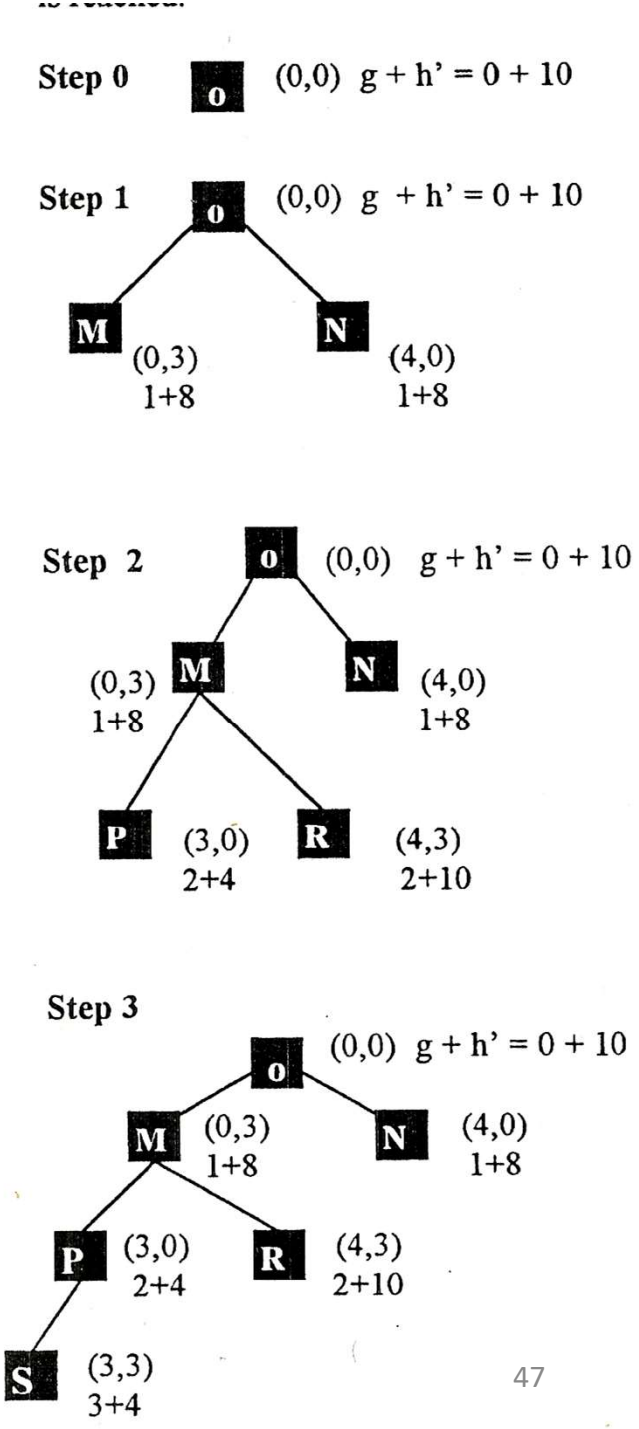


Fig. 3.2: The state-space for the water-jug problem.

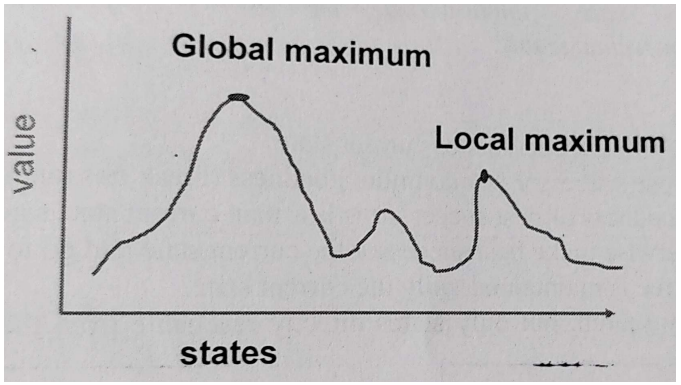
Heuristic Search

h(x) = 2, when 0<X<4 AND 0<Y<3,
= 4, when 0<X<4 OR 0<Y<3
= 10, when i) X = 0 AND Y = 0
OR ii) X = 4 AND Y = 3
= 8, when i) X = 0 AND Y = 3
OR ii) X = 4 AND Y = 0



Hill Climbing Algorithm

- It is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.



