D Krishna Kethan Reddy
(23x080198).

ALM-1

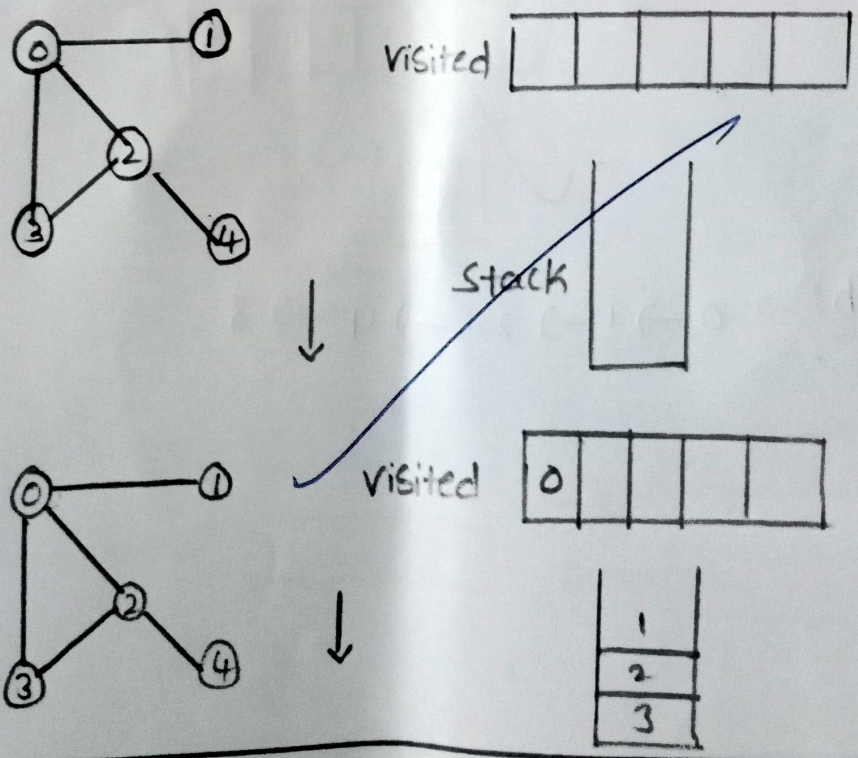
Uninformed Search:

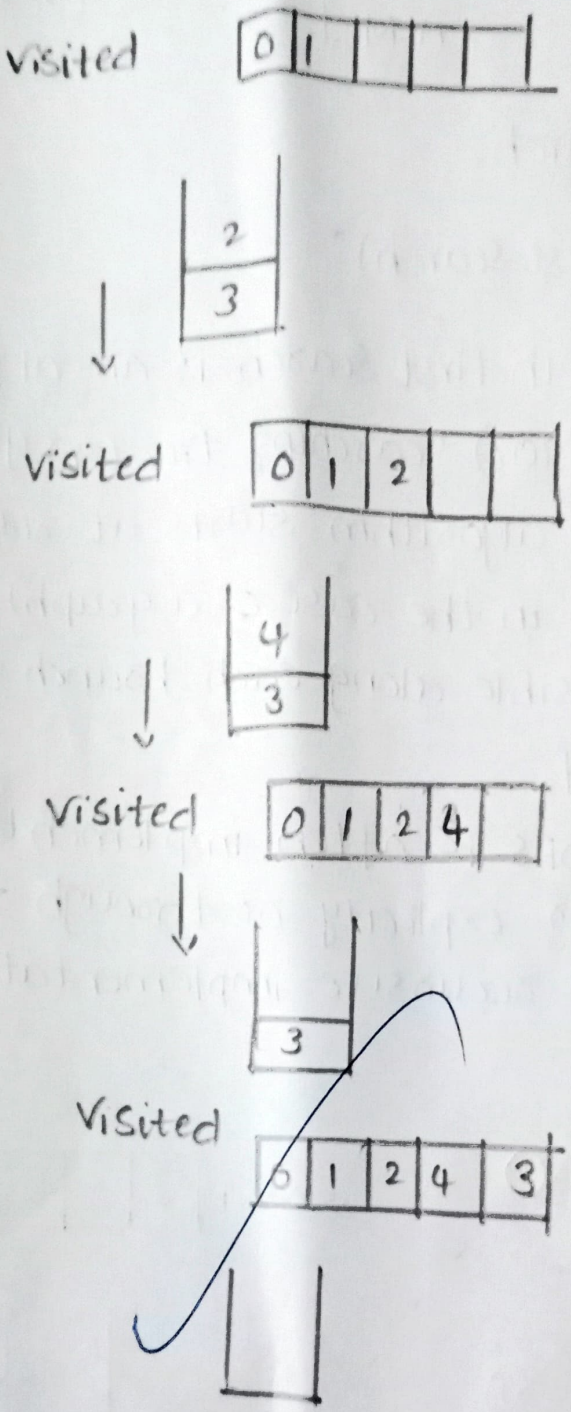
DFS (Depth First Search):

