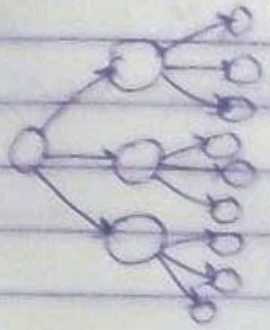
AI (UNIT-2)

* 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

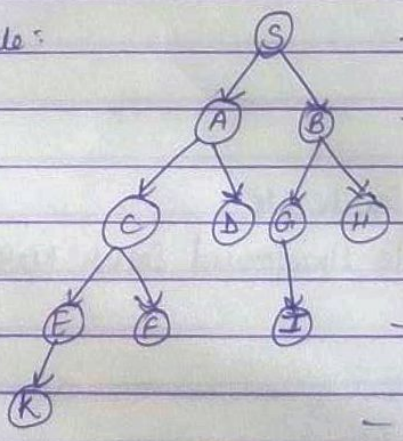
(*) Breadth first search:

① Most common search strategy for traversing a tree or graph.
② Searches breadthwise.
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.

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

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:

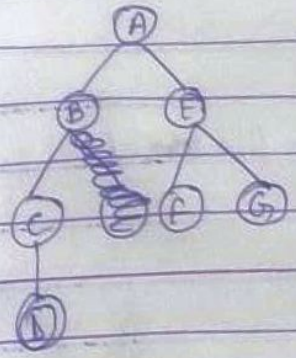


root node → S
Goal node → K.

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.

(*) 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:

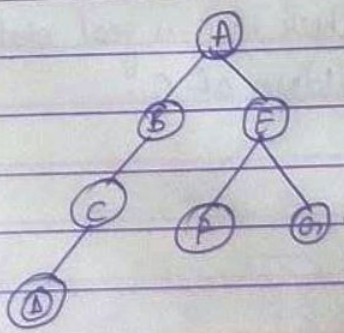
① Requires very less memory.
② Takes less time to reach goal node (than BFS).

Disadvantages:

① Possibility that many states keep reoccusing. No guarantee to find 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.

(x) Depth Limited Search: (Extended and refined version of DFS)

① Similar to Depth first search but with a predetermined limit. Solves the problem of infinite path in Depth first search algo.

② It can be terminated with two kinds of failure.

(a). Standard failure value: Indicates that problem does not have any solution.

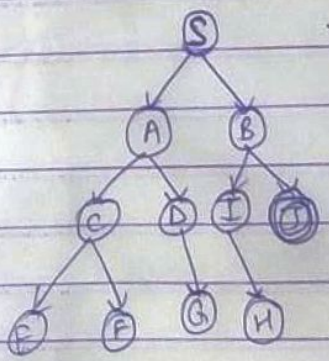
(b). Cut off failure value: defines no solution for the problem within a given depth limit.

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 → (I)

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

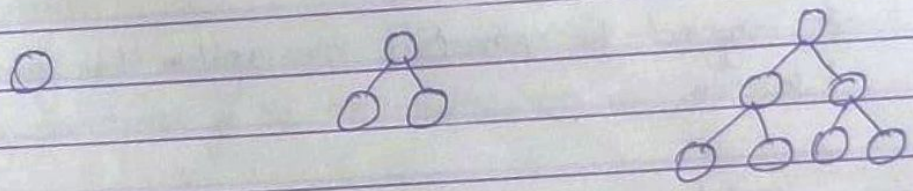
Time complexity: $O(b^l)$

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

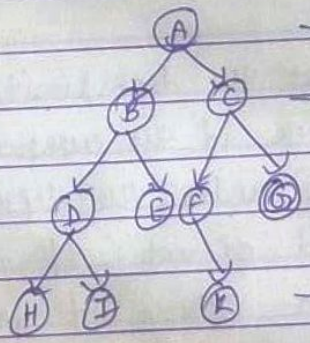
(x) Iterative Deepening Search:

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.

limit = 0 limit = 1 limit = 2



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.

(x) 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.

