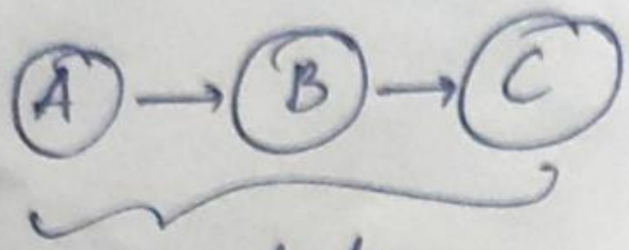
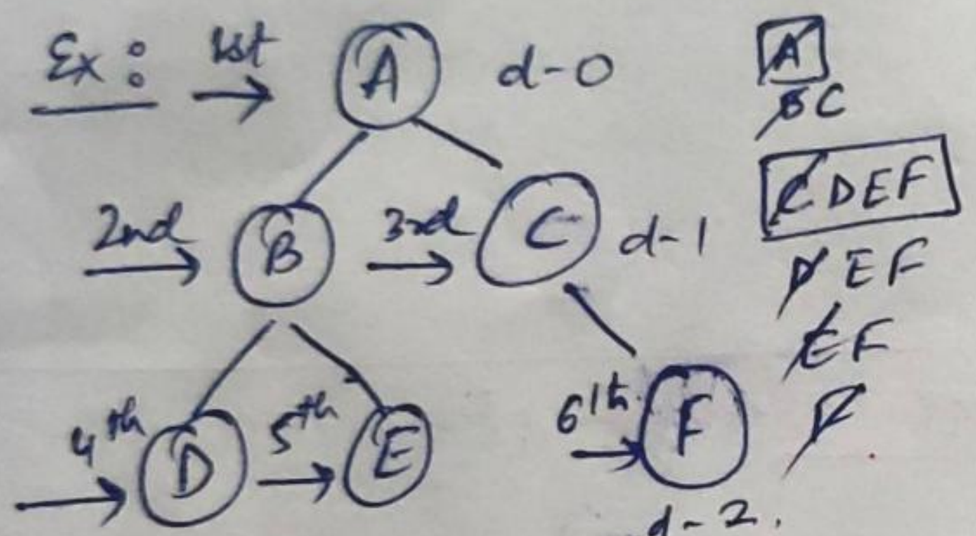
BFS Breadth first search

Brute force
→ [redacted] Algo → Uninformed search Technique
→ No Assumption how much time & space is required or cost to reach the goal.
→ Blind search Technique
→ Level order



No info. regarding any distance or time

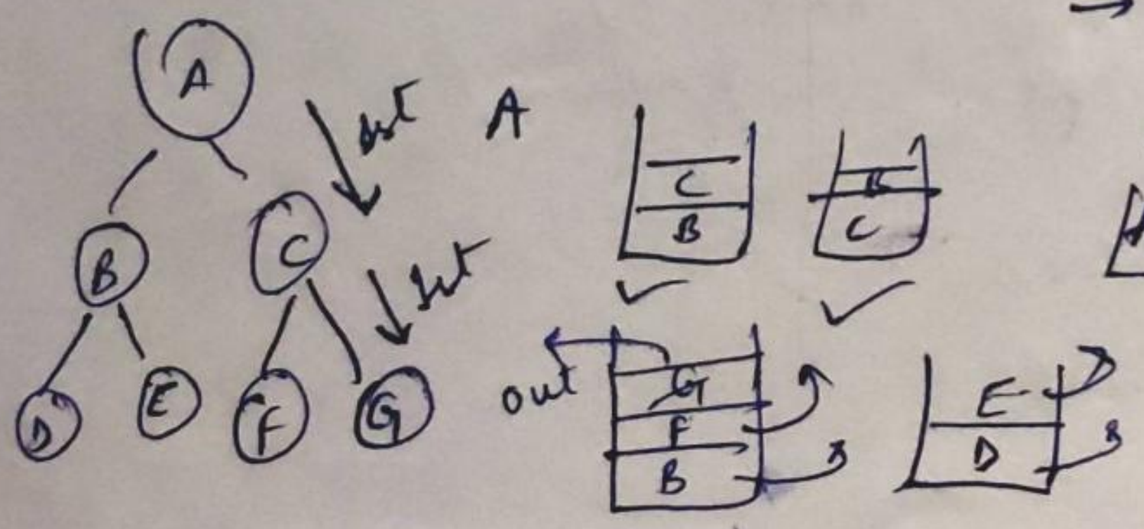
→ Shallowest Node
→ complete
→ optimal (shortest)
→ Time Complexity
$O(b^d)$ → depth. (Root → Leaf)
↓
branch factor
↓
how many max. children a node have. (n-ary tree)



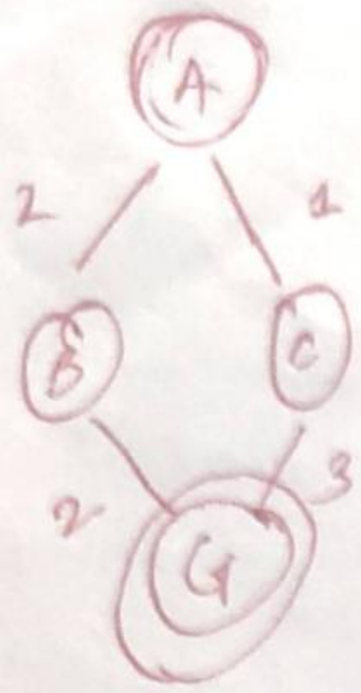
Ex: → If we're searching D.
d = 2
b = 2
2^2 - O(4)
with 4 search we'll get the sol. surely.

DFS. Depth first search

→ Uninformed search Technique → Stack (LIFO)
→ Estimation / Brute force / Blind search → Deepest Node
→ Sequence can be multiple → Incomplete (at times stuck in infinite search) → won't give Sol.
→ Non-optimal
→ Time Complexity



Uniform Cost



- Each goal state
- Optimal
- Minimizing Cost.

Depth Limited Search

→ It is similar to DFS with a pre-determined limit. Also, it has solved the major drawback of going into the infinite path in DFS. The node at the depth limit will treat as it has no successors nodes further.

→ DLS can be terminated using
- standard failure value (problem doesn't have any sol.)
- Cutoff failure value (will define no sol. for the problem)

→ Advantages: 1) It is memory efficient
2) Easy

→ Disadvantages: 1) Incompleteness.
2) Not optimal.

