ARTIFICIAL INTELLIGENCE (A.I.)

~~BLIND SEARCH~~ UNINFORMED SEARCH/	INFORMED SEARCH
1. Search without Information	1. Search with information
2. NO knowledge	2. Use knowledge to find steps to solution.
3. Time consuming.	3. Quick solution
4. More Complexity (Time, space)	4. Less Complexity (Time, space)
5. DFS, BFS etc	5. A*, Best first search

HEURISTIC

'Heuristic in AI' (Rule of thumb) [What, Why, How]

→ It is a technique designed to solve a problem quickly.

BLIND SEARCH

8 puzzle

$(O(b^d))$

(3^{20}) → Time complexity is more

Such problems are known as NP (Non polynomial) problem

↳ Time complexity is Exponential

↳ Advantage → optimal solⁿ

Heuristic Search

NP → P

→ Reduce time

→ Multiple solⁿ

↓

Explore best

→ Quickly solve the problem

→ Best Solution

→ doesnot gaurantee optimal solⁿ

How to find heuristic value :-

1. Euclidian Distance (straight Line Distance) :

→ find distance

$$E.D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

2. Manhattan distance (used when we want to find vertical, horizontal distance)

eg:-

1	3	2
6	5	4
	8	7

Start State

1	2	3
4	5	6
7	8	

Goal State

In Blind search (3^{20}) → then only you get optimal solⁿ
↳ search space

① To find heuristic value → use Manhattan distance

M.D $= 0+1+1+2+0+2+2+0$ (position of tiles)

↳ Explore all possible position
↳ select lowest Manhattan value

② No of Misplaced tiles

⑤ → less misplaced tile
→ select that path

1: BRANCH AND BOUND SEARCH (UNIFORM COST SEARCH)

→ In branch and bound search method, cost function (denoted by g(X)) is designed that assigns cumulative expense to the path from start node to the current node X by applying the sequence of operators.

→ Branch and bound search expands the least-cost path at each iteration till we reach goal state.

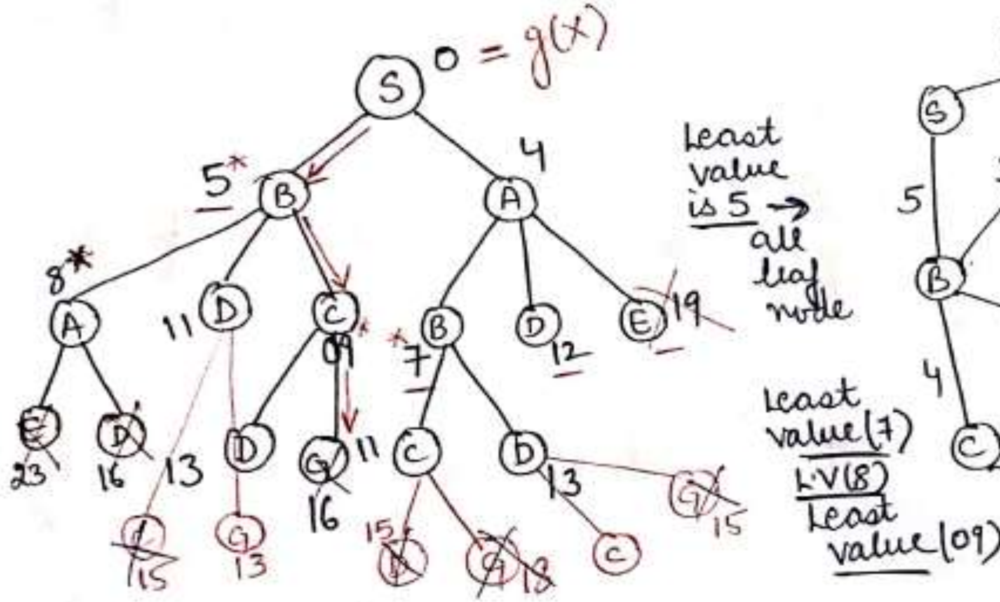
→ Sometimes it is also called a Uniform cost search.

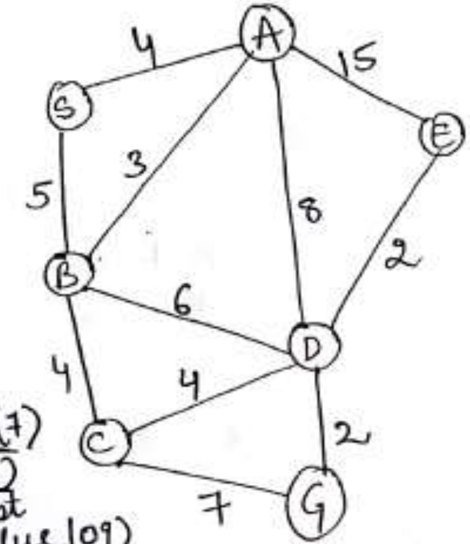
→ Two parts
- → Branch : Several choices are found
- → Bound : Setting bound on solⁿ Quality
  - ↳ Pruning trimming off branches where solution quality is poor.