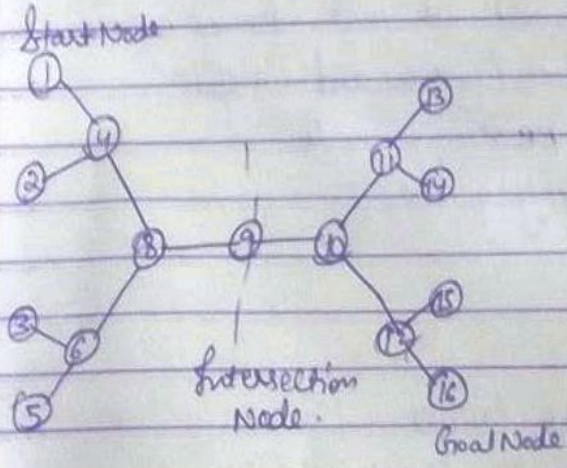
③ Search stops when these 2 graphs intersect each other.

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

Advantages: ① It is fast
② It requires less memory.

Disadvantages: ① Implementation is difficult
② 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). Its 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)

Best first Search

↳ selects the path which is best
↳ DFS + BFS combination
↳ uses Heuristic H(n) & search
↳ at each step, most promising node can be used.
↳ Greedy algorithm by priority queue
$[f(n) = g(n)]$
↳ informed

Example

Suppose, we have the graph as follows:

→ Advantage

- Can switch DFS & BFS
- More efficient

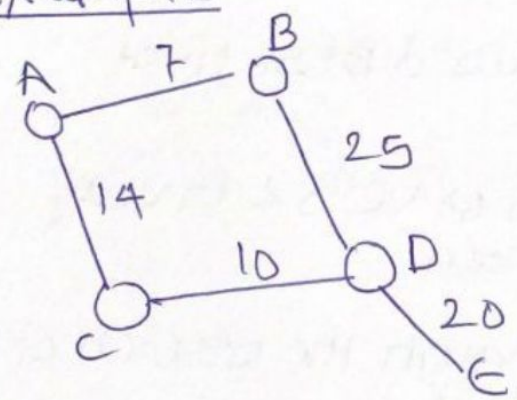
→ Disadvantage

→ behaves as unguided

→ Not optimal

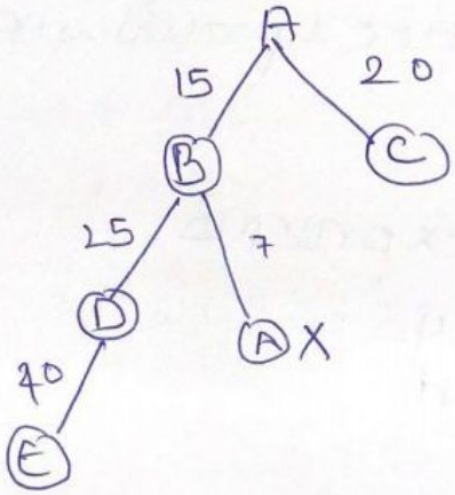
$O(b^m)$ → m depth

Example



Heuristic Values

A → E	10
B → E	15
C → E	20
D → E	40
E → E	0



Path followed

A → B → D → E

A* Algorithm

→ Most commonly used Best first search

→ Combined features of UCS & Greedy Best First Search

→ Shortest path through the search space using Heuristic

→ expands less nodes & provides optimal solution

→ complete

→ solve complex problems

→ does not always produce the shortest path

→ more memory.