Iterative Deepening Search

Adv. - Complete

Disadv - consumes much memory

BFS DFS

combination of both search

combine the advantages of both searches

Adv - consumes less memory

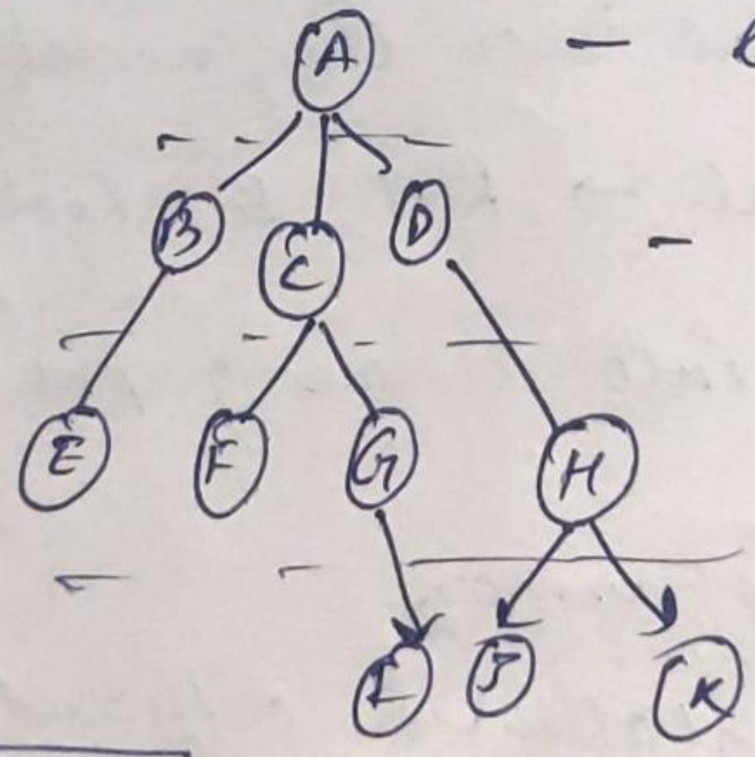
disadv - Incomplete

→ level wise search - (level 0 → level n)

search goal node in all levels & move forward if not found.

→ Example:

Goal Node - K

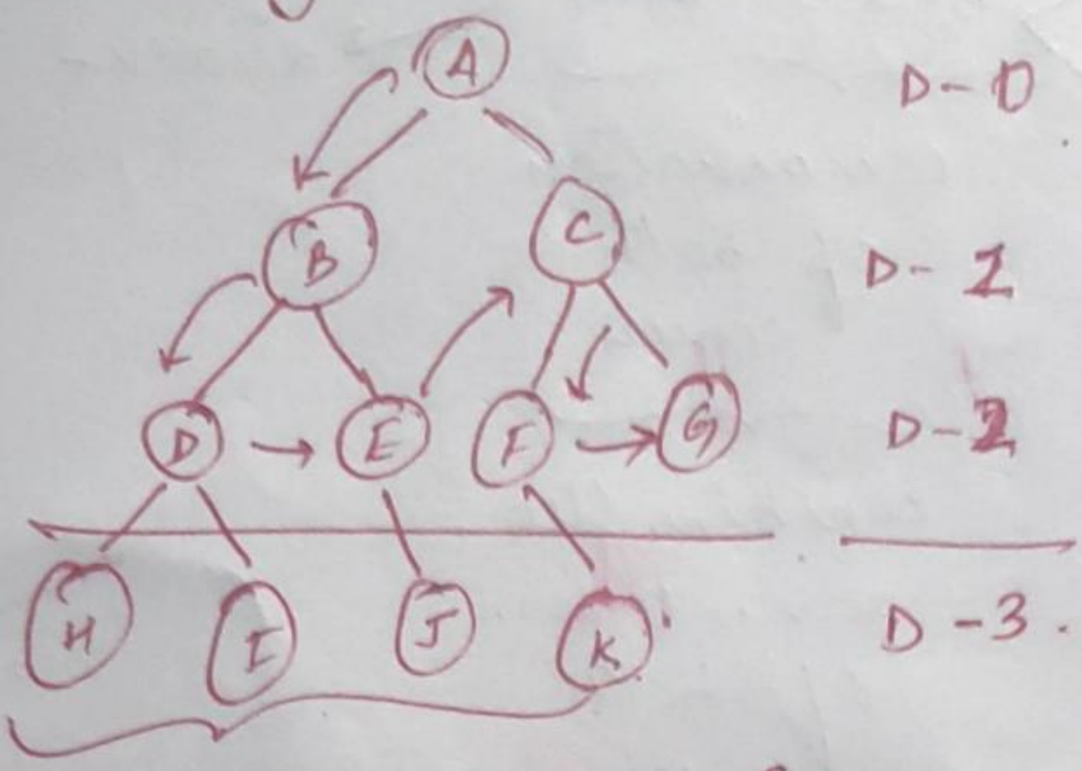


- level 0
- level 1
- level 2
- level 3.

level	Nodes Traverse
0	A
1	A B C D
2	A B E C F G D H
3	A B E C F G I D H J K → Goal Node.

limit = (depth = 2).

so, all the nodes at level 2 has to assume that they don't have any successor ahead.



Bidirectional Search

→ Searching into 2 ways → start from initial state / root node
                        → start from goal node

→ i.e we should know both the nodes priorly (Goal, Root)

