Chapter - 12 Game playing
(of Book)

Chapter - 10 of GTU Syllabus

→ In 1960, Arthur Samuel had succeeded in building the first significant, operational game-playing Program. His Program played checkers and could learn from its mistake and improve its performance.

⇒ Reasons why game is important domain in machine Intelligence

Valid (1) They Provide a structured task in which it is very easy to measure Success or failure.

(2) They did not require large amounts of knowledge, They were thought to be solvable by search from the starting to winning Position

not Valid because in chess

(1) avg branching factor is around 35

(2) avg game, each player might make 50 moves

(3) To examine complete tree, we need to examine 35^{100} Positions

→ So to develop a program that simply does a straightforward search of the

game tree will not be able to select even its first move during the lifetime of its opponent.

→ heuristic search procedure is ~~necessary~~ necessary.

⇒ To improve effectiveness of a search-based problem solving

Prog:

(1) Improve the generate procedure so that only good moves are generated

(2) Improve the test procedure so that the best moves will be recognized and explored first.

⇒ Before we have used

Simple legal move generators

then plausible move generator In which only small number of promising moves are generated.

⇒ To find a solution to a problem is to generate moves through the problem space until a goal state is reached.

In some games like chess it is not possible to search until a goal state is found. The depth of the resulting tree and its branching factors are too great.

→ In amount of time available, it is usually possible to search a tree only ten or twenty moves (called ply in the game-playing literature) deep.

→ Then in order to choose the best move, the resulting board positions must be compared to discover which is most advantageous.

⇒ Static evaluation function :-

This could is done using this, which uses whatever information it has to evaluate individual board positions by estimating how likely they are to lead eventually to a win.

↔ This fn is similar to heuristic fn h' in the A* algo in which absence of complete information, choose the most promising position.

↔ Not that much of story so apply to many levels, draw

→ SEF for chess based on piece advⁿ
was proposed by Turing.
Simply add the values of black's
pieces (B), the value of white's
pieces (W) and then compute the
quotient W/B.

→ SEF
~~SFB~~ can be a linear combination
of several simple functions, each
of which appeared as though it
might be significant.

★ Credit assignment problem :-
deciding which moves have
contributed to wins and which to
losses is not always easy.
suppose we make a very bad move,
but then, because the opponent
makes a mistake, we ultimately
win the game. we would not like
to give credit for winning to any
mistake.

⇒ The problem of deciding which of
a series of actions is actually
responsible for a particular
outcome is called the credit
assignment problem.

⇒ Two know" based components of a good game - playing program
(1) a good plausible - move generator
(2) a good static evaluation function

⇒ In some game playing methods the moves totally depends on opponents moves as values are passed back up, different assumptions must be made at levels where the program chooses the move and at the alternating levels where the opponent chooses,

⇒ one of the method used to do this is minimax search procedure

Minimax Procedure

⇒ Starting with current position apply ~~Possi~~ 'plausible - move generators to generate the set of possible Successors positions.

→ Then apply the static evaluation function to those positions and simply choose the best one.

→ SEF returns large values to indicate good situations for us. so our goal is to maximize the value of the static evaluation fun^n of the next branch position

→ Our goal is to maximize value

Example

↳ Static evaluation function that returns value from -10 to 10

→ +10 indicating a win for us.

→ -10 indicating a win for opponent

→ 0 indicates a draw

→ Our goal = Maximize the value of heuristic function

→ According to following figure - 1 to maximize we will select B(8) In a one - ply search

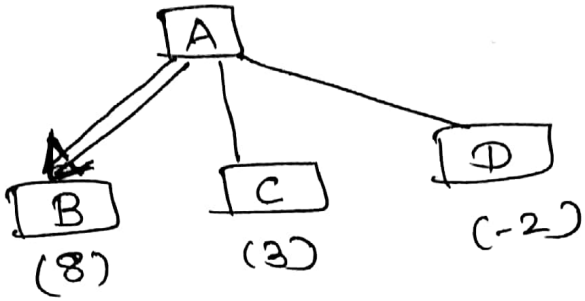


figure - 1 one - ply search

→ Now the opponent's goal is to minimize the value so, he/she will select F(-6)

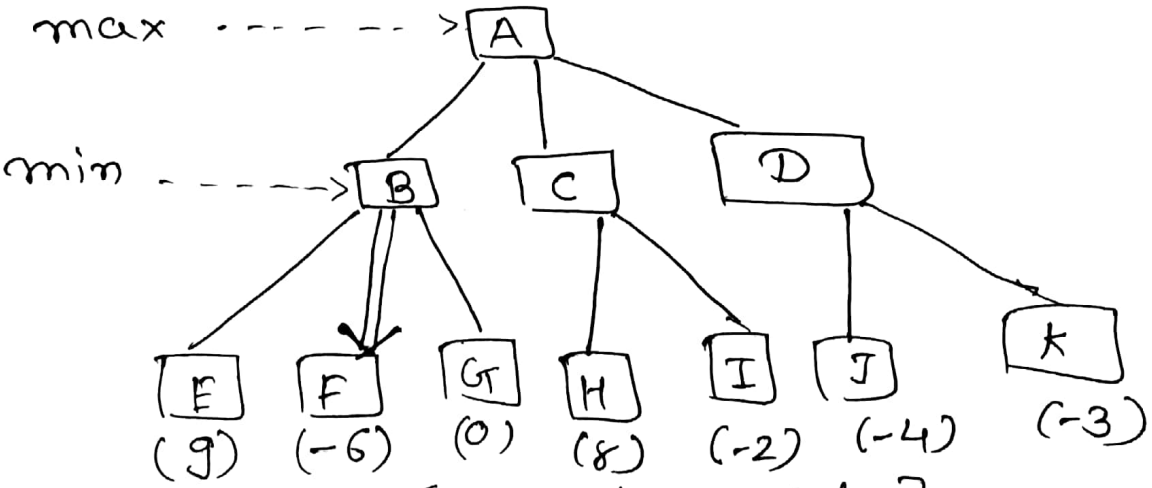


figure - 2 - [Two ply search]

