UNIT-2

part- A (problem Solving : State space Search and Control Strategies)

1. Introduction
2. General problem Solving.
   a. production Systems
   b. State- space Search
   c. Control Strategies
3. Characstics of problem
4. Exhaustive Searches
   a. Breadth - first Search
   b. Depth - First Search
   c. Depth - first iterative Deepening
   d. Bi directional Search
   e. analysis of Search methods.
5. Heuristic Search (methods) Techniques.
   a. General - purpose Heuristics
   b. Branch and Bound Search
   c. Hill climbing
   d. Beam Search
   e. Best- first Search
   f. A* Algorithm
   g. optimal Solution by A* Algorithm
   f. monotonic function
6. Iterative Deepening (A*)
7. Constraint Satisfaction

part-B (probelem Reduction and Game playing)

1. Introduction
2. probelem Reduction
3. Game playing
4. a. Game probelem Vs State Space probelem
   b. Statics Labelling procedure in Game tree
   c. Nim Game probelem
4. Alpha - Beta proning.
5. a. Refinements to Alpha- Beta proning.
   b. Alternative to Alpha- Beta proning minimax procedure.
   c. iterative Deepening.
5. Two - player perfect Infoomation Games.

1. INTRODUCTION

probelem solving is a method of deriving solutions step by step which begins from initial description of the probelem to the Desireed solution.

The task is solved by a series of actions that minimises the differences between given situation and desired goal.

In A.I, The probelem are frequently modelled as "State-Space -probelem" which is set of all possible states.

The Two types of probelem solving methods are,

- General purpose methods
- Special purpose methods.

The General purpose methods are used to solve wide variety of probelems.

The Special purpose methods are used to solve particular specific probelem.

The order of Application of rules to the current state is called "Control Strategy"

2. GENERAL PROBLEM SOLVING

General problem Solving uses,

a. production System.

b. State space Search

c. Control strategyies

a. production System:- (P.S)

production System is a mechanism which helps the A.I programs to Search / Run more conviently in State-Space probelem.

The production System consists of Initial state & final state of the probelem along with database of That particular task.

A production System consists of production Rules, which has Left Rule and & Right Rule.

The Left Side of Rule. Consists of current State. where as Right Side of The Rule describes new State. That is obtained after applying The rule.

EX:- Water Jug probelem.

probelem State ment:- we have 2 Jugs, one is a 5 Litre and other a 3 Litre with no measuring marks.

There will be. Continuous Sipply of water.

Goal:- To have. only 4 Litre. of water in a 5Lt Jug

Solution:- The State Space search is described as ordered pairs of Integers (x,y), such that x represents 5 Lt Jug and y Represents 3 Lt Jug.

The possible operations that can be used in this problem are;

1. Fill & Empty 5 Lt Jug.
2. Fill & Empty 3 Lt Jug.
3. pour water from 5 Lt Jug to 3 Lt Jug.
4. pour water from 3 Lt Jug to 5 Lt Jug.

Based on the above operations, the production Rules for water Jug problem as mentioned below;

Rule No.	Left Side of Rule	Right Side of Rule:	Description:
1.	$(x,y \mid x<5)$	$(5,y)$	Fill 5L Jug.
2.	$(x,y \mid x>0)$	$(0,y)$	Empty 5L Jug.
3.	$(x,y \mid y<3)$	$(x,3)$	Fill 3L Jug.
4.	$(x,y \mid y>0)$	$(x,0)$	Empty 3L Jug.
5.	$(x,y \mid x+y<=5 \wedge y>0)$	$(x+y,0)$	Empty 3L into 5L Jug.
6.	$(x,y \mid x+y<=3 \wedge x>0)$	$(0, x+y)$	Empty 5L into 3L Jug.
7.	$(x,y \mid x+y>=5 \wedge y>0)$	$(5, y-(5-x))$	pour 3L to 5L until 5L Full.
8.	$(x,y \mid x+y>=3 \wedge x>0)$	$(x-(3-y), 3)$	pour 5L to 3L until 3L full.

Solution path 1: For water Jug problem (6 steps)

Rule applied	5L Jug	3L Jug	Step No.
Start State.	0	0	1
1	5	0	2
8	2	3	3
4	2	0	4
6	0	2	5
1	5	2	6
8	4	3	

Goal State: 4 - -

The Goal of filling 4L water in 5L Jug is obtained within 6 steps, which is optimal Solution.

b. State Space Search

State - space Search is Similar to production System which represents the facility to Search for a better path.

while using this method, we find a path from Start State to Goal State to Solve a probelem.

A State Space basically consists of Components,

1. A Set S Contains Starting States of probelem.
2. A Set G Contains Goal States of a probelem.
3. Set of nodes (States) in Graphs/Tree are represented to Solve. The probelem.
4. Set of Arcs (which connects Nodes).

Eg:- (Missionaries & Cannibals probelem)

Missionaries : Good, Indicated by M, No.of missionaries : 3

Cannibals : Bad, Indicated by, No. of Cannibals: 3

No.of Boats : 1

Goal :-

To move all 3 Missionaries and Cannibals from one Side of The river to other Side of The River using boat.

Conditions :

Min 1 can row the Boat (either. M or C) Eg: M, C

Max 2 can row Boat (either M or C) Eg: MM, CC, MC

(M count) >= (C count), else game over.

To generate a Search Space, we use valid operations using Tree. mentioned below,

Goal State

Solution path: 11 steps:

IMIC ->R Start State (331: 000)

IMOC <-L

OM2C ->R

OM1C <-L

2MOC ->R

1M1C <-R

2MOC ->R

OM1C <-L

OM2C ->R.

OM1C <-L

OM2C ->R Goal State (000: 331)

Finally all the 3 missionaries and 3 cannibals are moved from one side of the river to the other side using a boat with in 11 steps.

C. Control Strategies:

Control strategies is one of the most important components of problem Solving which describes the order of Application of Rules to the current State.

The Control Strategy must explore the Solution Space in Systematic manner.

If we Select a Control Strategy where we Select a rule randomly from the applicable rules, then it definitly causes motion and finally leads to a solution.

They are;

1. Data-Driven Search (Also called as forward Chaining)
   It starts from Start State.
2. Goal-Driven Search (Also called as Backward Chaining)
   It starts from Goal State.

Forward Chaining:-

Here, the process begins with known facts and proceeds towards a Solution.

The States of the Next level of the tree are generated by finding the new State which matches the goal or not.

Backward Chaining:- It is a Goal-Directed Strategy which begins with Goal State and proceeds working backwards. It Concludes its goal when it reaches its Start State.

3. CHARACTERSTICS of Probelem:

To find a pro solution for a probelem, first we need to analyse the probelem based on below mentioned factors,

1. Type of the probelem.
2. Decomposition of the probelem.
2+1 Role of Knowledge.
3+1 Consistency of knowledge.
4+1 Requirement of the Solution.

1. Type of the probelem:

There are 3 Types of probelems like.

- Ignorable probelem (Simple control Strategy)
- Recoverable probelem (water Jug probelem)
- Irrecoverable probelem.

2. De Composition of the probelem:-

Decomposition of the probelem is the process of deviding the probelem into set of independent Smaller Sub probelems and later combines the Solved Sub Solutions to get the final Solution

This technique is also called as "Divide & Conquer" method.

It can also be used for parallel processing.

3. Role of Knowledge:

The Knowledge plays an important role in Solving any kind of probelem.

Knowledge can be in the form of rules & facts which help in generating Search Space to find the Solution.

Consistency of Knowledge:-

The Consistency of Knowledge base is very important to Solve a probelem. If The Knowledge base is Inconsistent, it may lead to wrong Solution.

Requirement of The Solution:-

Requirement of The Solution needs to analyze. The probelem whether we need an absolute or relevant Solution.

4. EXHAUSTIVE SEARCHES

Exhastive Search is an Uniformed Search technique which considers mostly all the element in the Search process.

Some of the Exhaustive Search techniques are,

1. Breadth First Search (BFS)
2. Depth (first Search (DFS)
3. Depth first itarative deepning (DFID)
4. Bidirectional Search.

1. Breadth First Search (BFS) :-

BFS expands all the States from the start State in 1st Level and expands all the states in the next level and so on until it reaches its Goal State.

2. Algoritom Explanation:-

The Search using BFS is implemented using two lists (OPEN & CLOSE)

The OPEN List Contains the States that are to be expanded.

The CLOSED List Contains the States that are already expanded

OPEN List is implemented as Queue, whereas CLOSED List is implemented as Stack.

Algorithm:-

Input: START State, Goal State

Variables: OPEN, CLOSED, State_X, SUCC, FOUND

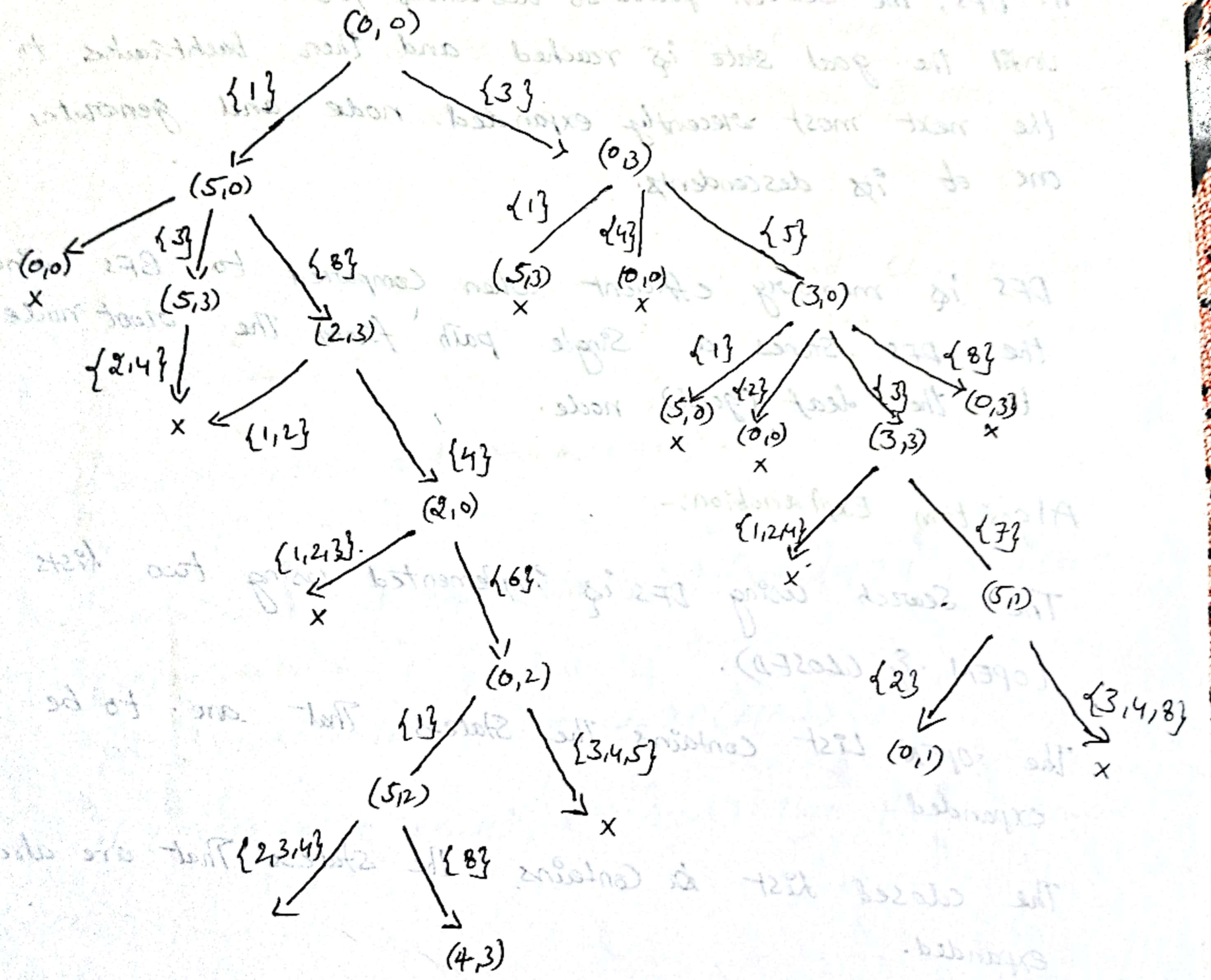
output: YES & NO.

