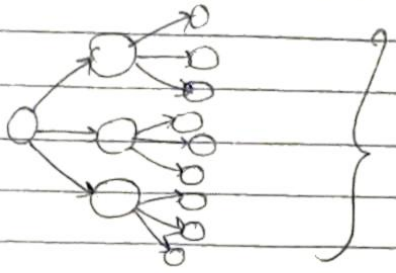
AI (UNIT-2)

(*) Uninformed Search: (AKA Blind Search)

① Search without information

② Brute force Method / Blind searching



We search all the state spaces.
→ It is always complete
→ High cost

③ No knowledge., Time Consuming, More Complexity (both Time & Space)

④ Examples include DFS, BFS, etc.

⑤ Does not use additional information to guide the search process.

⑥ Does not consider the cost of reaching the goal.

⑦ Inefficient for complex problems

⑧ Do not guarantee an optimal solution

(*) Informed Search: (AKA Heuristic Search).

① Search with information. (Information is known as Heuristic).

② Use knowledge to find steps to solution.

③ Quick Solution

④ Less Complexity.

⑤ Examples: A*, Heuristic DFS, Best first search, etc.

⑥ Efficient problem solving. As compared to Uninformed Search Method.

⑦ Goal directed & cost based.

⑧ May guarantee an optimal solution.

⑨ May or may not be complete.

⑩ Low Cost

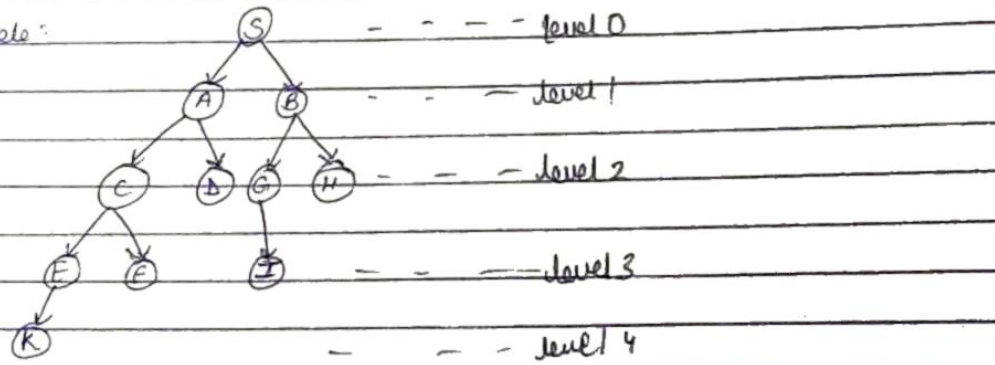
(*) Breadth First Search

① Most common search strategy for traversing a tree or graph.
② Searches breadth wise.
③ Starts searching from root node and expands all successor nodes at the current level before moving to the nodes of next level.
④ Example of General Graph-Search Algorithm.
⑤ Implemented using FIFO Queue Data Structure.

① Advantages:
→ Always provides a solution, if exists.
→ If there are multiple solutions, then it provides the minimal soln which requires least number of steps.

② Disadvantages:
→ Requires a lot of space to save each level of tree.
→ Needs a lot of time if the soln is far from root node.

Example:

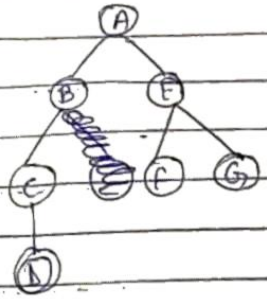


root node → S
Goal node → K.

Solution: S → A → B → C → D → G → H → E → F → I → K

→ Time Complexity can be calculated by the number of nodes traversed until the shallowest node.
→ BFS is complete, if the shallowest node is at some finite depth, then BFS will find a solution.
→ BFS is optimal

Algorithm



Queue	Check
A	Removed A, check if A is goal state No? Add Children of A
BF	Removed B, check if B is goal state No? Add Children of B.
EC	Removed E, check if E is goal state. No, add Children of E
CFG	Removed C, check if C is goal state. No? Add Children of C.
FGD	Removed F
GD	Removed G
D	Removed D. Check if D is goal state Yes!!

Time & Space Complexity

$O(b^d)$

where,
d = depth of shallowest soln.
b = node at every state.

(iii) Depth first Search:

① Recursive algorithm for traversing a tree or graph Data structure.
② Starts from root node and follows each path to its greatest depth node before moving to next path.
③ Uses stack for its implementation.