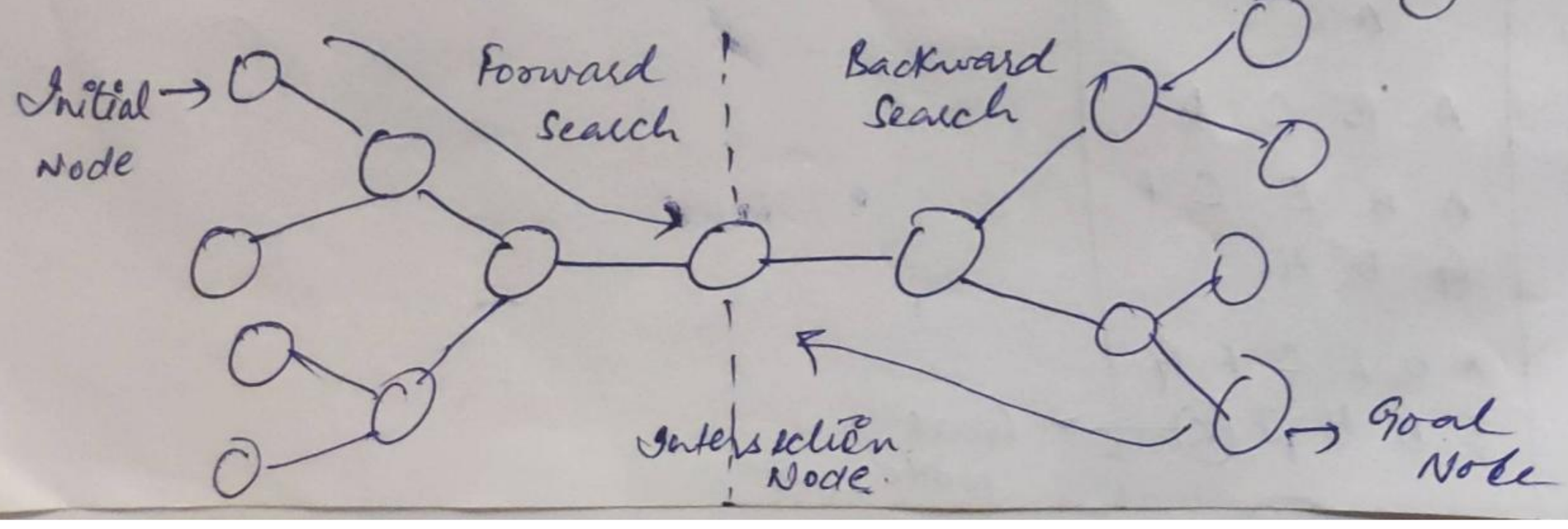
→ Reach from Goal → Root & Root → Goal.

→ Divide the graph into 2 sub-graphs → Initial vertex
                                      → Goal vertex

→ Complete

Advantage - Fast & less memory

Disadvantage - Implementation is difficult
               - Goal Node should be known priorly.



Informed Search Techniques

A- Star Algorithm (A* - search)

→ Finds shortest path through the search space using the heuristic function.

→ It uses h(n), & cost to reach the node n from the start state g(n).

(fitness No.) $f(n) = g(n) + h(n)$ ---- eq. 1

→ The algo. expands less search tree & provides optimal result faster.

→ It is similar to UCS except that it uses $g(n) + h(n)$ instead of $g(n)$.

→ A* uses search heuristic as well as the cost to reach the node. Hence, both costs are mentioned above in equation 1.

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

- $F(n)$ ↓ Estimated cost of the cheapest solution
- $g(n)$ ↓ cost to reach the node n from start state
- $h(n)$ → cost to reach from node to goal node.

Algorithm (working):

1) Check, if the list is empty or not; then return the failure & stop.

2) Select the node from the OPEN List which has the smaller value of evaluation function ($g+h$), if node n is goal node then return SUCCESS & STOP; else

3) Expand node n & generate all the successors & put n in the CLOSE LIST.

→ For Each successor 'n', check whether 'n' is already in a OPEN & CLOSED list.

→ If not, then compute evaluated function for 'n' & place it in OPEN list

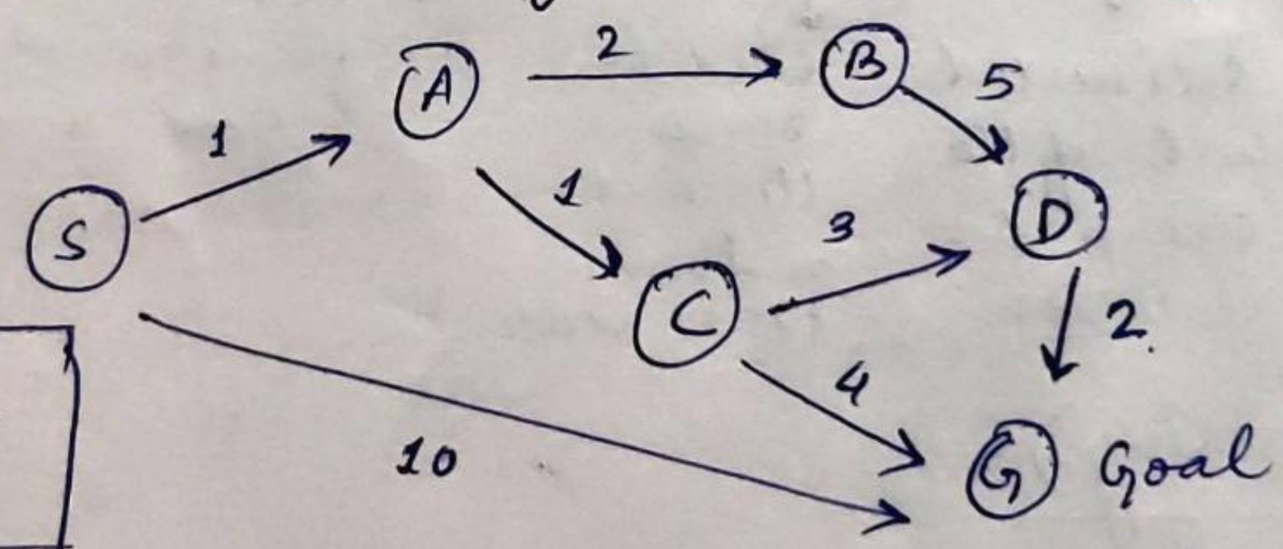
4) Else if node 'n' is already in OPEN & CLOSED, then should be attached to the back pointer which reflects the lower g(n) value.

5) Return to Step 2

Advantages: 1) Best algorithm than other search algo.
2) optimal & complete.
3) It can solve complex problems.