Depth First Search is an algorithm used for travelling (or) Searching tree (or) graph data structure. This algorithm starts at root (or any arbitary node in the case of a graph) and explores as far as possible along each branch before back tracking.

DFS is often implemented using a stack either explicity or through the system's call stack in a recursive implementation.

Ex:



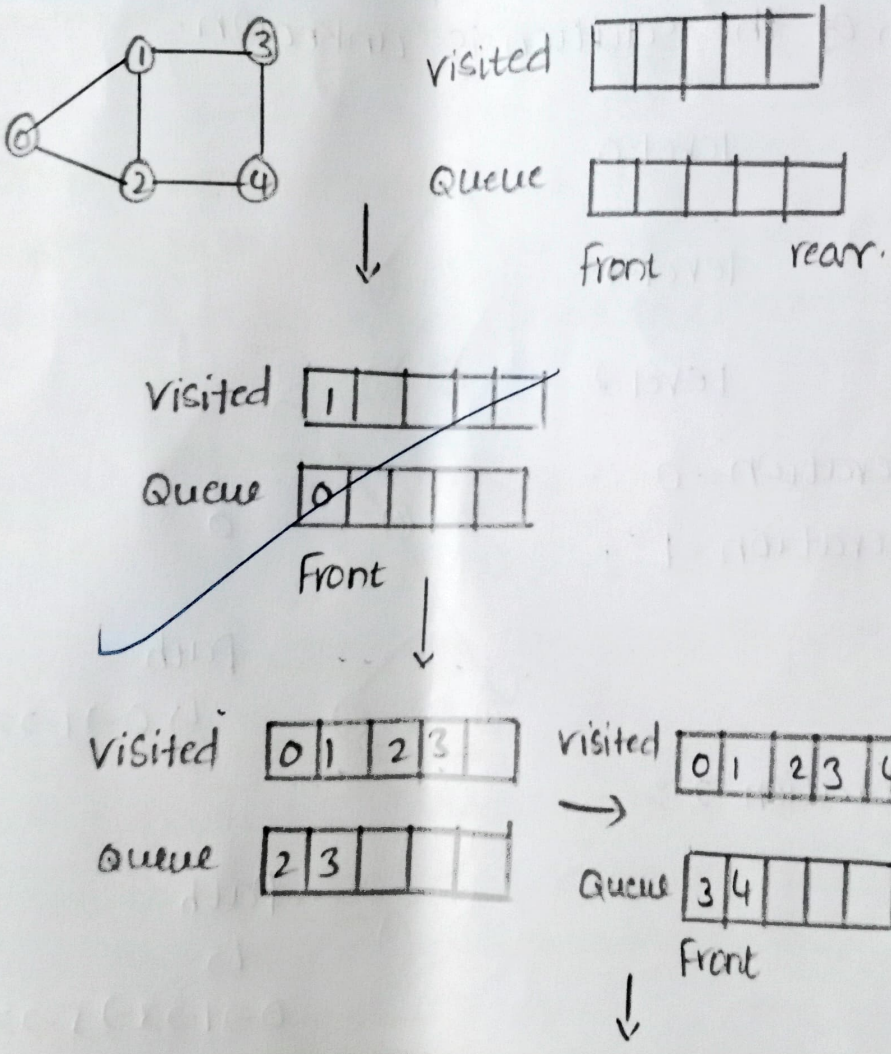


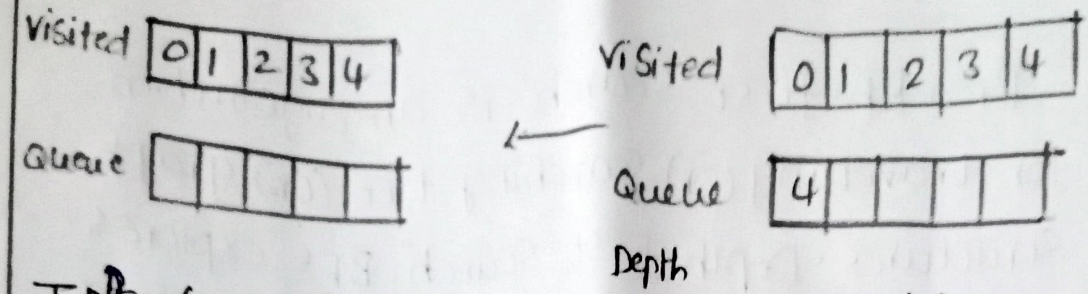
Path: → 0 → 1 → 2 → 4 → 3

BFS (Breadth first Search):

Breadth first Search is an algorithm used for travelling (or) searching tree (or) graph data structures. Depth first search. BFS explores the neighbour nodes at the present depth prior to moving on to nodes at the depth length.

BFS is often used to find the shortest path in an unweighted graph. as it explores all nodes as it the aurent depth before moving deeper.

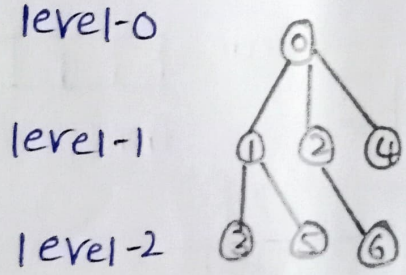




IDFS (Iteration deepening First Search):

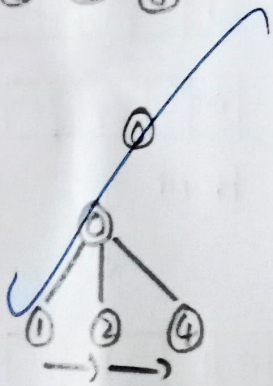
Iteration deepening depth First Search is an algorithm that combines both DFS and BFS concept of exploring the search space level by level. IDFS is particularly useful in scenarios where the search and space is large and the depth of the solution is unknown.

Ex:



iteration -0 :-

iteration -1 :-

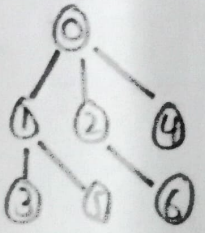


0

Path

↳ 0→1→2→4

iteration -2 :-



Path

↳

0→1→3→5→2→6

Informed Search:

A* Search algorithm:

A* Search algorithm is a popular and efficient algorithm used for finding the shortest path between nodes in a graph. It is widely used in various applications, such as path finding in games, robotics and AI. A* is both complete and optimal, meaning it will always find the shortest path if one exists and it does so efficiently by combining aspects of both depth-First-Search (DFS) and Breadth First Search (BFS).