Advantages:

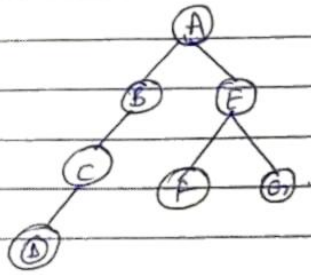
① Requires very less memory.
② Takes less time to reach goal node (than BFS).

Disadvantages:

① Possibility that many states keep reoccuring. No guarantee to find ~~the~~ sol.
② It may sometimes go to the infinite loop.

Flow: Root node → Left node → Right node.

Example:



Stack

A ← top.

Pop A, is it goal? No, push its children on stack.

B ← top E

Pop B, is it goal? No, push its children.

C ← top F

Pop C

D ← top F

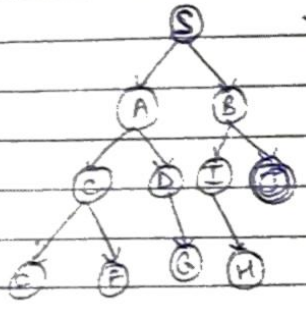
Pop D, is it goal? Yes.

Pop all the contents, path found.

* Depth limited search: Standard and refined version of DFS.
It is to depth first search but with a predetermined limit.
Solves the problem of infinite path in depth first search also.
It can be terminated with two kinds of failure
1. Standard failure value indicates that problem does not have any solution
2. cut off failure value defines no solution for the problem within a given depth limit

Advantage: It is memory efficient.
Disadvantages: → Incomplete.
→ It may not be optimal if the problem has more than 1 solution.

The depth limit is denoted by l.



— level 0
— level 1
— level 2
— level 3

We set the depth limit, then we check if the current node is lying under the depth limit specified or not. If no, then we do nothing, if yes, then we explore the other nodes in the same way.

Depth limit → 2

Start State → (S)
Goal State → (J)

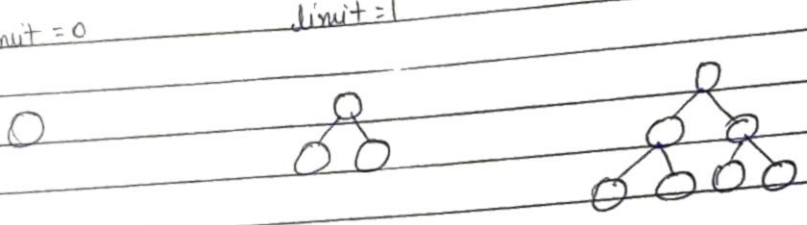
Solution (S) → (A) → (C) → (D) → (B) → (I) → (J)

Time complexity: $O(b^l)$
Space complexity: $O(b \times l)$

(*) Iterative Deepening Search:

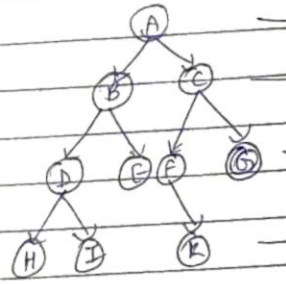
① Combination of DFS and BFS algorithms.
② Finds the best depth limit by gradually increasing the limit until a goal is found.
③ Combines the benefits of DFS's memory efficiency & BFS's fast search.
④ Used when search space is large and depth of goal node is unknown.
⑤ This algo. is complete if Branching factor is finite.
Main disadvantage: Repeats all the work of previous phase.

①



②

③ Example:



Start → (A)
Goal → (G).

1st Iteration → A
2nd Iteration → A B C
3rd Iteration → A B D E C F G
4th Iteration → A B D H I E C F K G

Time complexity: $O(b^d)$ (worst case)
Space Complexity: $O(bd)$

b → branching factor
d → depth

(*) Bidirectional Search Algorithm:

① Runs 2 simultaneous searches. One from initial state called as forward search and one from goal node called as Backward Search

② It replaces one single search graph with 2 small subgraphs in which one starts search from initial vertex and other from goal vertex.

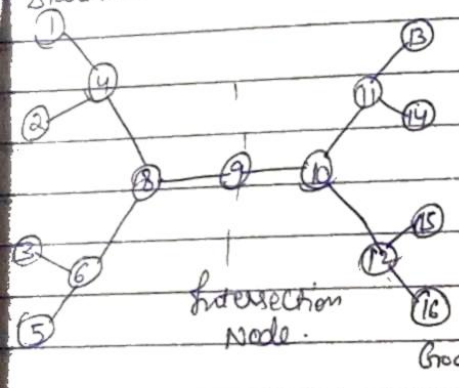
③ Search stops when these 2 graphs intersect each other.