→ This means that if we make move B then it is good for opponent and Bad for us as shown in figure 3

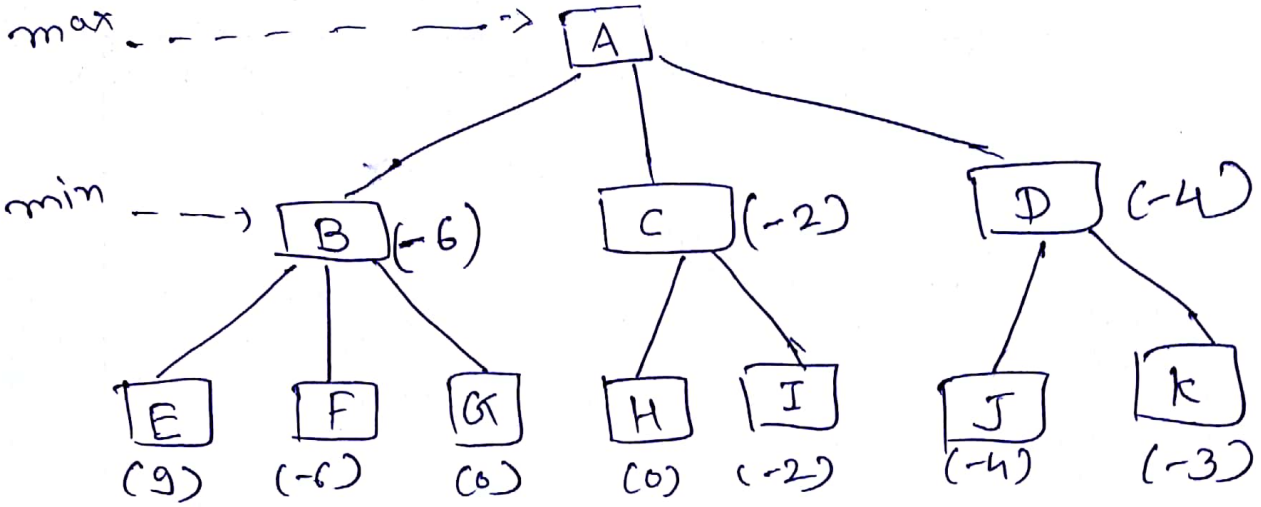


fig - 3 [Backing up the value of Two ply search]

→ Because goal of opponent is to minimize the value of the evaluation function.

→ as in figure-3 At this level representing the opponents choice, minimum value was chosen and backed up

→ At the level representing our choice, maximum value was chosen.

→ The alternation of maximizing and minimizing at alternate ply when evaluation are being pushed back up corresponds to the opposing strategies of the two players and gives this method the name minimax.

→ So, good choice is to select C(-2) for us

→ minimax procedure is straightforward Recursive procedure with two procedure.

(1) MOVEGEN (Position, player)
plausible-move generator, which returns a list of nodes representing the moves that can be made by player in position.

(2) STATIC (Position, player)
Static evaluation function, which return a number representing the goodness of position from the standpoint of player.

⇒ As with any recursive program a critical issue in the design of the minimax procedure is when to stop. the recursion and simply call the Static evaluation function.

→ There are a variety of factors that may influence this decision,
(1) Has one side won?
(2) How many ply have we already explored?
(3) How promising is this path?
(4) How much time is left?
(5) How stable is the configuration?

Algorithm : MINIMAX (Position , Depth, player)

(1) If DEEP-ENOUGH (Position , Depth) then Return the Structure

VALUE = STATIC (Position , player);

PATH = nil

This indicates that there is no path from this node and that its value is that determined by the static evaluation fun^{n}

(2) Otherwise, generate one more ply of the tree by calling the function MOVE-GTEN (Position, player) and setting SUCCESSORS to the list it returns.

(3) If SUCCESSORS is empty, then there are no moves to be made, so return the same structure that would have been returned if DEEP-ENOUGH had returned true.

(4) If SUCCESSORS is empty, then examine each element in turn and keep track of the best one. This is done as follows

Initialize BEST-SCORE to the minimum value that STATIC can return. It will be updated to reflect the best score that can be achieved by an

element of SUCCESSORS.

for each element SUCC of SUCCESSORS, do the following:

(a) Set RESULT-SUCC to

MINIMAX (SUCC, Depth +1, OPPOSITE(player))

This recursive call to MINIMAX will actually carry out the exploration of SUCC.

(b) Set NEW-VALUE to - VALUE (RESULT-SUCC)

This will cause it to reflect the merits of the position from the opposite perspective from that of the next lower level.

(c) If NEW-VALUE > BEST-SCORE, then we have found a successor that is better than any that have been examined so far. Record this by doing the following:

(i) Set BEST-SCORE to NEW-VALUE

(ii) The best known path is now from CURRENT to SUCC and then on to the appropriate path down from SUCC as determined by the recursive call to MINIMAX. So set BEST-PATH to the result of attaching SUCC to the front of PATH (RESULT-SUCC)

(5) Now that all the successors have been examined, we know the value of position as well as which path to take from it. So return the structure.