Algorithm *(Branch and Bound)*

```
Input: START and GOAL states
Local Variables: OPEN. CLOSED. NODE. SUCCs. FOUND:
Output: Yes or No
Method:
 • initially store the start node with g(root) = 0 in a OPEN list: CLOSED = ϕ:
   FOUND = false:
 • while (OPEN ≠ ϕ and FOUND = false) do
   {
    • remove the top element from OPEN list and call it NODE:
    • if NODE is the goal node. then FOUND = true  else
      {
       • put NODE in CLOSED list:
       • find SUCCs of NODE. if any. and compute their 'g' values and store them
         in OPEN list:
       • sort all the nodes in the OPEN list based on their cost-function values:
      }
   } /* end while */
 • if FOUND = true then return Yes otherwise return No:
 • Stop
```

Example :-





S-B-C-G = 16

1. Bound set = 16 → value greater than 16 is pruned off.
2. Now explore the path having value less than 16
3. Now discard path having value greater than 13.

$\boxed{S-B-D-G = 13}$ optimal path

2. HILL CLIMBING (Local search algo, Greedy approach, NO backtrack)

Local search algo:

→ knowledge of local domain

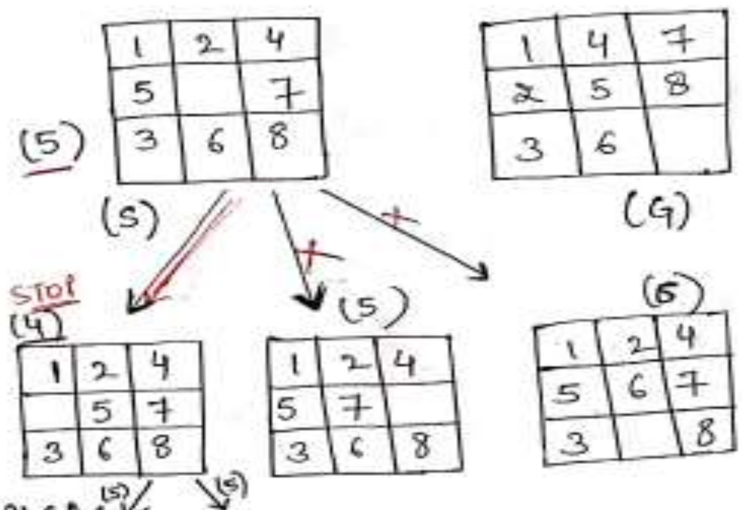
→ No knowledge about global domain

Greedy approach:

→ move till it gets the best move otherwise it will stop.

NO backtrack:

If it doesnot find the best move it will not backtrack.



Space complexity is less as the node which are in used only saved in memory

ALGO:

1. Evaluate the initial state
2. Loop until a solution is found or there are no operators left
   - select and apply a new operator
   - Evaluate the new state:
     - if goal then quit
     - if better than current state then it is a new current state.

Algorithm *(Simple Hill Climbing)*

Input: START and GOAL states

Local Variables: OPEN, NODE, SUCCs, FOUND;

Output: Yes or No

Method:

- store initially the start node in a OPEN list (maintained as stack); FOUND = false;
- while (OPEN ≠ empty and Found = false) do

  {

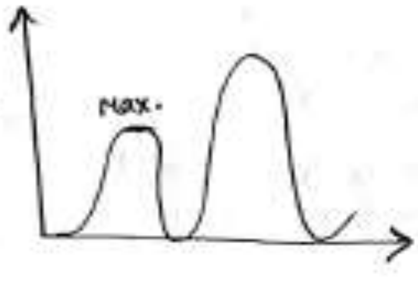
  - remove the top element from OPEN list and call it NODE;
  - if NODE is the goal node, then FOUND = true else
    - find SUCCs of NODE, if any;
    - sort SUCCs by estimated cost from NODE to goal state and add them to the front of OPEN list;

  } /* end while */
- if FOUND = true then return ***Yes*** otherwise return ***No***;
- Stop

PROBLEMS IN HILL CLIMBING

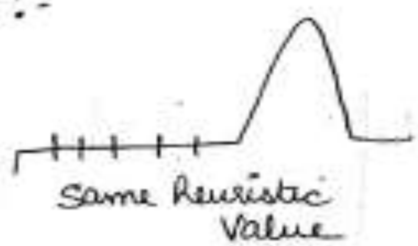
Local Maximum :-

It is a state better than all its neighbours but not better than some other states which are far away.