④ Use search techniques like BFS, DFS, DLS, etc.

Advantages: ⓐ It is fast
ⓑ It requires less memory.

Disadvantages ⓐ Implementation is difficult.
ⓑ One should know the goal state in advance.

Example:



→ divides one graph/tree in two subgraphs.
From node 1 → forward search.
From node 16 → Backward search.
→ Terminates at node 9 where two searches meet.

→ It is complete if we use BFS in both searches.
→ It is optimal.

Time complexity: $O(b^d)$
Space complexity: $O(b^d)$

(*) Heuristics function

① Used in informed search. It finds the most promising path
② Takes the current state of agent as I/P and estimates how close the agent is from goal.
③ It might not always give the best soln but guarantees to find a good solution in reasonable time
④ Represented by $h(n)$. It's value is always positive.
⑤ Admissibility is given as: $h(n) <= h^*(n)$

where, $h(n)$ = Heuristic cost
$h^*(n)$ = Estimated cost

GENERATE AND TEST

Algorithm ① Generate a possible soln
② Test if the possible soln is real one, compare with goal state.
③ Check soln, if true return soln. else go to step 1.

(*) Best first Search

① Always selects a path which appears best at that moment
② Combination of DFS and BFS. Uses heuristic function and search.
③ At each step, we can choose the most promising node.
④ Greedy algorithm, implemented by priority queue.

$f(n) = g(n)$

Algorithm

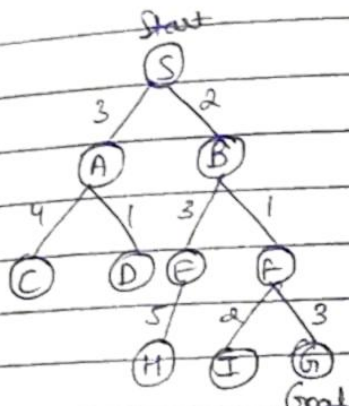
① Place the starting node into OPEN List
② If the OPEN list is empty, stop and return failure.
③ Remove node n from OPEN list which has the lowest cost (lowest value of $h(n)$) and place it in the CLOSED list.
④ Expand the node n and generate successors of node n.
⑤ Check each successor and find whether any node is goal node or not. If goal node found then terminate and return success, else proceed.
⑥ For each successor node, the algo. checks for evaluation function $f(n)$

and then check whether the node is in OPEN or CLOSED list if it is not in both the lists, then add it to OPEN list
③. Repeat Check Return to step 2

Advantages ① Can switch b/w DFS and BFS by gaining advantages of both.
② More efficient than BFS and DFS

Disadvantages ① Can behave as unguided depth first search in worst case.
② It can get stuck in a loop.
③ It is not optimal.

Example:



① S → Open list
S → Closed, AB → Open
SB → Closed, AEF → Open
SBE → Closed, AEFG → Open
SBEG → Closed, AEF → Open

Solution
S → B → F → G

Time complexity: $O(b^m)$ m ⇒ maximum depth of search tree.
Space complexity (Worst) $O(b^m)$

→ Greedy Best first search is incomplete and not optimal.

(*) A* Algorithm

① Most commonly known form of Best first search.
② Has combined features of UCS and Greedy Best first search, by which it can solve the problem efficiently.
③ It finds the shortest path through the search space using Heuristic function.
④ Expands less search tree and provides optimal solutions faster.

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

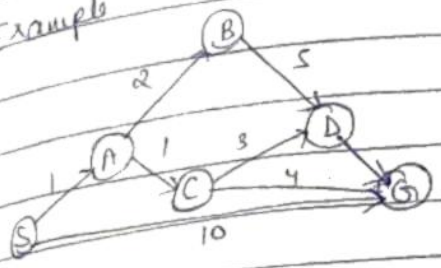
- $f(n)$: estimated cost of cheapest solution.
- $g(n)$: cost to reach node n from start state.
- $h(n)$: cost to reach from node n to goal node

Algo: ① Place the starting node in OPEN list.
② Check if the OPEN list is empty or not. If empty, then return failure
③ Select a node from open list with the least value of (g+h). If it is goal node, then return success
④ Expand node n and generate all successors.
⑤ Find the lowest value.
⑥ Return to Step 2.