- Initialize. OPEN List from START and CLOSED = NULL
  - FOUND = False.
- while. (OPEN ≠ NULL and FOUND = False) do.
  {
  - remove. the first state from OPEN and Call it as State -X.
  - Send The. STATE-X into CLOSED
  - if STATE-X = GOAL. Then FOUND = True.

  else.
  - Expand. STATE -X, which produces Succs.
  - append Succs to The end of OPEN List

  }
- if FOUND = true Then. return "YES" else. return "No"
- Stop.

Start State.



Goal State.

2. DEPTH FIRST SEARCH (DFS)

In DFS, The Search process descends from current state until the goal state is reached. and then backtracks to the next most recently expanded. node and generates one of its descendents.

DFS is memory efficient when compared to BFS Since the DFS stores a Single path from the root node to the leaf (goal) node.

Algorithm Explanation:-

The Search using DFS is implemented using two lists (OPEN & CLOSED).

The OPEN List contains The States That are to be expanded.

The closed list contains The states That are already expanded.

∴ Both The OPEN & The CLOSED List is implemented within Stacks.

Algorithm:-

Input: START state , GOAL State.

Variables OPEN, CLOSED, STATE-X, SUCC, FOUND, RECORD-X

output: YES (path Sequence from START to GOAL) or NO.

Method:

- Initialize OPEN List with (START , Nil) and CLOSED = NULL
- FOUND = false

- while (OPEN ≠ NULL and FOUND = false) do

{

- Send The initial node (START , nil) from OPEN to RECORD-X of CLOSED
- put RECORD-X in front of CLOSED (STACK)
- If (STATE-X of RECORD-X = GOAL) Then FOUND = true.

else.

{

- expand state-X called Succ's and connect with parent
- Remove The Succ's which are already in CLOSED
- Insert The Succ's in front of OPEN

3.

3

- if FOUND = True Then return The traced path else return "No"
- Stop.

Eg:- Search Tree using DFS for water Jug problem

WATER Jug problem

Search tree generation using DFS	OPEN LIST	CLOSED LIST
START State. (0,0)	[((0,0), nil)]	
↓ {1} (5,0)	[((5,0), (0,0))]	[((0,0), nil)]
{2} ↙ ↘ {3} X (5,3)	[((5,3), (5,0))]	[((5,0), (0,0), (0,0), nil)]
↓ {2} (0,3)	[((0,3), (5,3))]	[((5,3), (5,0), ((5,0), (0,0)), (0,0)), nil)]
{1,4} ↙ ↓ {5} X (3,0)	[((3,0), (0,3))]	[((0,3), (5,3)), ((5,3), (5,0)), (5,0), (0,0)) ((0,0), nil)]
↓ {3} (3,3)	[((3,3), 3,0)]	[((3,3), (3,0)), ((0,3)), (5,3)), ((5,3), (5,0)), (5,0), (0,0) ((0,0), nil))]
{1,2,4} ↙ ↓ {7} X (5,1)	⋮	⋮
↓ {2} (0,1)		
{1,2,4} ↙ ↓ (1,0)		
{6,2} ↙ ↓ {3} X (1,3)		
{1,2,4} ↙ ↓ {5} X (4,3)		
Goal State.	[(4,0) (0,3)]]	[(4,0), (0,3)), ((1,3), (1,0)), ((1,0), (0,1)), (0,1), (5,1)), ((5,1), (3,3)), (3,3)), (3,0)), ((3,0), (0,3)), (0,3)), (5,3)), (5,3), (5,0)), (0,0), (0,0), nil)]

The path
path
(0,0

The path is obtained from the last stored in CLOSED. The solution path is:

$(0,0) \rightarrow (5,0) \rightarrow (5,3) \rightarrow (0,3) \rightarrow (3,0) \rightarrow (3,3) \rightarrow (5,1) \rightarrow (0,1) \rightarrow (1,0) \rightarrow (1,3)$

$\rightarrow (4,0)$

DEPTH FIRST ITARATIVE DEEPENING (DFID)

Definition:- The DFID uses the advantages of BFS and DFS to make the search process in a tree.

BFS has an advantage of reaching the GOAL and.
DFS has an advantage of using less memory

Algorithm Explanation:-

→ The DFID expands all the nodes at a given depth before expanding any node.

→ The DFID is used to find the shortest path / optimal solution from a start state to GOAL STATE

→ The DFID never searches deeper than the depth (D) where by it reduces space since the space consiption is O(d)

Algorithm:-

Input: START State, GOAL State.

Variable: FOUND

output: YES, No.

d (depth)

- Initialize d=1 and FOUND = False.

while (FOUND) = False)

{.

perform a DFS from start to d.

if GOAL is obtained, FOUND = true.

- else discard that node and check for other nodes of depth d = d=d+1.

3. If FOUND= true. Then return the 'YES' else return 'No'

- Stop:

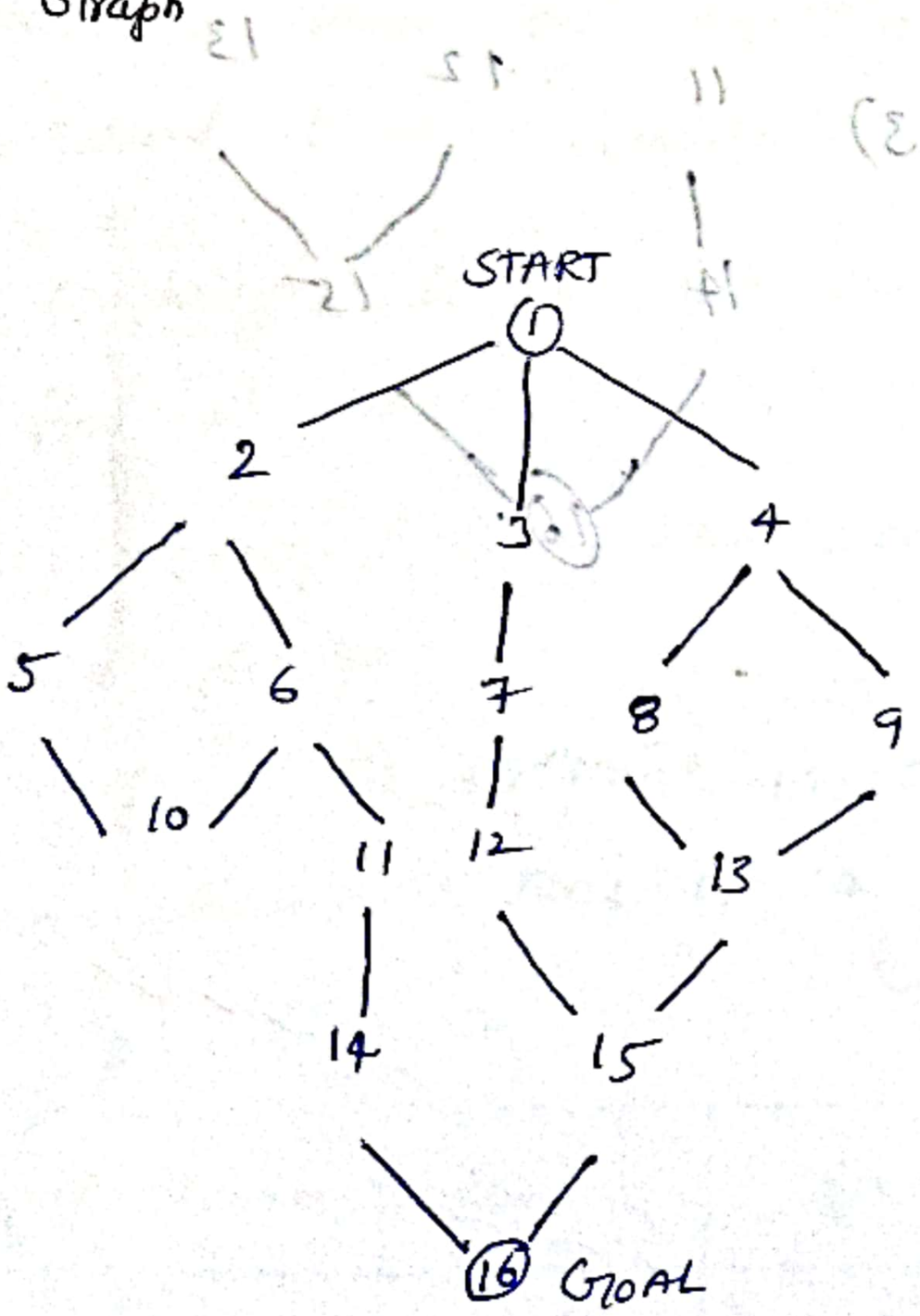
BIDIRECTIONAL SEARCH

→ Bidirectional Search is a graph Algorithm which runs two searches simultenasly

→ one Search starts from start state and other search moves backward from GOAL state.

→ when both the search meet in the middle then the search process stops.

Eg:- Graph



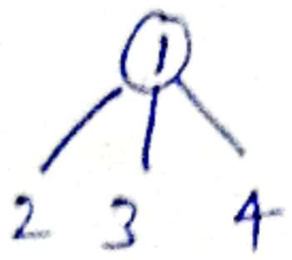
Our object is to find the best path, to reach the goal. using bidirectional strategy (TOP to down and Down to Stop similtaneously)

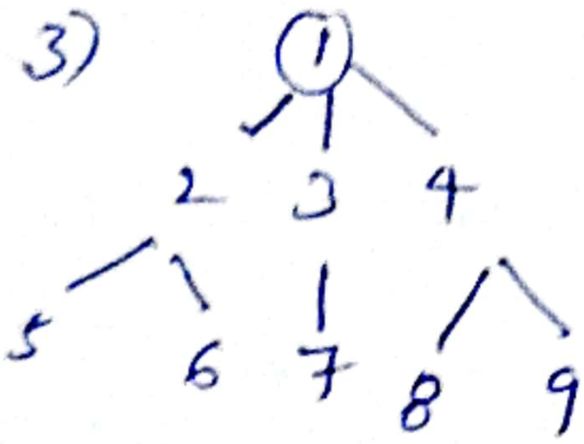
Tracing the Bidirectional Space.

In the figure, we have several iterations,

The top-down iterations are;

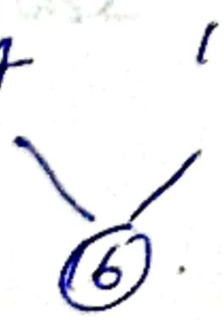
1) ● (1) (Start)

