* Artificial Intelligence *

⇒

State-space search :-

→ A problem can be defined formally by 5 components :

{Initial state, Actions(s), Result(s,a), Goal test, Path cost function}

* Search strategies :-

- **Completeness :** Determines if the search technique guarantees finding a solution if one exists within the search space.
- **Optimally :-** Determines wheather the search technique can find the best or optimal soln among all possible solutions.

* Uniformed searching :-

- Search without Information
- No knowledge about the solution.
- Time Consuming
- More Complexity (Time / space)

Exa – DFS, BFS ~~etc~~. UCS etc.

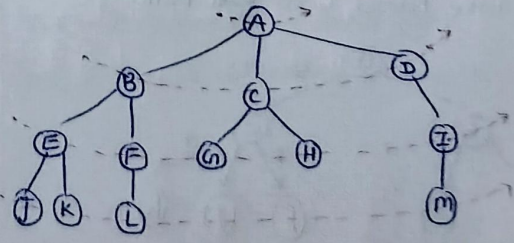
* Informed searching :-

- Search With Information
- Use knowledge to find steps to solution.
- Quick solution.
- Less Complexity.

Exa – A*, Heuristic DFS, Best first search

* BFS (Breath first search) :-

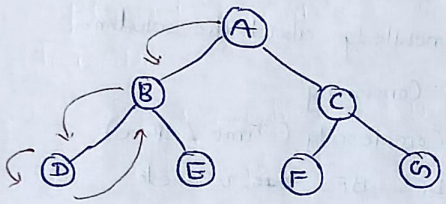
- Uninformed search technique
- FIFO (Queue)



→ Level order Traversal
→ Shallowest Node
→ Complete
→ optimal
→ Time - Complexity - $O(b^d)$
  { b - branching factor
  { d - depth of shallowest ~~Node~~ Goal Node.
→ Space - Complexity - $O(b^d)$, It Stores all the nodes at each level in the search tree in memory.

* DFS :-

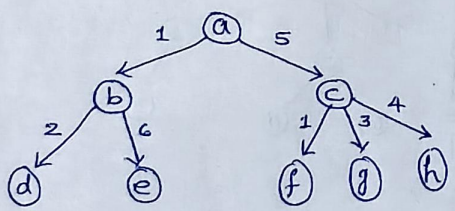
→ Uninformed search techanique
→ Stack ds based.
→ ~~Bac~~ Getting deepest most Node. Then Backtrack



→ Incomplete
→ Not optimal
→ Time Complexity - $O(b^m)$
  m - maximum depth of search tree
→ Space Complexity - $O(bm)$

* Deapth Limited search :- (UCS)

→ Extension of BFS.
→ It Calculate cost to reach each Node and take Lowest cost ~~Post~~ Path.



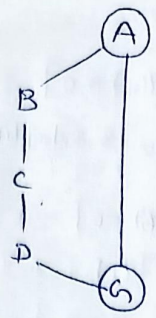
⇒ a → b → d → c → f → e → g → h
→ Implement by using Priority queue
→ Compte - Yes (if search space is finite)
→ optimal
→ Time Complexity - $O(b^d)$
→ Space Complexity - $O(b^d)$

* Bi-directional search :-

→ Two Simultaneous search from an initial node to Goal and backward from goal to initial, stopping when two meet
→ Complete - Yes in BFS, No in DFS
→ optimal -
→ Time - Complexity - $O(b^{d/2})$
→ Space - Complexity - $O(b^{d/2})$

* Iterative deepening search :-

→ depth Limited search
→ for depth ← 0 to ∞
  res ← DEPTH_Limited_Search(problem, depth)
  if result ≠ cutoff
    then return result
→ Time - Complexity -
$T.C = (d+1)b^0 + db^1 + (d-1)b^2 + \dots + b^d$
$= O(b^d)$
→ Space - Complexity -
$S.C = O(bd)$
→ overhead = 11%.
→ optimal - yes, if step cost = 1



BFS : A B G

DFS : A B A B A B ---

IDS : (A), (A B G)

* Informed search Techniques :-

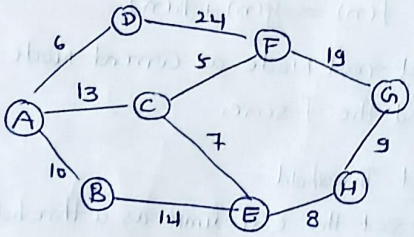
- **Best First search :-**

→ use an Evaluation function f(n) for node (n).