- **Features of Hill Climbing Algorithm:**
  - Generate and Test variant
  - Greedy approach
  - No backtracking
- **Simple Hill Climbing**: It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state.
  - Less time consuming
  - Less optimal solution and the solution is not guaranteed

Hill Climbing Algorithm

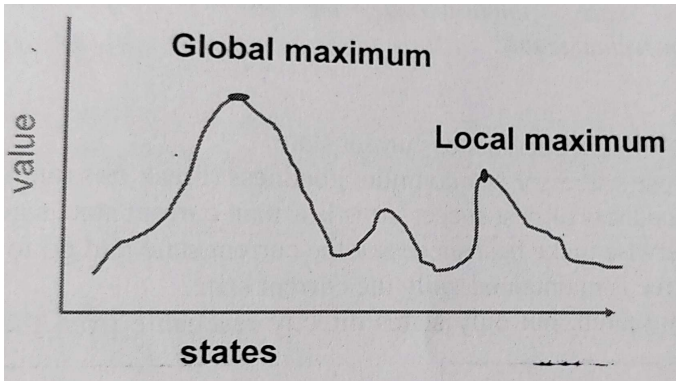
The generate & test type of search algo. expands the search space and examines the existence of the goal in that space. But in this method, the feedback from test procedure helps the generator to decide which direction to move in the search space.

In this method a function $f(x)$ is employed that would give an estimate of the measure of distance of the goal from node 'x'.

After $f(x)$ is evaluted for all possible initial nodes 'x', the nodes are sorted in ~~as~~ ascending order of their function values. So the stack top element has the least function value. It is now popped out and compared with goal state. If the stack top element is not the goal then it is expanded and function is measured for each of its children. Again the previous steps ~~are~~ are repeated until the goal state is ~~ach~~ achieved.

Problems in Hill Climbing Algorithm

- **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.
- **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.
- **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.



Simulated Annealing

Annealing is a process of metal casting, where metal is melted at high temperature beyond its melting point and then it is allowed to cool down, until it returns to solid state. Thus in the physical process of annealing, the hot material gradually looses its energy and finally at one point of time reaches at a state of minimum energy.

Here the term objective function is used rather than or in place of heuristic function. Here the attempt is to minimize the object function rather than maximize. Thus actually it describes the process of valley descending rather than hill climbing.

Simulated Annealing

The concept of rolling ball can be used here. Let, consider a rolling ball that falls from a higher energy state to a valley and moves up to a little higher energy state. The probability of such transition to a higher energy state is very small and is given by

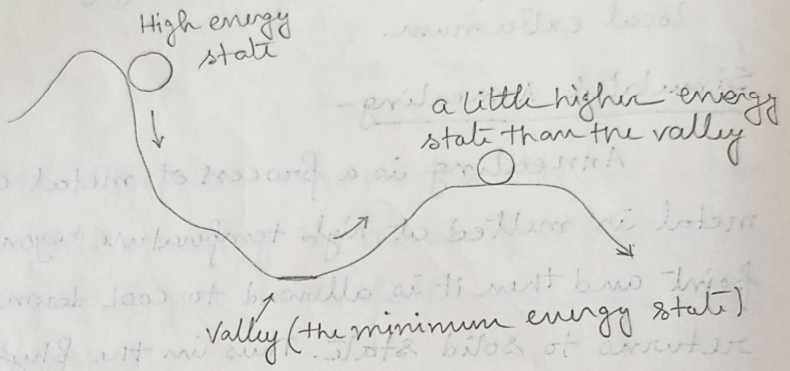
$$p = \exp(-\Delta E/KT)$$

p – probability of transition from lower to higher energy state

ΔE – positive change in energy

K – Boltzman constant

T – temperature



So, ΔE small then p is higher

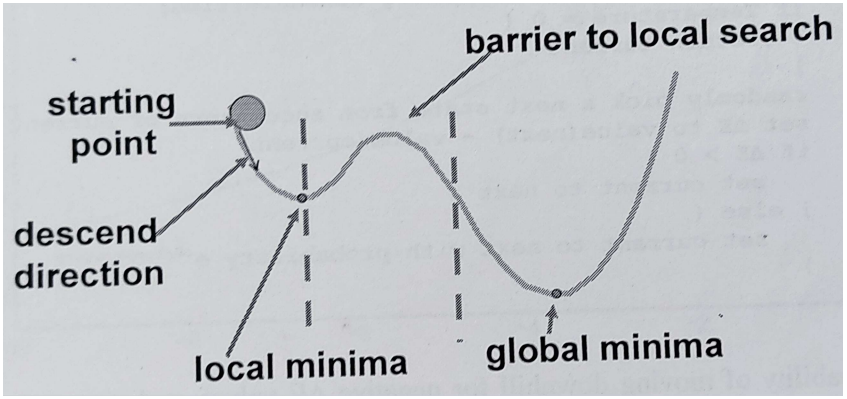
ΔE large then p is smaller

i.e., the probability of transition to a slightly higher state is more than the probability of transition to a very high state.