Disadvantages: 1) Always produces shortest path
2) Practical for various large-scale problems

Example:



state	heuristic value
S	5
A	3
B	4
C	2
D	6
G	0

① S → A : f(n) = 1 + 3 = 4. ✓
S → G : F(n) = 10 + 0 = 10 ×
F(n) = g(n) + h(n)

② S → A → B : f(n) = 3 + 4 = 7 ×
S → A → C : f(n) = 2 + 2 = 4 ✓

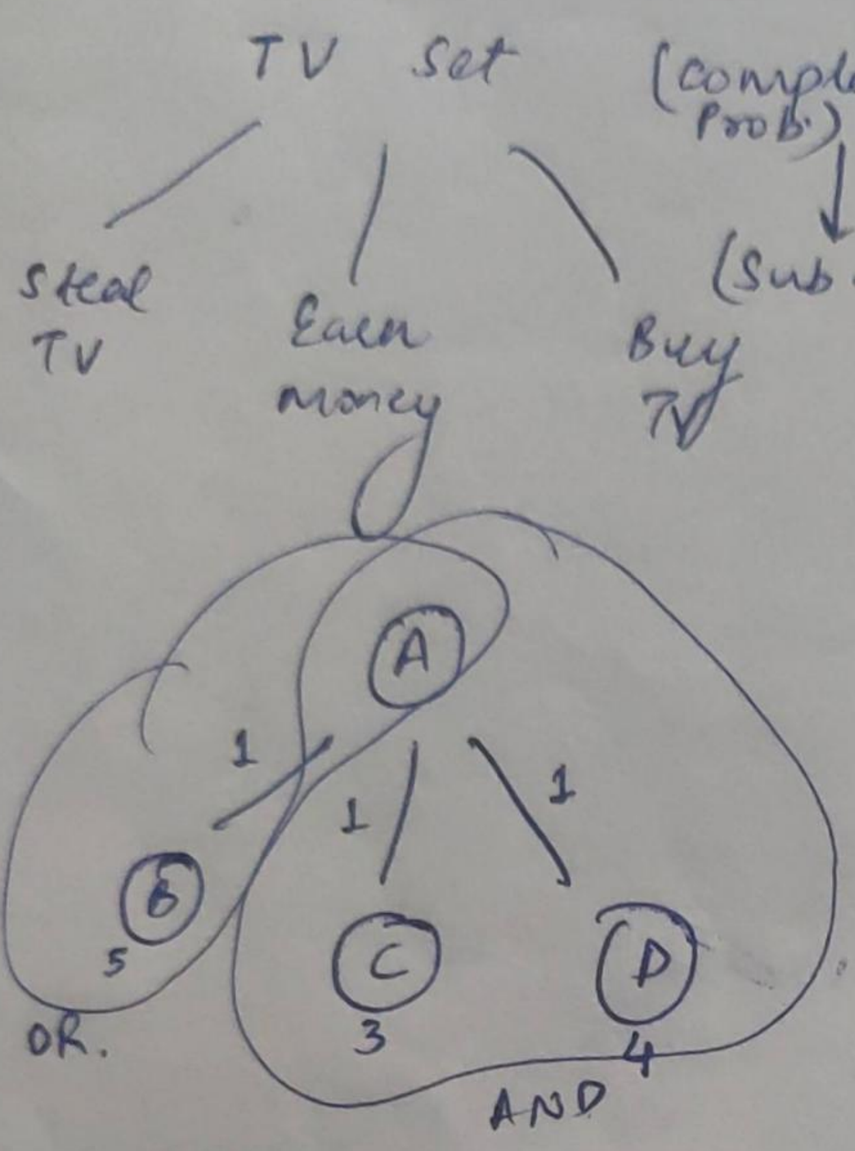
③ S → A → C → D : 5 + 6 = 11 ×
S → A → C → G : 6 + 0 = 6 ✓

least cost = 6

AO* Algorithm (AND-OR) Graphs

(3+1) + (4+1) = 9

→



AND.

Earn Money + Buy TV = Watch TV

OR.

Steal TV = Watch TV

5 + 1 = 6.

Easy to opt.

→

→ If there is any situation where AND-OR Graph are generated, then it can't be solved by A* Algo. alone. So AO* Algo. is required.

Simple Hill climbing Algo. (Local search Algo., Greedy approach; NO backtrack)

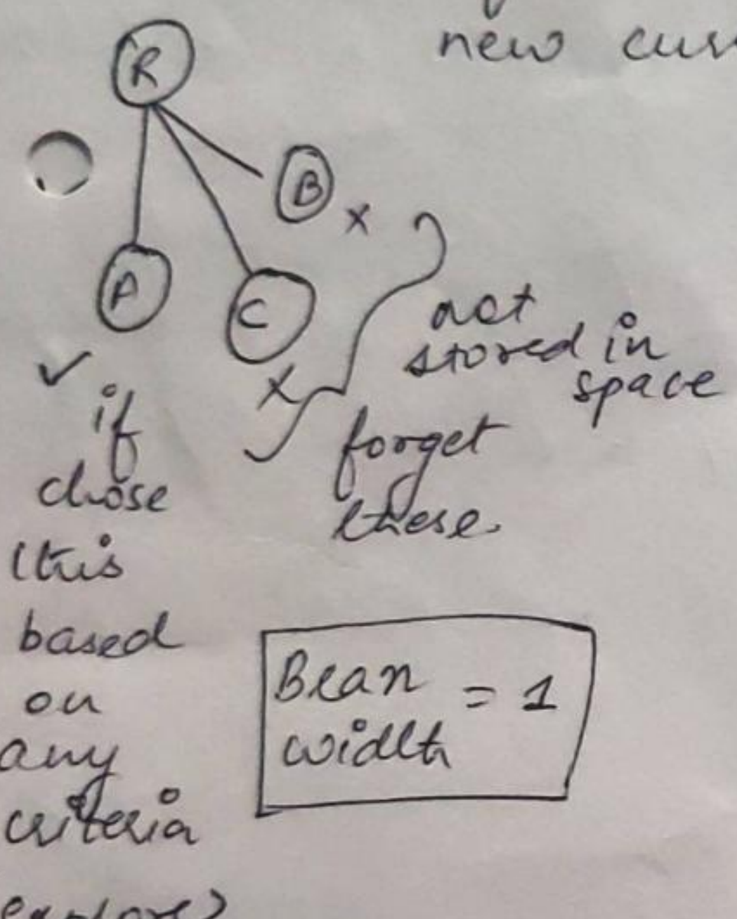
→ knowledge of only local domain, no idea of global domain.