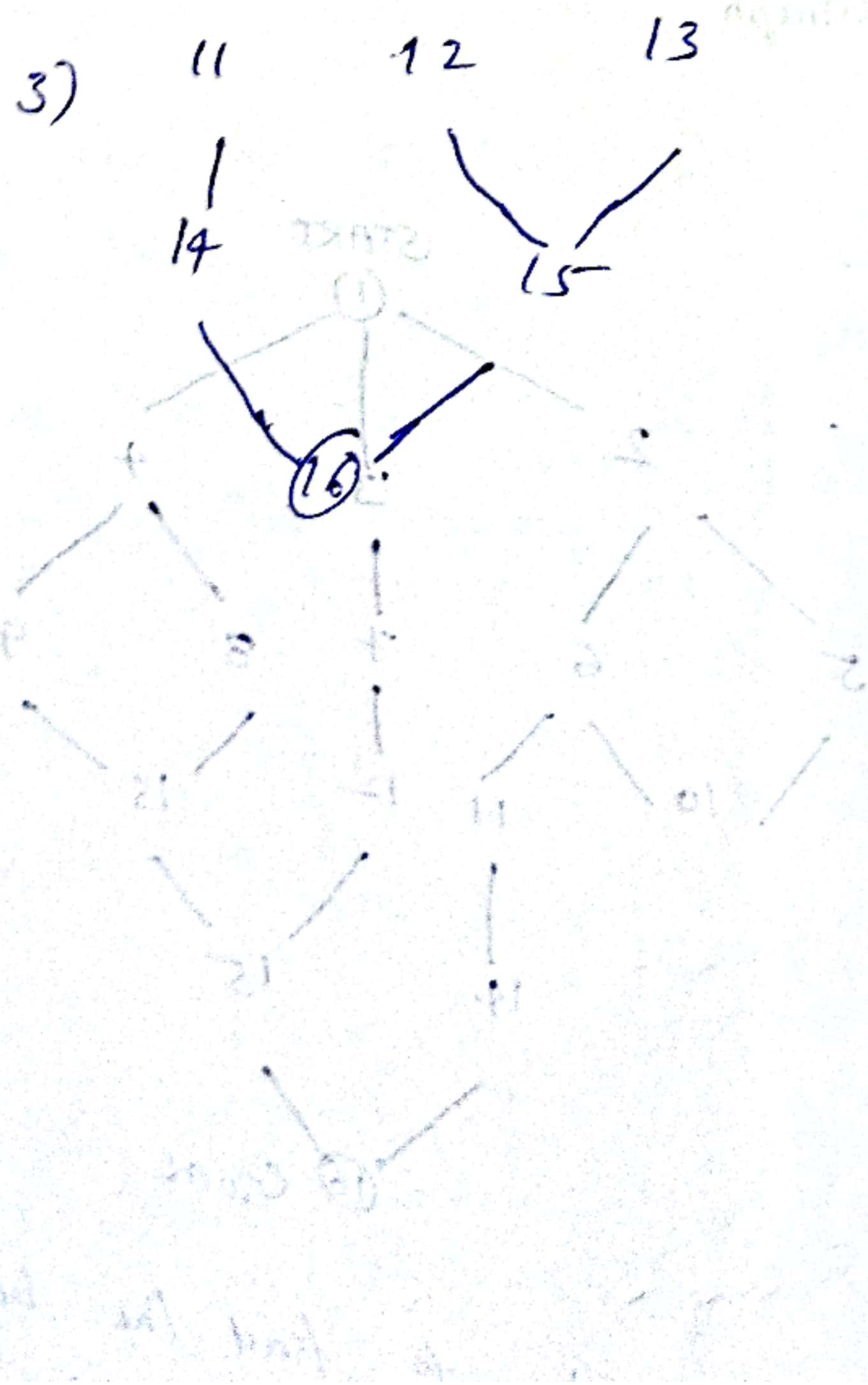
2) 

3) 

The bottom-up iterations are.

1) (16) GOAL

2) 

3) 

ANALYSIS OF SEARCH METHODS

The effectiveness of any Search Strategy in problem solving is measured in terms of-

1) Completeness
2) Time complexity
3) Space complexity
4) optiomdogy.

1) Completeness:- When an Algorithm guarantees a solution, it is considered to be. complete.

2) Time complexity:- It is time required by an Algorithm to find a solution.

3) Space complexity:- It is space required by an Algorithm to find a solution.

4) optiomology:- The Algorithm is optimal if it finds best solution from different solutions of a problem.

Search Techniq	Time	Space	Solution
BFS	$O(b^d)$	$O(b^d)$	optimal Sol
DFS	$O(b^d)$	$O(d)$	....
DFID	$O(b^d)$	$O(d)$	optimal Sol
Bi-directional	$O(b^{d/2})$	$O(b^{d/2})$	....

{ where:
b= branch of tree
d= depth of tree. }

5. Heuristic search techniques

a) General-purpose Heuristics

(i) One of the most important analysis of search process is to find the number of nodes in a complete search tree

(ii) This simple analysis improves the exhaustive searches and to find an upperbound on search time

(iii) This search strategy which uses some domain knowledge are called informed search strategy

5. b) Branch and Bound Search :-
(Uniform cost Search)

*) Here, The cost function (denoted by g(n)) is used to now. Commulative expose to The path from start node. current node 'x'

*) while generating search space a least cost path is is expanded at each iteration till goal state is reached.

*) Since branch and bound search expands The least-cost partial path, it is also called 'Uniform cost Search'.

Algorithm :-

Input : START and GOAL State

Variables : OPEN CLOSED, NODE, SUCC, FOUND

output : YES & NO.

Method :-

- Intialize The start node ($g(root) = \phi$) in OPEN List :

  CLOSED $= \phi$;

- FOUND = False;

- while (OPEN ϕ and FOUND = False) do.

  {

  - remove top element from OPEN LIST and call it NODE

  - if NODE = GOAL NODE, The round = true else.

    {

    - put NODE in CLOSED list.

    - find Succ's of NODE, if found, Then store. Their 'g' values in 'OPEN' LIST

- Sort all nodes in the open list based on cost function

3.

3. if found = true. Then return YES, else return NO.

- Stop.

5.c) Hill climbing:

*) Hill climbing is one of the variant of "Generate and Test" algorithm.

Generate and test Algorithm:

Start

- Generate a possible solution
- Test/ check it it is the goal.
- if not, go to start, else Quit

Stop.

*) Hill climbing can be used to solve problems that have many solutions where some solutions are better than others.

*) Hill climbing uses DFS with added quality measures ie (measure of remaining cost from current to goal)

eg:- Travelling Sales Man problem.

*Algorithm:

input: START AND GOAL states

Variables: OPEN, NODE, SUCC's, FOUND

output: YES, & NO.

Method :-

- Initialize START NODE in an OPEN list and FOUND = false
- while (OPEN ≠ empty and FOUND = false) do

{

- remove top element from OPEN CALL it NODE
- if NODE = goal NODE, Then FOUND = true.

else.

- ~~found S~~
- find Succ's of NODE if any, Sort them by cost goal. State and add them to OPEN List

3.

if FOUND = true. then return YES else NO.

- Stop.

5.d) Beam Search.

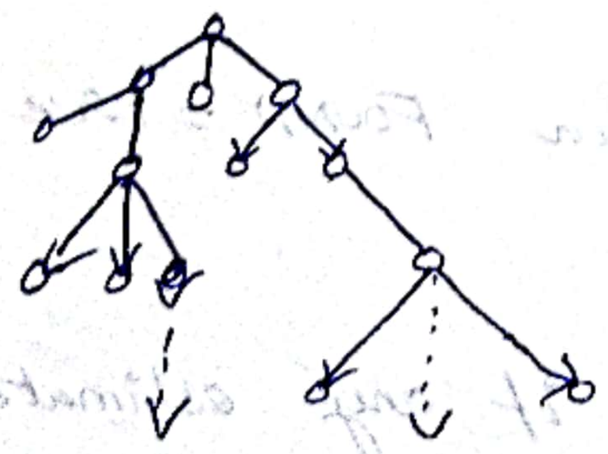
*) Beam Search is a Heuristic Search algorithm in which 'w' number of best nodes at each level is expanded.

*) It progress level by level and moves downward only. from the best 'w' nodes at each level.

*) Beam Search uses 'BFS' to build its Search tree.

Eg:- 'w' indicates No. of Best states on each level and 'B' indicates The Branching factor

ie: 'w' indicates No. of B NODES will only be consider.



w = 2 B = 1
w = 2 B = 2
w = 2 B = 3.

In above graph, w=2 and B=3.

Algorithm:-

Input : START AND GOAL STATES
Variables: OPEN, NODE, SUCCS W-OPEN, FOUND
Output : YES, & NO.

Method :-

- NODE = Root NODE and FOUND = False.
- If NODE = goal NODE then FOUND = True.

else.

find Succs of NODE, if any, estimate cost and Store. in OPEN list

- while (FOUND = false) do

{

- Sort OPEN List
- Select two 'w' elements from OPEN and ~~pull~~ PULL in w - OPEN and empty OPEN LIST
- For each node in w-open List

{

- if Node = Goal, Then FOUND = true

else.

find Succs of Node, if any estimate. cost and store. in OPEN.

}.

}.

- if FOUND = true. Then return yes or No.
- Stop.

5a) Best-First Search:

*) It is based on expanding the best partial path from current node to goal node.

*) The cost of partial paths is calculated using "Heuristic"

ie: if the state has been generated earlier and new path is better than the previous one, then change the parent and update the cost.

*) In hill climbing, sorting is done on sucessive nodes, but in BEST_First_sorting is done on entire list

*) N the performance depends on the heuristic function.

Algorithm:-

Input: START and GOAL states.

Variable:- OPEN, CLOSED, NODE, FOUND

Otput:- YES, or NO.

Method:

- Intialize OPEN list with root node, CLOSED = ϕ, FOUND = False
- while (OPEN = ϕ and FOUND = false) do

{

- if first element = Goal node, Then FOUND = true.
  else, remove from OPEN List and put in CLOSED list
- Add its Successors, if any, in OPEN LIST
- Sort the entire list with heuristic funtion to reach goal node.

- if FOUND = true, Then return yes, else return No.
- Stop.

A* Algorithm:

) A Algorithm is a Heauristic search teamique which is useful to find the shoit cut path through the search space using the 'Heauristic' function.

*) Heauristic function is denoted by h(n)

*) Cost to reach node(n) from stoart node is denoted by g(n)

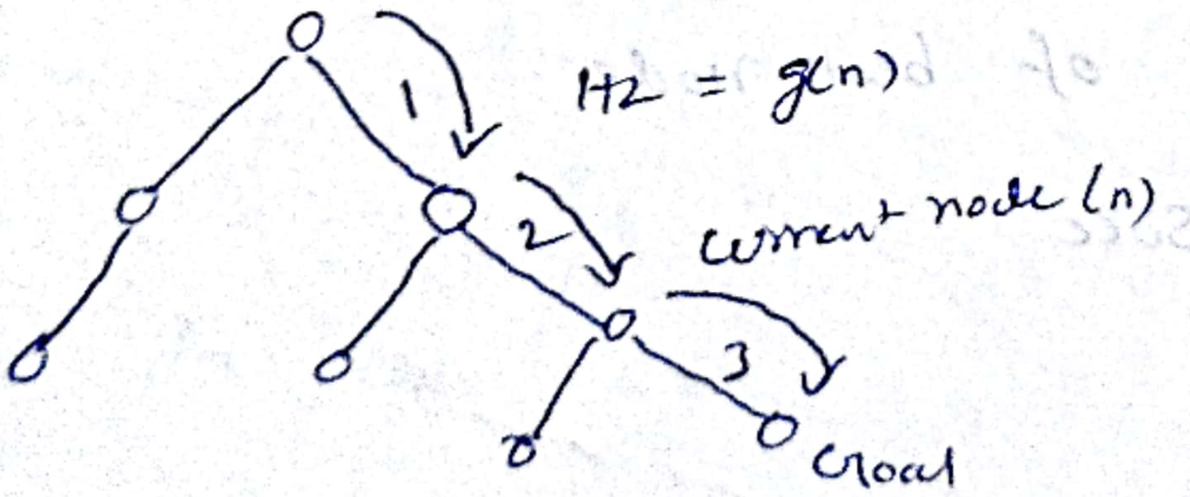
) Since A Algorithm uses both Heauristic + Cost values, the estimated cost of cheapest solution, denoted by f(n) is given by $f(n) = h(n) + g(n)$.

where, f(n) = estimated cost to cheapest Solution.

g(n) = cost to reach node(n) from stoart state

h(n) = cost to reach from node(n) to goal state.

Eg:-



Algorithm :- The algorithm proceeds with lowest $f(n)$ value at each node;

Input: START and GOAL STATES

Variables: OPEN, CLOSED, BEST node, SUCC, OLD, FOUND

Output:- YES, or NO.

Method:-

- Initialize. OPEN List with start node CLOSED = ϕ, $g = \phi$, $f = h$, FOUND = false.
- while (OPEN $\neq \phi$ and FOUND = false) do

{.

- remove node with low value of f from OPEN and store in CLOSED and call it best-node.
- if (best-node = Goal) then found = true.

else.

{

- Expand Succ of best node.
- for each Succ

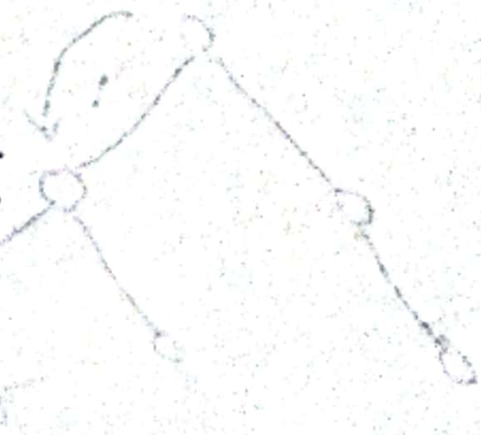
do

{

- estblish parent link of Succ and compute.