→ Always choose the node from fringe that has Lowest f value.

heuristic function –

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



A → 39 F → 16

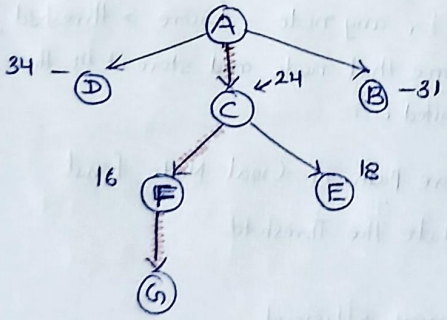
B → 31 H → 9

C → 24 G → 0

D → 34

E → 18

↳ h(n) = straight line dist.



→ Complete :- No (get stuck in Loops)

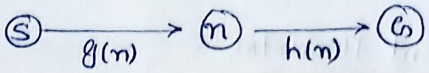
→ optimal :- No

→ T.C :- $O(b^m)$

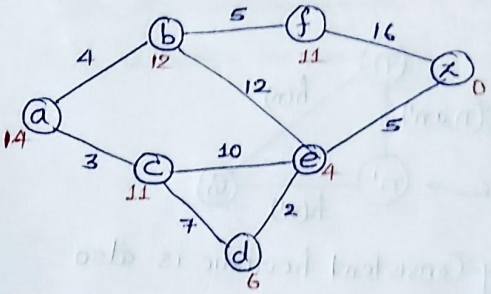
→ S.C :- $O(b^m)$

- **A* search :-**

→ Combines the advantages of both UCS and Greedy Best-First search.



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



$f(b) = g(b) + h(b) = 16$

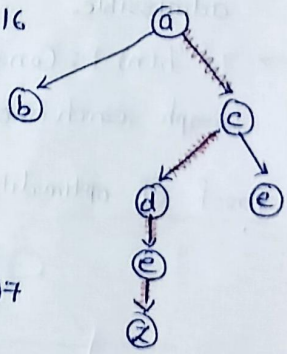
$f(c) = g(c) + h(c) = 14$

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

$= 10 + 6 = 16$

$f(e) = 13 + 4 = 17$

Cost = 3 + 7 + 2 + 5 = 17



→ Complete :-

Yes, if search space is finite and the heuristic fn is admissible.

→ optimal :-

Yes, if h(n) is admissible and consistent

→ T.C – Exponential (Worst Case)

→ S.C – Keeps all nodes in the memory.

* Admissible heuristics:-

→ h(n) is admissible if for every node n,

$h(n) \leq h^*(n)$ → (under estimation)

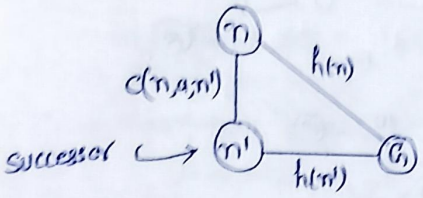
where $h^*(n)$ is the true cost to reach the goal state from n.

→ If h(n) is admissible, A* using TREE-SEARCH is optimal.

* Consistent Heuristics:-

→ h(n) is Consistent if -

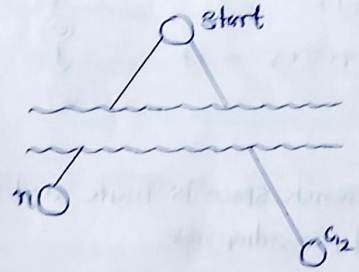
$h(n) <= c(n,a,n') + h(n')$



→ Every Consistent heuristic is also admissible.

→ If h(n) is Consistent, A* using Graph search is optimal.

