SEARCH STRATEGIES:

Search Algorithm

uninformed/Blind (6)
- → Breadth first search
- → uniform cost search
- → depth first search
- → depth limited search
- → Iterative deeping depth first search
- → Bi-directional search

Informed search (5)
- → Best-first search
- → A* search
- → AO* algo
- → problem reduction
- → Hill climbing

Types of search algo
- → uninformed (Blind search) search
- → informed (Heuristic search) search

depth first search: (DFS)

* recursive / non recursive
* tree / graph traversal

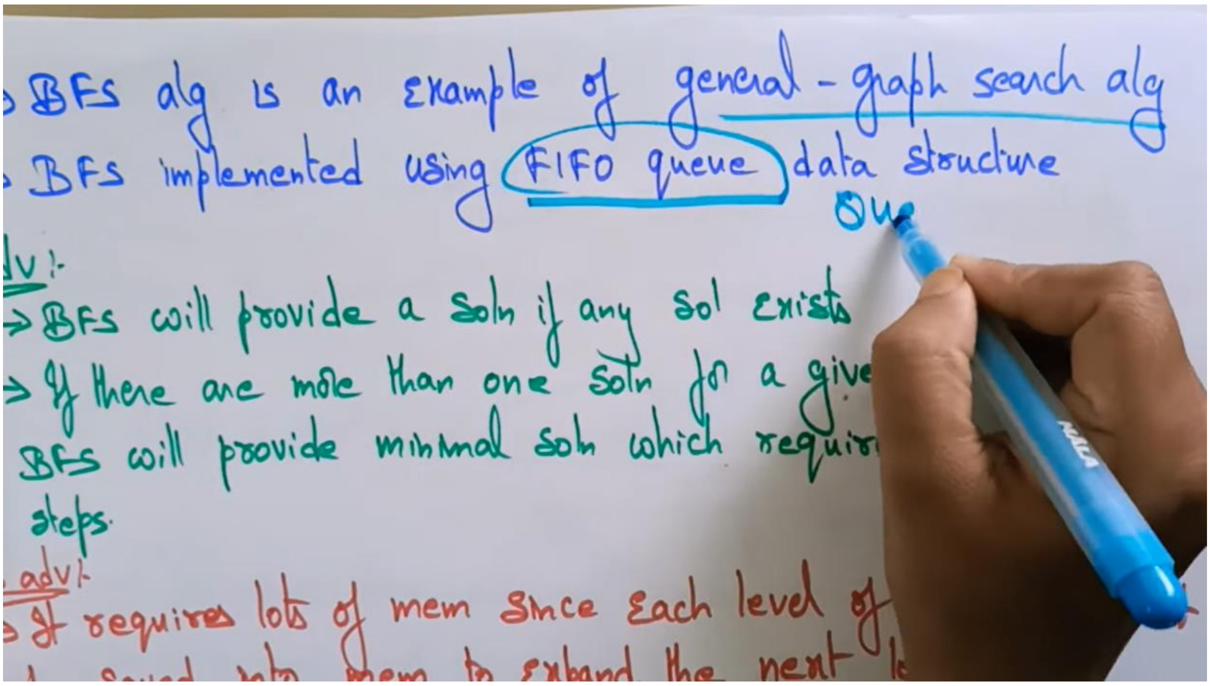
UNINFORMED SEARCH STRATEGIES: BFS

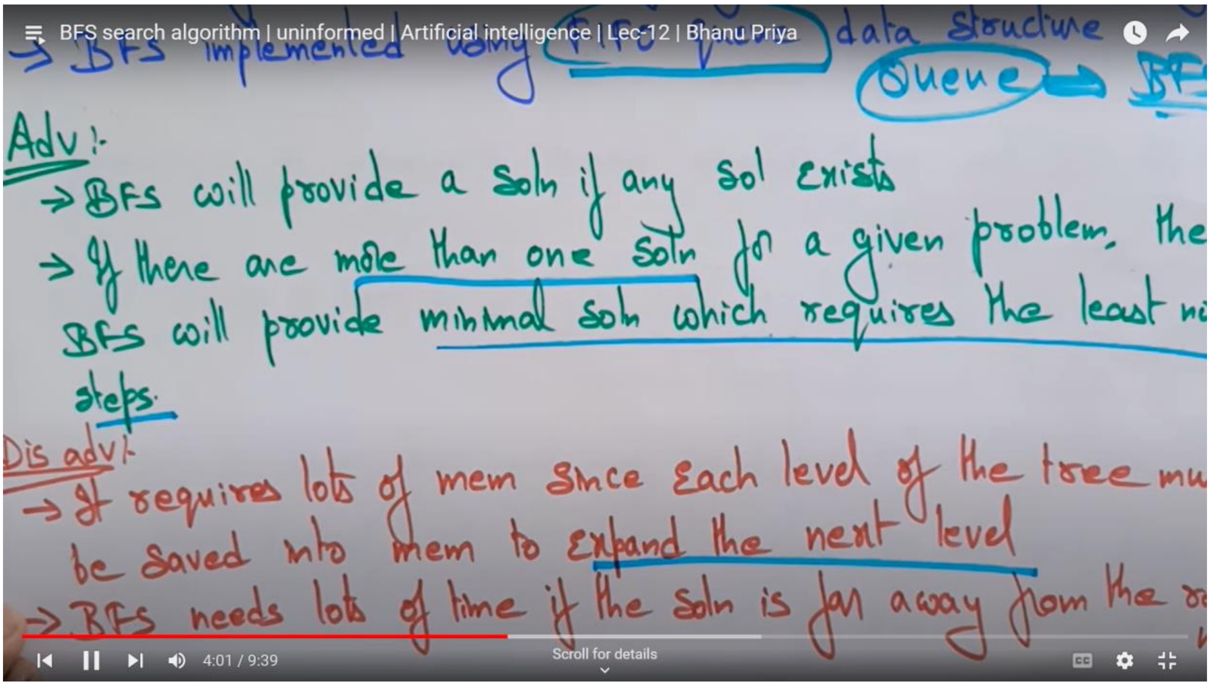
① Breadth-first Search (BFS) →

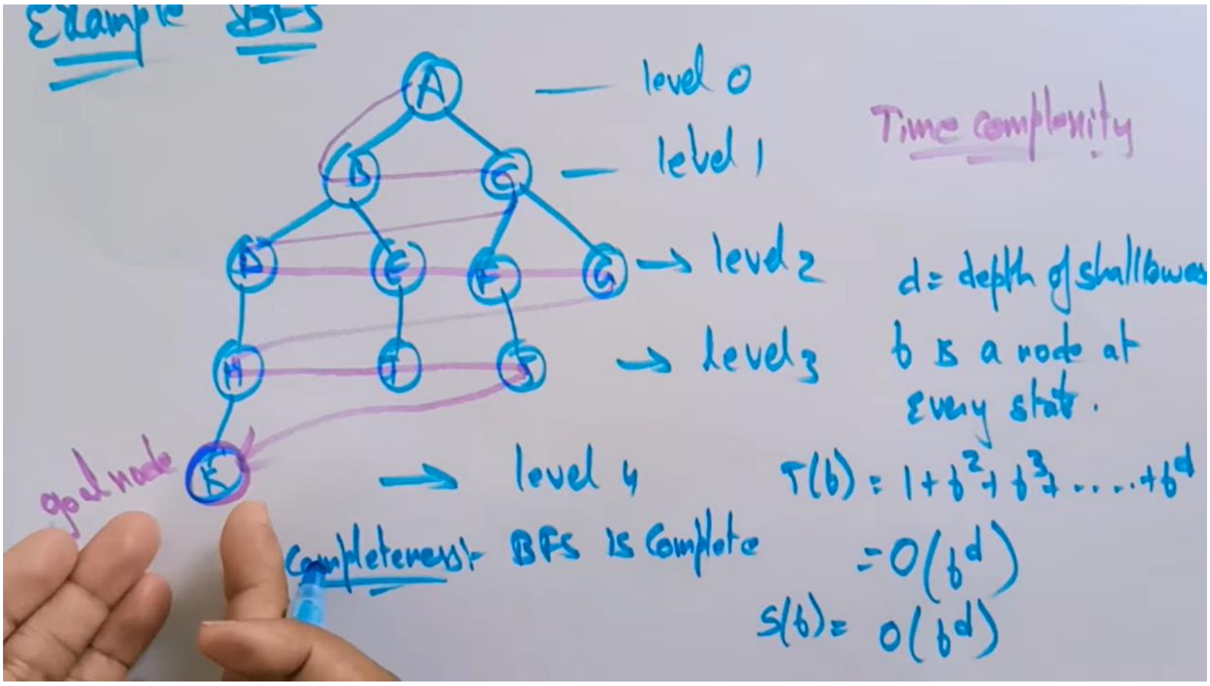
→ It is the most common search strategy for traversing a tree or graph.