VALUE = BEST - SCORE

PATH = BEST - PATH

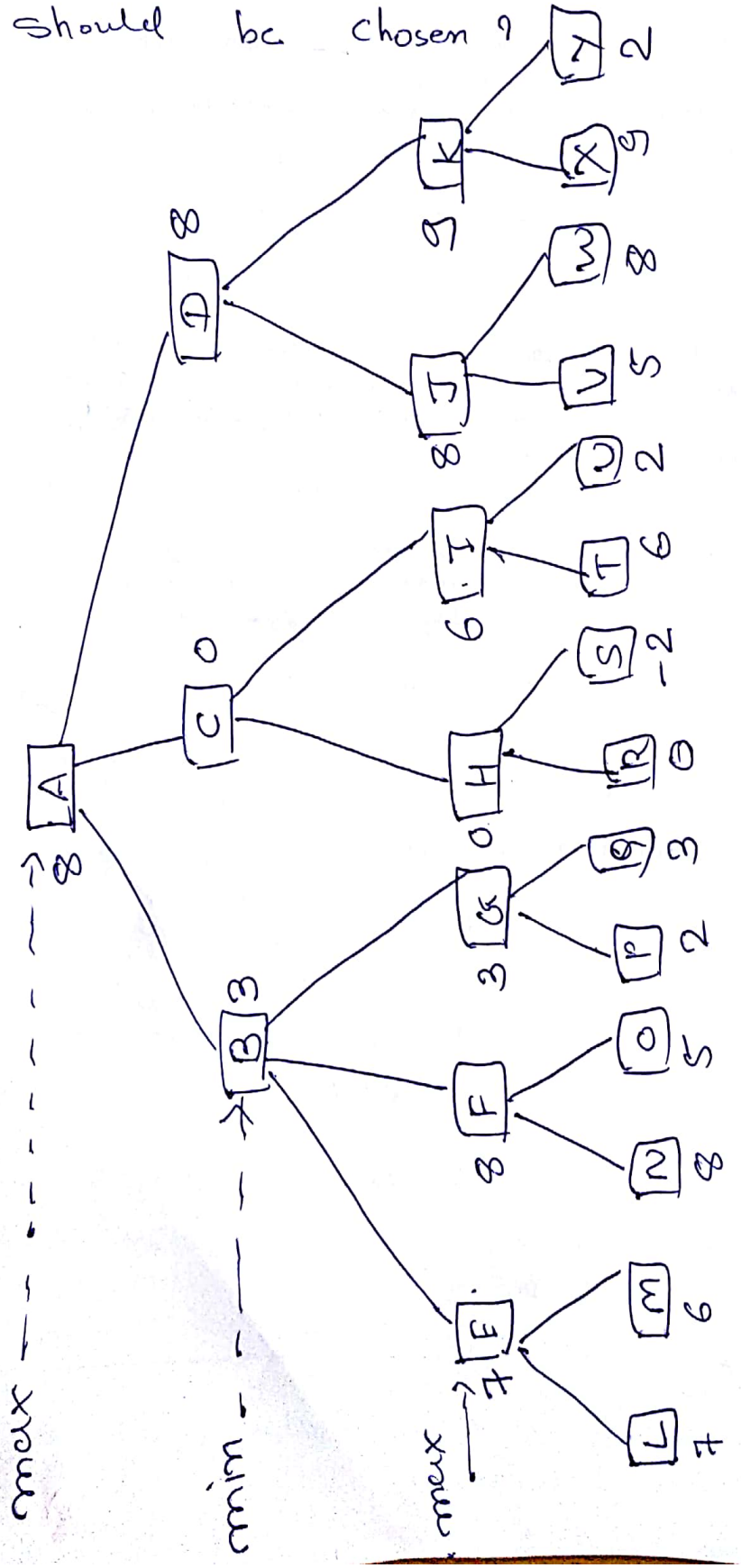
⇒ For the general MINIMAX procedure we append to a function, DEEP-ENOUGH, which is assumed to evaluate all of these factors and to return TRUE if the search should be stopped at the current level and FALSE otherwise.