$g(Succ) = g(best\text{-}node) + cost$ (from best-node to Succ)

- if Succ $\in$ OPEN (not processed) Change. parent



- if succ ∈ CLOSED, (processed), Then Ignore.
- if succ ∉ OPEN & CLOSED

{

- add to best_nodes succ's
- compute. $f(succ) = g(succ) + h(succ)$
- Store succ on OPEN with its f value.

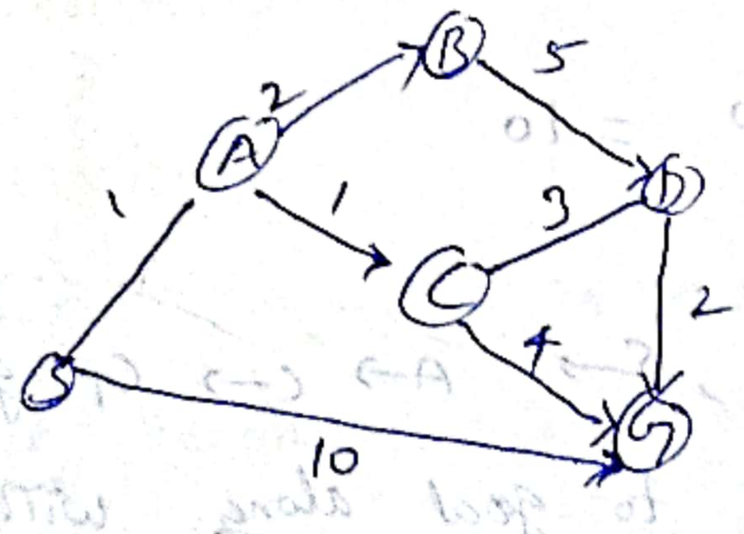
3.

3.

3.

- if FOUND = true then return YES, else return NO.
- Stop.

Eg:-



Suppose h(n) for each states

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

1) $S \rightarrow A \Rightarrow f(n) = g(n) + h(n)$

$f(n) = 1+3 = 4$

(or)

$S \rightarrow G \Rightarrow f(n) = \cancel{3+4=7}\ 10+0 = 10$

} (least and best

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

(or)

$S \rightarrow A \rightarrow C \Rightarrow f(n) = \cancel{5+6=A}\ 2+2=4$

(or)

$\cancel{S \rightarrow A \rightarrow C \rightarrow D \Rightarrow f(n) = 6+0=6.}$

}

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

(or)

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

}

4) $S \rightarrow G \Rightarrow f(n) = 10+0 = 10$

Out of all above paths, $S \rightarrow A \rightarrow C \rightarrow G$ gives the Complete path from start to goal along with $S \rightarrow G$

Now Consider the lowest $f(n)$ among above two paths,

ie, $S \rightarrow A \rightarrow C \rightarrow G = 6$

and $S \rightarrow G = 10$

from above, $S \rightarrow A \rightarrow C \rightarrow G$ is lowest $f(n)$ is 6.

So Consider its as the shortest path.

Q. optimal Solution by A* Algorithm

To get the optimal solution by A* Algorithm we need to consider 3 factors.

1) Underestimation
2) overestimation
3) Admissibility.

1) Underestimation:- Here, A* never estimates actual value from current to goal, The A* Algorithm ensures to find an optimal path from a goal.

2) overestimation:-

Here, The heuristic value of each node in the graph/tree is overestimated.

3) Admissibility:-

A Search Algorithm is admissible, if for any graph, it always terminates in an optimal path from path from Start state to goal state, if path exits.

11. Monotonic function

A: heuristic function (h) is monotonic if,

1) $\forall$ states x_i and $x_j \rightarrow x_j$ is successor of x_i

ie; $h(x_i) - h(x_j) \leq \text{cost}(x_i, x_j)$

2) $h(\text{Goal}) = 0$

Each monotonic heaustastic function is admissible.

*) A cost function (f) is monotonic if $f(N) \leq f(\text{Succ}(N))$

*) for any admissible cost function .f, we can construct a monotonic admissible function

6. Iterative Deepening A* (IDA*)

) It is a combination of depth first Iterative deepening and A Algorithm.

*) Here, The Successive Iterations are corresponding to increasing Values of The total cost of path rather than increasing depth of a search.

Algorithm:-

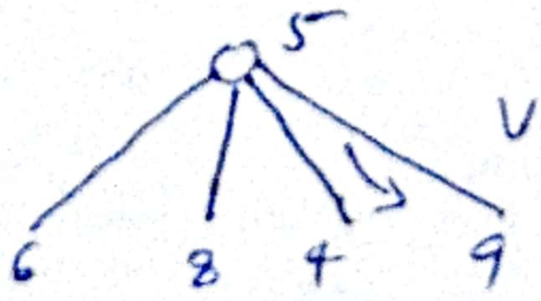
*) For each Iteration, perform DFS for a branch when its total cost (g+h) exceeds a given threshold.

*) Let, The Intial threshold starts at estimatic cost of The start state and increases for each Iteration of The algorithm.

*) The Threshold used for next Iteration is The minimum cost of all Values exceeding The current Threshold.

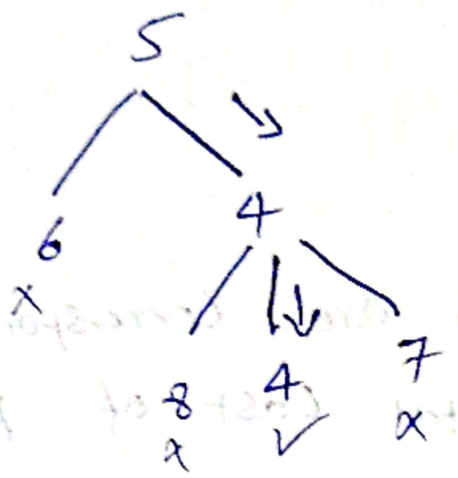
*) Repeat above steps until goal state is reached.

Eg:- 1st Iteration;



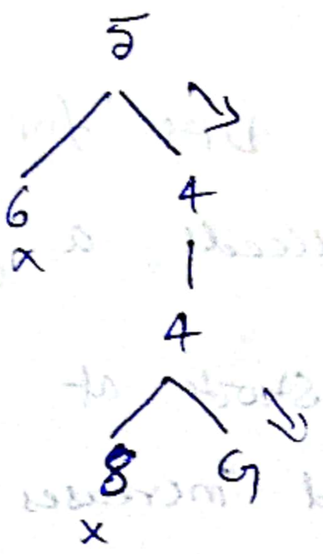
Value less than Threshold (5) is choosen.

2nd iteration :-

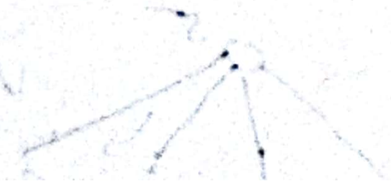


4 is choosen.

3rd iteration :-



Goal is reached.



7) Constraint Satisfaction

*) Many A.I problems can be viewed as problems of constraints satisfaction in which the goal is to solve some problem state that satisfies a given set of constraints instead of finding optimal path to the solution. Such problems are called. Constraint Satisfaction (CS) problem.

Examples of (CS) problems are.

a) Cryptography

b) n- Queen problem

c) map colouring

d) crossword puzzle .. etc

Cryptography problem :-

C_4 C_3 C_2 C_1

B A S E

\+ B A L L

G A M E S.

Constraints Equations are.

$E+L = S \longrightarrow C_1$

$C_1 + S+L = E \longrightarrow C_2$

$C_2 + 2A = M \longrightarrow C_3$

$C_3 + 2B = A \longrightarrow C_4$

INTRODUCTION

PART-B

1) An effective way of solving a complex problem is to reduce it to simple parts and solve each part separately, This method is called problem reduction

2) To solve each kind of complex problems we use. AND-OR graph or tree.

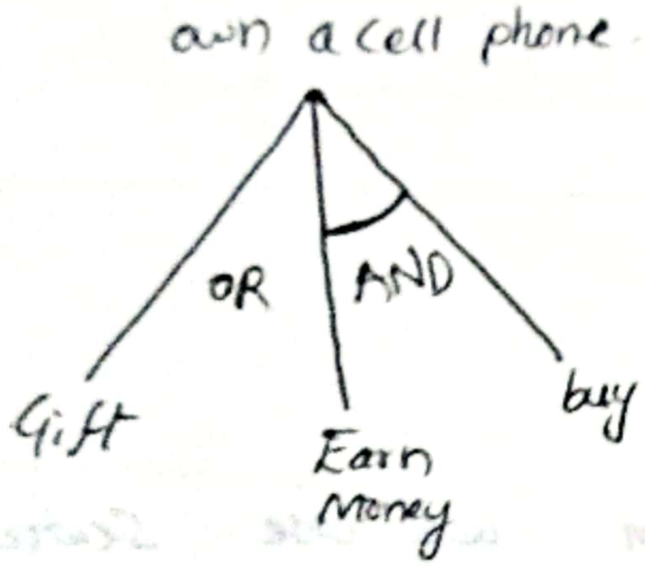
3) we use AND-OR Graphs in 'Game playing' which uses State space search problem solving mechanism.

4) The effectiveness of a Search may be improved by using generate-test procedures.

problem - Reduction

1. A given complex problem can be solved in many ways, here we use AND-OR graph, tree to find the solution.

Eg:-



2. By default we don't have Arc, it is considered to be ~~AND~~ and if a represent with an Arc between paths, it is considred to be 'AND'

3) From above example, if we want to own a cellular phone. we have two choices, They are.

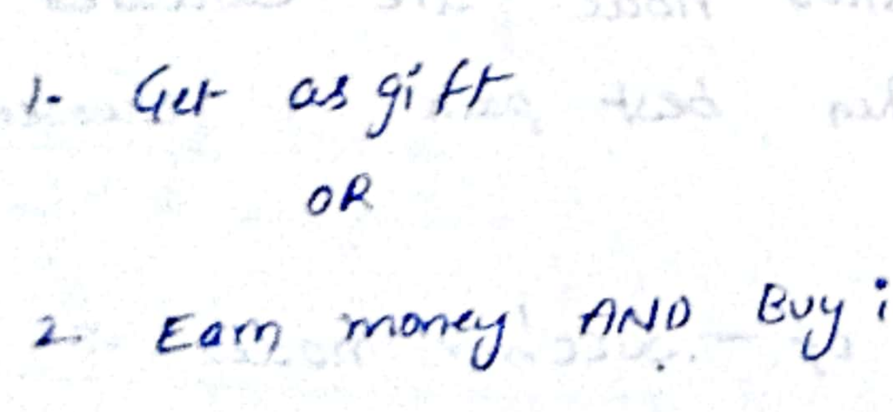
1- Get as gift

OR

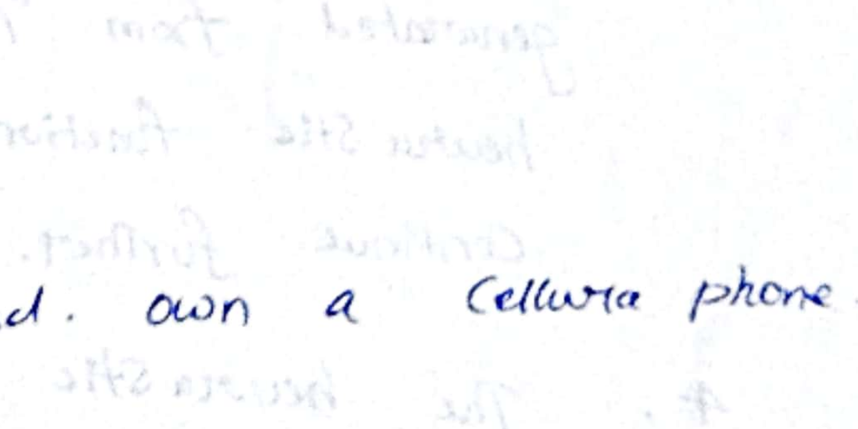
2- Earn money AND Buy it and own a Cellular phone