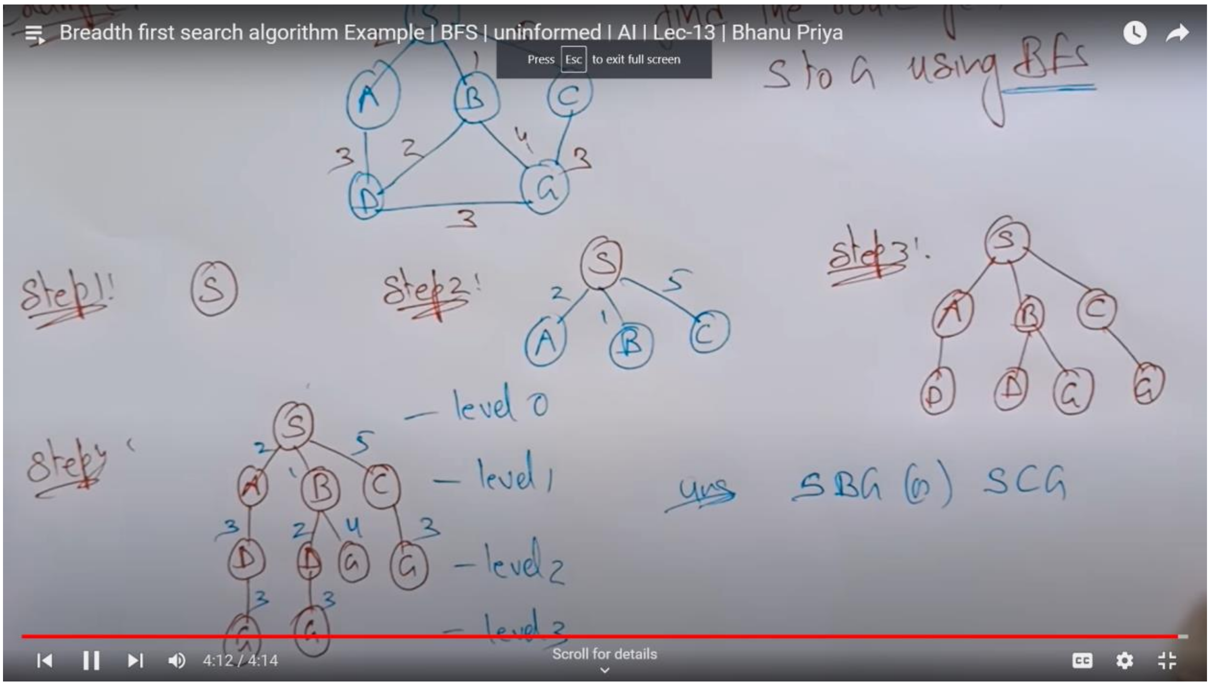
→ This alg searches breadthwise in a tree or graph, so it is called breadth-first search

→ BFS alg starts searching from the root node of the tree and expands all successor node at the current level before moving to node of next


BFS alg is an example of general - graph search alg
BFS implemented using FIFO queue data structure
Adv:-
→ BFS will provide a soln if any sol exists
→ If there are more than one soln for a
BFS will provide minimal soln which
steps.
→ It requires lots of mem since each level of


BFS search algorithm | uninformed | Artificial intelligence | Lec-12 | Bhanu Priya
data structure
Queue
Adv:-
→ BFS will provide a soln if any sol exists
→ If there are more than one soln for a given problem, the
BFS will provide minimal soln which requires the least
steps.
Dis adv:-
→ It requires lots of mem since each level of the tree
be saved into mem to expand the next level
→ BFS needs lots of time if the soln is far away from the
4:01 / 9:39
Scroll for details


Example BFS
level 0
level 1
level 2
level 3
level 4
goal node
Time complexity
d = depth of shallowest
b is a node at every state.
T(b) = 1+b²+b³.....+b^d
= O(b^d)
S(b) = O(b^d)
Completeness:- BFS is Complete


Breadth first search algorithm Example | BFS | uninformed | AI | Lec-13 | Bhanu Priya
Press Esc to exit full screen
S to G using BFS
Step 1:
Step 2:
Step 3:
Step 4
level 0
level 1
level 2
level 3
ans SBG (or) SCG
4:12 / 4:14
Scroll for details

DEPTH FIRST SEARCH

2) Depth first Search

→ It is a recursive alg for traversing a tree ~~of~~ or graph ds.

→ It is called DFS because it starts from the root & follows each path to its greatest depth node before moving to the next

→ DFS uses a stack ds for its implementation

→ The Process is similar to BFS alg.

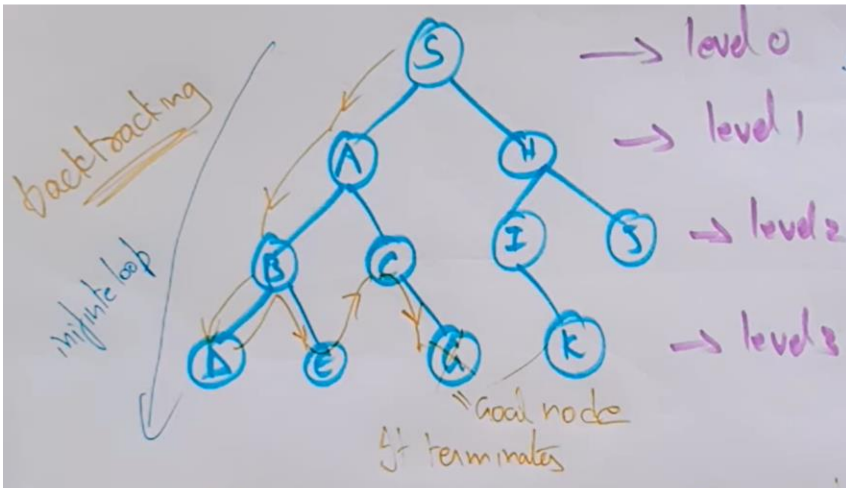
Adv:-

→ It requires very less memory as it only needs to store a stac of the nodes on the path from root node to the current node

Dis adv:-

→ There is the possibility that many states keep ~~re-counting~~, re-occurring, & there is no guarantee of finding the soln.

→ ~~B~~DFS alg goes for deep down searching and sometime it ma go to the infinite loop.



$s(c) = \cancel{O(b^m)}$

$= O(bm)$

Optimal :- non-optimal

Completeness: It is complete within finite state space

Time Complexity: $T(n) = 1 + n^2 + n^3 + \dots + n^m = O(n^m)$

m = max depth of any node

③ **Depth-limited Search Algorithm:**

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

Depth-limited search can be terminated with two conditions of failure:

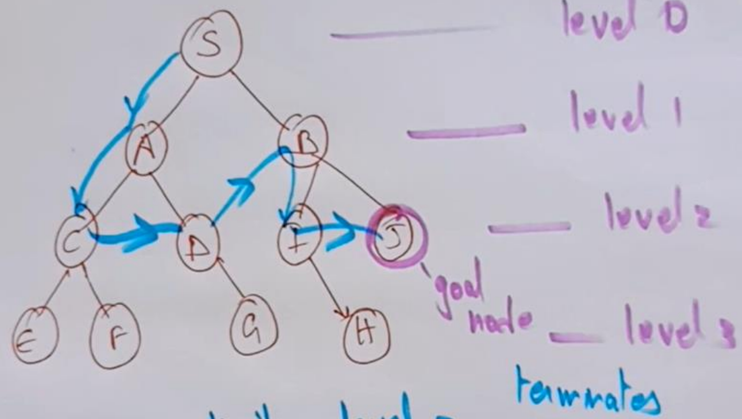
- Standard failure value - It indicates that problem does not have any soln.
- Cutoff failure value :- It defines no soln for the problem within a given depth limit.

Advantages :-

- It is memory efficient.

Disadvantages :-

- incompleteness
- It may not be optimal if the problem has more than one solution.



limit - level 2

It is not optimal.