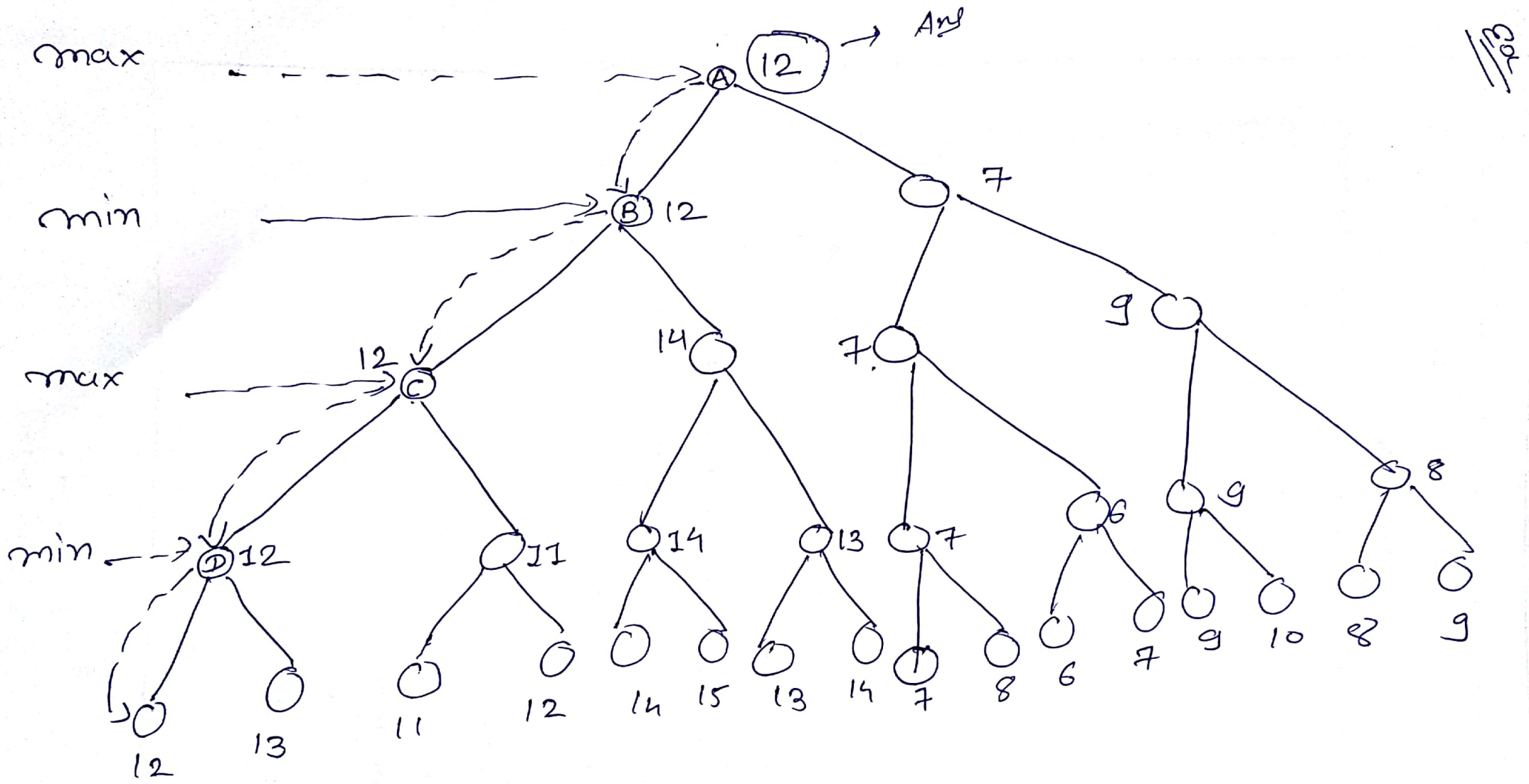
→ One problem that arises in defining MINIMAX as a recursive procedure is that it needs to return not one but two results.

→ (1) backed-up value of the path it chooses

(2) The path itself.

Eg Consider the following game tree in which static scores are all from the first player's point of view. Suppose the first player is the maximizing player, what move should be chosen ?





Alpha - Beta cutoffs

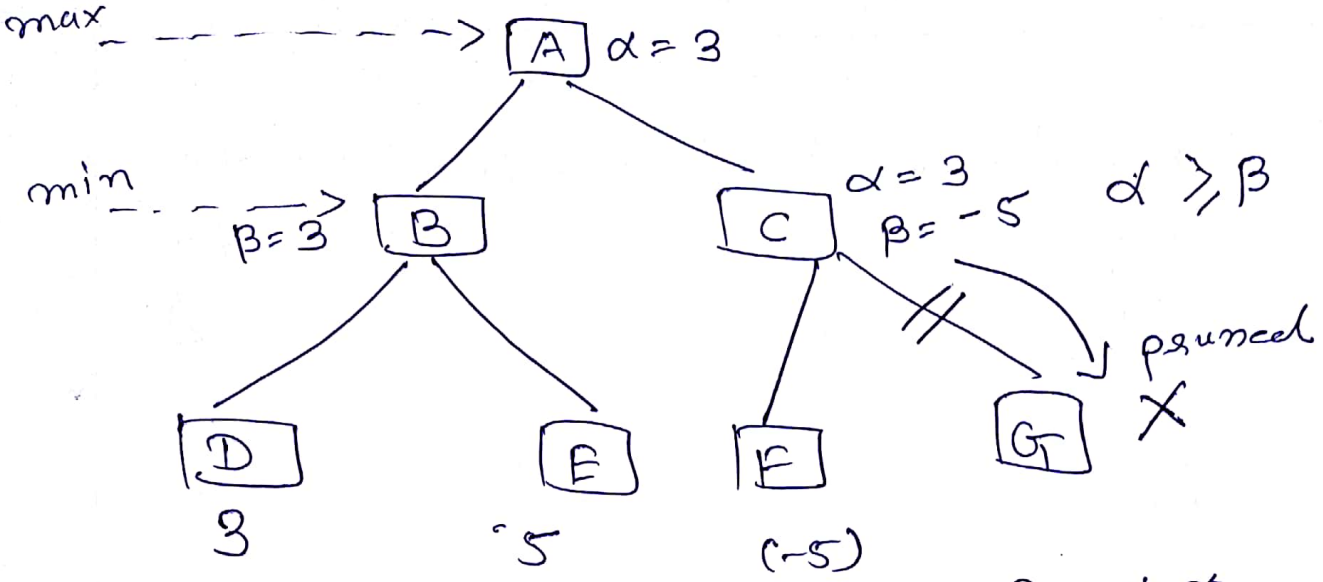
→ minimax procedure is a depth-first process., efficiency of it can often be improved by using branch-and-bound techniques in which partial solutions that are clearly worse than known solution can be abandoned early.

→ The branch-bound strategy for modifying it two players, including two bounds one for each of the player is called as alpha-beta pruning

(1) Alpha → value of the best possible move you can make, that you have computed so far. (max player)

(2) Beta → value of best possible move your opponent can make, that you have computed so far. (min player)

⇒ requires the maintenance of two threshold values.



Initial value of $\alpha = -\infty$, $\beta = +\infty$

here for α = max player change α

for β = min player change β

conditions to check for node where α and β both can be get

(1) $\alpha \geq \beta$ then no need to traverse the new node.