→ $f(N) = g(N) + h(N)$

$g(N)$ → Actual cost

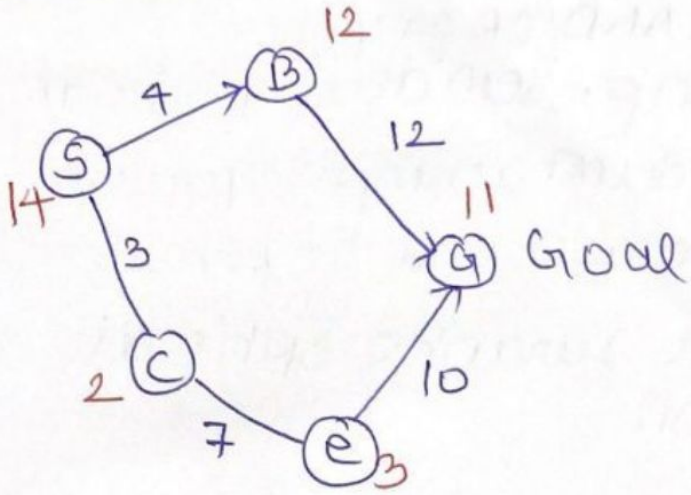
$h(N)$ → Estimation cost

→ $TC = O(V + E)$

$= O(b^d)$

$SC = O(b^d)$

Example



$f(D) = g(D) + h(S)$

$= 0 + 14 = 14$

$S \rightarrow B$

$4 + 12 = 16$

$S \rightarrow C$

$3 + 5 = 8$

$SC \rightarrow e$

$10 + 3 = 13$

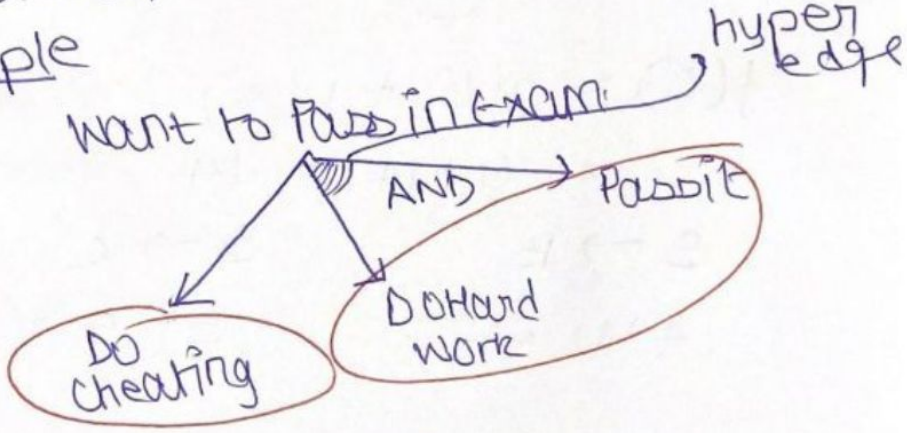
$SCe \rightarrow G$

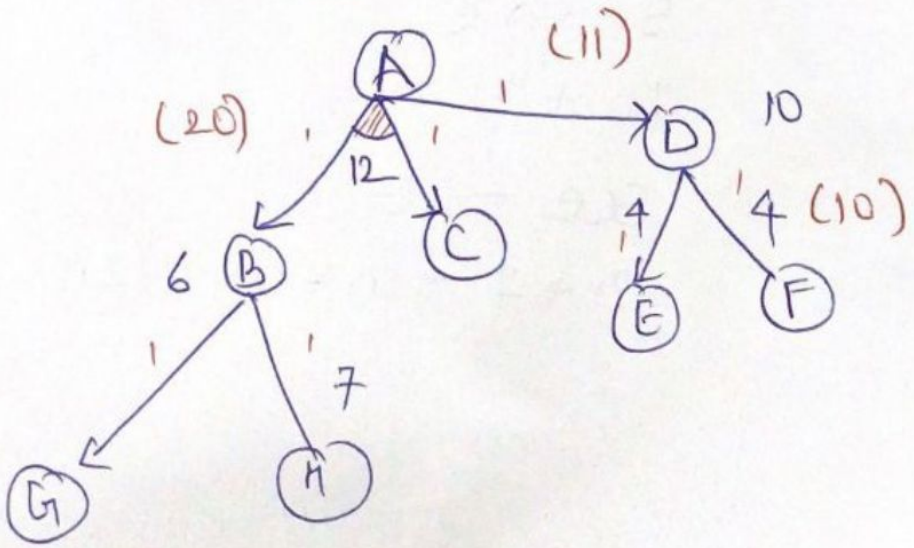
$3 + 7 + 10 + 0 = 20$

AO* Algorithm

→ Based on AND/OR graph
→ works on problem decomposition
→ breaks down a complex problem
→ does not explore all the paths
→ does not guarantee optimal solution

Example





Local Search Algorithm

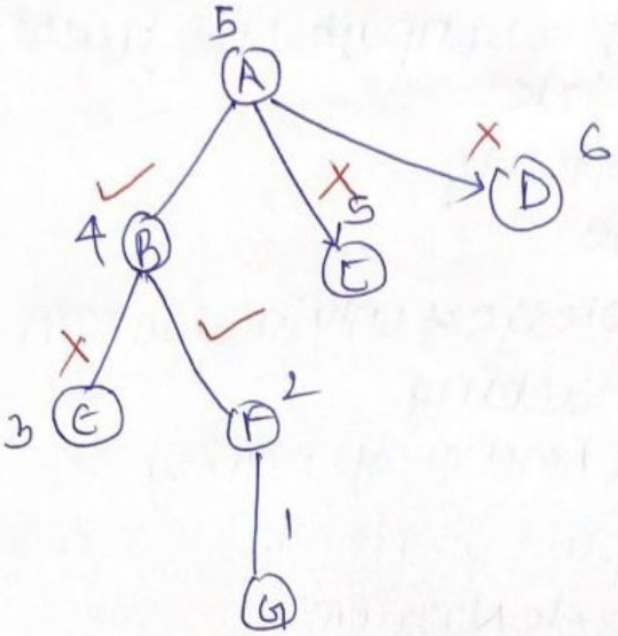
a) does not focus on path, only focuses on solution state
b) works on greedy
c) best move
d) Has knowledge of only local domain
e) No Backtracking
f) constant Amount of memory
g) Better space complexity
h) Incomplete Algorithm