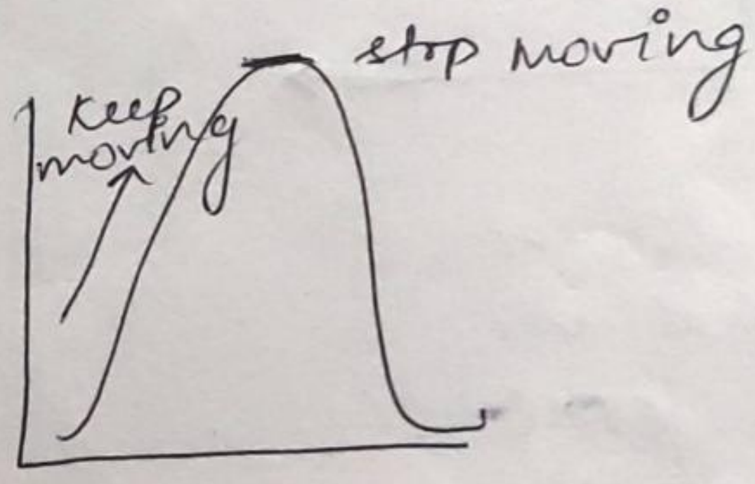
→ step 1 - Evaluate the initial state.

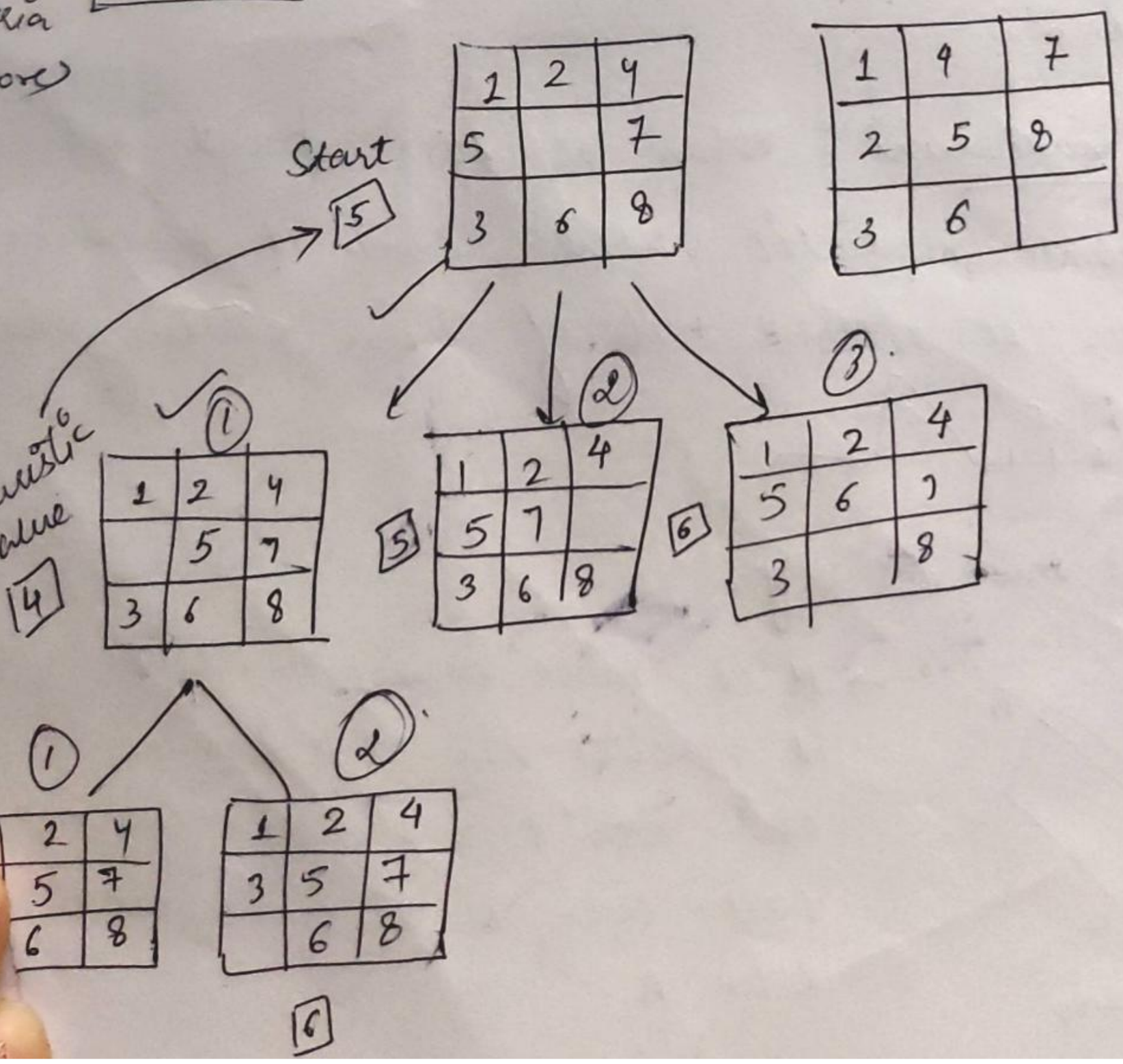
2 - Loop until a sol. is found or there are no operators left.

- select & apply a new operator.
- Evaluate a new state.
  - if goal found then quit.
  - if better than current state then it'll be the new current state.

(Move up an hill, blind folded) till reach saturation.