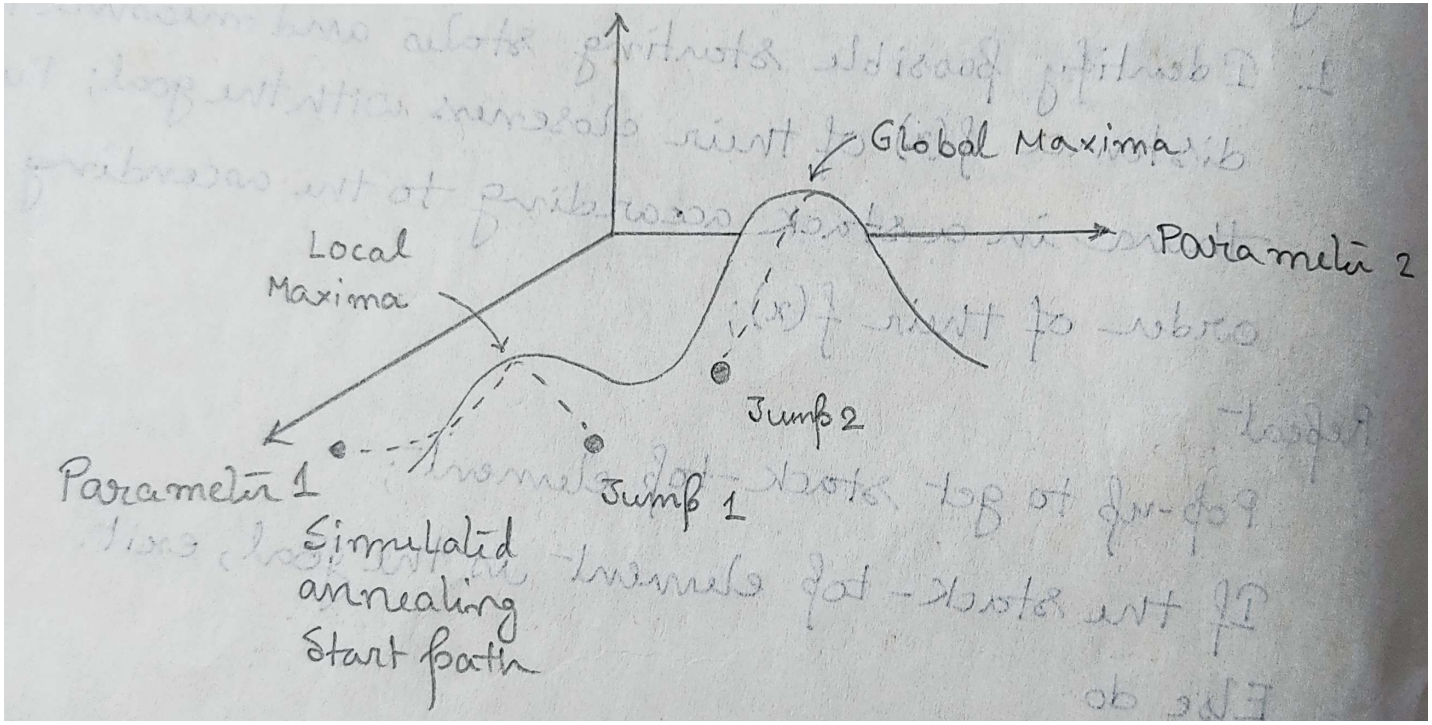
Now under this circumstances ΔE is calculated for all possible legal next states and p' is also evaluted for each such next state by the formula

$$p' = \exp(-\Delta E/T)$$

A random number in the closed interval of $[0,1]$ is then computed. and p' is compared with the random number. If p' is more than it is selected for next transition.



```
set current to start state
for time = 1 to infinity {
    set Temperature to annealing_schedule[time]
    if Temperature = 0 {
        return current
    }
    randomly pick a next state from successors of current
    set ΔE to value(next) - value(current)
    if ΔE > 0 {
        set current to next
    } else {
        set current to next with probability e^(ΔE/Temperature)
    }
}
```



Game Playing

- The second major application of **heuristic search** algorithms in AI is game playing.
- One of the original challenges of AI, which in fact predates the term artificial intelligence, was to build a program that could play chess at the level of the best human players.
- **Competitive environments** in which goals of multiple agents are in conflict (often known as games).
- Games problems are like **real world problems**.
- Game theory views as **multi-agent environment** as game.
- Define **optimal move** and algorithm for finding it.