Types — Simple

Hill Climbing Search

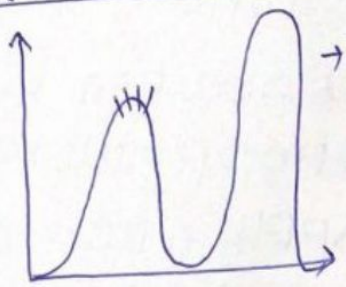
a) evaluate the initial state
b) Loop until a solution is found or there are no operators left
c) select & APPLY a new operator
d) Evaluate new state
   → if goal then quit
   → if better than current state then it is new state.

Example



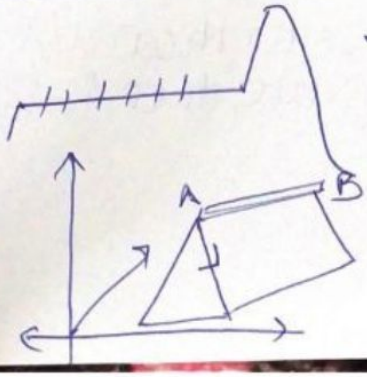
Problems in Hill climbing

a) Local Maximum



→ Local or non better value

b) Plateau/Flat Maximum



→ similar value

c) Ridge

→ one direction

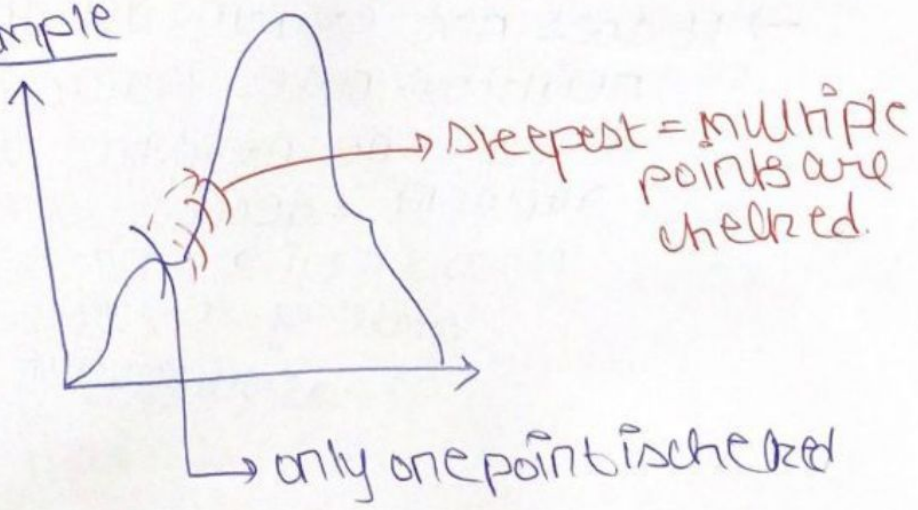
Steepest Ascent Hill climbing

→ optimal solution
→ Time consuming.
→ In steepest Ascent Hill climbing multiple points are checked
→ Selects the best among the children states that are better than the current state.