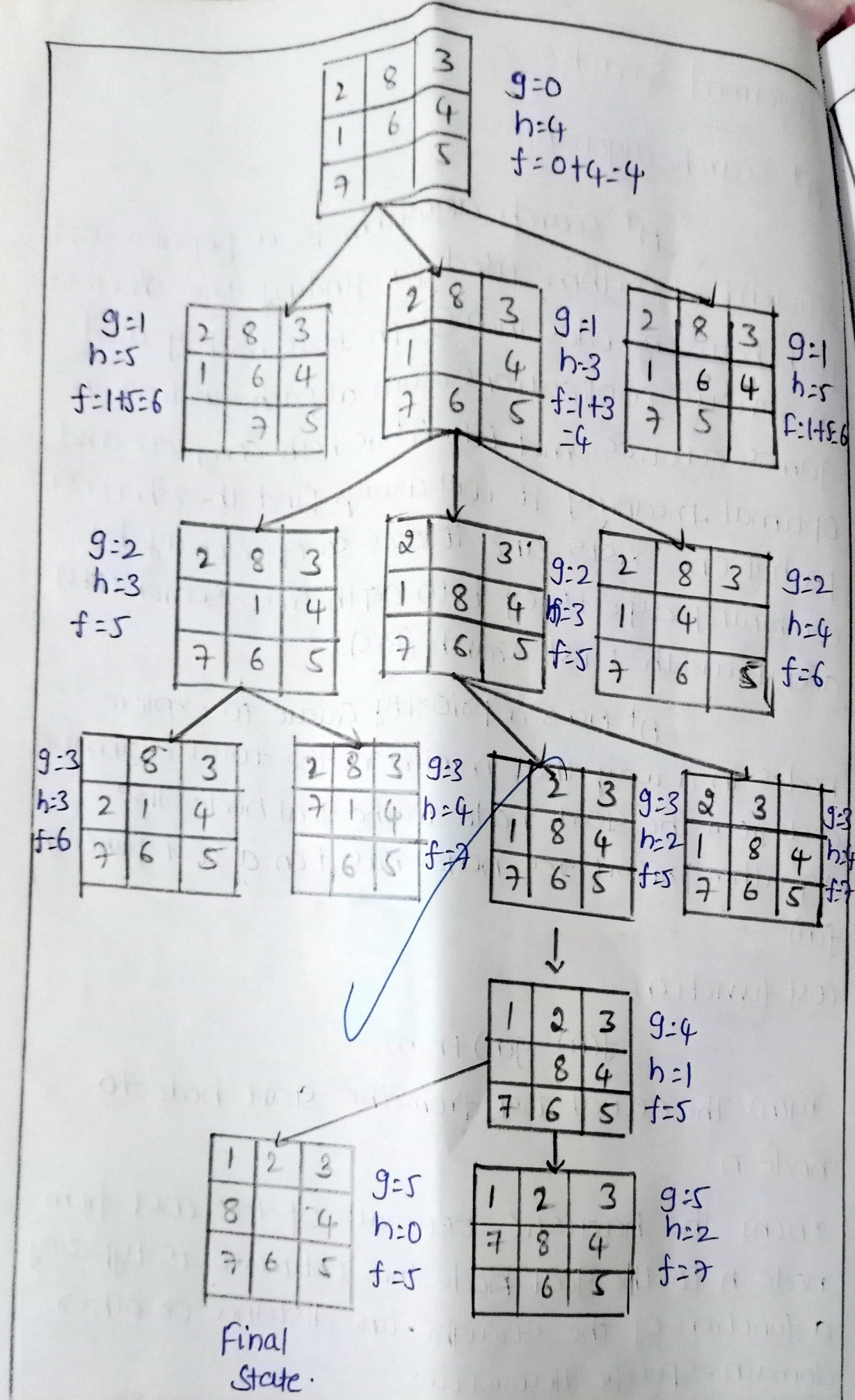
A* uses a priority Queue to explore nodes in a way that minimizes the total estimated cost from the start node to the goal node. The algorithm prioritizes nodes based on a cost function, f(n).

Cost function:

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