Advantages: ① Best algo. than other search algorithms.
② Optimal & complete.
③ Can solve very complex problems.

Disadvantages: ① Does not always produce the shortest path.
② Has some complexity issues.
③ Requires more memory, not practical for various large scale problems.

Example



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

Initialization : $\{(S, S)\}$

Iteration 1 : $\{(S \to A, 4)\ (S \to G, 10)\}$

2 : $\{(S \to A \to B, 7)\ (S \to A \to C, 4)\ (S \to G, 10)\}$

3 : $\{(S \to A \to B \to D, 14)\ (S \to A \to C \to D, 11)\ (S \to A \to C \to G, 6)\ (S \to G, 10)\}$

Here, we get the final result, $S \to A \to C \to G$ provides a path with cost 6 (Optimal path).

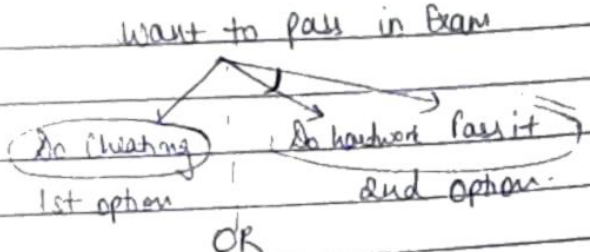
→ It returns the path which occured first and does not check for remaining paths.

① It is complete as long as branching factor is finite.
② It is optimal if it follows 2 conditions. ⓐ. Admissible, ⓑ. Consistency
③. Time Complexity : $O(b^d)$ where b is branching factor.
④. Space Complexity : $O(b^d)$

(*) AO* Algorithm

① Based on AND/OR graph.

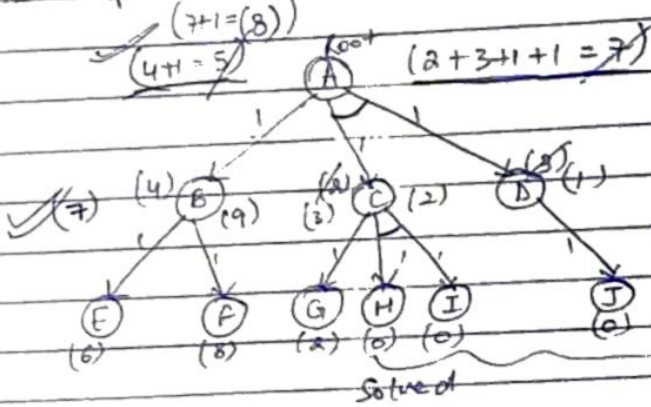
② Works on Problem Decomposition Breaking down a complex problem into pieces & and then find the solution.



→ A* gives optimal solution But AO* does not guarantee an optimal solution.

→ It does not explore all the solution paths once it got a solution.

Example:



Ans: 5

(*) Local Search Algorithms

① Does not focus on path cost. Only focuses on solution state needed to reach goal state.
② Works on Greedy Technique
③. Follows best move
④ Has knowledge of only local domain.
⑤ No Backtracking.
⑥ Uses very little / constant amount of memory. Better space complexity.
⑦ No High quality solution, Incomplete Algorithm.

Different types of local searches: ⓐ Hill Climbing Search
ⓑ Local Beam Search.

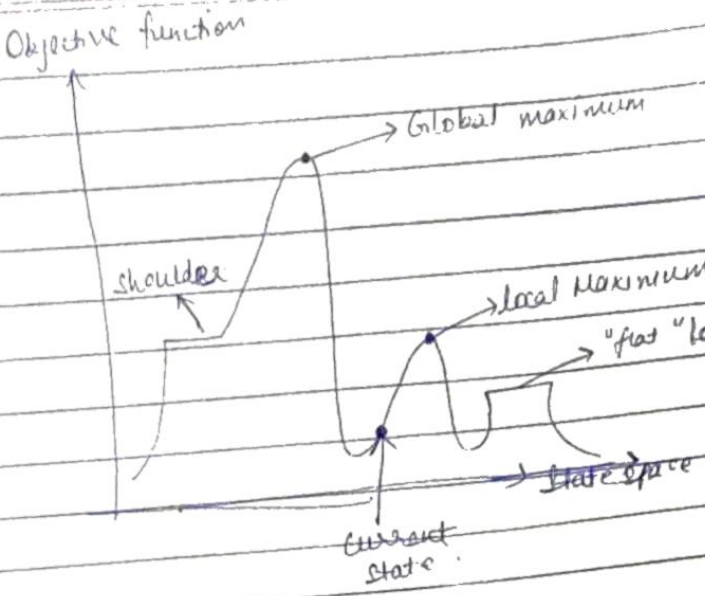
→ The selection of the move to be performed at each step is based on cost function.