(2) $\alpha < \beta$ then required to create new node.

At B :

⇒ Starting with left side of the tree at B, we need to compare $\beta(+\infty)$ and D(3) So we get,

$\min(+\infty, 3) = 3$ So, at B we get $\alpha = -\infty$, $\beta = 3$

Now, compare $\beta(3)$ with E(5)

$\min(3, 5) = 3$ So,

at B we get $\alpha = -\infty$, $\beta = 3$

At A : max level

$\alpha = -\infty$, $\beta = +\infty$

→ at max level change $\alpha(-\infty)$ ~~compute~~ compare $\alpha(-\infty)$ with the value of $\beta(3)$

$\max(-\infty, 3) = 3$

so, $\alpha = 3$, $\beta = +\infty$

At C : min level , change β

Now, we ~~get~~ can propogate the value of α, β at A to α, β at C

so, $\alpha = 3$, $\beta = +\infty$

$\beta(+\infty)$, $F(-5)$

So, $\min(+\infty, -5) = -5$

so , $\beta = -5$

Now, $\alpha = 3$, $\beta = -5$

So, $\alpha \geq \beta$ satisfied

$3 \geq -5$

so, we can prune the G path from C

Now, At A : max level

Compare $\alpha(3)$ of A with $\beta(-5)$ of C

$$\max(3, -5) = 3$$

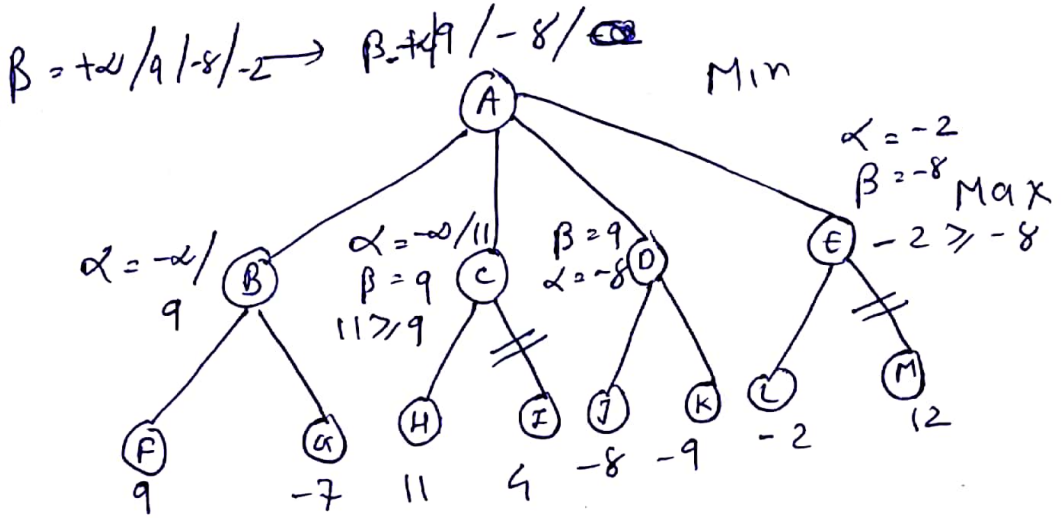
So, $\alpha = 3$

$\beta = +\infty$

$\Rightarrow$ So, Answer is with α value 3

A → B → D

Q-B 5)



Ans: ADJ.

C → I
E → M
} B-cut.

1) at B : Max
α = -∞
β = +∞
Max (+∞, 9) = 9.
Max (+9, -7) = 9.
α = 9
β = +∞

2) at A : Min
α = -∞
β = +∞
Min (+∞, 9) = 9.
α = -∞
β = 9.

3) at C : Max
α = -∞
β = 9
Max(-∞, 11) = 11
α = 11
β = 9
→ 11 ≥ 9.

So, no need to expand C → I.

4) at A : Min
α = -∞
β = 9 (Min (9, 11))

5) at D : Max
α = -∞
β = +9
Max (-∞, -8) = -8
α = -8
β = +9
α < β → -8 < 9
So, expand D → K.
Max (-8, -9) = -8

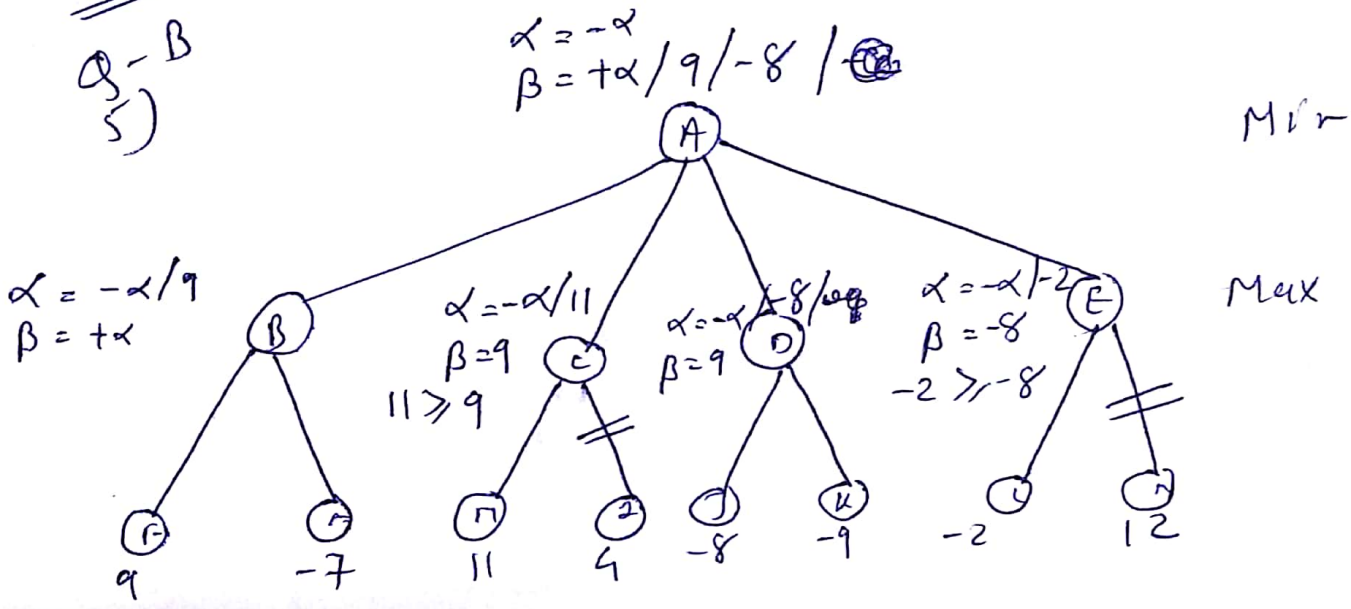
6) at A : Min
α = -∞
β = -8 (Min (9, -8))