*g(n): The actual cost from the start node to node n

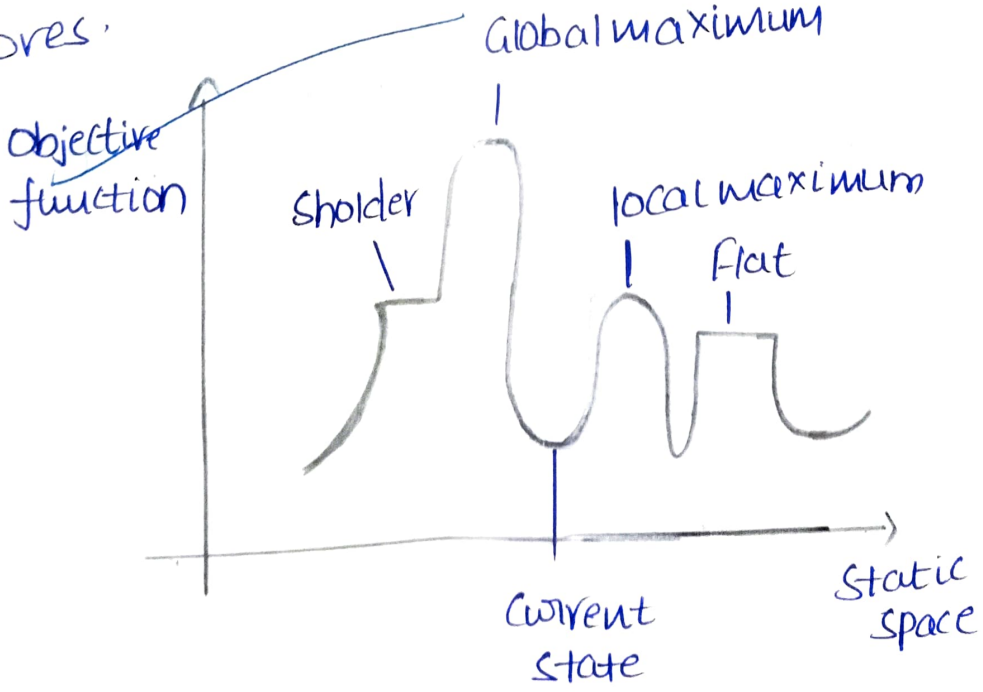
*h(n): The heuristic estimate of the cost from node n to the goal node this estimate is typically a function of the straight-line distance or other domain-specific heuristics.