(*). Hill Climbing Search:

①. Continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem.
②. Terminates when it reaches a peak where no neighbour has a higher val.
③. Technique used for optimizing mathematical problems.
④ Also known as greedy local search as it only looks to its good immediate neighbour state and not beyond that.
⑤. Mostly used when a good Heuristic value is available.
⑥. We don't need to maintain/handle the search tree or graph.

→ In the State space diagram.

ⓐ If the function on y-axis is cost, then the goal of search is to find global minimum and local minimum.
ⓑ If the function of y-axis is objective function, then the goal of search is to find global maximum and local maximum.



→ Local maximum: State which is better than its neighbours states but there is also another state which is higher than it.

→ Global maximum: Best possible state of state space landscape. It has the highest value of objective function.

→ Current state: State in the landscape diagram where an agent is currently present.

→ Flat local maximum: Flat space in landscape where all the neighbours states have same value

→ Shoulder plateau region which has an uphill edge.

Types.

- Simple Hill Climbing
- Steepest Ascent Hill Climbing.
- Stochastic Hill Climbing.

→ Simple Hill Climbing:

① Simplest way to implement Hill Climbing Algorithm
② Only evaluates the neighbour node state at a time
③ It only checks its one successor state and if it finds better than the current state, then move else be in the same state.
④ Less time consuming. Less optimal. Solution is not guaranteed

Algorithm

① Evaluate the initial state. If GOAL → Quit
② Loop until solution is found or no new operators are left to be applied on current state.
   (a) Select and apply operator to produce next state
   (b) evaluate new state.
      (i) if GOAL → QUIT
      (ii) if it is better than current state, then assign it as current
      (iii) if not better, then continue in the loop.
③ Exit.

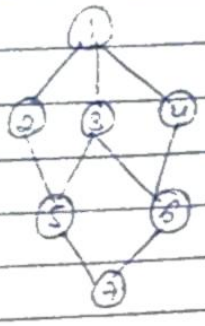
→ Steepest Ascent Hill Climbing:

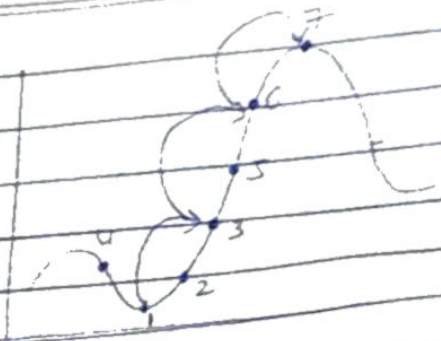
① Variation of Simple Hill Climbing Algorithm
② Examines all the neighbouring nodes of current state and selects one neighbour node that is closest to goal state.
③ Consumes more time as it searches for multiple neighbours.
④ Optimal Solution

Algorithm

① Evaluate Initial State If GOAL → QUIT else Initial State : Current State

eg:





① 1 → successors → {2, 3, 4}
↑
Best (Jump from 1 to 3).

② 3 → Successors → {5, 6}
↑
Best (Jump to 6)

Takes bigger steps.

→ Stochastic Hill Climbing

① It does not examine all the neighbour nodes before, it select one neighbour at random and decides whether to choose it as current state or examine another state.

Problems in Hill Climbing algorithm.

① Local Maximum Solution is backtracking (create a list of promising paths so that the other paths can also be explored.

② Plateau Solution: Take big steps and Randomly select a state which is far away from the current state to find non-plateau region.

③ Ridges: It has an area which is higher than its surrounding areas but itself has a slope and cannot be reached in a single move.