7) at E : Max
α = -∞
β = -8
Max (-∞, -2) = -2
α = -2
β = -8
-2 ≥ 8
So, no need to expand E → M

8) at E : Min
α = -∞
β = -8 ((Min (-8, -2))
→ A.

Or.

Q-B
5)



→ follow the steps of previous example.

Ans: ADJ.

Now, At A : max level

Compare $\alpha(3)$ of A with $\beta(-5)$ of C

$\max(3, -5) = 3$

So, $\alpha = 3$

$\beta = +\infty$

⇒ So, Answer is with α value 3

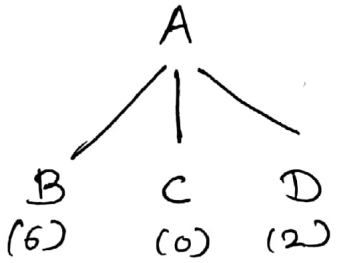
A → B → D

★ Additional Refinements

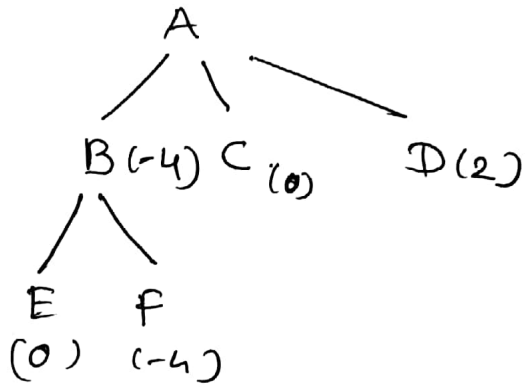
(1) Waiting for Quiescence

→ when we select any path according to ply if we look one move ahead our estimation can be changed drastically.

ex



We select B



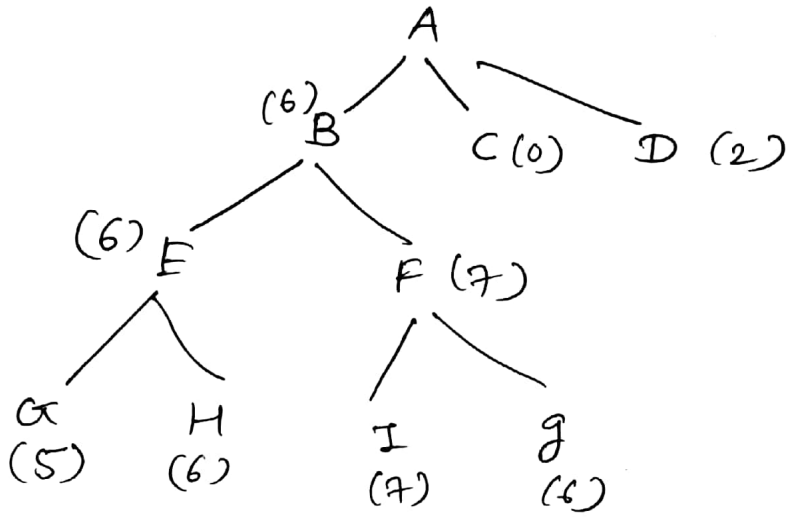
Now, after one more ahead we select C, by deciding B is not good move

→ but To make sure that such Short-term measures do not unduly influence our

choice of move. We should continue the search until no such drastic change occurs from one level to the next.

→ This is called waiting for Quiescence.

→ If we do one more move, we might get



→ Now move B again looks like a reasonable move for us to make since the other half of the piece exchange has occurred.

⟹ Waiting for Quiescence helps in avoiding the horizon effect

In which an inevitable bad event can be delayed by various tactics until it does not appear in the portion of game tree.

→ It is a phenomenon that arises when a program is facing a move by opponent that cause wide damage and is unavoidable.

→ This effect may make a move look good despite the fact that the move might be better if delayed past the horizon.

(2) Secondary Search :-

One good way of combating the horizon effect is the double-check.

→ a chosen move to make sure that a hidden pitfall does not exist a few move further away.

(3) Using Book Moves :-

For complicated games taken as whole possible solution but it is not feasible.

→ But for some segments of some games this approach is reasonable.