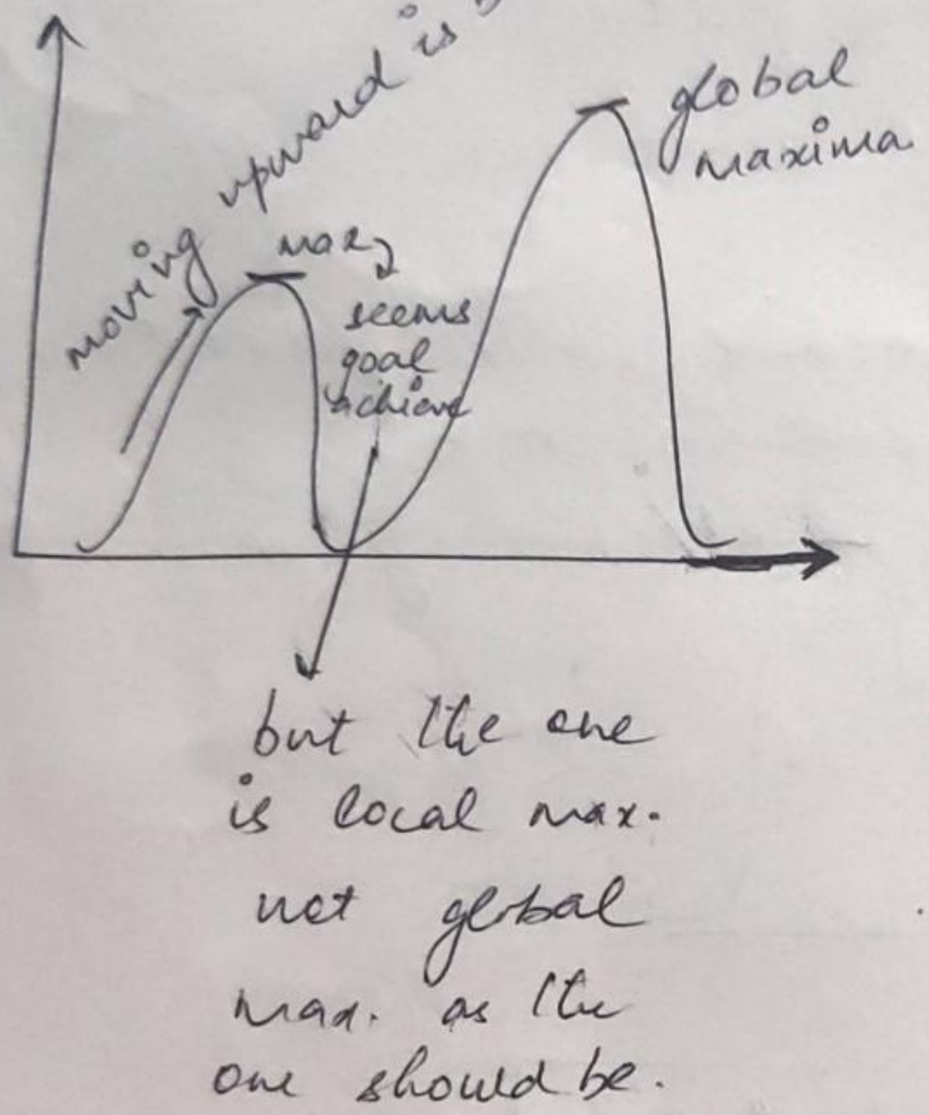






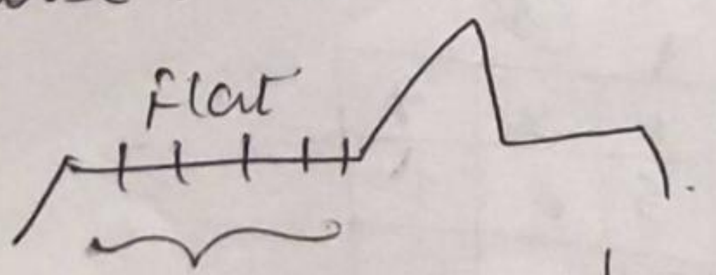
→ Limitations of Hill Climbing Algo:

1) Local Maximum: As the algo. is working blindly for local or not having domain knowledge of global level & is limited to local level of spread.

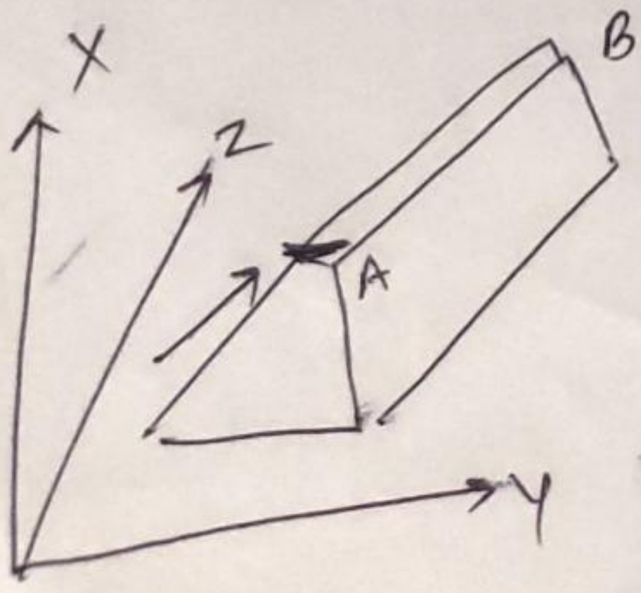


* But as you can see the example shared previously the logic is it'll move further as it'll get a better heuristic further. But in case if won't achieve then, it has to stop there.

2) Plateau / Flat maximum: When at some point all the neighbours are of same heuristic value, then it becomes difficult to choose. It restricts itself from reaching to solution.



3) Ridge:



→ It'll move in up direction till it reaches the top / max. height and won't change direction in any case. even if B is at more h than A.

→ Special case of local maximum

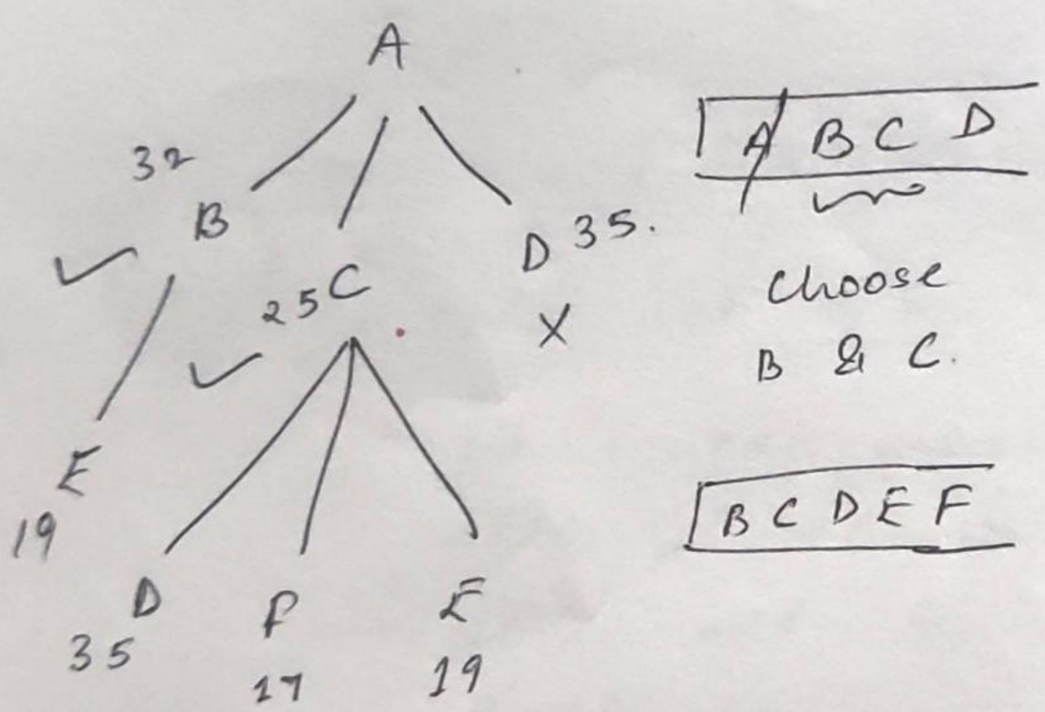
* Beam (Space) Search Algo - Based on Heuristic

Take care
Space complexity

→ Variation of Best first algo:

→ Beam width (B) is given.