Ex2:- 3 disks Towers of Hanoi

Start State

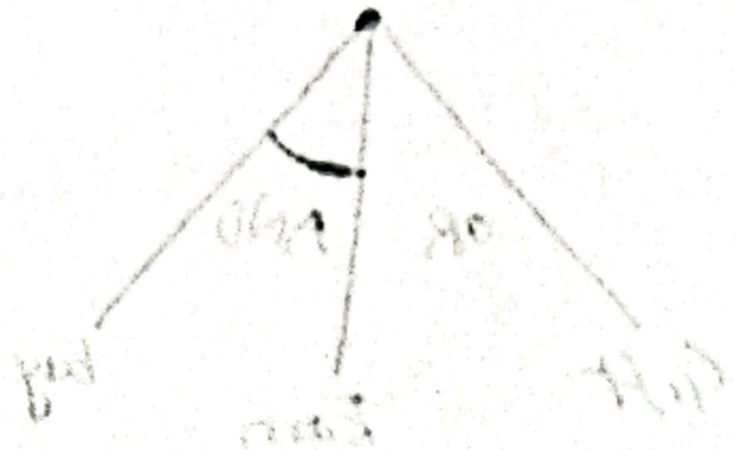


Goal State



Conditions :-

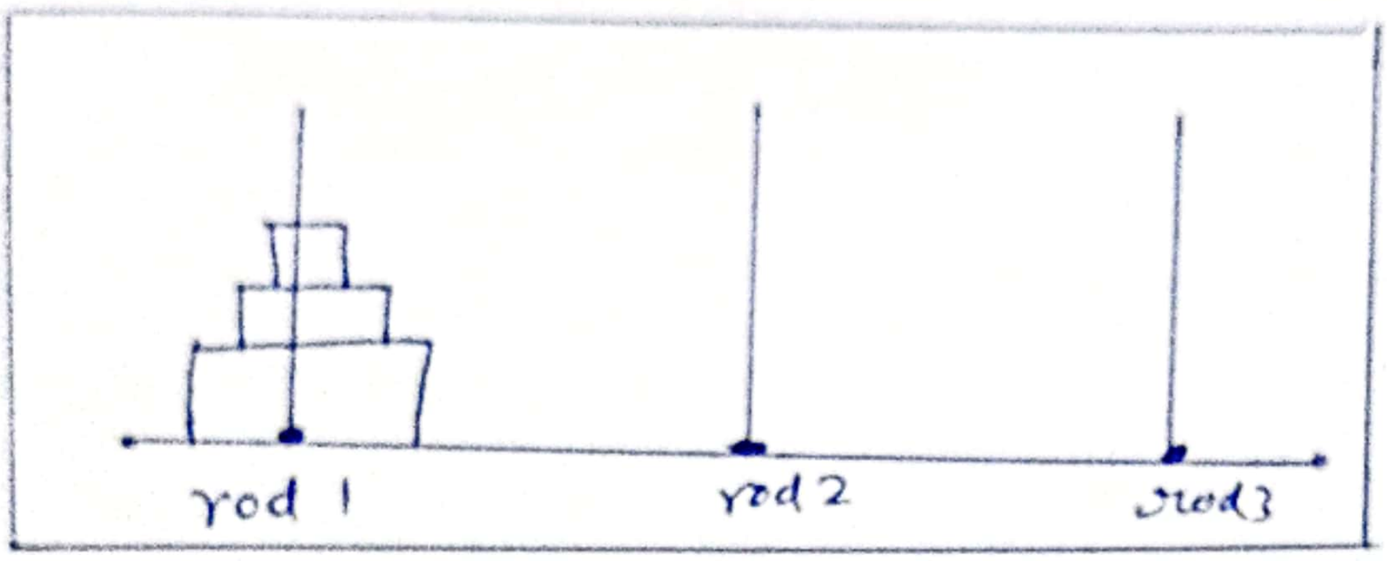
1. Only one disk can be moved at a time.
2. No large disk should be place on a small disk
3. move all the disks of rod 1 to rod 2 with the same order.



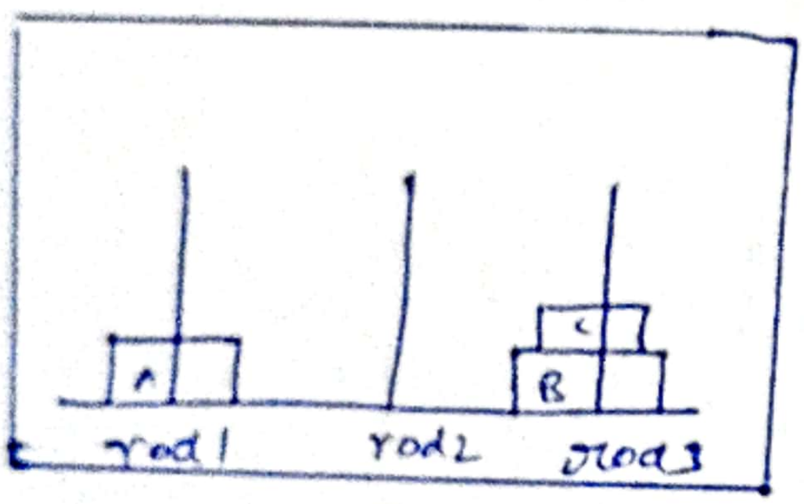
process :-

1. To solve the above probelem we use search space tree which is in the form of AND-OR graph
2. Here with Heurastic function denoted by 'F' for each node in AND-OR graph which is similar to compute and estimated value. (A*)
3. In AND-OR graph, the estimated costs for all paths generated from the start node are calculated by heurastic functions. Then best path is choosen to continue further.
4. The heurastic values of successer nodes are calculated and cost of parent nodes is revised Accordingly
5. The revised cost is propagated back to the start node through the choosen path

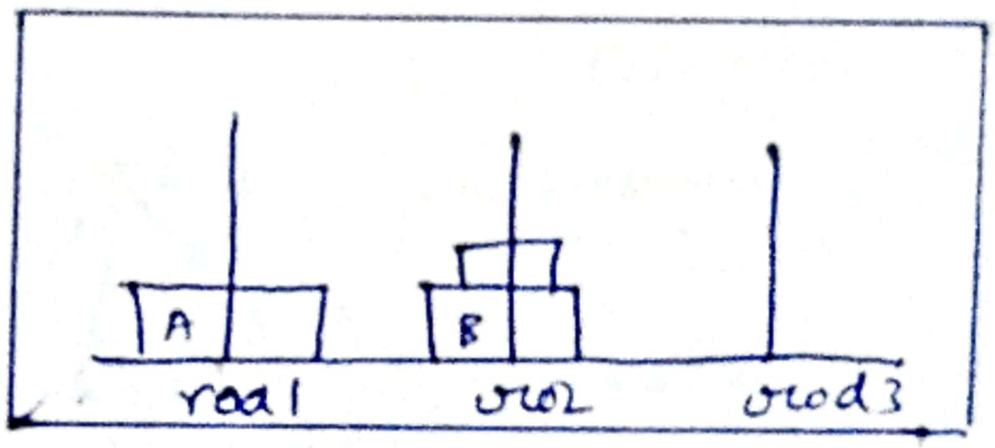
AND-OR Graph for 3 disks Towers of Hanoi problems.



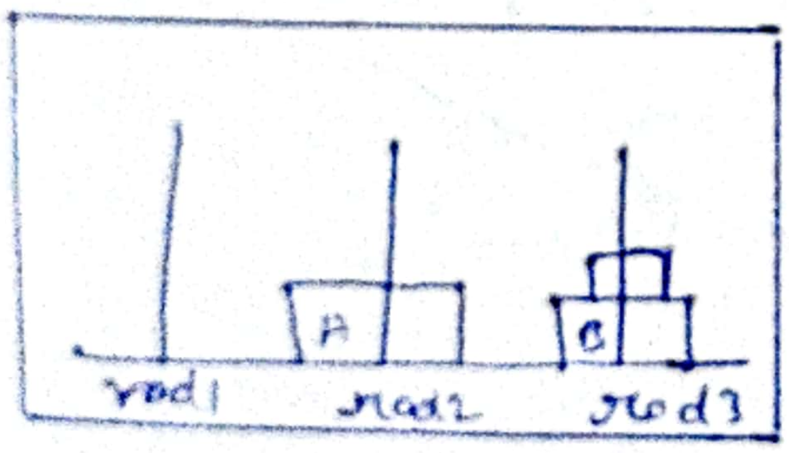
A



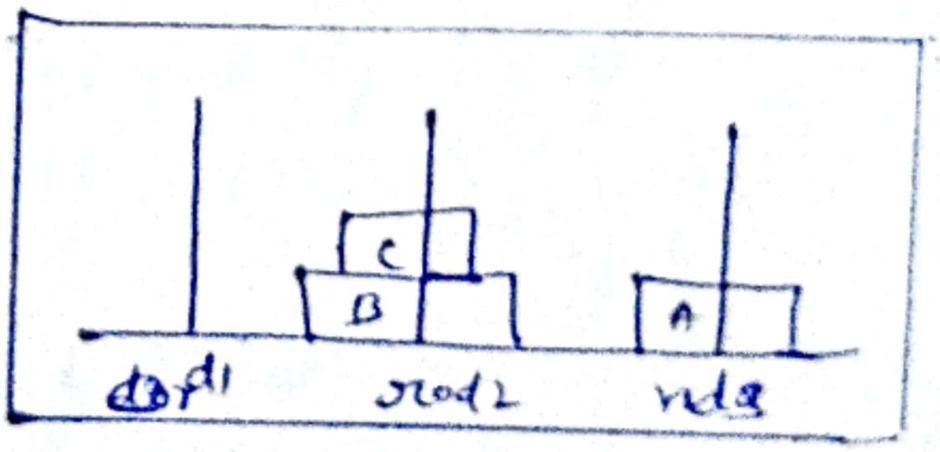
A'



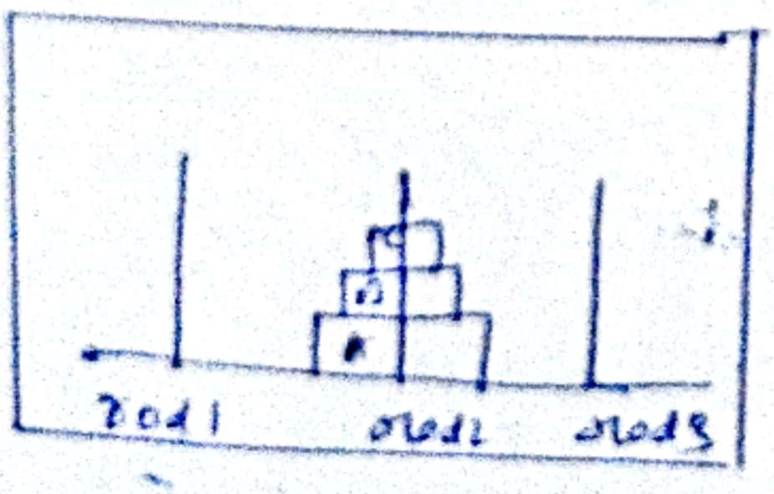
B



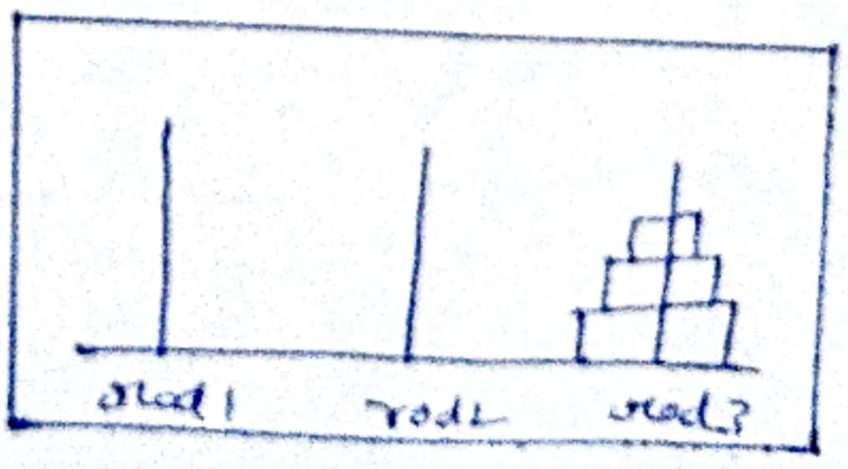
B'