Iterative Deepening :

→ used in program CHESS 4.5

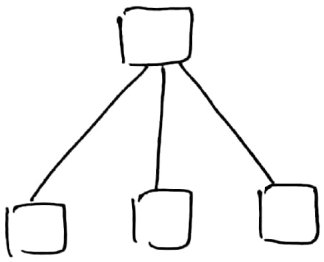
→ Rather than searching to a fixed depth in the game tree.

→ CHESS 4.5 first searched only single ply.

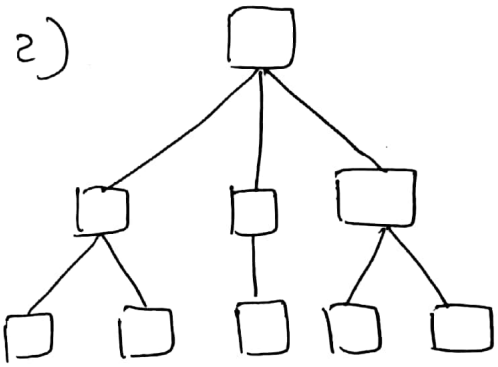
→ It then initiated a new minimax search, this time to a depth of two ply.

→ This way followed by a three ply and then a four-ply search etc.

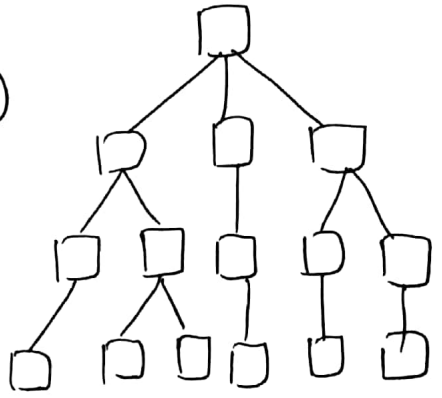
1)



2)



3)



→ on the face of it, this process seems ~~was~~ wasteful.

→ Why should we be interested in any iteration except the final one ?

→ There are several reason,

(*) Depth-first Iterative Deepening (DFID):-

→ Combined the best aspects of DFS and BFS.

Algorithm DFID:

(1) Set SEARCH - DEPTH = 1

(2) Conduct a DFS to a depth of Search-depth.
- If a solution path is found, then return it.

(3) Otherwise increment SEARCH - DEPTH by 1 and go to Step-2.

→ Clearly, DFID will find the shortest solution path to the goal.

→ Maximum amount of memory used by DFID is proportional to the no. of nodes in that solution path.

→ Only disturbing fact is that
- all interation but the final one are essentially wasted.
- However this is not a serious problem.
- DFID is only slower than DFS.

→ The problem with DFS is that there is no way to know in advance how deep the solution lies in the search space.

→ DFID avoids the problem of chosing cutoff without sacrificing efficiency.

→ DFID is the optimal algorithm (in terms of space & time) for uninformed search.

→ Iterative Deepening also used to improve the performance of the A*.

→ A* require large amount of memory.

Algorithm ID - A*

(1) Set THRESHOLD - the heuristic evaluation of the start state.

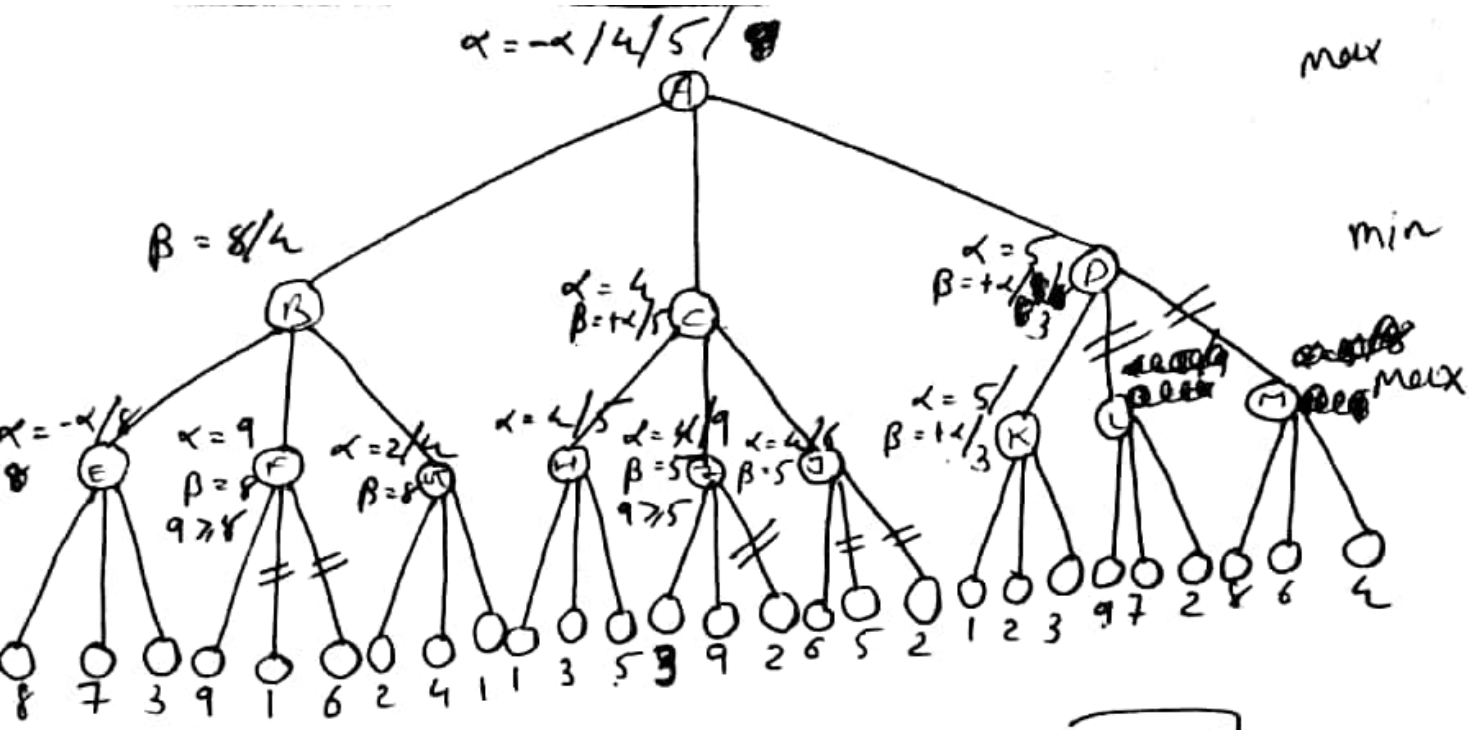
(2) Conduct a DFS, pruning any branch when its total cost function (g+h') exceeds THRESHOLD
- If solution bound → return it.