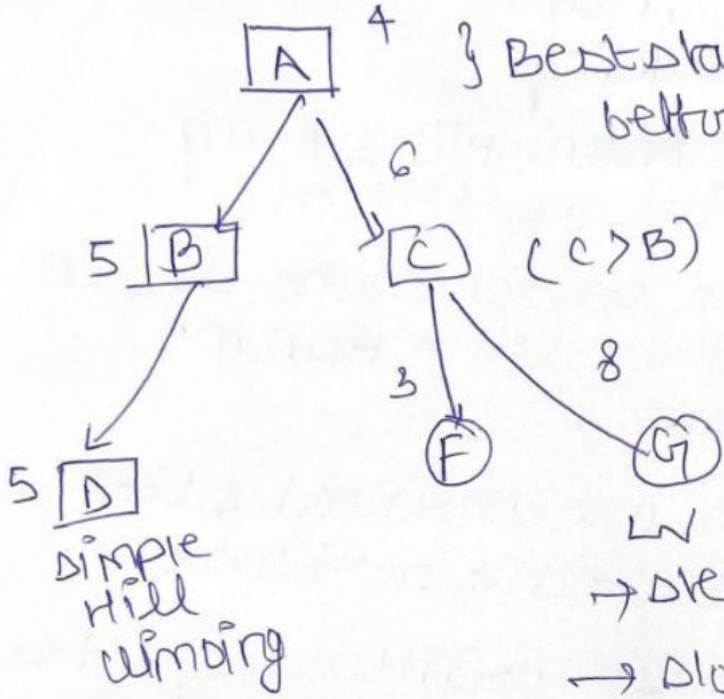
↳ All moves are considered & best one is selected as next state

↳ examines all neighbouring nodes and selects nodes closest as the next node

Example



Example



} Best state is node with better heuristic value (>)

Simple Hill climbing

LV

→ Steepest Ascent

→ Slower than Simple Hill

Stochastic Hill climbing

→ It does not examine all the neighbour nodes. Instead it selects one neighbour at random & decides whether to choose it as the current state or examine another state.

Simulated Annealing

a) Simulated Annealing allowed downward.

b) whereas, downwards are not allowed in the simple hill climbing.

c) Easy to code for complex problems also

d) Gives good solution to a problem

e) statiscally gwarntees finding

f) Slow Process

g) can't tell whether an optimal solution is found (some other method is required.

h) Annealing schedule is maintained. whereas, it is not done in the Hill climbing.

i) Moves to worst state may be expected

j) Best state found so far is maintained.

Local Beam Search

→ Take care of space complexity (constant)

→ Beam width is given (B)

→ Heuristic Search Algorithm

→ optimized version of best first search

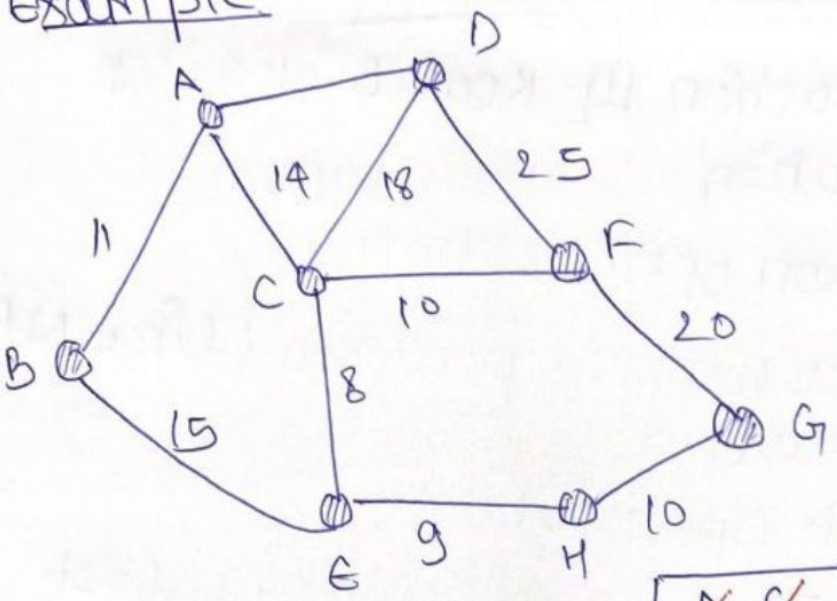
→ Greedy Algorithm use.

→ explores a graph by expanding the most promising node in a limited set.

→ B is predetermined no of best partial solution are kept as candidates

→ Class of problems includes in Machine translation, job scheduling, vehicle Routing