$TC \Rightarrow O(b^{l})$

$SC \Rightarrow O(b \times l)$

DEPTH FIRST ITERATIVE DEEPENING SEARCH ALGORITHM:

I) Depth First Iterative deepening Search:

→ It is a combination of BFS & DFS

→ It will find Goal node at any way

→ Both Advantages of BFS & DFS are achived

→ it find goal node

→ it consumes less memory

adv:

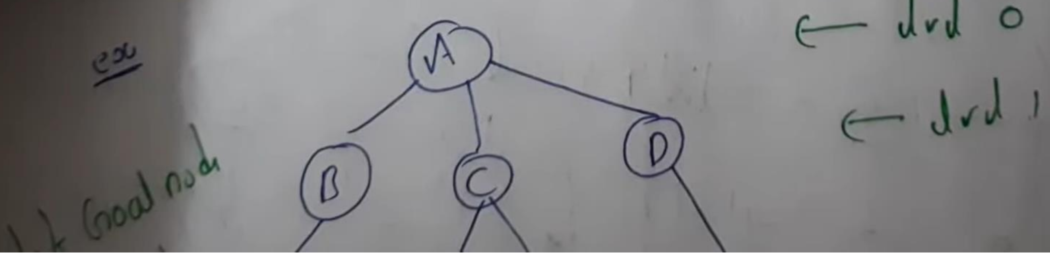
→ It will find goal node

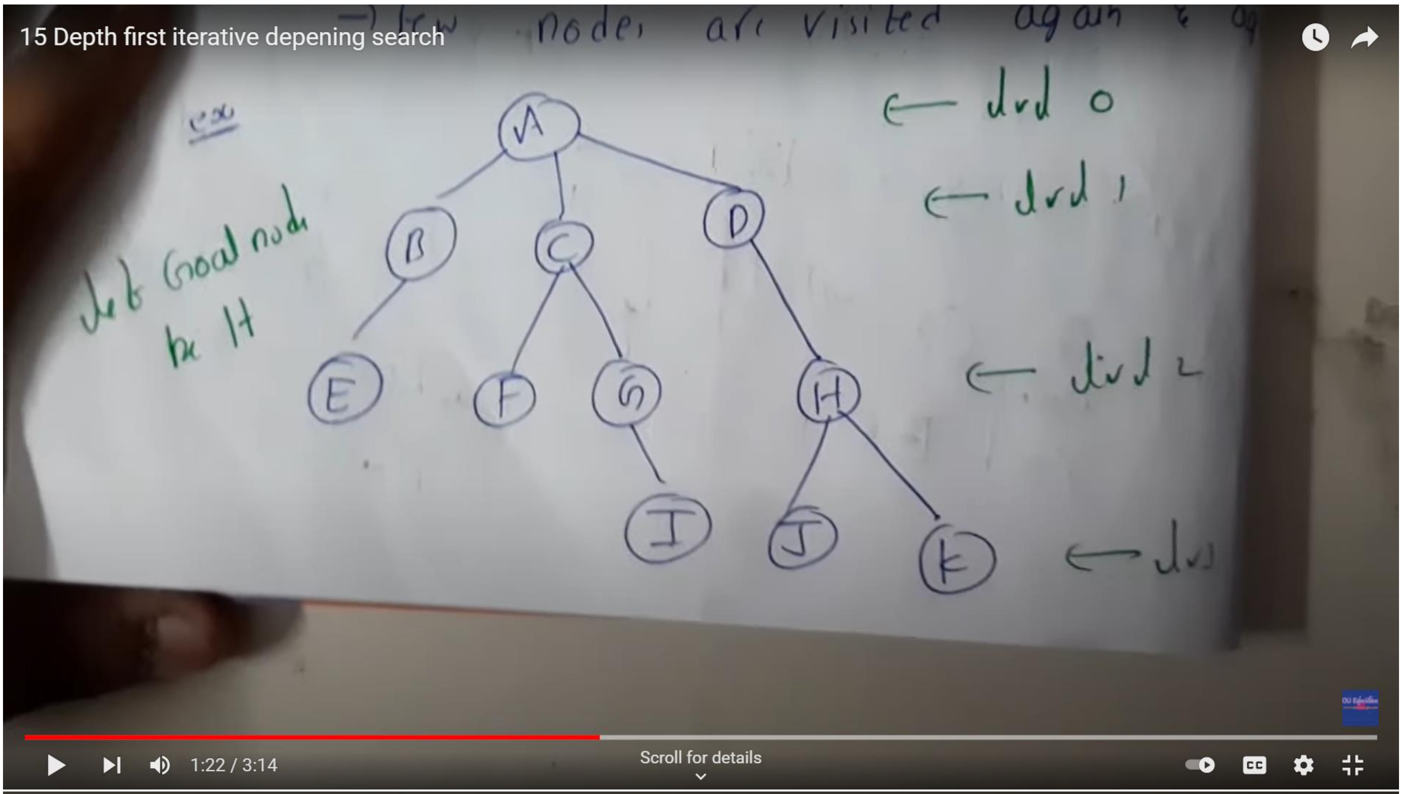
→ It consumes less memory

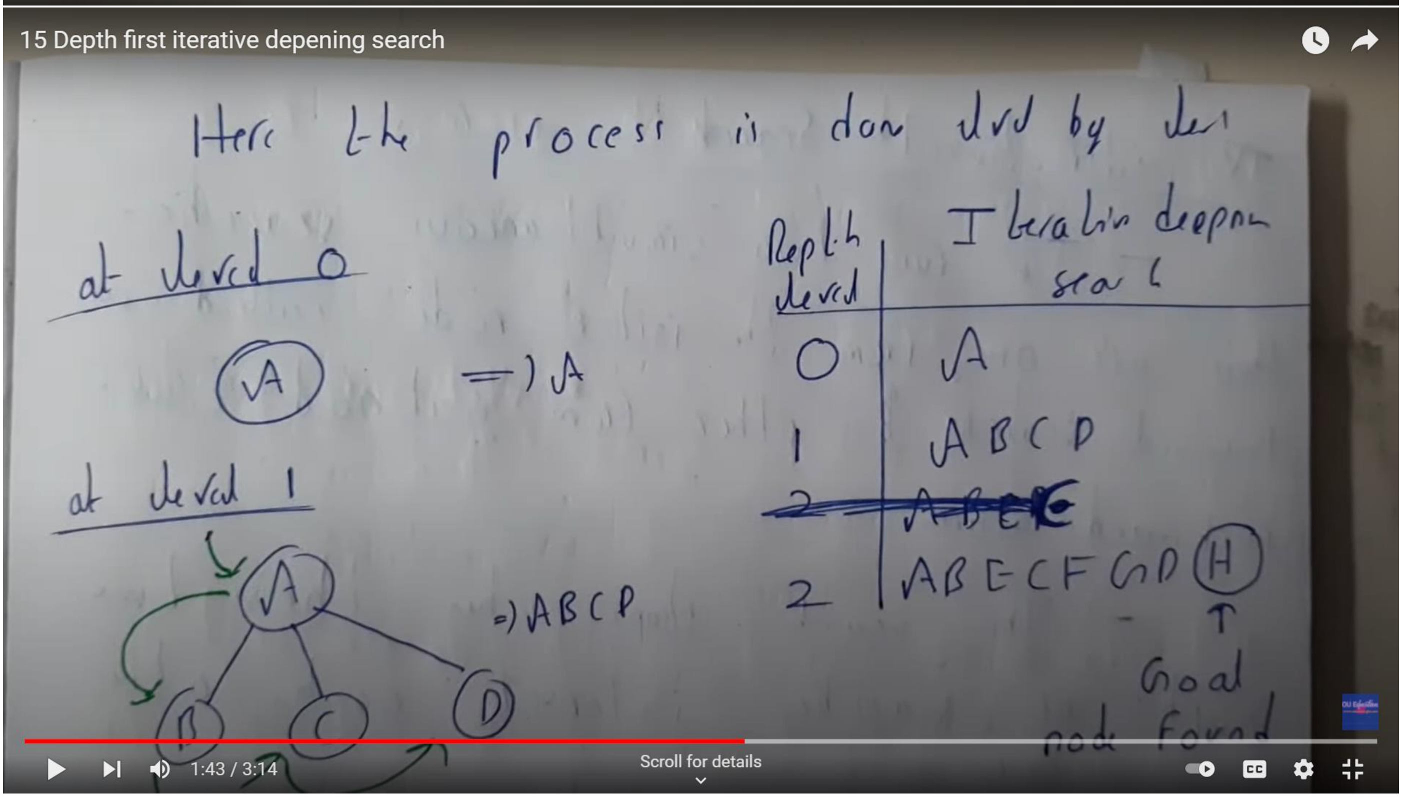
disadvantage

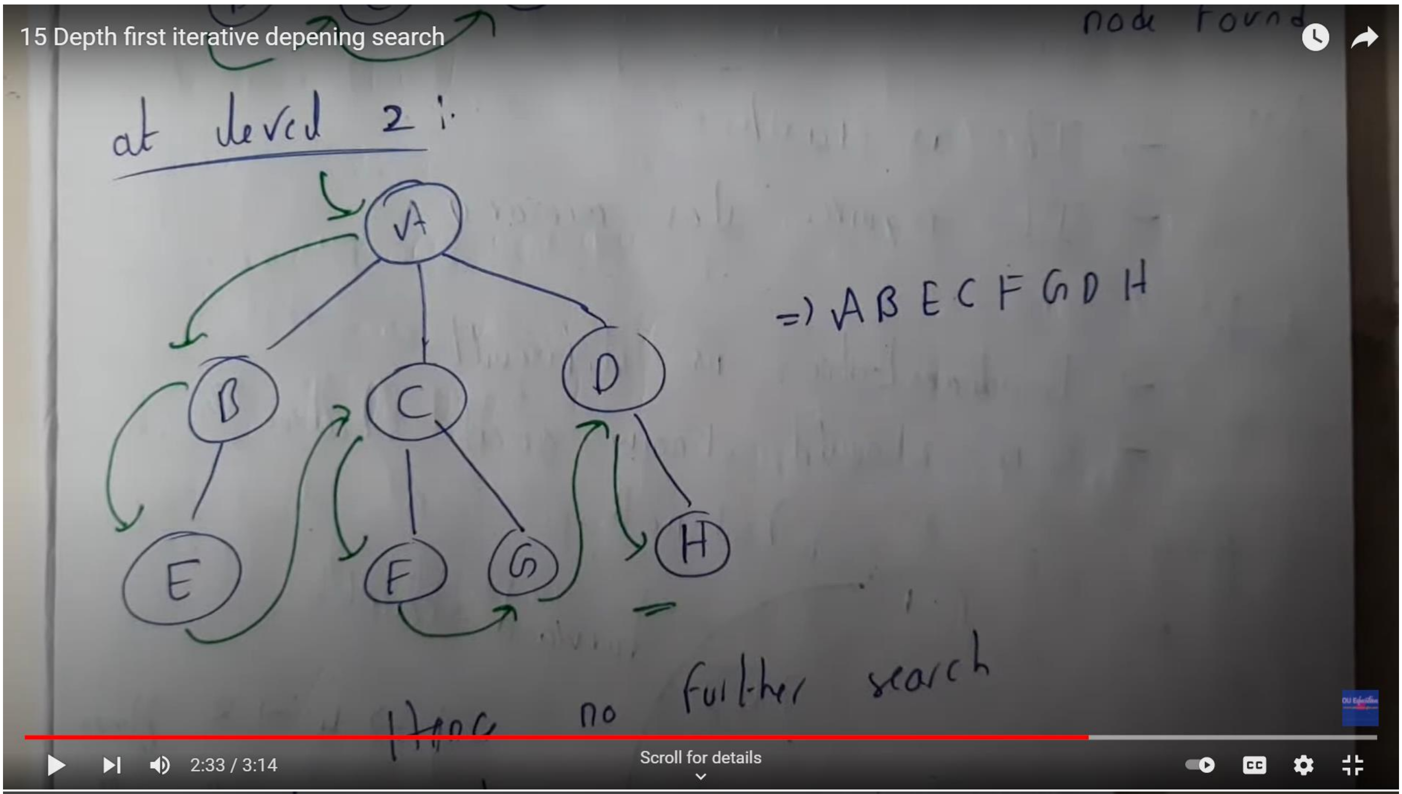
→ few nodes are visited again & ag

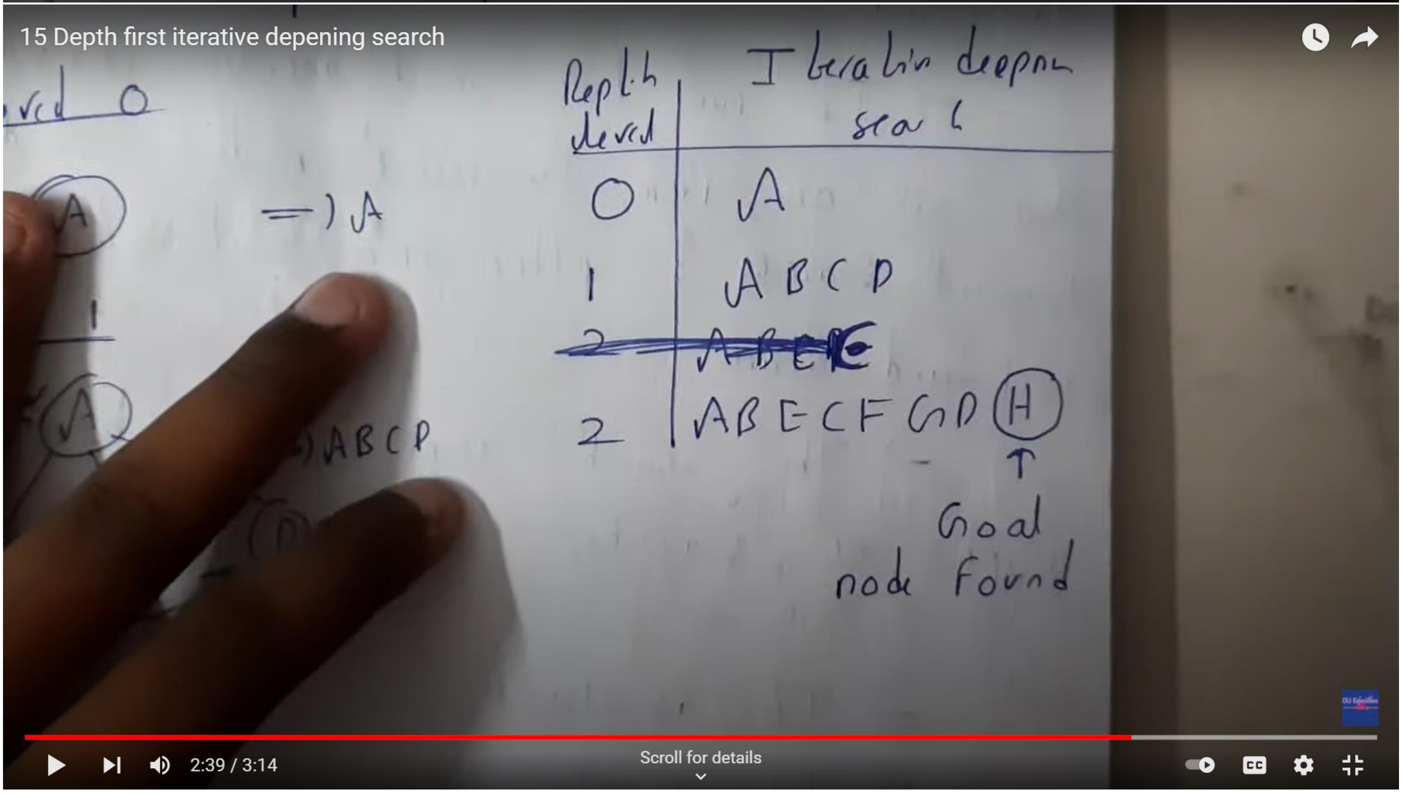
ex




15 Depth first iterative depening search
ex
let Goal node be H
A
B C D
E F G H
I J K
1:22 / 3:14
Scroll for details


15 Depth first iterative depening search
Here the process is done level by level
at level 0
⇒ A
at level 1
⇒ ABCD
Depth level
Iterative deepening search
0 A
1 ABCD
2 ABECFGDH
Goal
1:43 / 3:14
Scroll for details


15 Depth first iterative depening search
at level 2 :
A
B
C
D
E
F
G
H
=) A B E C F G D H
Hence no further search
2:33 / 3:14
Scroll for details


15 Depth first iterative depening search
Depth level
Iteration deepen search
0 A
1 A B C D
2 A B E C F G D H
Goal node found
=) A
)A B C D
2:39 / 3:14
Scroll for details

UNIFORM COST SEARCH ALGORITHM:

4) Uniform-Cost Search Algorithm:

-> It is used for traversing a weighted tree or graph.

-> It comes into play when a different cost is available for each edge.

-> The goal of UCS is to find a path to the goal node which has the lowest cumulative cost.

-> It expands nodes according to their path costs from the root

-> It can be used to solve any graph/tree ~~with th~~ where the optimal cost is in demand.

-> A UCS alg is implemented by the priority queue.