Ridge

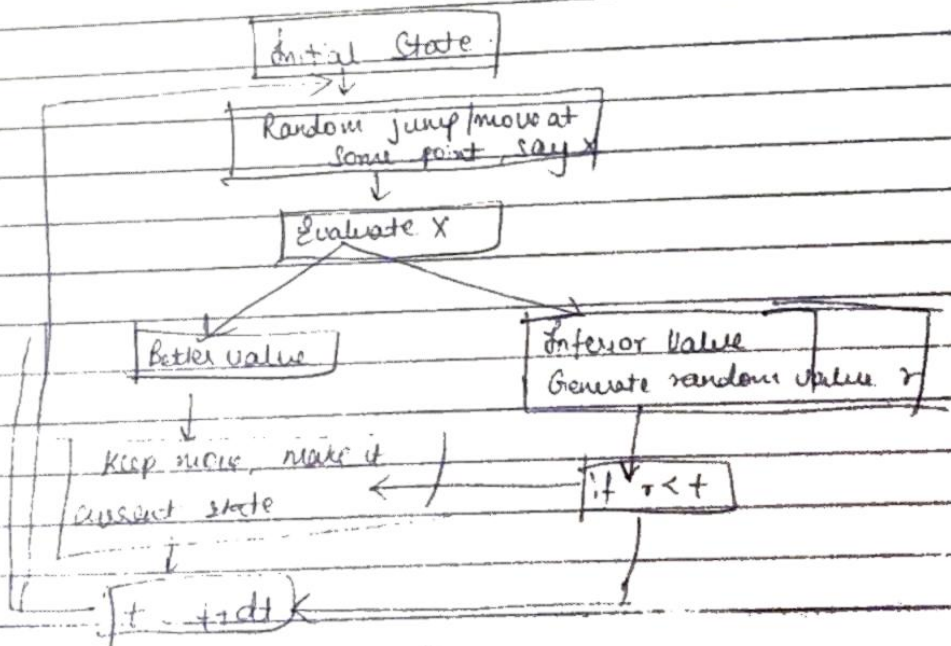
Solution: Use of bidirectional search

(b) SIMULATED ANNEALING:

① A Hill climbing Algorithm which never makes a move towards lower value is guaranteed to be incomplete b/c it may get stuck at local maximum.
Simulated Annealing is an algorithm which yields both, efficiency and completeness.

② The algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path otherwise it moves downhill and chooses another path.

③ Compared to hill climbing, the difference is that it allows downwards steps.



Basic Steps.

(x) Local Beam Search:

① Developed in attempt to achieve optimal solution without consuming too much memory.

② Used in many machine translation systems.

③ It is a Heuristic search Algorithm Eg N-Queens.

④ This class of Problems include:

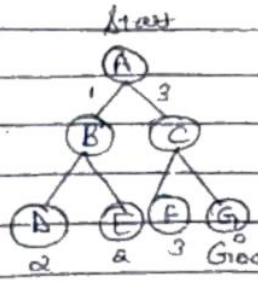
→ Machine Translation:
→ Job Scheduling
→ Vehicle Routing
→ Travelling Salesman Problem
→ Network Optimization
→ Factory floor layout.
→ Integrated Circuit Design.

Machine Translation. Many different ways of translating the words appear. The top best translations according to sentence structures are kept. Rest are discarded.
The translator then evaluates the translations and chooses the translation which best keeps the goals.

→ Beam search is an optimization of Best first search that reduces Memory Requirements.
→ It is incomplete. A more accurate heuristic function and a larger beam width can improve Beam Search's chances of finding the goal.

When $\beta = 2$: It means that we will take 2 best states.

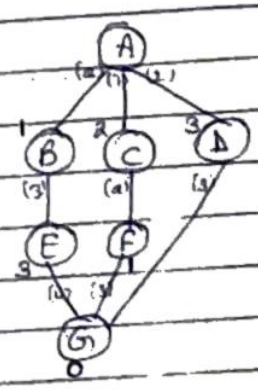
Example: Given β = 2



① OPEN → A
② OPEN → {B, C} CLOSED → A
③ OPEN → {D, E} CLOSED → {A, B}
↑
C will be removed because we will keep best 2 states

In the given example, open set becomes empty without finding the goal node.
With β = 3, the algo succeeds to find the Goal Node.

→ This algorithm is not guaranteed to be optimal.



Example: β = 2

① Open → {A}
② Open → {B, C} Closed → {A}
③ Open → {E, C}
④ Open → {F, E}
⑤ Open → {E, G}

Path found. Path → A → C → F → G
Optimal Path → A → D → G (Can be found by A*)

Time complexity depends upon accuracy of heuristic function.

Worst case Time Complexity: O(β* m), Space complexity O(βm)
Best case constant

where, β → Beam width
m → maximum depth of any path in the search tree.