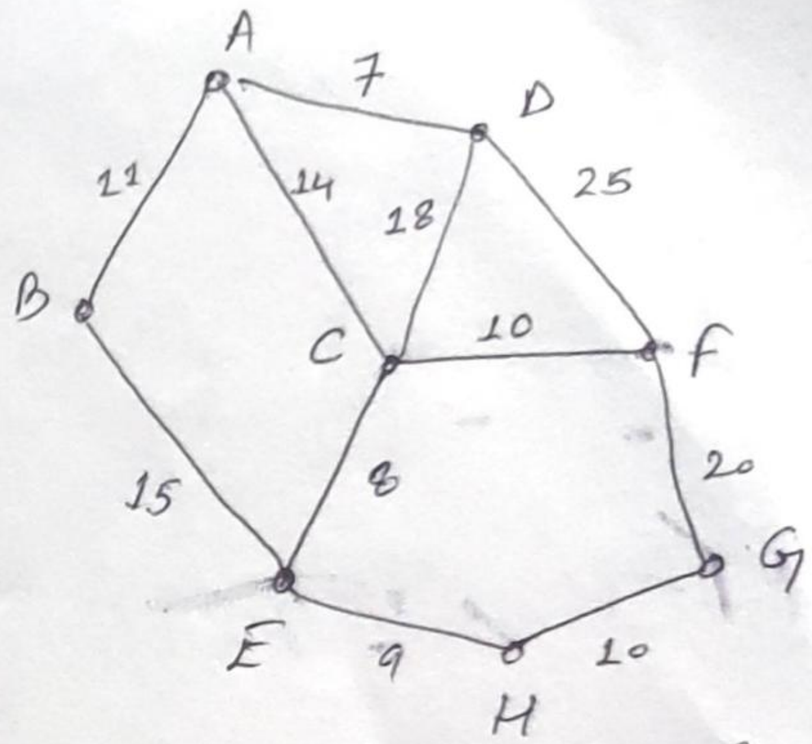
If B = 2 → defines to keep the best 2 values / nodes in priority queue.



A	B	C	D

Choose B & C.

B	C	D	E	F



straight line dist.

A → G - 40
B → G - 32
C → G - 25
D → G - 35
E → G - 19
F → G - 17
G → G - 10
G → G - 0

Adversarial Search (Game Playing).

→ Relates to competitive environment in which the agent goals are in conflict giving rise to adversarial search.

→ Types of Game Problems: 1) Adversarial Games.
2) Co-operative Games

1) Win of 1-player is loss of other
2) players have common interest & utility function.

Adversarial Games ← focused on. → Fully co-operative Games

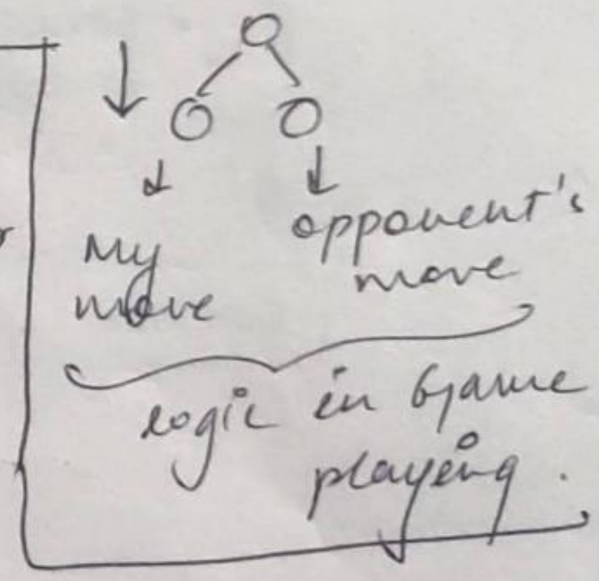
→ Min-Max Procedure / Algorithm: • Backtracking Algo. (move to leaf node & then come back after calculating values at leaf node to root node)

Game-Playing - Why not BFS approach

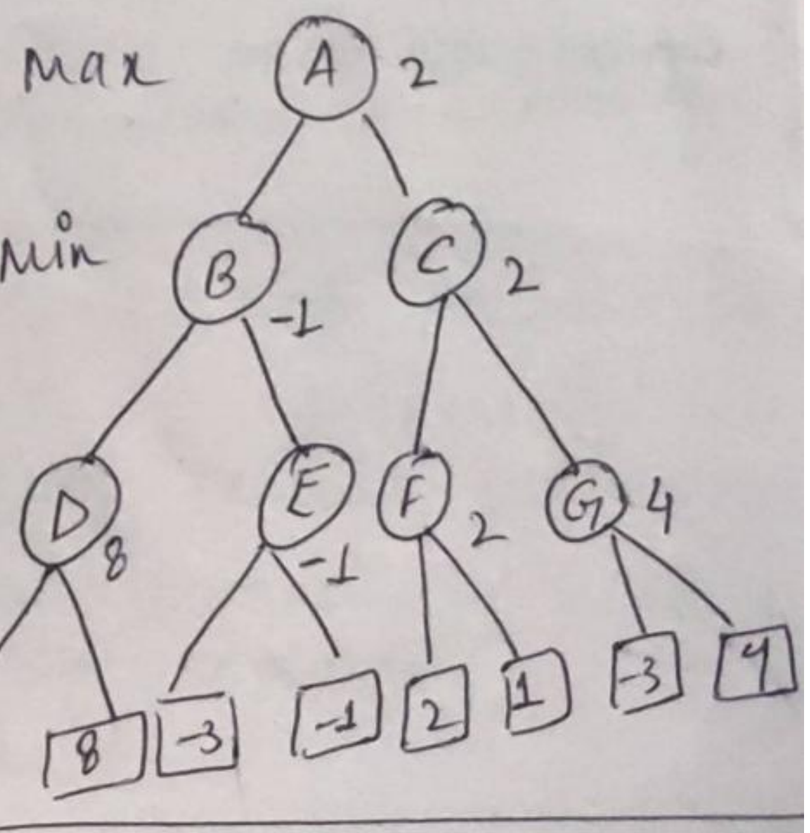
→ Horizontal moves aren't allowed in Game Playing
→ It'll not be implemented Practically.