-> It gives maximum priority to the lowest cumulative cost

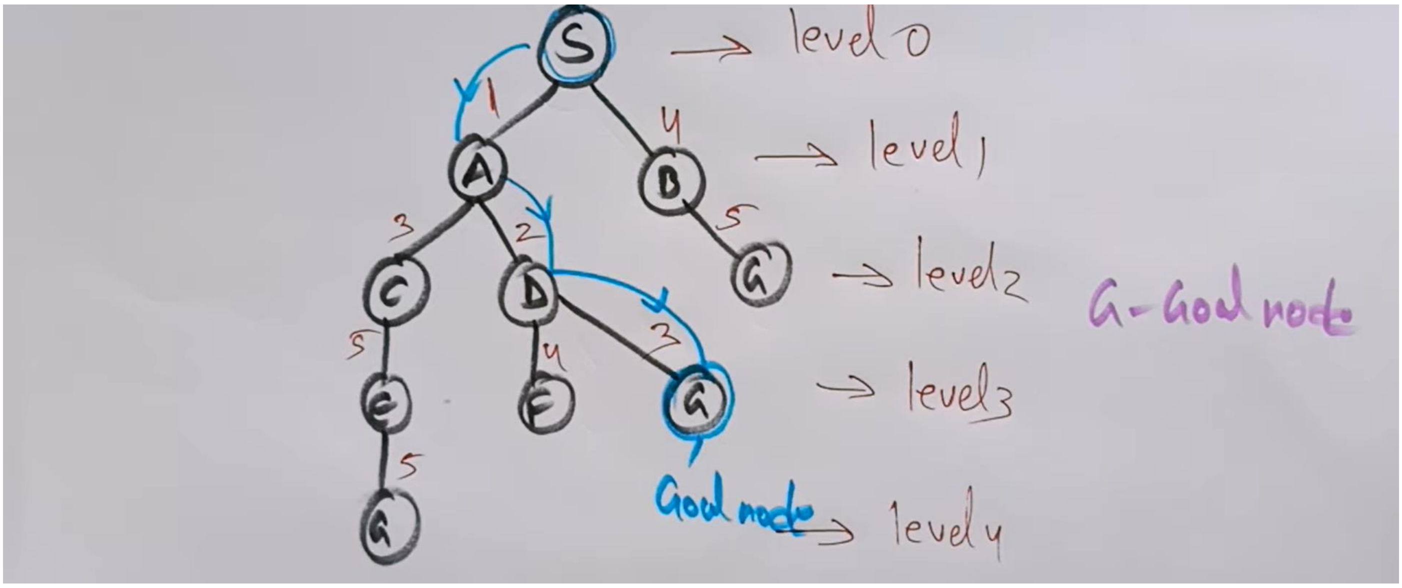
-> It is equivalent to BFS if the path cost of all edges is the same.

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 no: of steps involve in searching and only concerned about path cost. Due to which this alg ma stuck in an infinite loop.



BIDIRECTIONAL ALGORITHM:

5 Bidirectional Search Algorithm :-

(B)

→ It runs two simultaneous searches, one ~~from~~ from initial st
called as forward search & other ~~from~~ from goal node calle
as backward-search, to find goal node.

→ It replaces one single search graph with two small subgr
in which one starts the search from initial vertex &
other starts from goal vertex.

→ The search stops when these two graphs intersect each o

→ Bidirectional search can use search techniques such as
BFS, DFS, DLS, etc.,

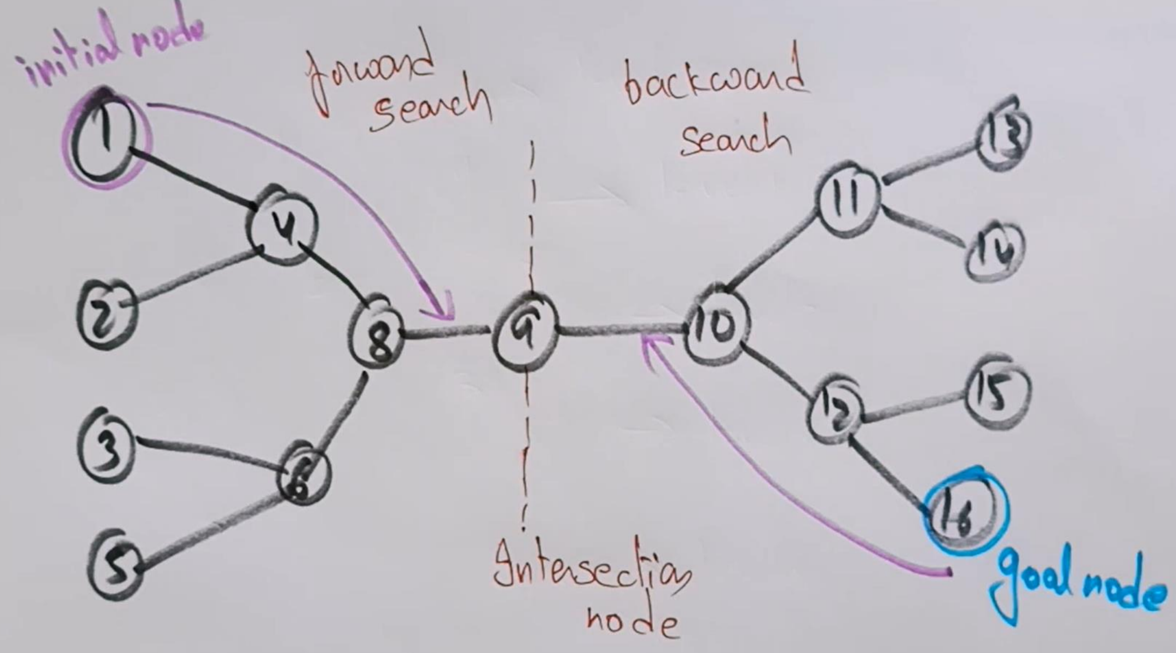
→ Bidirectional search is fast

→ Bidirectional search requires less memory.

Disadvantages:

→ Implementation of the bidirectional search tree is difficult.

→ In bidirectional search, one should know the goal state in advance.



INFORMED SEARCH STRATEGIES:

Informed Search Algorithm

① **Best-first Search Algorithm (Greedy Search)**

⇒ It always selects the path which appears best at that moment.

⇒ It is combination of DFS & BFS

⇒ It uses the heuristic function $h(n) <= h^*(n)$ and sea

$h(n)$ = heuristic cost

$h^*(n)$ = estimated cost

⇒ The greedy best first alg is implemented by priority que

Best first Search Algorithm:-

step1:- Place the starting node into the open list

step2:- If the open list is empty, stop & return failure

step3:- Remove the node n, from the open list which has lowest value of $h(n)$, & places it in the closed list.

step4:- Expand the node n, and generate the successors of node n.

step5:- check each successor of node n, and find whether node is a g[illegible] node or not. If any successor node is

Step 6 :- For each successor node, algorithm checks for evaluation function f(n), & then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to OPEN list.

Step 7 :- Return to step 2.

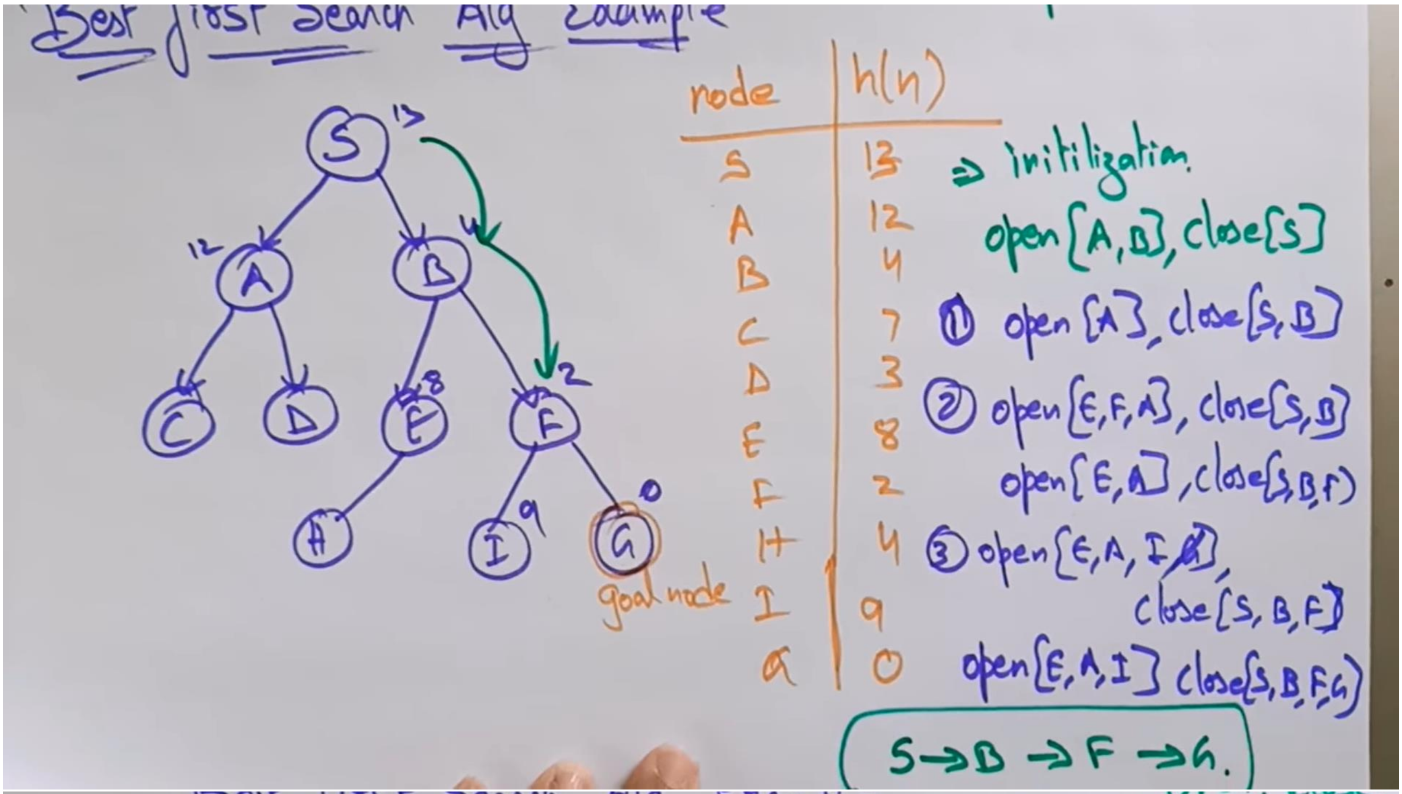
Advantages :