Example

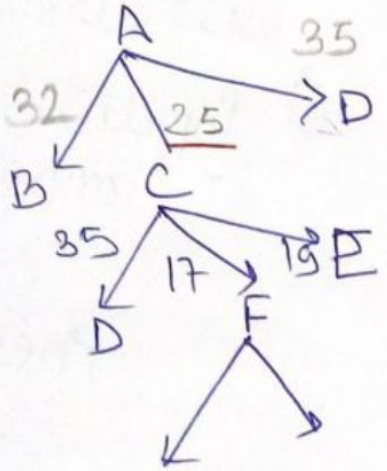


Sorting

~~A~~ ~~C~~ B D F E

Heuristic values

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

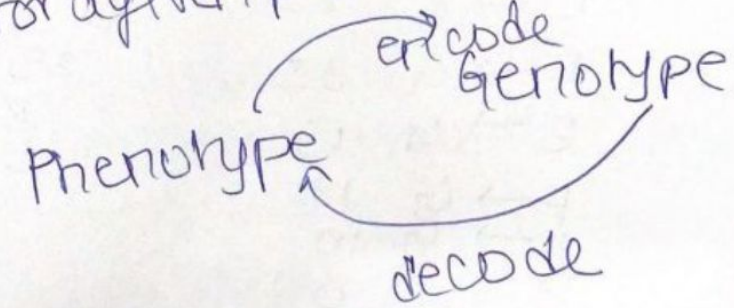


$\beta = 2$ (keep any best two & remove others).

$O(b^d)$.

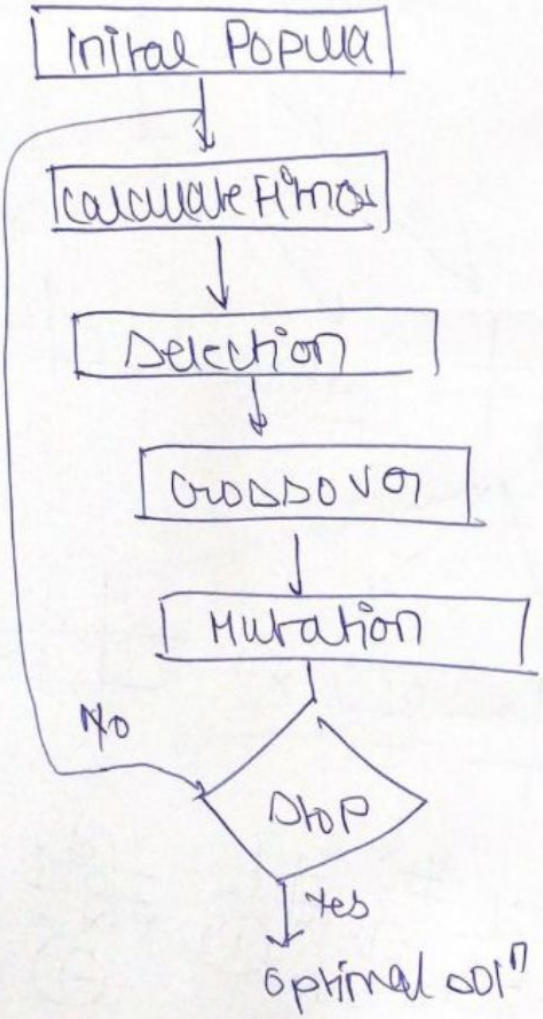
Genetic Algorithm

1) Abstraction of Real Biological Evolution
2) Famous optimization
3) Solves complex problems (like NP hard)
4) Search Space is large
5) From a group of individuals best will survive.
6) Population of possible solution for a given problem.