* Proof of optimality of (Tree) A* :-



G1

→ Say some sub-optimal goal state G_2 has, n be an unexpanded state such that n is an optimal path to the optimal goal G_1.

Focus on G_2 :

$f(G_2) = g(G_2)$ $[\because h(G_2) = 0]$

$g(G_2) > g(G_1)$ [Since G_2 is suboptimal]

G_1: $f(G_1) = g(G_1)$ $[\because h(G_1) = 0]$

$f(G_2) > f(G_1)$ —— (1)

∵ h is admissible -

$h(n) \leq h^*(n)$

$g(n) + h(n) \leq g(n) + h^*(n)$

$f(n) \leq f(G_1)$ —— (2)

$f(G_2) > f(n)$ and A* will never select G_2 for expansion.

Explained (2)

* Iterative deepening A* search Algorithm:-

F-score -

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

S1:- Set root Node as Current Node and find the f-score.

S2:- Set Threshold -

↳ set the cost limit as a threshold for a node i.e max f-score allowed for that node

S3:- Expand the Current Node to its children and find fscore.

S4:- Pruning -

↳ if for any node - fscore > threshold, Prune that node and store it in the visited List.

S5:- Return path if Goal Node found

S6:- Update the Threshold

⟹ memory Efficient

* Weighted A* :-

- Prioritizes speed over optimality.

→ $f(n) = g(n) + W \cdot h(n)$

if $W=1$ ⇒ We get standard A*

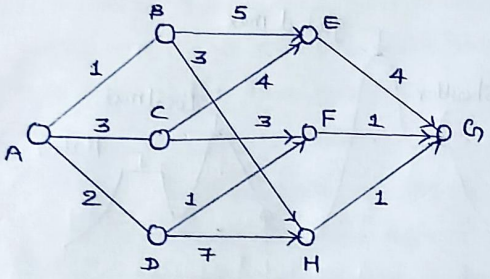
if $W>1$ ⇒ Faster but less optimal.

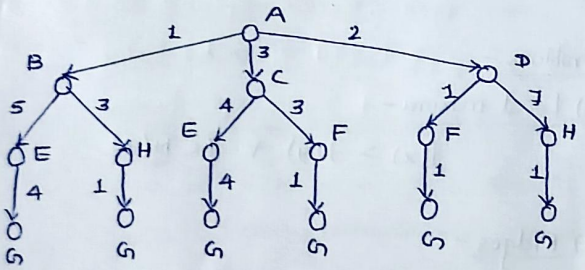
if $W >> 1$ ⇒ becomes more likely Greedy Best first search.

* Depth First Branch and Bound :-

Branch – mechanism to generate branches when searching the solution space.

Bound – mechanism to generate a bound so too many branches can be terminated.





for ABHG → bound = 5

for ADFG → bound = 4

8- Puzzle Problem -

2	8	3
1	6	4
7		5

→

1	2	3
4	5	6
7	8	

$h_1(n)$ = Number of misplace tiles

$h_2(n)$ = total manhattan distance

$h_1(n) = 6$

$h_2(n) = 1+1+0+2+2+1+0+2 = 9$

if $h_2(n) \geq h_1(n)$ for all n (both Admissible)

then h_2 dominates h_1.

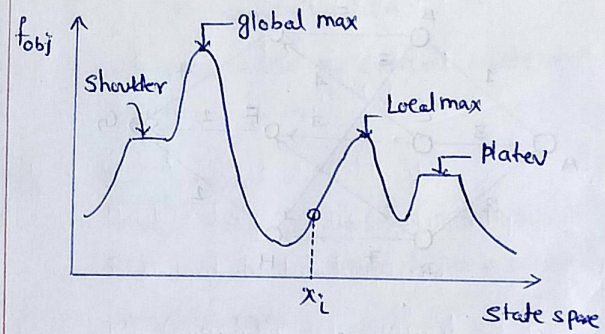
h_2 is better for search.

* Hill climbing Algorithm :-

↳ Greedy local search

Algorithm –

1) Start with an initial state (random or Predefined)
2) Loop until a stopping condition is met :
   a) Evaluate current state using h(n)
   b) Generate the neighboring states of the current state.
   c) Select the best neighboring state with the highest h(n).
   d) if selected neighbor is better than the current state, move to it.
   e) if no better neighbor exists, terminate the Algorithm.
3) Return the final state as the solution.