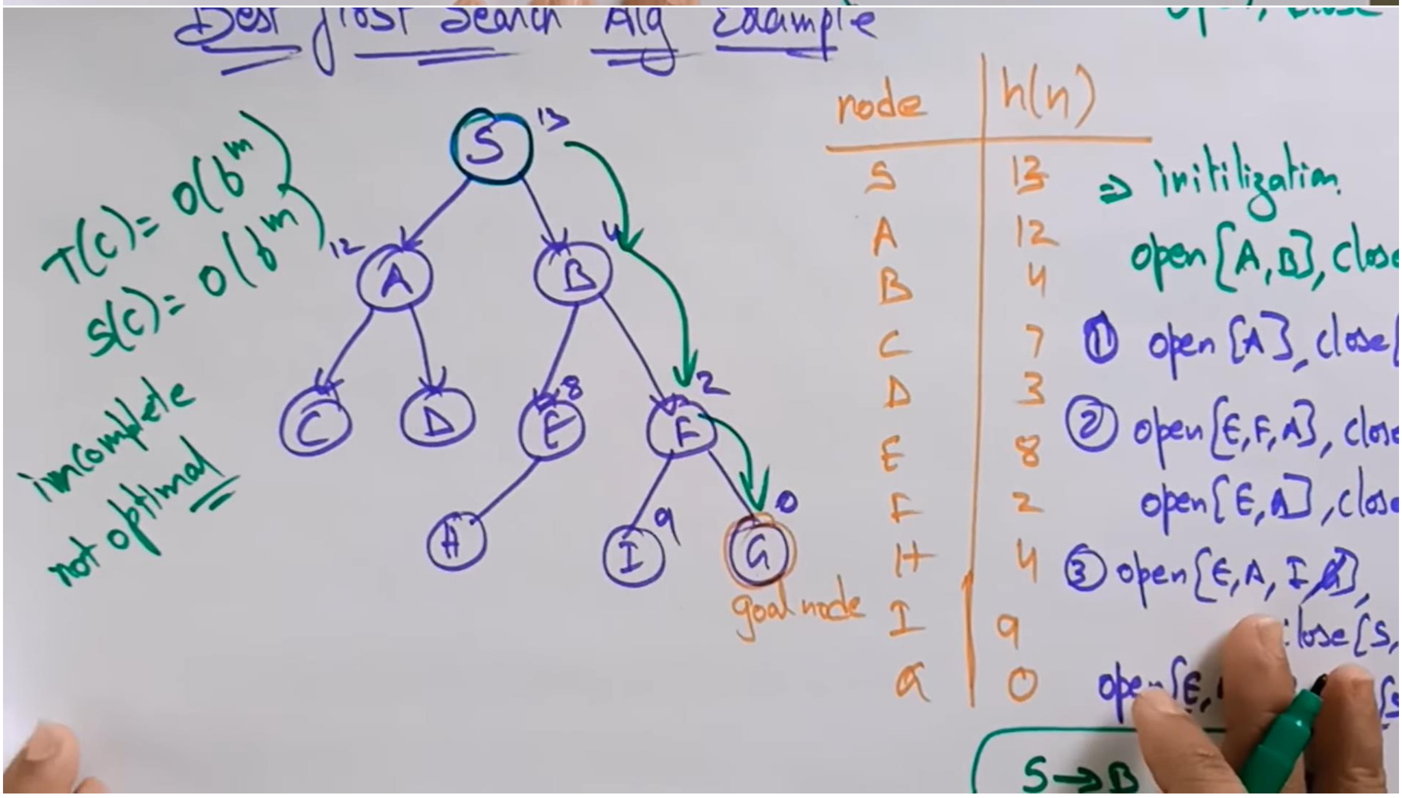
→ Best first search can switch between BFS & DFS by gaining the advantages of both the algorithm

This algorithm is more efficient than BFS & DFS algorithm

→ Disadvantages :

- It can behave as an unguided depth-first search in the worst case scenario
- It can get stuck in a loop as DFS
- This algorithm is not optimal.


Best first search Alg Example
node | h(n)
S 13
A 12
B 4
C 7
D 3
E 8
F 2
H 4
I 9
G 0
goal node
⇒ initilization
open[A, B], close[S]
① open[A], close[S, B]
② open[E, F, A], close[S, B]
open[E, A], close[S, B, F]
③ open[E, A, I, G], close[S, B, F]
open[E, A, I] close[S, B, F, G]
S → B → F → G.


Best first search Alg Example
T(c) = O(b^m)
S(c) = O(b^m)
incomplete
not optimal
node | h(n)
S 13
A 12
B 4
C 7
D 3
E 8
F 2
H 4
I 9
G 0
goal node
⇒ initilization
open[A, B], close
① open[A], close
② open[E, F, A], close
open[E, A], close
③ open[E, A, I, G],
S → B

(2) A* Search Algorithm:

$h(n)$

$h(n)$ $g(n)=$

→ A* Search Alg finds the shortest path through the search space using the heuristic function. $A(n) = g(n) + h(n)$

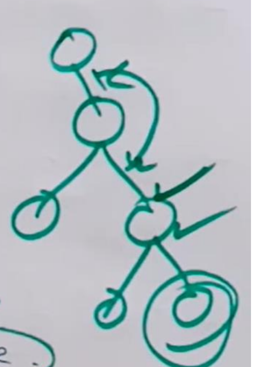
→ It uses $h(n)$, & Cost to reach the node n from the start st $g(n)$.

→ This alg expands less search tree and provides optimal resu faster.

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

A* use search heuristic as well as the cost to reach the node. Hence we combine both costs as,

$$\boxed{f(n) = g(n) + h(n)}$$ {fitness number}



$f(n)$ = estimated cost of the cheapest soln

$g(n)$ = Cost to reach node n from start state

$h(n)$ = Cost to reach from node n to goal node.

lgorithm of A* Search

Step1: Place the starting node in OPEN list

Step2:- check if the OPEN list is empty or not, if the list is

Empty then return failure & stops.

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

Step 4: Expand node n & generate all of its successors, & put n in 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 & CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to step 2

Advantages:

- It is best alg than other search alg
- It is optimal & complete
- It can solve very complex problems.