Genetic operators

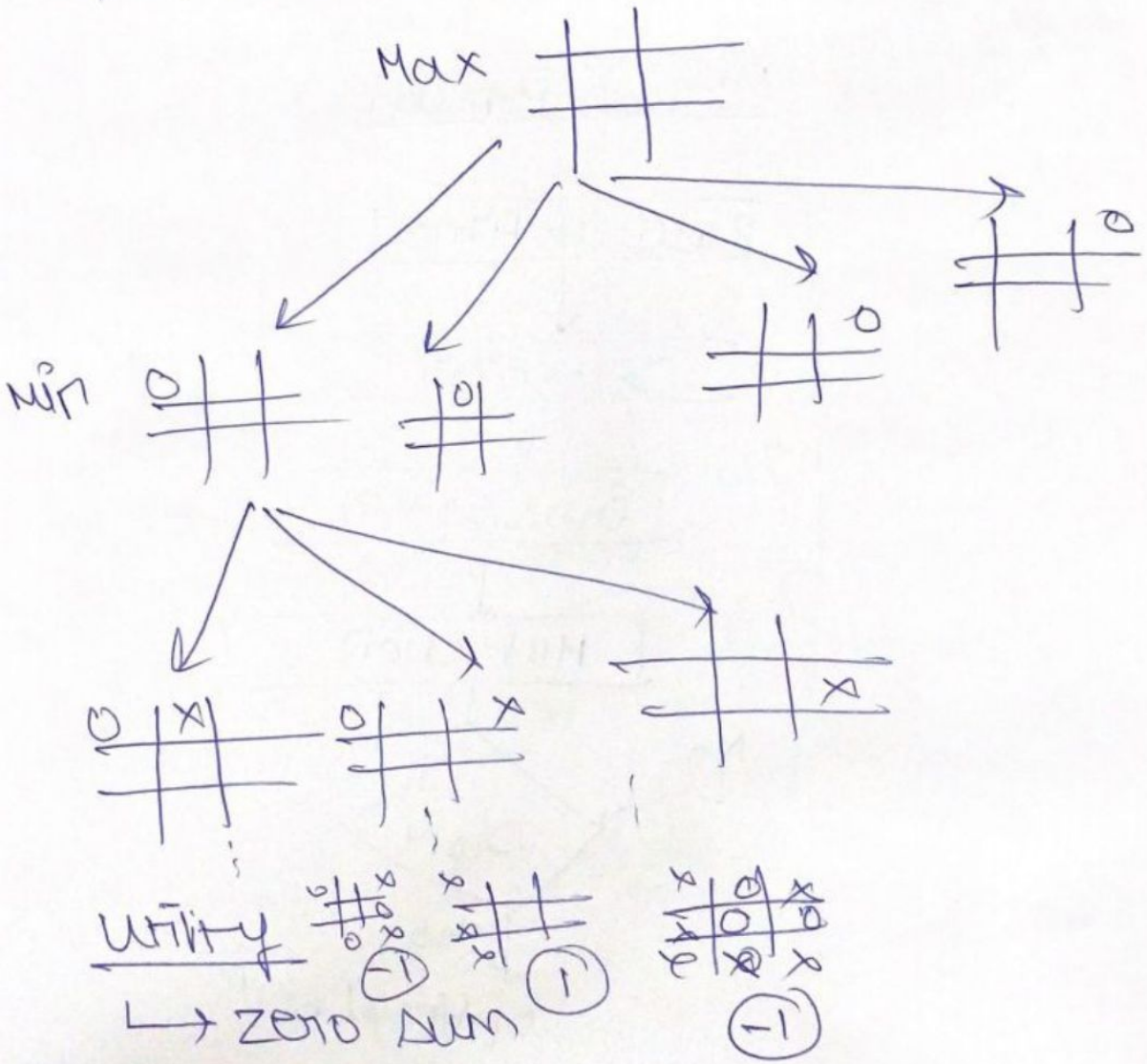
1) Selection
2) Mutation
3) Crossover.



Game Playing Algorithm

a) Minimax Algorithm

b) Alphabeta (α-β) Pruning.