• Best Move Strategy: Both players will try their best to win themselves & make other one lose the game. (Human ↔ Machine)
↓ can beat machine — ↓ can beat human

• Max-Min Strategy:
- will try to maximize its utility (best move)
- will try to minimize its utility (worst move)



Game Tree



• Time Complexity: $O(b^d)$ → depth.
↓ branching factor (children traverse possibly)

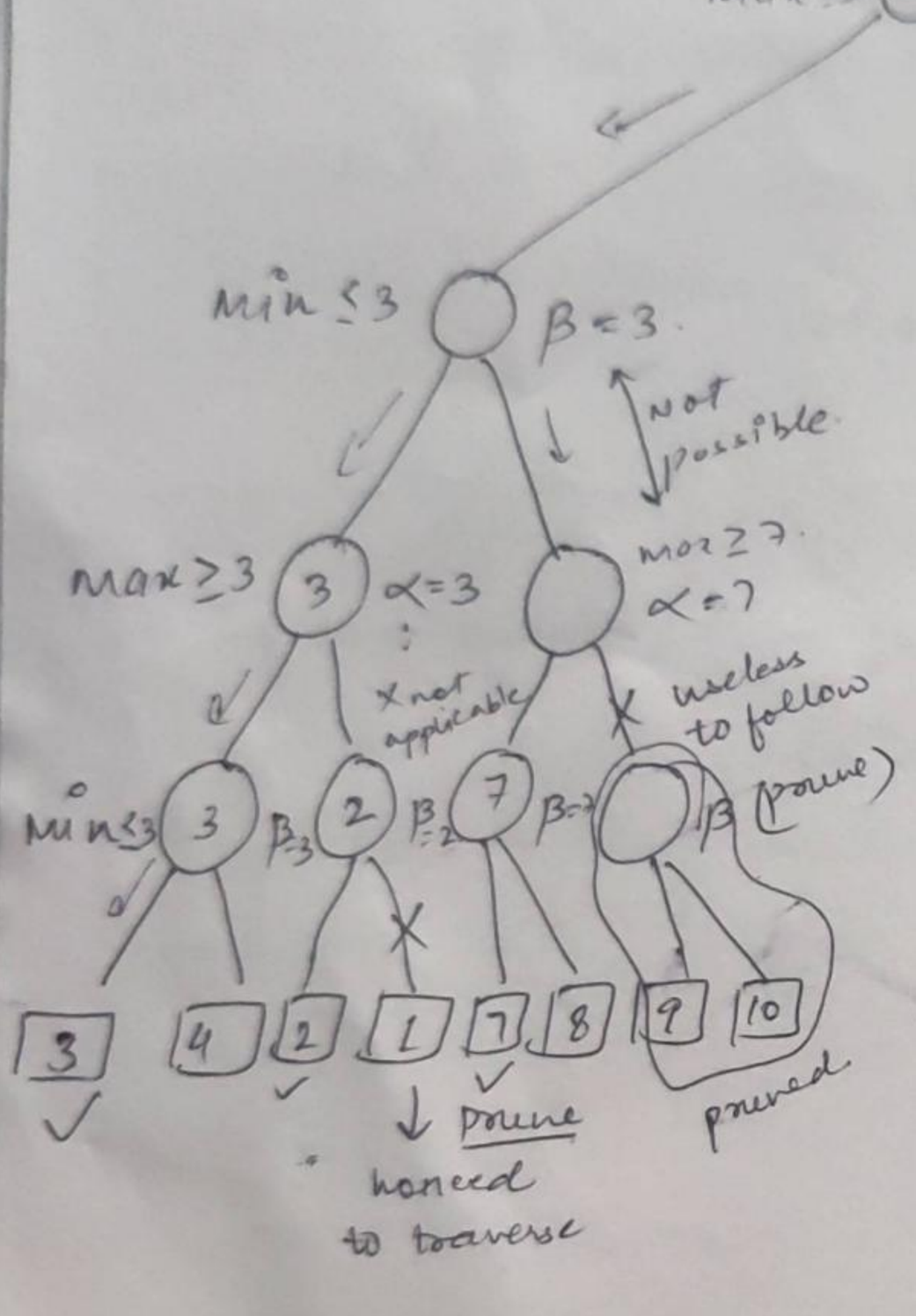
Feasible ✓

→ Tic Tac Toe - Not Feasible

For Chess: $(35)^{100}$ → 50 for each players
↓ choices possible

(α-β) Alpha-Beta Pruning

→ Ex: Chess is not feasible to followed by Min-Max algo.
→ Advanced version of Min-Max Algo.
→ Cut-off search by exploring less no of nodes.
→ Pruning is done for the nodes useless or time consuming



Best / Avg. Case - $O(b^{d/2})$ Worst case - $O(b^d)$

→ Thus, Alpha-beta pruning is much better than Min-Max Algo.

Due to complexity issues of Minmax Algo- α-β pruning is preferred.

1) Dynamic Pruning of redundant branches of search tree.
α-β pruning — identify the probable suboptimal branch of the search tree before it is fully explored.
— Eliminate the suboptimal branch.
2) Early cutoff of the search tree: It uses imperfect minimax value estimate of non-terminal states (positions).

Introduction to Game Playing

→ Introduce multi-agent environment.
→ Min-max
→ Alpha-beta pruning
→ Game Tree / Search Tree / Space Graph.
→ It is defined as a search problem using:
- Initial state
- Successor function
- Goal Test
- Path Cost / Utility / Pay off function.

→ A Game must feel natural-
- obey laws of the game.
- characters aware of environment
- Path finding
- decision making
- planning

Game AI is about illusion of human behaviour-
- smart to certain extent
- non-repeating behaviour
- Emotional influence
- being integrated in the environment

→ Game AI needs:
- Knowledge Based system
- ML
- Multi-Agent system
- CG & animation
- DS