C



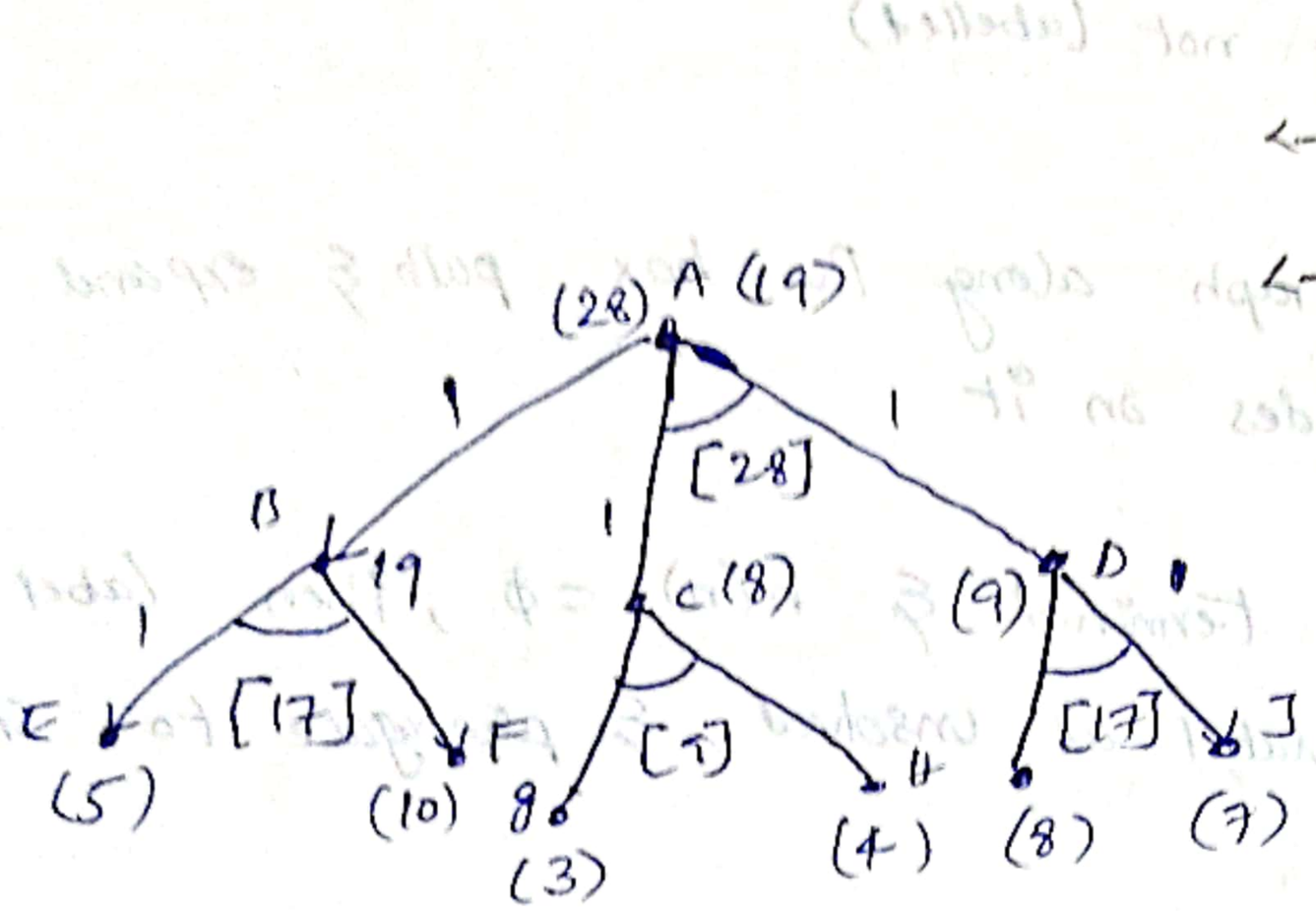
C'



AO* Algorithm

AND-OR graph algorithm is also called "AO* Algorithm" which is used to search for a solution in AND-OR graph

AO* Algorithm:- Eg:-



← Intial estimation Values
← revised Values
← Estimated Value
← revised Values
← estimated Values.

let $h(n) = 1$ for all nodes &

$g(n)$ estimates values are represented by ( ) &

revised values are represented by [ ].

- from above Eg, Intialy estimated values are 28 & 19, for (A→B) & (A→C & D), so we choose (A→C & D)
- Since we choose (A→C & D), we now expanded C & D & change the estimated values and also compare the revised values
- After comparing & changing the revised values, a New revised value occurs for (A→C & D), is [28] instead of (19), So now we compare [28] revised value with (A→B) value. ie [28]

- Hence now we change our path to (A→B) values, a new the optimal solution to reach the goal

AO* Algorithm:-

- Initialize graph with start node.
- while. (start is not labelled)

{

- Traverse the graph along the box. path & expand all unexpanded nodes on it
- if node is terminal & h(n) =ϕ, Then label the node, else label as unsolved, & propagate to start node.
- if node is Non terminal, add. The.
- Revise cost of expanded nodes & propagate to start node
- Choose the current best path

3.

- if (start node = solved), The leaf nodes of best path from root is the solution nodes, else., No solution exits

3. Game playing

- Game playing requires contain well defined states & rules
- A game is delayed as a sequence of choices which are made from a number of discreate alternatives
- Games can be classified into two types,

a) perfect information Games (Tic-Tac-Toe, Chess...)
both players have Access to same information

b) Imperfect information Games (cards, dice, ...)
both players do not have complete information

a) Game probelem Vs state space probelem.

Game probelem	state space probelem.
Board position	States
legal moves	Rules
winning position	Goal.

- A Game starts with intial state & ends with a win/loss/ draw
- Game probelem is defined by using "Game Tree"

In 'Game Tree', one level is treated as OR node level & other as 'AND' node level

The philosophy of games is, minimize. The maximum possible loss & maximize. The minimize. gain

- Two players are named as MAX & MIN.

) Status Labeling procedure. in Game Tree.

- Status labelling procedure for a node with WIN, Loss & draw for a "game Tree" is,

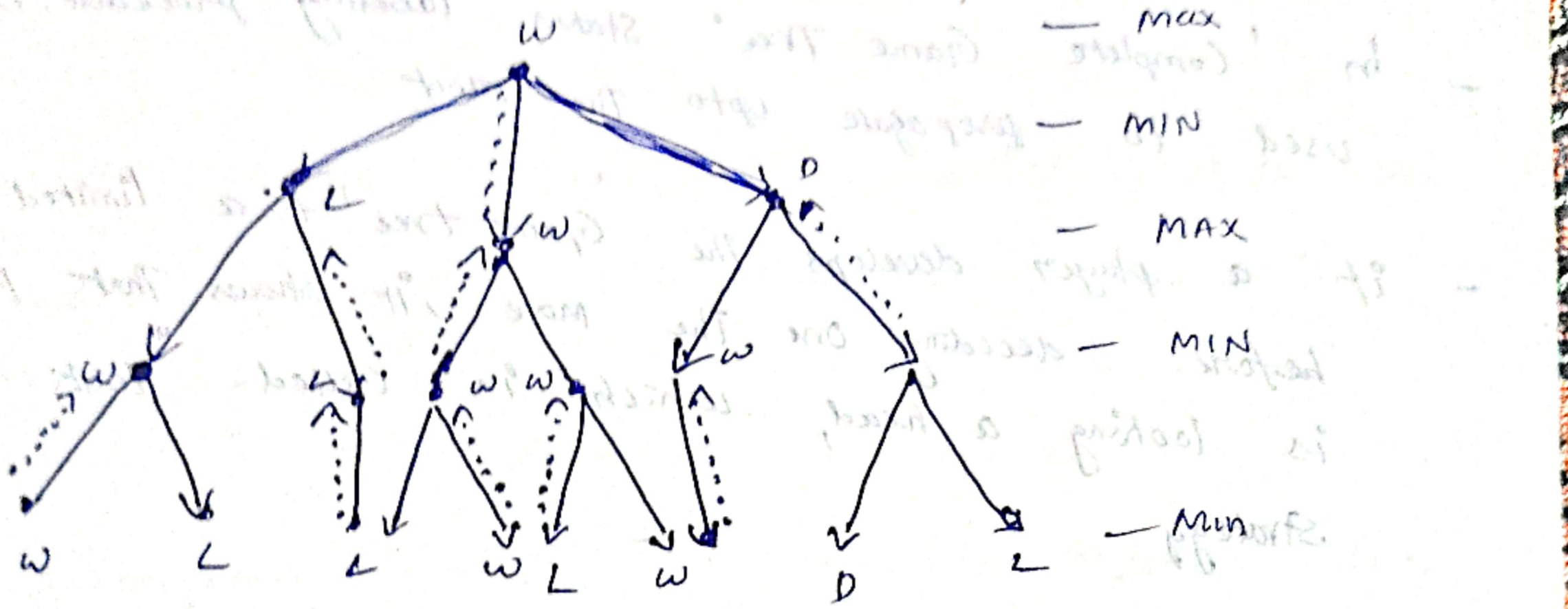
• if i is a non-terminal maxnode, then

$$\text{Status}(i) = \begin{cases} \text{win}, & \text{if any of i's successors is a win} \\ \text{loss}, & \text{if all i's successors is are loss} \\ \text{DRAW}, & \text{if any of i's succs is draw \& none is win} \\ & \text{(NONE)} \end{cases}$$

• if i is a Non-terminal MIN node, then

$$\text{Status}(i) = \begin{cases} \text{WIN}, & \text{if all i's successors are WIN} \\ \text{loss}, & \text{if any of i's succs is a loss} \\ \text{DRAW}, & \text{if any of i's succs is a draw \& none is loss} \end{cases}$$

Eg: Complete 'Game Tree' with "MAX" playing First move.



Let us Indicate win with 'W' loss with 'L', Draw with 'D'

c. Nim Game problem

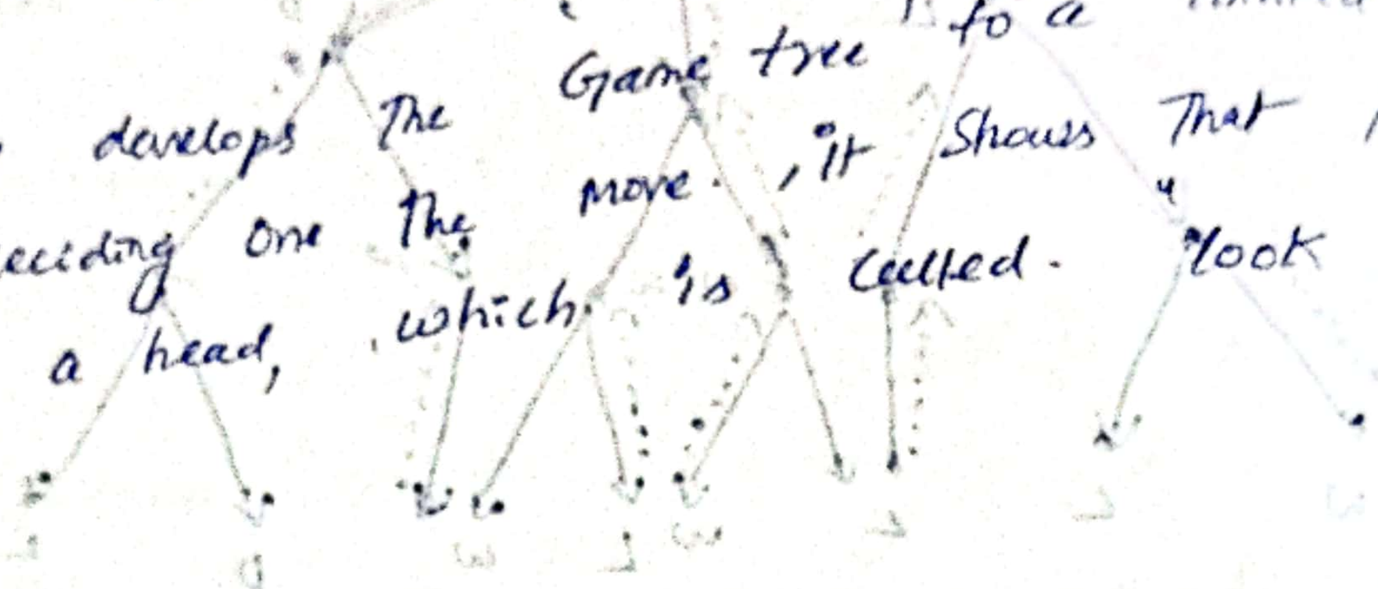
It is a two player game which has single pile of matchsticks (>1). Moves are made by players alternately in a move, each player can pick a maximum of half the no. of matchsticks in pile.