g=0
h=4
f=0+4=4
g=1
h=5
f=1+5=6
g=1
h=3
f=1+3=4
g=1
h=5
f=1+5=6
g=2
h=3
f=5
g=2
h=3
f=5
g=2
h=4
f=6
g=3
h=3
f=6
g=3
h=4
f=7
g=3
h=2
f=5
g=3
h=4
f=7
g=4
h=1
f=5
g=5
h=0
f=5
g=5
h=2
f=7
Final State.

Hill Climbing Algorithm:

Hill climbing is a simple optimization algorithm used in AI to find the best possible solution for a given problem. It belongs to the family of local search algorithms and is often and in optimization problems where the goal is to find the best solution from a set of possible solutions.

* In hill climbing, the algorithm starts with an initial solution and then iteratively makes small changes to it in order to improve the solution. There changes are based on a heuristic fimation that evaluates the Quality of the solution. The algorithm continues to make there small changes until it radius a local maximum, meaning that no further improvement can be made with the current set of moves.



Min Max Algorithm:

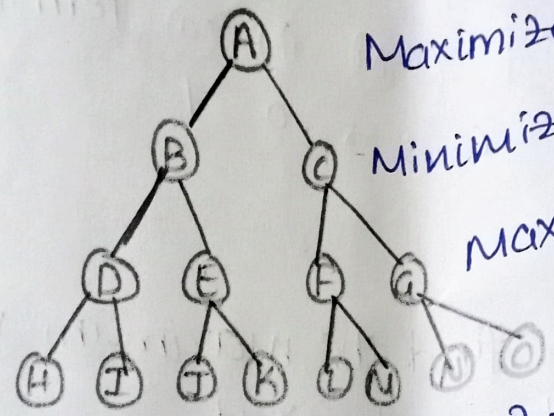
- Min Max Algorithm is a recursive or back-tracking algorithm which is used in decision making and game theory. It provides an optimal move for the players assuming that the opponent is also playing optimally.
- Min Max Algorithm is mostly used for game playing in AI. such as Chess, checkers, tic-toe, go and various two-players game. This algorithm computes the min-max decision for the current state.
- In this algorithm two players play the game one is max and other is Min.
- Both players of the game are opponent with each other where MAX will select the maximized value and Min will select the minimized value.
- The minmax algorithm performs a DFS Algorithm for the exploration of complete game tree.

Step-3:

For finding the minimizes take the minimum value of the child nodes.

B = MIN (D, E) = MIN (4, 6) = 4

C = MIN (F, G) = MIN (-3, 7) = -3



Maximizes

Minimizes

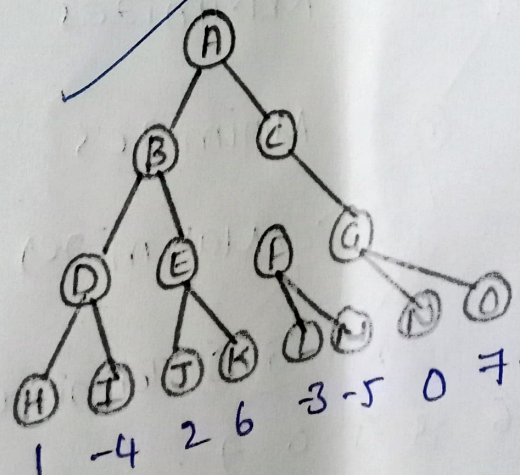
Maximizes

Terminal Nodes.