(*) Genetic Algorithm

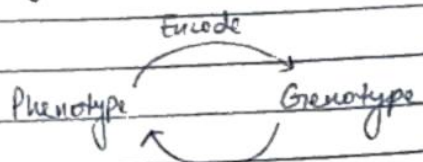
① Abstraction of Real Biological Evolution.

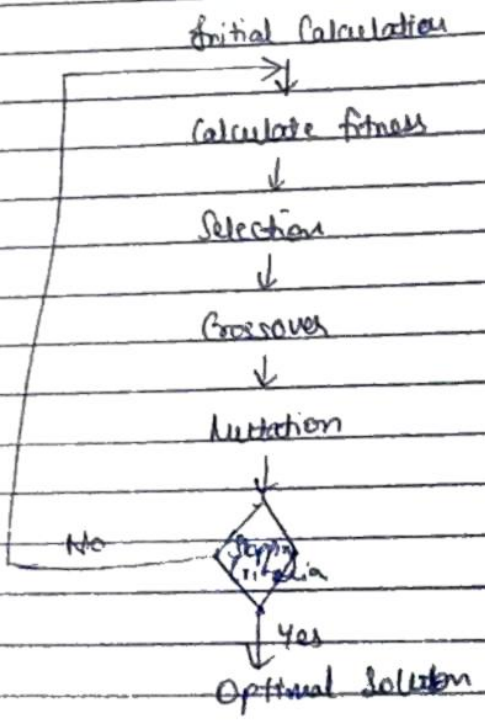
② Focuses on Optimization.

③ Solves complex problems. (like NP hard problems)

④ Search Space is large.

⑤ From a group of individuals, the best will survive.

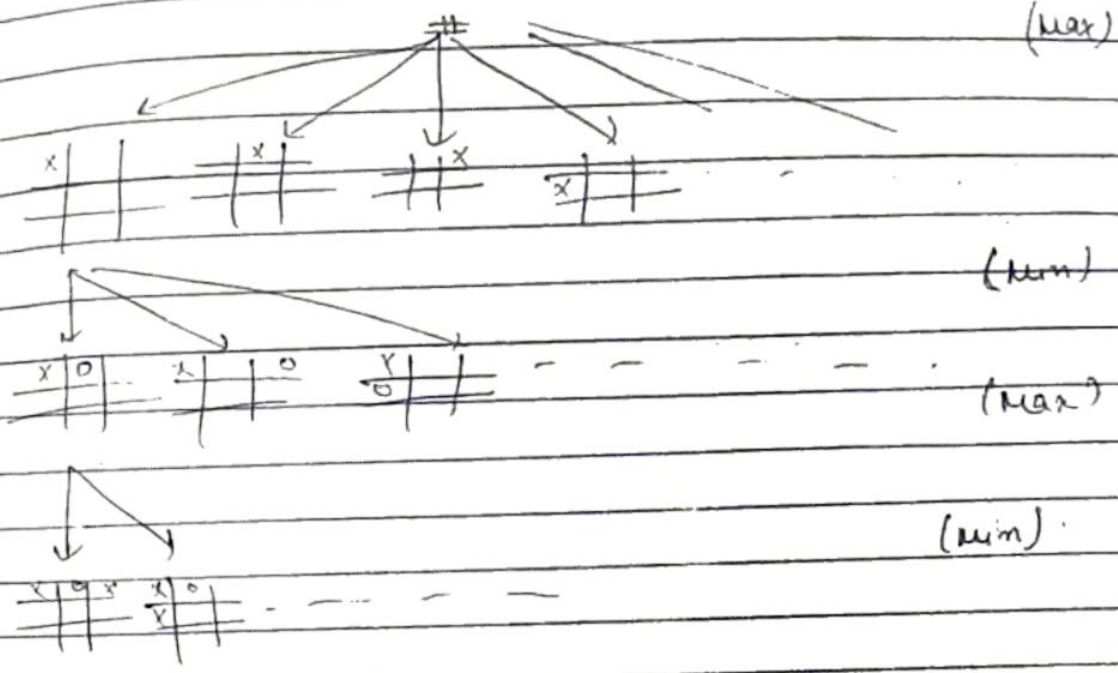




Genetic Operators: ① Selection operator
② Mutation
③ Crossover

(*). Game Playing Algorithm:

Eg: tic tac toe.