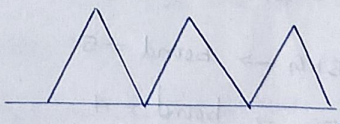


Problems –

(i) **Local maxima -**

$$f(x) \geq f(y) \quad \forall \quad y \in N(x)$$

(ii) **Ridges -**



Ridges result in a sequence of local maxima that is very difficult ~~to~~ for greedy Algo to navigate.

(iii) Plateau :-

→ This is a flat local maxima from which no uphill exit exists, or a shoulder from which progress is possible.

* Random Restart hill Climbing :-

⇒ To avoid local optima, you can perform multiple runs of the algorithm from different random initial states.

• Tabu search :-

→ To prevent the algorithm from getting stuck in a loop or revisiting states, a list of previously visited states can be maintained (Tabu list).

→ Add most recent state to queue; drop oldest.

• Local Beam search :-

→ To tackle plateau and local optima, this variation of hill climbing keeps track of K states rather than just one.

→ It begins with K randomly generated states. At each step, all the successors of all K states are generated, if any one is a goal, it halts. Else select the best K successors from the complete list and repeats.

* Hill Climbing with Random Walk :-

→ In this, with some probability P, the algorithm chooses a random move instead of the best move.

Algorithm -

1) ~~Stat~~ Start with an Initial state s.
2) Repeat until a stopping condition is met:
   a) Generate the neighbors of current states
   b) with probability P :
      - Pick a random neighbor and move to it.
   c) otherwise -
      - Pick the best neighbor based on Evaluation fn and move to it.
   d) If no move is possible (plateau), terminate.
3) Return the final state as the solution.

* Hill Climbing with both :-

At each step do one of the three -

- Greedy move
- Random walk
- Random Restart

* Simulated Annealing :-

→ Annealing is the process in which physical substances as metal are melted and then ~~Gradu~~ gradually cooled until solid state is reached.

→ Probability that the metal will jump to a high energy level is given by —

$$P = e^{\Delta E/T}$$

```
function SIMULATED_ANNEALING (Problem, shedule):
    Current ← initial_state
    for t=1 to ∞ do:
        T ← schedule(t)
        if(T=0) then return current
        next ← randomly selected neighbor
        ΔE ← VALUE(next) - VALUE(current)
        if ΔE > 0 then:
            current ← next
        else:
            with prob e^(ΔE/T), move to ~~ne~~next
```

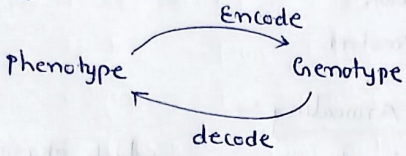
→ high T : Probability of 'locally bad' move is higher

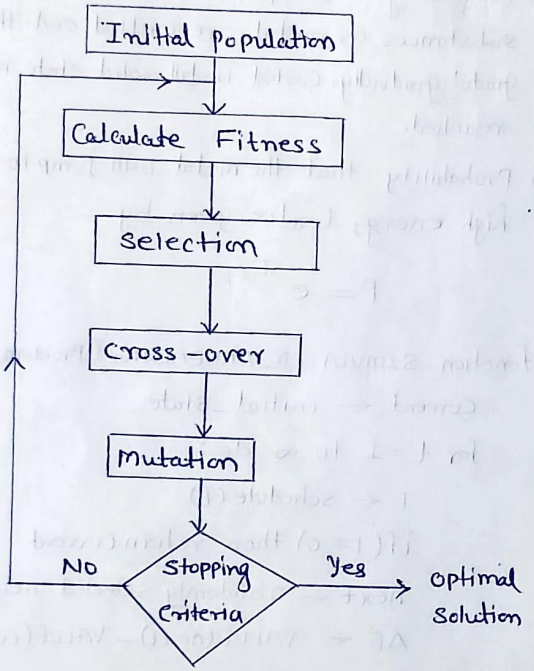
→ Low T : " " " move is lower.