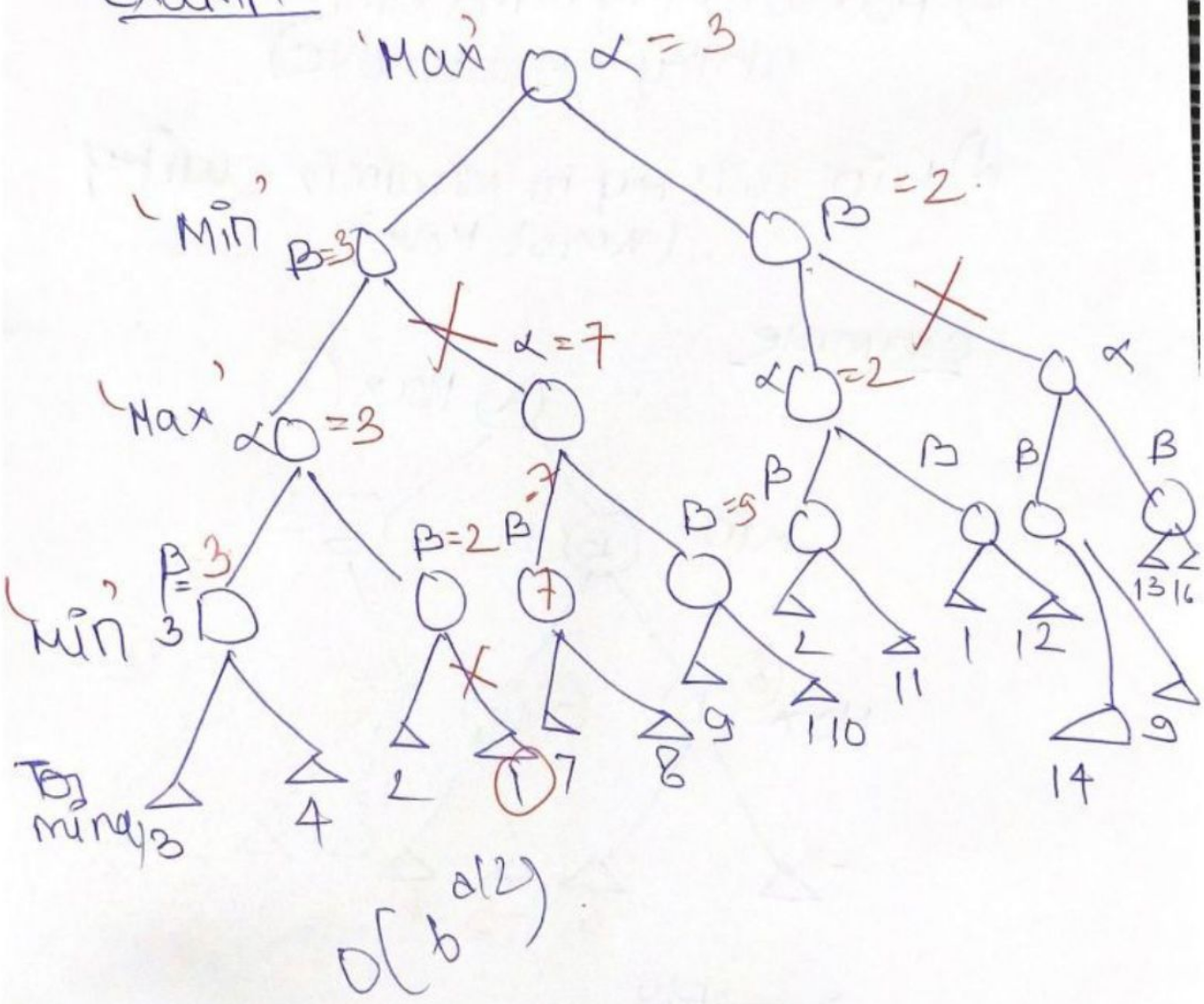


→ Zero Sum

Alpha Beta Pruning

→ cut off search by exploring less no of nodes.

Example

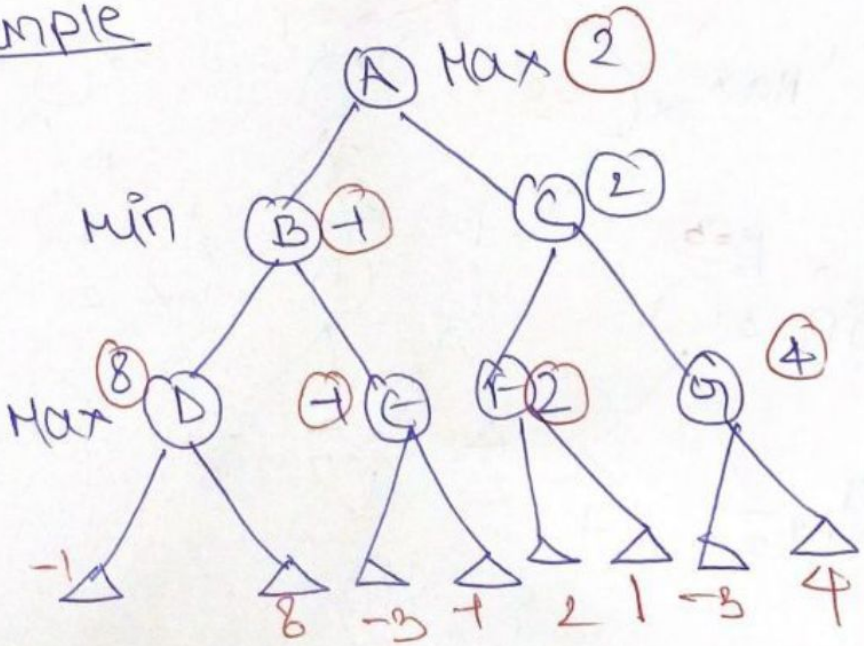


$O(b^{d/2})$

Min-Max Algorithm

a) Back tracking Algorithm
b) Best move strategy used
c) Max will try to maximize its utility (Best move)
d) Min will try to minimize utility (worst move)

Example



$O(b^d)$ → ply.

b → branching factor