Minimax Search

- Mini-max algorithm is a ***recursive or backtracking algorithm*** which is used in decision-making and game theory.
- It provides an ***optimal move*** for the player assuming that opponent is also playing optimally.
- Algorithm searches ***forward to a fixed depth*** in the game tree, limited by the amount of time available per move.
- At this search horizon, a ***heuristic evaluation function*** is applied to the frontier nodes.
- the heuristic is a function that takes a board position and returns a number that indicates how ***favorable that position is for one player*** relative to the other.

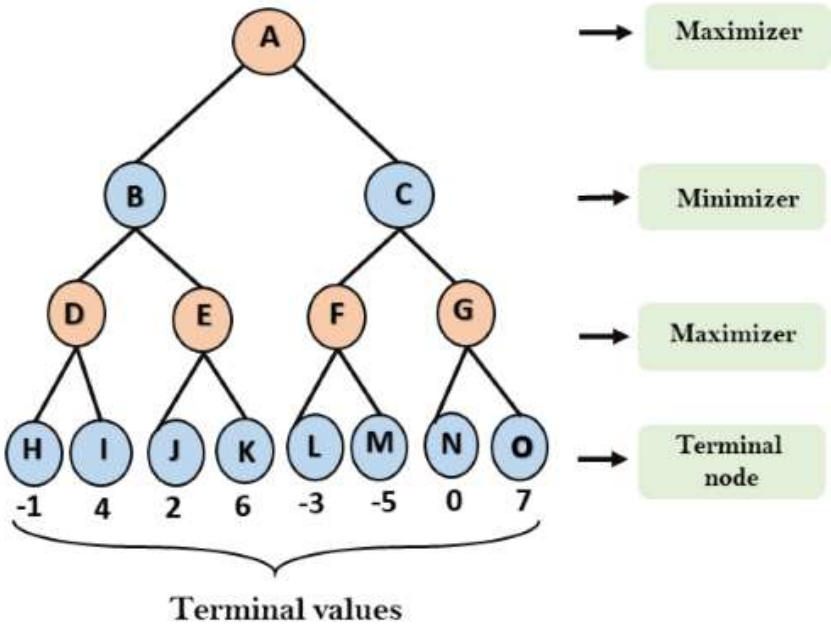
Minimax Search

- For example, a very simple heuristic evaluator for chess would ***count the total number of pieces on the board for one player***, ***weighted by their relative strength, and subtract the weighted sum of the opponent's pieces***.
- Thus, large positive values would correspond to **strong positions for one player, called MAX**, whereas **negative values of large magnitude would represent advantageous situations for the opponent, called MIN**.
- MAX attempts to maximize its score.
- MIN attempts to minimize MAX's score.

Minimax Search

- first step, the algorithm generates the entire game-tree and apply the **utility function to get the utility values for the terminal states**.

maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.