→ T ↓ Algorithm runs longer

* Genetic Algorithm :-

- Abstraction of Real biological Evolution.
- Solve Complex problems (like Np-Hard)
- focus on optimization
- Populations of possible solutions for a given Problem.





- Produce the next generation of states by "simulated evolution".
  - → Random selection
  - → Crossover
  - → Random mutation

* for 8 queen problem-

→ fitness fn :- No. of Non-attacking pairs of queens

$$^8C_2 = 28$$

for- (24748552)

f(n) = 24

24748552	24
32752411	23
24415124	20
32543213	11

24752411
32748552
24415213
32543124

Cross-over ↓

32748152
24752411
24415417
32252124

mutation

⊕ → Global search Capability
Handles large space Efficiently
Resistant to local optima

⊖ → No Guarantee of optimal solution
Require Careful parameter Tuning
Inefficient for simple problem

* Optimization of Continuous Functions :-

(a) Discretization :-

- Convert Continuous Values into discrete ones and apply hill climbing.
- Not very Efficient

(b) Gradient Descent Algorithm :-

- It is a first-order optimization Algo. for minimizing continuous functions by iteratively moving in the direction of steepest descent.

- Algorithm -

Let $f(x_1, x_2, \ldots, x_N)$

(i) Compute the gradient of Each Variable

$$\frac{\partial f}{\partial x_i}, \quad x_i \in X$$

It points in the direction of the steepest increase of the function.

(ii) update Each Variable -

$$x_i \leftarrow x_i - \lambda \frac{\partial f}{\partial x_i}$$

(iii) Repeat until Convergence or max iterations

(iv) if the Change in $f(x)$ is Smaller than ϵ then stop.

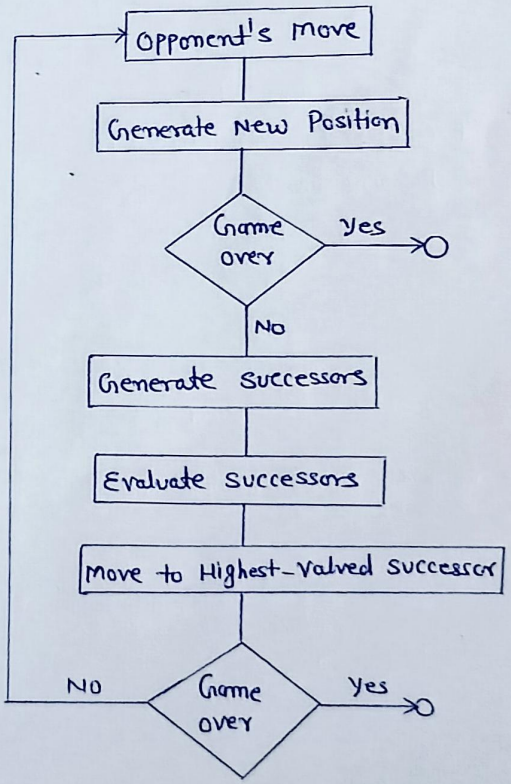
(v) Return x.

* λ is too small → Converges Slowly
* λ is too large → overshoot the minimum
* This Algo moves in the direction of the steepest decrease.

* Game Playing :-

Adversorial search or game-tree search is a technique for analyzing an adversorial game in order to try to determine who can win the game and what moves the players should make in order to win.

- States - board Configurations
- Initial state - the board position and which player will move.
- Successor fn - return list of (move, state) pairs, each indicating a legal move and the resulting state.
- Terminal ~~state~~ test - determines when the game is over.
- Utility function - gives a numeric value in terminal states. (e.g. → -1, 0, +1 for loss, tie, win)



* Mini-max Algorithm :-

- Two Player – maximizing player (MAX) and minimizing player (MIN)
- MAX tries to maximize the score.
- MIN tries to minimize MAX's score.
- Game is represented as a tree,
  - Nodes – game states
  - Edges – Possible moves
  - Leaf Nodes – game outcomes (win/loss/draw)