Disadvantages:

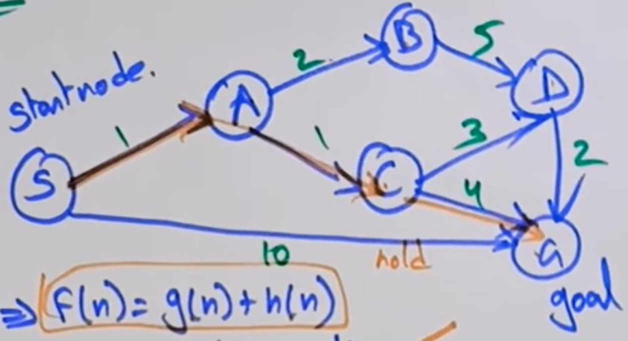
- It does not always produce shortest path

- It is not practical for various large-scale problems

A* Search Algorithm

Example 1

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



① $S \rightarrow A \Rightarrow \boxed{f(n) = g(n) + h(n)}$
$= 1 + 3 = 4$ ✓

$S \rightarrow G = f(n) = 10 + 0 = 10$ hold ✗

② $S \rightarrow A \rightarrow B \Rightarrow f(n) = 3 + 4 = 7$ hold ✗

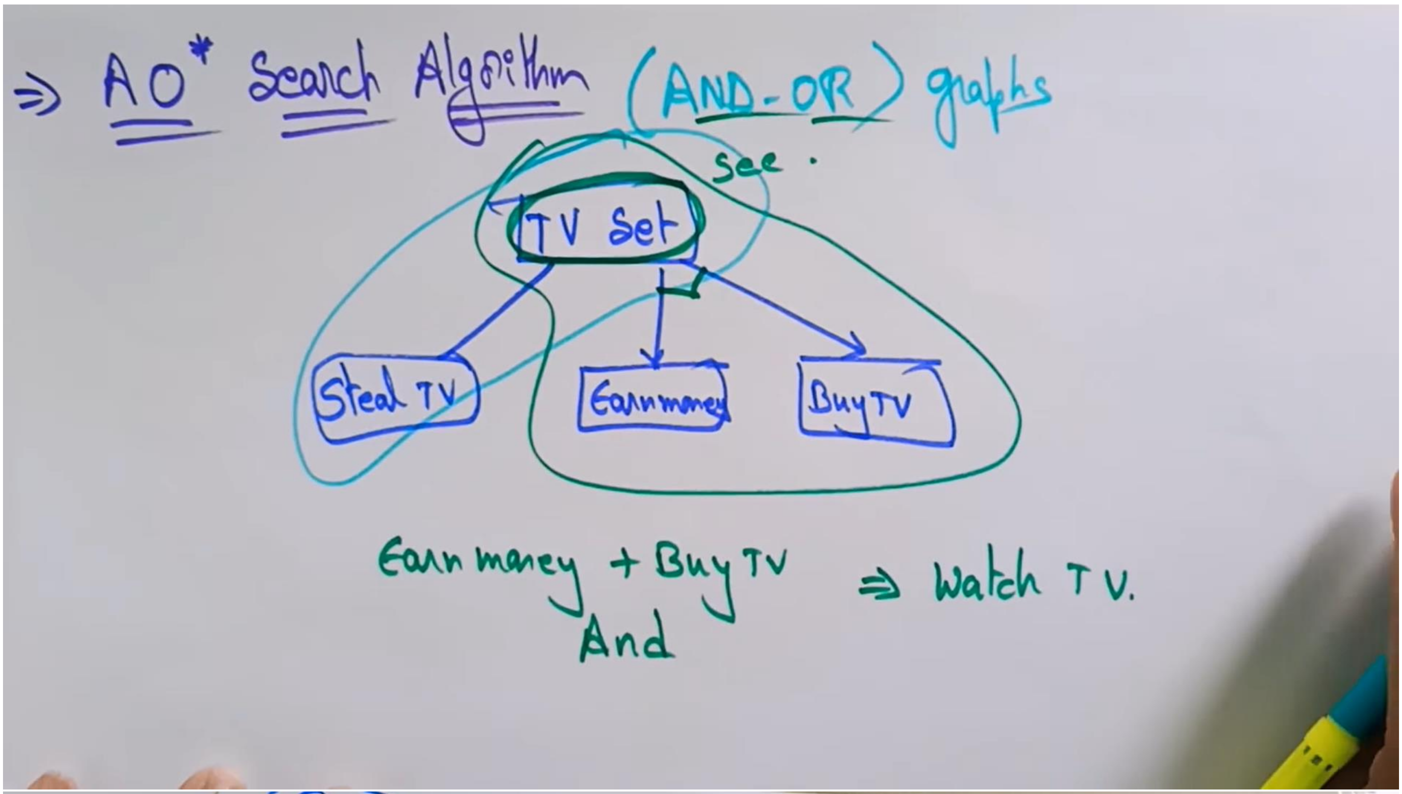
$S \rightarrow A \rightarrow C \Rightarrow f(n) = 2 + 2 = 4$ ✓

③ $S \rightarrow A \rightarrow C \rightarrow D \Rightarrow f(n) = 5 + 6 = 11$ hold ✗

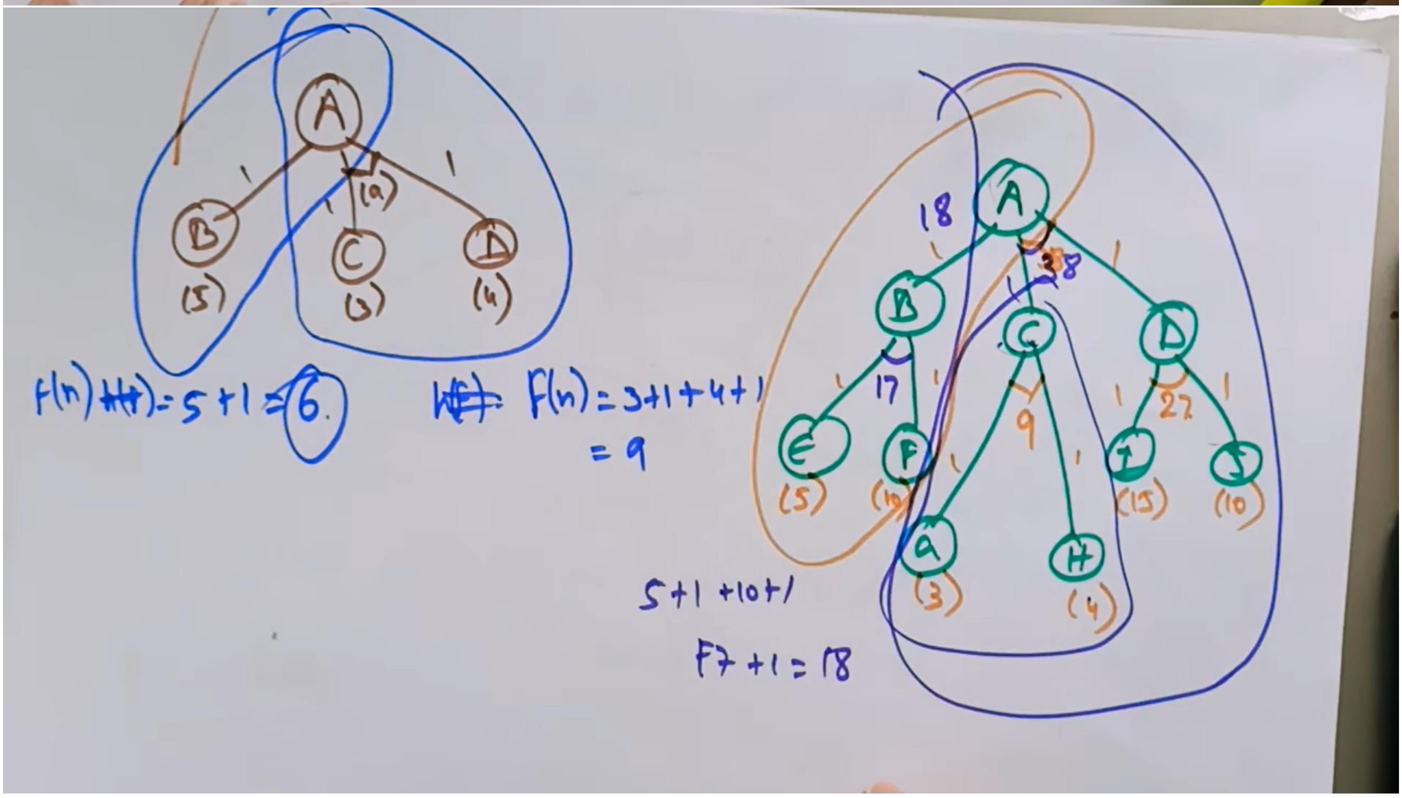
$S \rightarrow A \rightarrow C \rightarrow G \Rightarrow f(n) = 6 + 0 = 6$ ✓

$S \rightarrow A \rightarrow C \rightarrow$

Cost = 6.