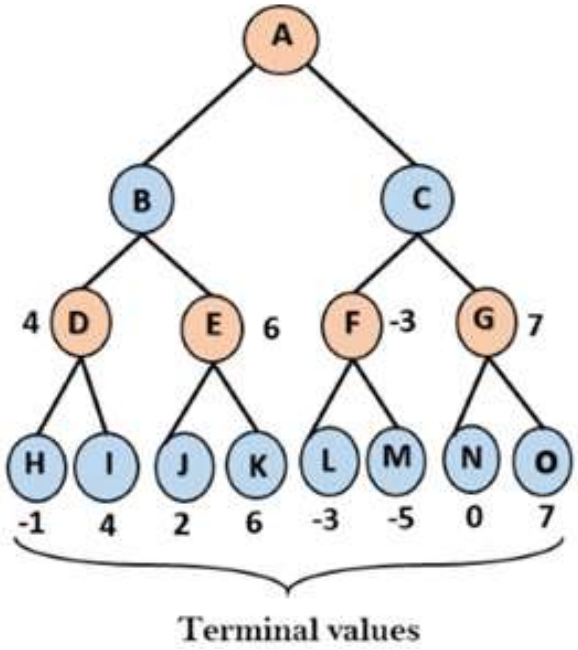


For node D max(-1,- -∞) => max(-1,4)= 4

For Node E max(2, -∞) => max(2, 6)= 6

For Node F max(-3, -∞) => max(-3,-5) = -3

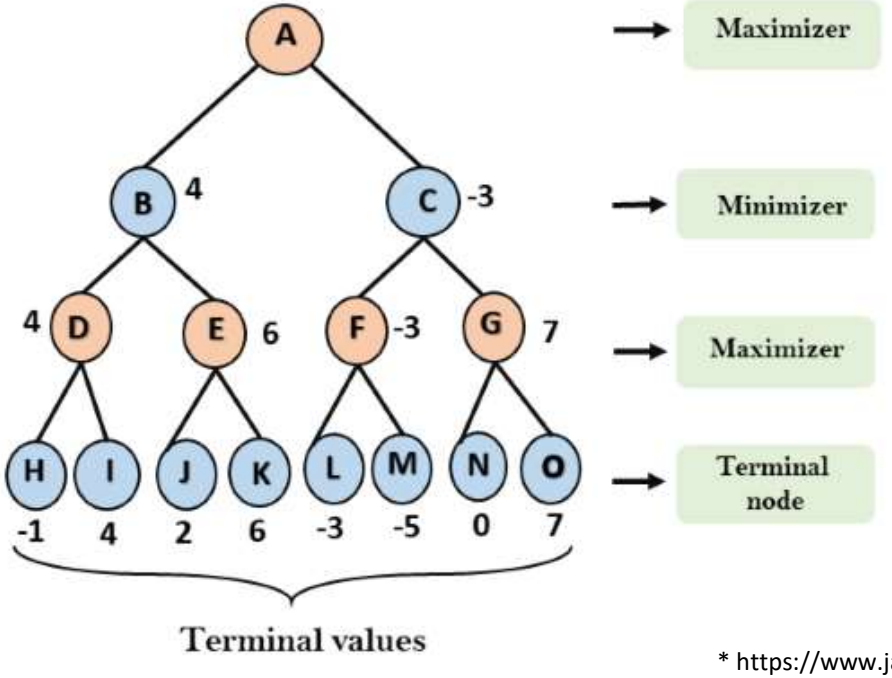
For node G max(0, -∞) = max(0, 7) = 7



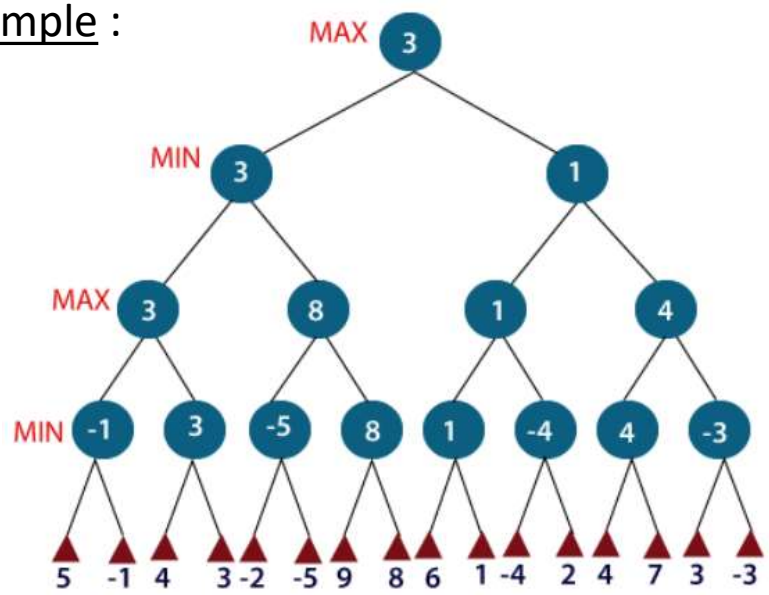
* https://www.javatpoint.com/mini-max-algorithm-in-ai

Minimax Search

- find the utilities value for the Maximizer, its initial value is -∞, so compare each value in terminal state with initial value of Maximizer and determines the higher nodes values.
- In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.
- Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



Example :



* https://www.javatpoint.com/mini-max-algorithm-in-ai