(3) Otherwise increment THRESHOLD by the minimum amount it was exceeded during the previous step, and go to step 2.

→ IDA* guarenteed to find optimal solution.

→ IDA* very efficient w.r.t space.

→ IDA* was first heuristic search algorithm to find optimal solution by the 15-puzzle with reasonable time & space.



Ans: ~~Redacted~~

at A $\alpha = 5$

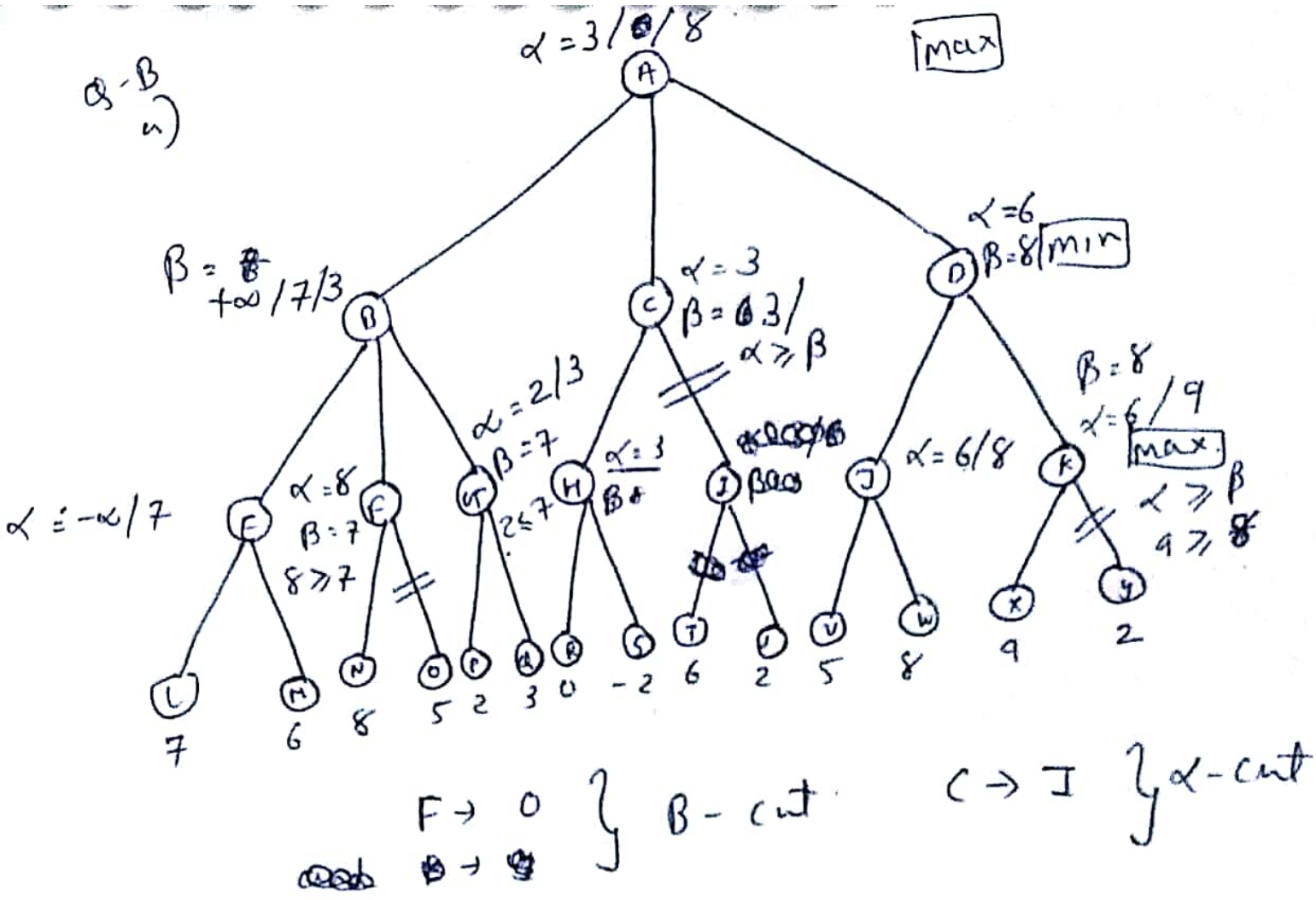
Ans: A C H

Note:

for: Min and Max both level,
only check Condition → $\alpha \geq \beta$.
if this Condition Satisfied, then no need to expand other nodes.
if $\alpha < \beta$, then you have to expand other nodes.

at Max level, if path is cut → it's called β-cut
at Min level, if path is cut → it's called α-cut

Q. B
ii)



F → O } β - cut

C → I } α - cut

Ans : A D J W