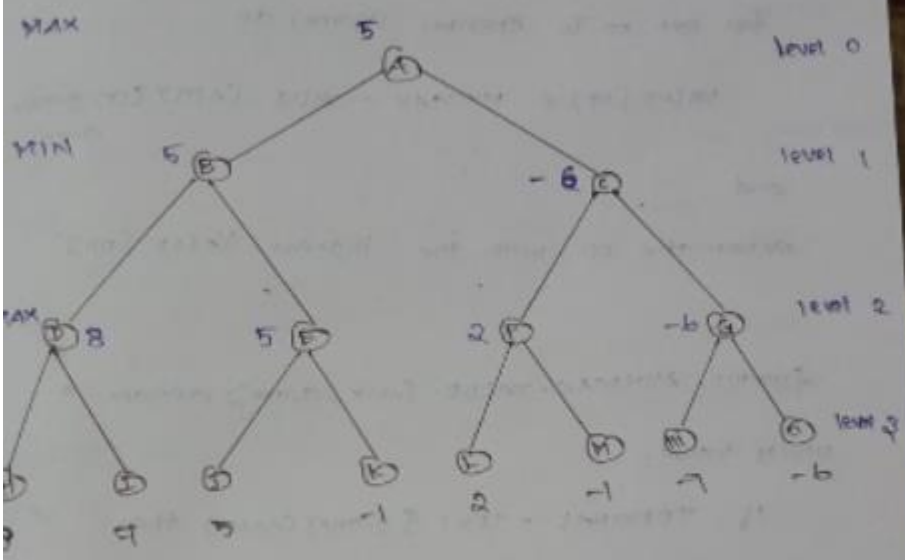
⇒ AO* Search Algorithm (AND-OR) graphs
see.
TV Set
Steal TV
Earn money
Buy TV
Earn money + Buy TV ⇒ Watch TV.
And



A
B
C
D
(5)
(3)
(4)
F(n) = 5+1 = 6.
F(n) = 3+1+4+1
= 9
18
38
17
9
27
E
F
G
H
I
J
(5)
(10)
(3)
(4)
(15)
(10)
5+1 +10+1
17+1 = 18

8) Perform Min-Max algorithm.

Graph



Hence the maximum node value of root node is 5

Min-Max algorithm concentrates on 2 point of views

player → try to maximize the winning possibility

Agent → try to minimize the winning possibility.

Min Max algorithm.

function MINMAX-DECISION (game) returns an operator

for each op in Operators [game] do

VALUE [op] ← MINMAX - VALUE [APPLY (op, game), game)

end

return the op with the highest VALUE [op]

function MINMAX-VALUE (state, game) returns a utility value.

If TERMINAL - TEST [game] (state) then

return UTILITY [game] (state)

else if MAX is to move in state then

return the highest MINMAX - VALUE of SUCCESSORS (state)

else

return the lowest MINMAX - VALUE of SUCCESSORS (state)

Properties

* The algorithm is complete, meaning in a finite search tree, a solution will be certainly found.

* It is optimal if both the players are playing optimally.

* Due to the Depth first search, for the game tree the time complexity of the algorithm is $O(b^m)$.
where b is the branching factor, m → max. depth of tree.

* Space complexity → $O(bm)$

As the name indicates, it starts from leaf node, depends upon parent min or max, it decides the value and stores in node → root node is always max.

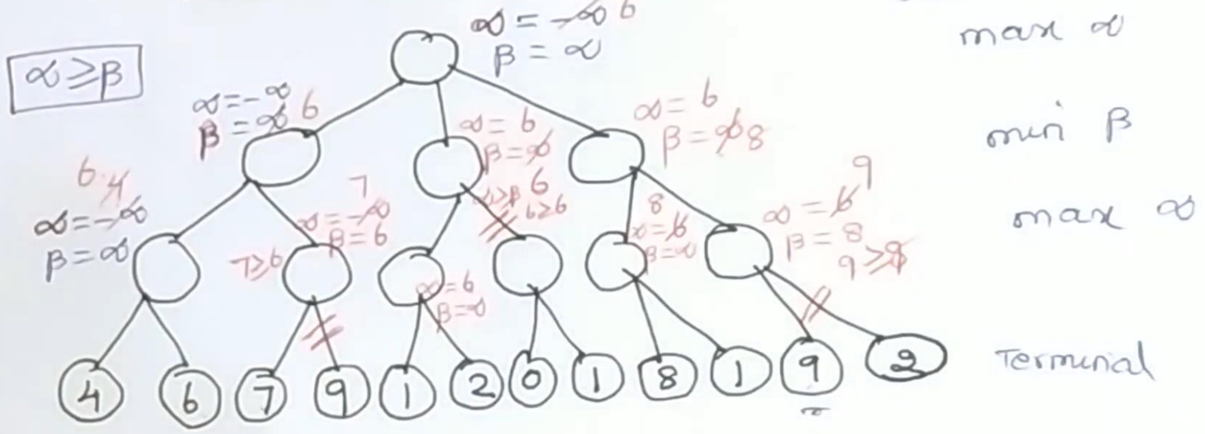
Advantages:

* A thorough assessment of the search is performed.
* Decision making in AI is easily possible.
* New and smart machines are developed with this algorithm

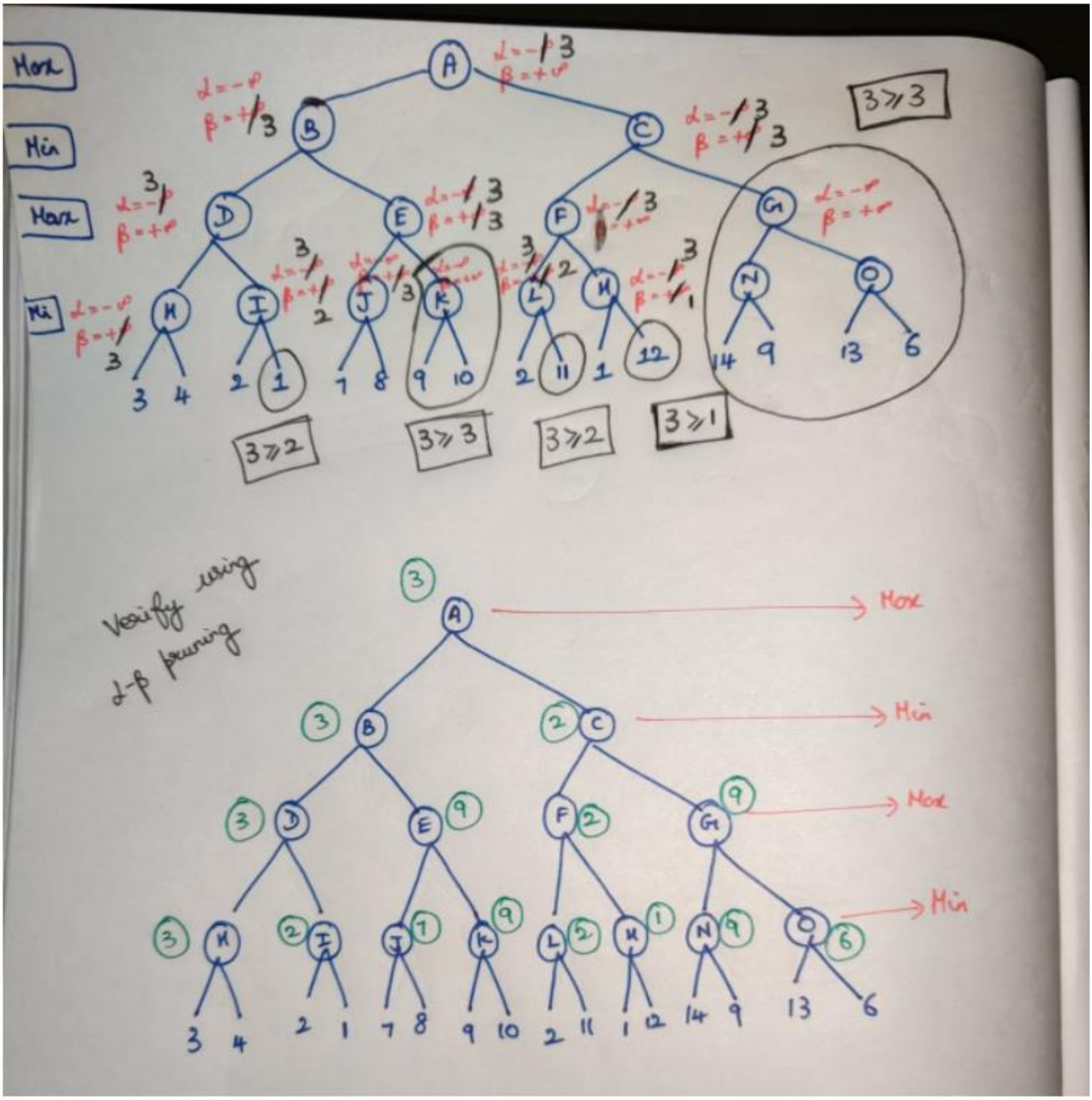
Disadvantages:

* Process of reaching the goal is slower because of the huge branching.
* Evaluation & search of all possible nodes and branch degrades the performance & efficiency of the engine.
* Both the players have too many choices to decide from.
* If any restriction of time & space complexity → difficult to explore the entire tree.

Alpha - Beta pruning - AI



⇒ DFS


Max
Min
Max
Min
3≥3
3≥2
3≥3
3≥2
3≥1
Verify using α-β pruning
Max
Min
Max
Min