Whoever takes the last matchstick losses

4. Bounded Look - Ahead Strategy & use of Evaluation functions.

- In 'Complete Game Tree' states labelling procedure is used to propagate upto the root

- If a player develops the Game tree for a limited step before deciding one the move, it shows that player is looking a head, which is called. 'look ahead' Strategy



- if a player is looking ahead n number of levels before making a move, it is called n-move 'look-ahead' strategy

- The Actual value of a terminal state is unknown Since it is not a Exhaustive Search - In this case, we can use an 'Evalution function'

Using Evalution functions :-

- The process of evalution of a game is determined by the structural features of the current state.

- The steps involved in the evalution procedure are,

  - First step is decide which features are of values in a game
  - Next-step is to provide each feature with a range of possible values

o last step to divise a set of weights in order to combine all the feature values into a single value.

MiniMAX :-

procedure:-

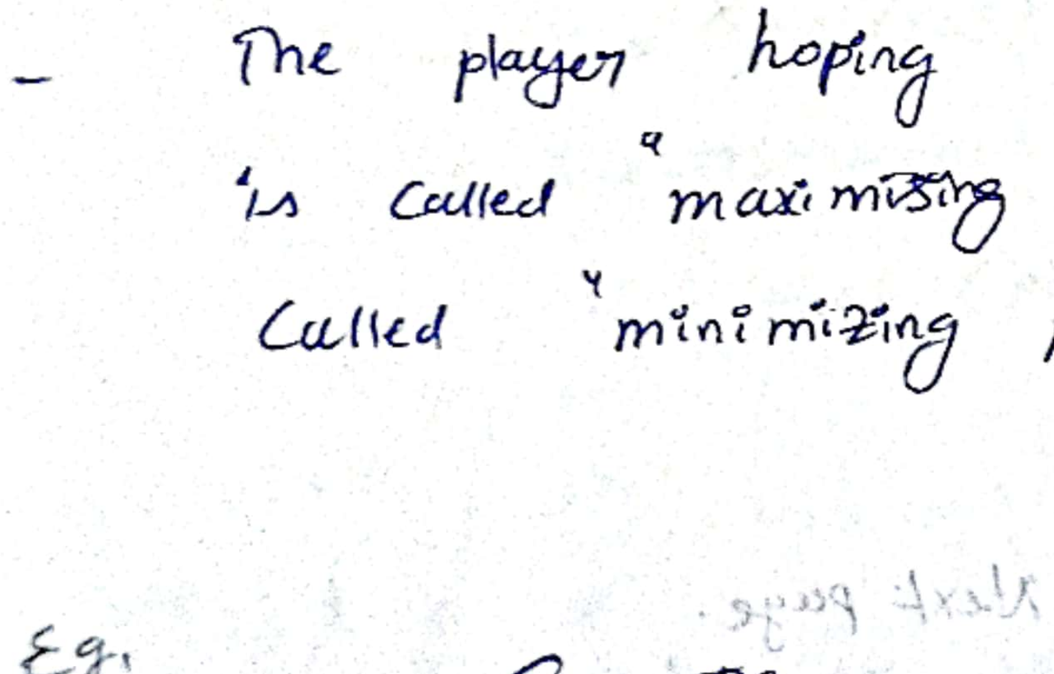
The estimated scores generated by a heuristic evalution function for leaf nodes are propagating to the root using 'MINIMAX' procedure.

- A 'MINIMAX' procedure is a recursive algorithm where, a ~~player~~ player tries to maximize its chances of win a simultenously minimizing the chances of win of approval

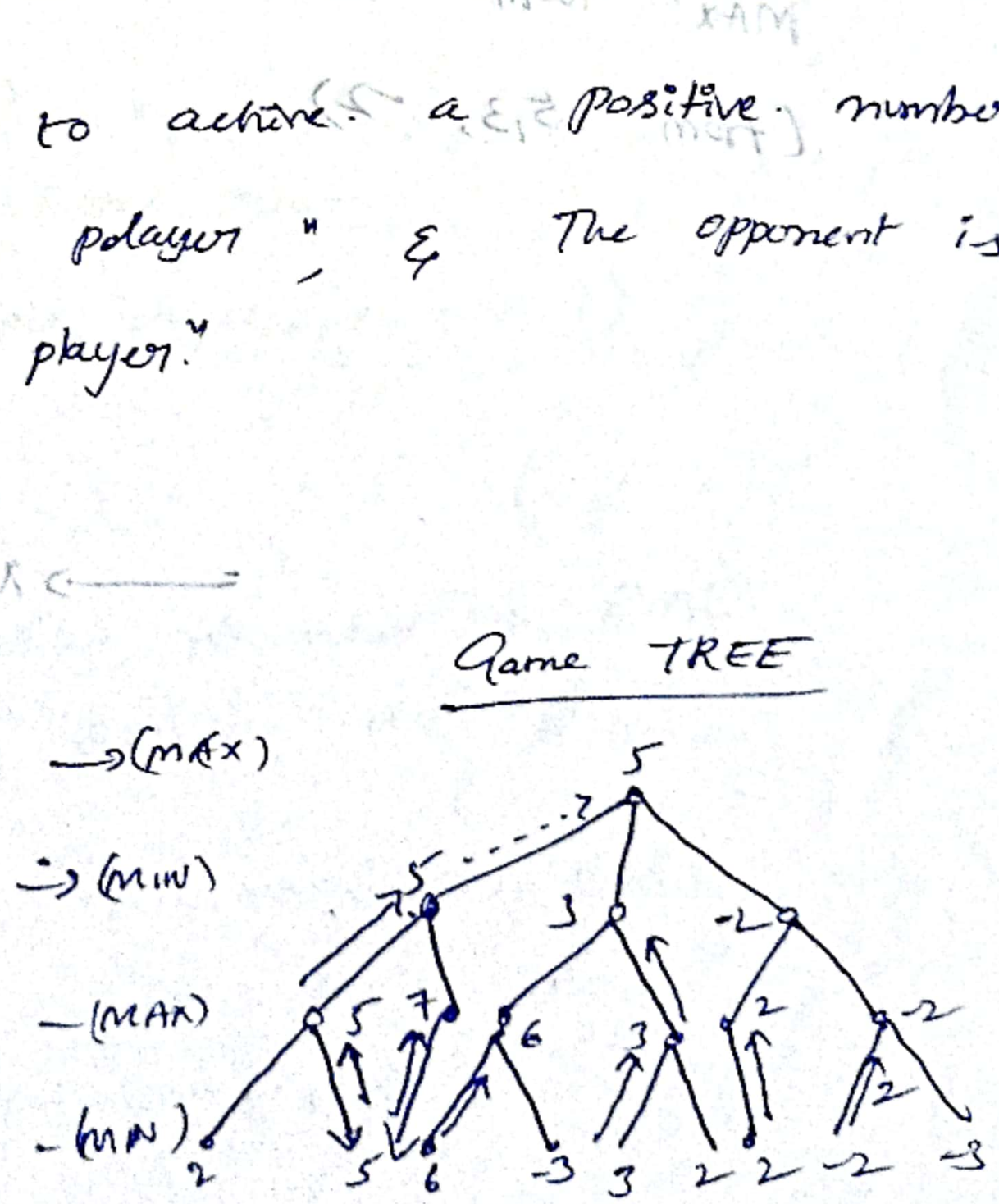
- The player hoping to achieve a positive number is called "maximizing player", & the opponent is called "minimizing player."

Eg:-

Game Tree



Game TREE



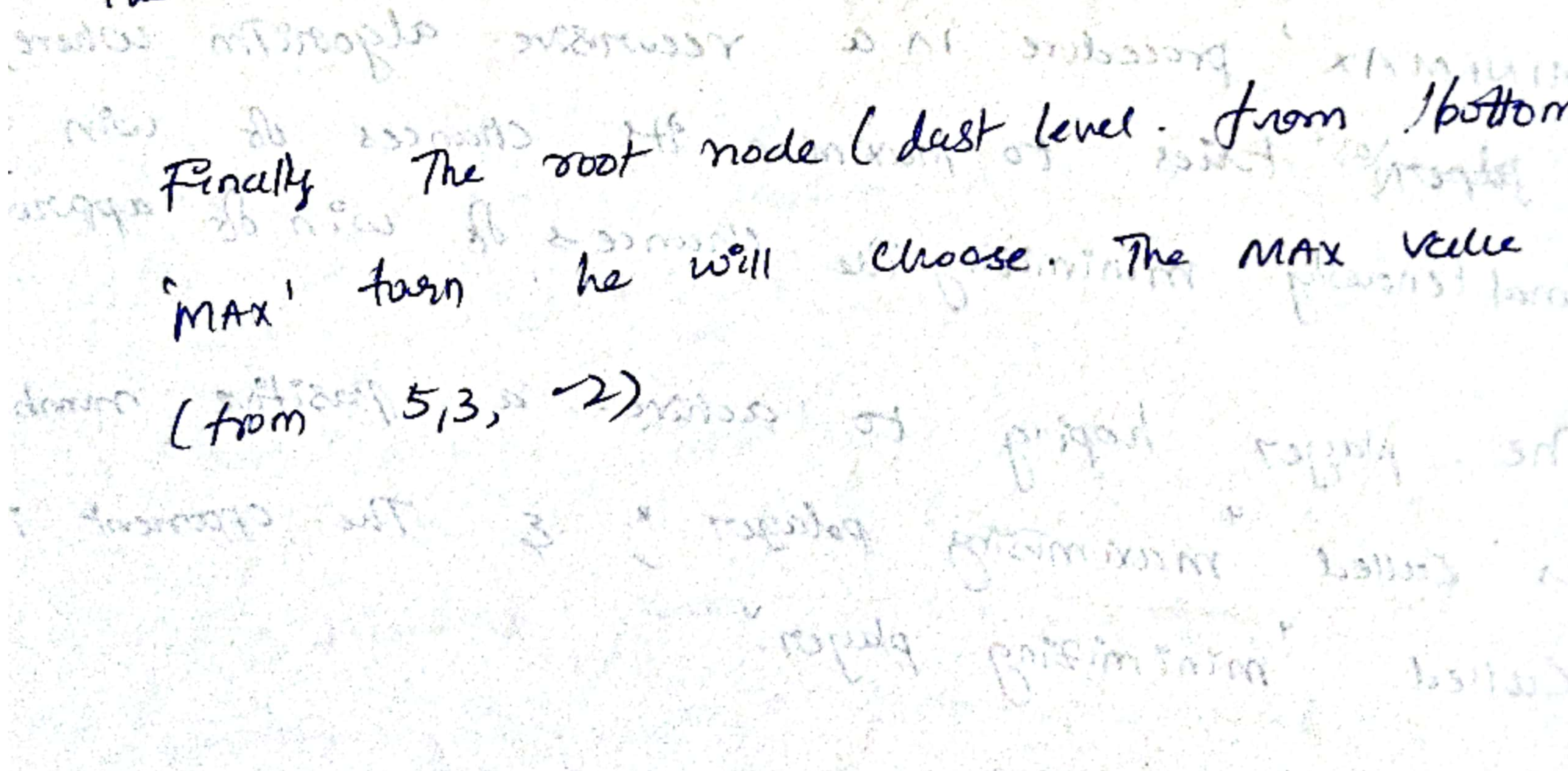
In above figure, The leaf nodes are to be used since They have heuristic values

Since The parent node of leaf node is 'MAX' turn, he chooses The MAXIMUM value from its leaf node is '5' from (2&5) & - so on

Since The above level of is 'MIN' turn, he chooses The minimum value is 5 from (5 & 7) and so on

Finally The root node (last level from bottom) is of 'MAX' turn he will choose. The MAX value i.e: 5 (from 5,3, 2)

⟹ Next page.



(MINIMAX)

MINIMAX Algorithm.