Game playing is a search problem defined by:
ⓐ Initial State
ⓑ Successor function
ⓒ Goal test
ⓓ path cost / utility

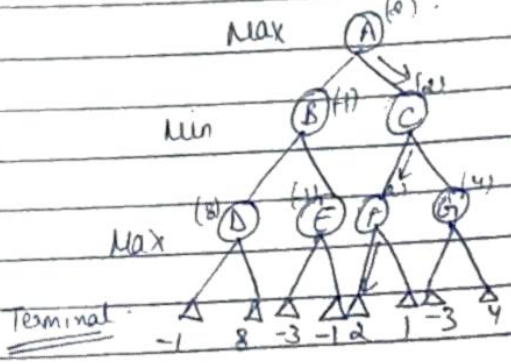
→ Works on minimax algorithm.

(*) Minimax Algorithm:

① It is a back-tracking Algorithm.

② Best move strategy is used

③ 2 players Max and Min Max will try to maximize the utility and Min will try to minimize Max's utility so that Max loses.



Find the best path for player → Max.

Game Tree.

Time complexity: $O(b^d)$ where b → Branching factor.
d → depth.

(*) Alpha - Beta Pruning:

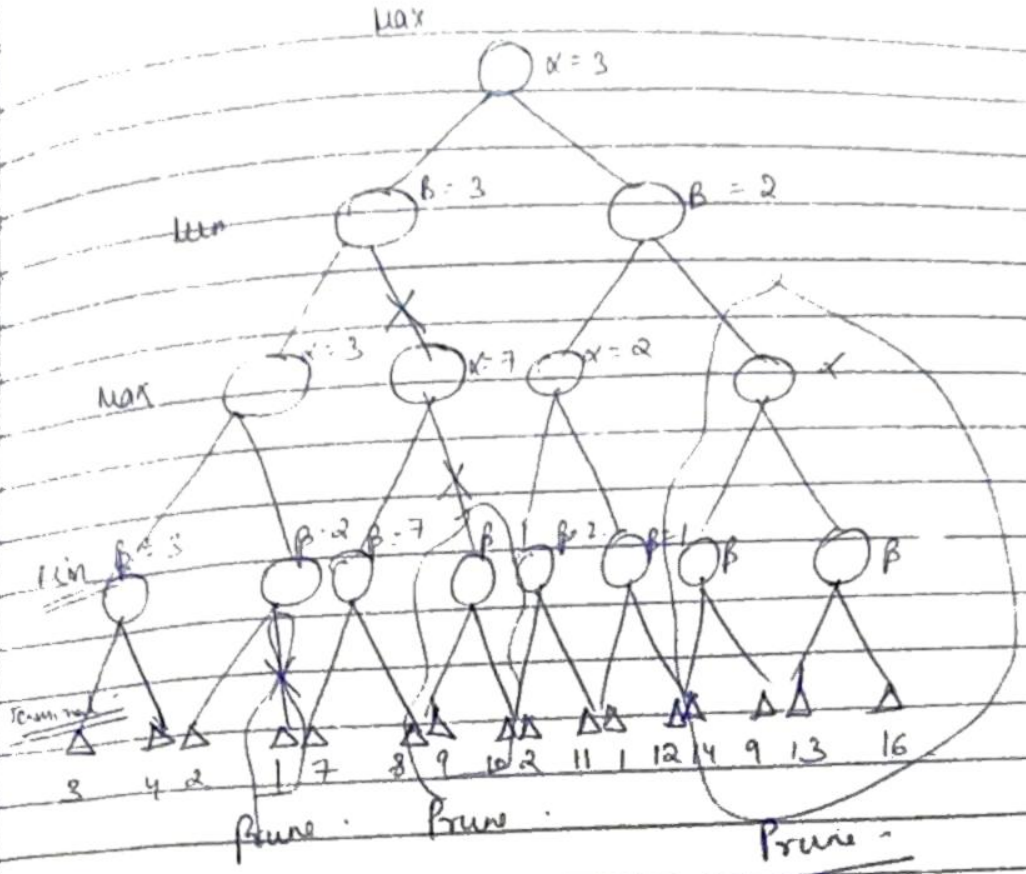
① If we have already found the best path then we will remove rest of the paths

α → value of node "Max"

β → value of node "Min"

Complexity is best case: $O(b^{d/2})$ / average case.

worst case: $O(b^d)$


Max
α = 3
Min
β = 3
β = 2
Max
α = 3
α = 7
α = 2
α
Min
β = 3
β = 2
β = 7
β
β = 2
β = 1
β
β
Terminal nodes
3 4 2 1 7 8 9 10 2 11 1 12 14 9 13 16
Prune
Prune
Prune