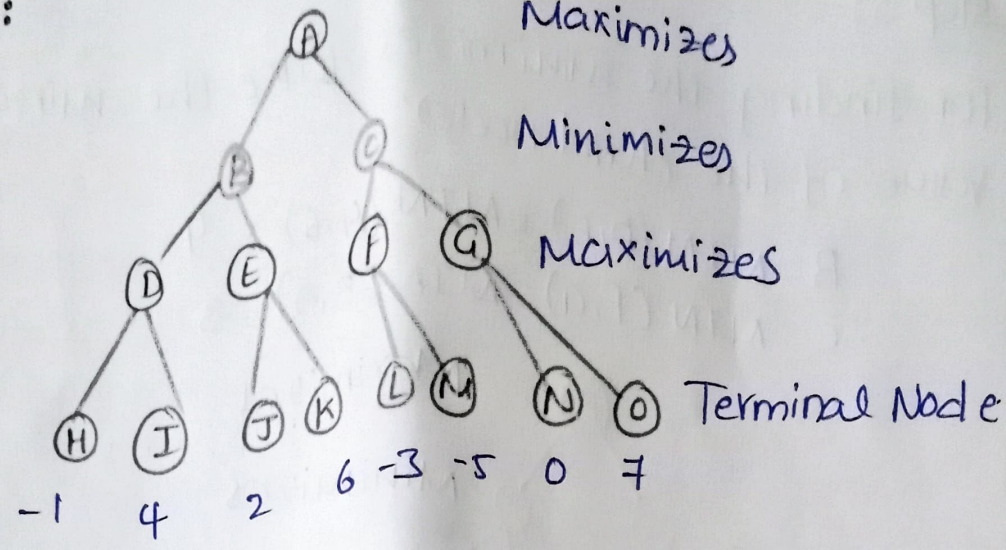
Step-4:

For finding the maximizes take the maximum values from the child nodes.

A = MAX (B, C) = MAX (4, -3) = 4



- Step-1:



Maximizes

Minimizes

Maximizes

Terminal Node

- Step-2:

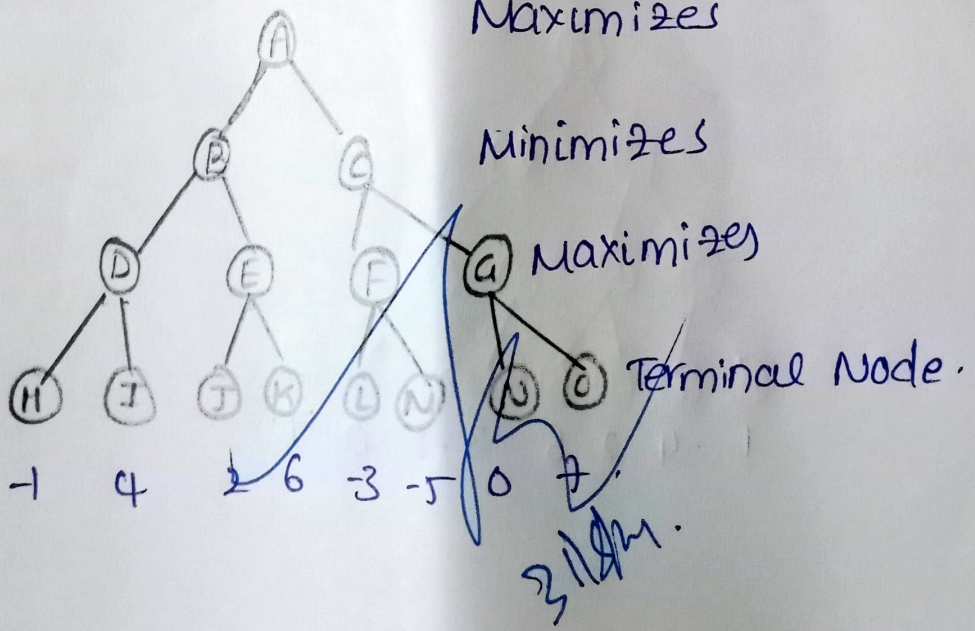
For the maximizes find the maximum values from the child nodes and fix it.

D = max(H, I) = max (-1, 4) = 4

E = max(J, K) = max (2, 6) = 6

F = max (L, M) = max (-3, -5) = -3

G = max (N, O) = max (0, 7) = 7.



Maximizes

Minimizes

Maximizes

Terminal Node.