→ It is a recursive / back tracing algorithm which is used for Decision making in Game Theory (which searches in (GAME TREE) (TIC-TAC-TOE ... chess)

```
MINIMAX (pos, Depth, player)
{
  if DEPTH (pos, Depth) then return
        ({VAL= Eval (pos, player) .path = Nil})
  else.
  {
    SUCC_LIST = GEN (POS)
    if SUCC_LIST = Nil then return
      (Eval = Eval (pos, player) . path = Nil})
  else
  {
    Best_val = Min Value returned by EVAL
    for each_succ's succ_list do
    { SUCC_Result = MINIMAX (succ, Depth+1, player)
      No's value = -Val of SUCC-Result
      if New_value > Best_val Then
      {
        Best_val = New_value.
        Best_path = Add (succ, path of SUCC Result)
      }
    }
    return (Eval = Best_val. path = Best-path})
  }
}
```

5. Alpha - Beta pruning
(Optimized Minimax Algorithm)

- The main drawback of Minimax algorithm is, it cannot handle very complex games (chess, go, etc). Since these games have ~~big~~ huge branching factor a players have lot of choices

- The above limitation can be overcome by 'Alpha-Beta' pruning.

Definition:-

- Alpha-Beta (α-β) pruning is a strategy which reveals the number of tree branches, which is also called Backward pruning.

- α-β pruning is a modified depth first procedure which reveales the amount of work done in generating useless nodes

procedure :-

- α-β pruning procedure requires 2 Threshold values α (Lower Bound) $\in \alpha$, & β (Upper bound)

Max Node has α which never decreases &

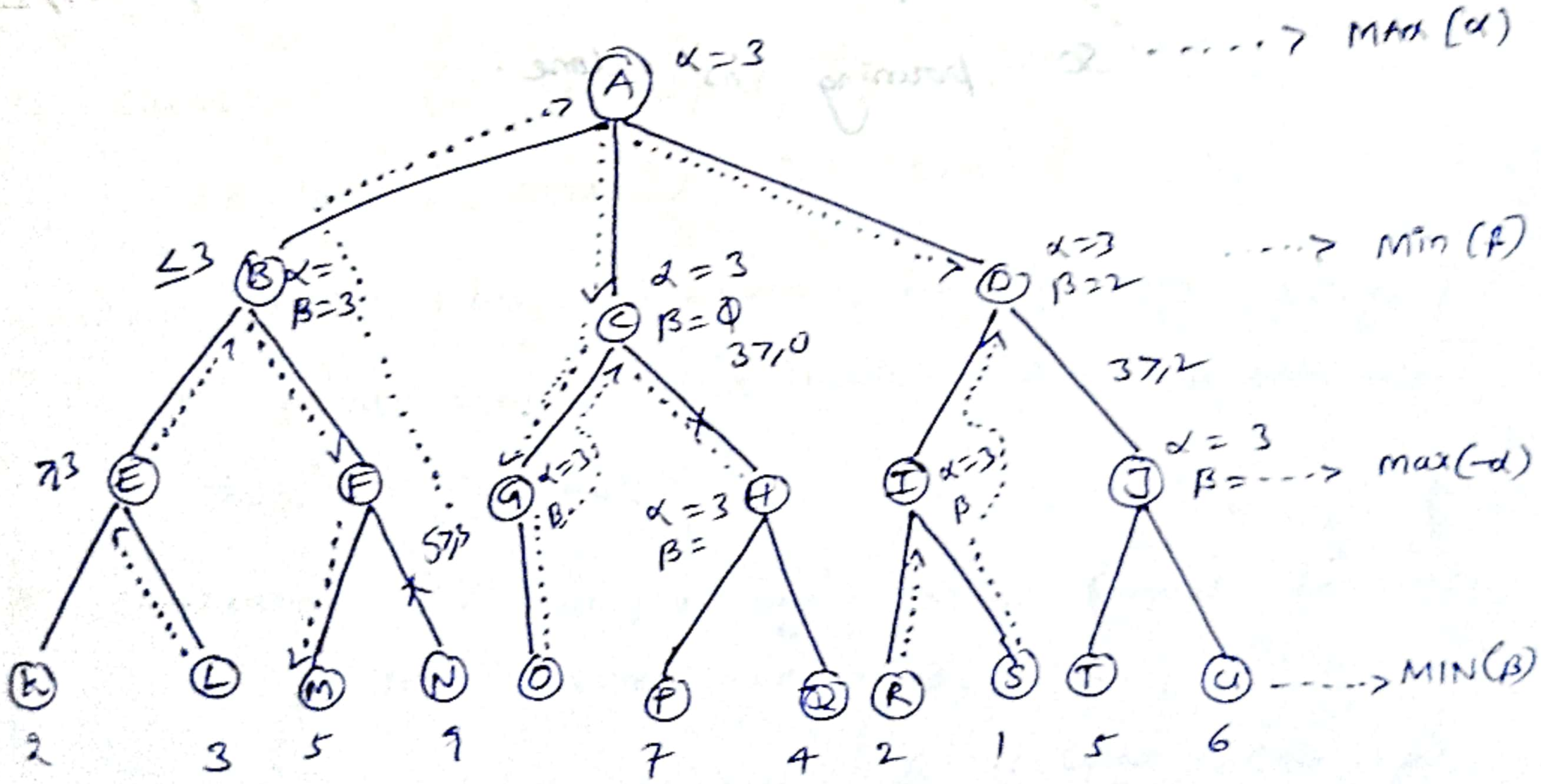
MIN Node has β which never increases $\leq$ (Threshold

During the traverse, α-β pruning uses DFS, & checks nodes where $\alpha \geq \beta$ during back traverse, & its condition satisfies, then pruning is done from that node.

MAX = α

MIN = β

pruning condition : $\alpha \geq \beta$.



Ex.

From above MINMAX Game Tree, α-β pruning is applied,

Since α-β pruning is applied, DFS is used

First (3) is extracted by α at E,

(3) is extracted by α at B.

(3) is corrected by down to β at F

(5) is extracted by α at F.

Now comparision at F is done, for α & β $\Rightarrow (\alpha \rightarrow \beta)$ & since $5 \geq 3$, Then pruning is applied for that branch.

w, (3) is extracted by α at A

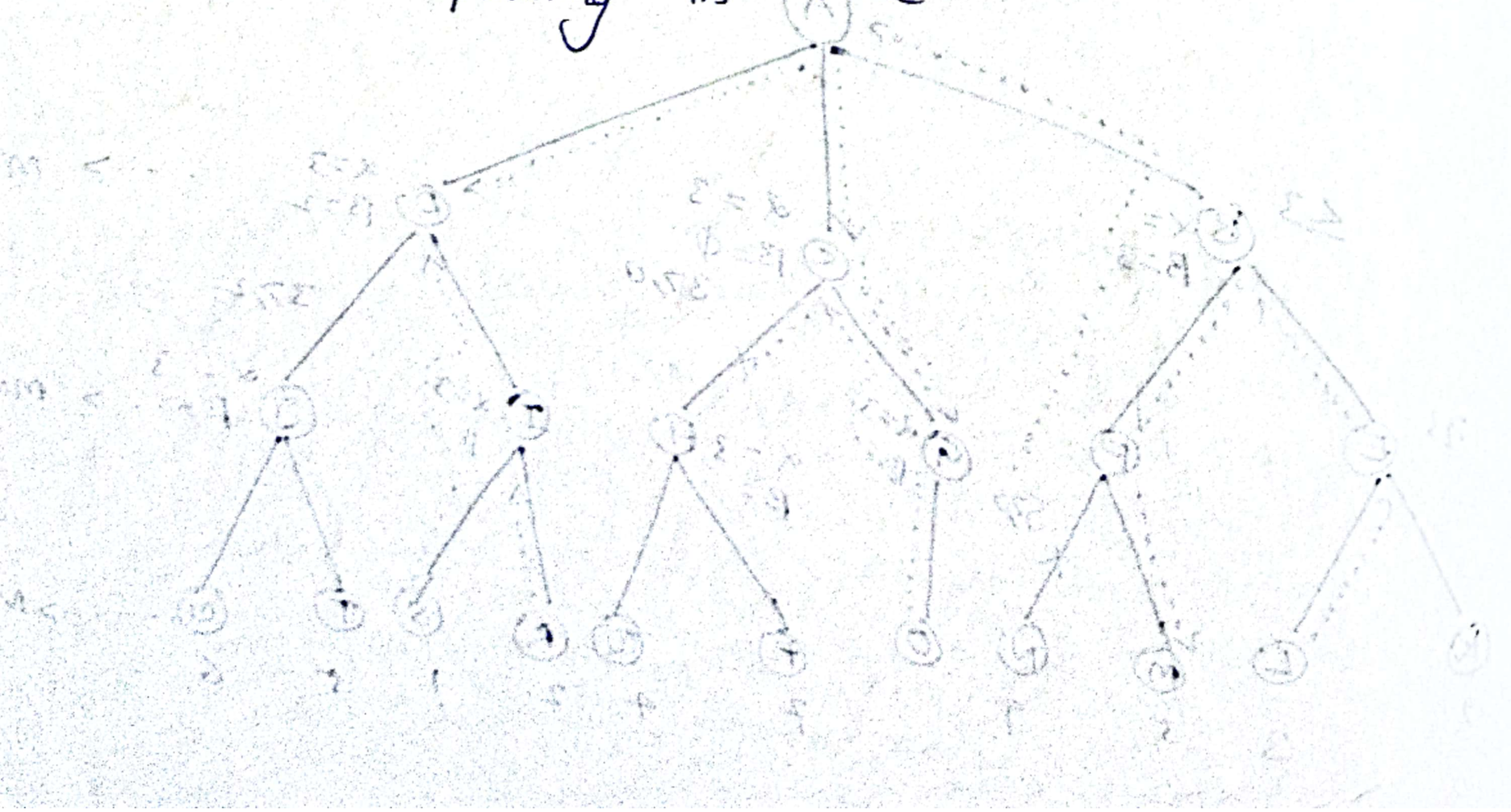
(3) is connected down to all Succes is, C, D, G, H, I, J (Non-terminal, Nodes)

(2) is extracted by β at C.

New comparision at C is done. $\alpha \geq \beta$, so pruning is done.

(2) is extracted by β at D.

New Comparision at D is done. $\alpha \geq \beta$, so pruning is done.



6. Two - player perfect Information Games.

Some of the two player perfect Information Games which are associated with combination of search & knowledge are,

1) Chess
2) Checkers
3) OTHello.
4) Go
5) Back grond.

1) chess:- (Developed by Green Balt in 1967)

64 Sources, arranged in 8×8.

each has, 1 king, 1 Queen., 2 rooks. , 2 kings.

& pawns Goal is to make. State opponent king: Check & Checkmate.

2) Checkers:- (Developed by Arthur Samual in 1967)

8×8 Squre board.

each player has 12 pieces of 1 Color each and. are placed in that Color in 3 rows

- closest row to player in king row., called kings & others are men.

- kings move diagonally, Fwd, bwd & men. Can move only daigonal Fwd.

- when men peices Jump over king pices They Transform to kings

Goal :- remove all opponent pices on the board.

6. Two-player perfect Information Games.

- Some of the two player perfect information Games which are associated with Combination of search & knowledge. are,

1) Chess
2) Checkers
3) OThello.
4) Go
5) Back ground.

1) Chess:- (Developed by Green Balt in 1967)

64 Sources, arranged in 8×8.

each has, 1 king, 1 Queen., 2 rooks. , 2 kings. , & pawns Goal is to make. State opponent king; Check & Check mate.

2) Checkers:- (Developed by ArThor Samual in 1967)

8×8 SQure board.

each player has 12 pieces & 1 Colour each and. are placed in That Colour in 3 rows

- Closest row to player is king row., called. kings & others are men.
- kings move. diagonally, Fwd, bwd. & men. Can move only daigonal Fwd.
- when men peices Jump over king pices They Transform to kings

3) checkers:- (also called Reversi) 8×8 square grid.
dev by Rosenbloom in 1982) Grid has
& dibtaut & light · Each face represent a player.

Goal:- make more peices of players to have.
of that colour face

4) Go 19×19 board.
each player uses stones either black & white.
Goal is occupy longer part of the Board.

5) Backgamon:- Developed by Leddon's in 1980 playing
pices are moved by using Dice! pkeyer
must choose from numerious Optimum.