Plateau | Flat Maximum :-

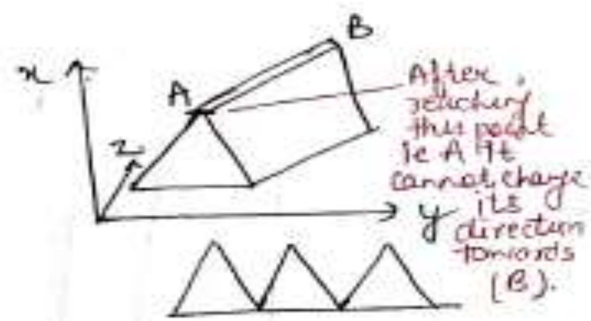
It is a flat area of the search space where all neighbouring states has the same value.



Ridge :-

It is an area of search space that is higher than surrounding areas but that cannot be traversed by single moves in any one direction.

It is a special kind of local maxima.



2. BEST FIRST SEARCH (INFORMED, HEURISTIC)

↳ works as Greedy method.

ALGORITHM :

Let 'OPEN' be a priority queue containing initial state

LOOP

if 'OPEN' is empty return failure

Node ← Remove-First (OPEN)

if Node is a Goal

then return the path from initial to Node

else generate all successors of Node and

put the newly generated Node into 'OPEN'

according to their f values

└ heuristic value

END LOOP

Algorithm *(Best-First Search)*

Input: START and GOAL states

Local Variables: OPEN, CLOSED, NODE, FOUND;

Output: Yes or No

Method:

- initialize OPEN list by root node; CLOSED = ϕ; FOUND = false;
- while (OPEN $\neq \phi$ and FOUND = false) do

  {

  - if the first element is the goal node, then FOUND = true else remove it from OPEN list and put it in CLOSED list;
  - add its successor, if any, in OPEN list;
  - sort the entire list by the value of some heuristic function that assigns to each node, the estimate to reach to the goal node;

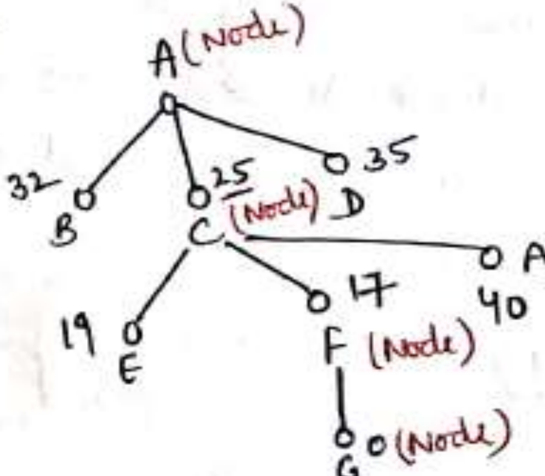
  }
- if FOUND = true then return ***Yes*** otherwise return ***No***;
- Stop

EXAMPLE

1: A (Node)
→ this is not a goal

else

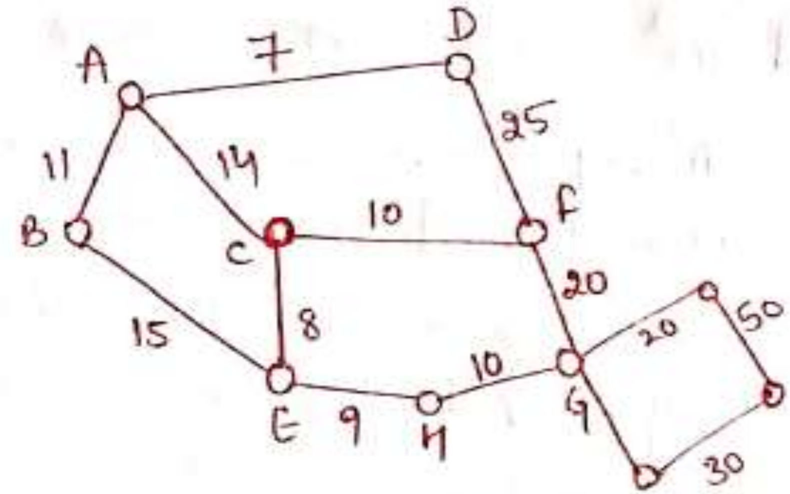
2.



already explored (add to CLOSE LIST)

OPEN LIST :- ~~A~~, ~~C~~, B, D, ~~F~~, E, G

3. if Node is a Goal then return path (A-C-F-G)



Straight line distance (Eucledian distance)

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

If you are working according to heuristic you will follow the path:

A – C – F – G (cost is 44 without heuristic)

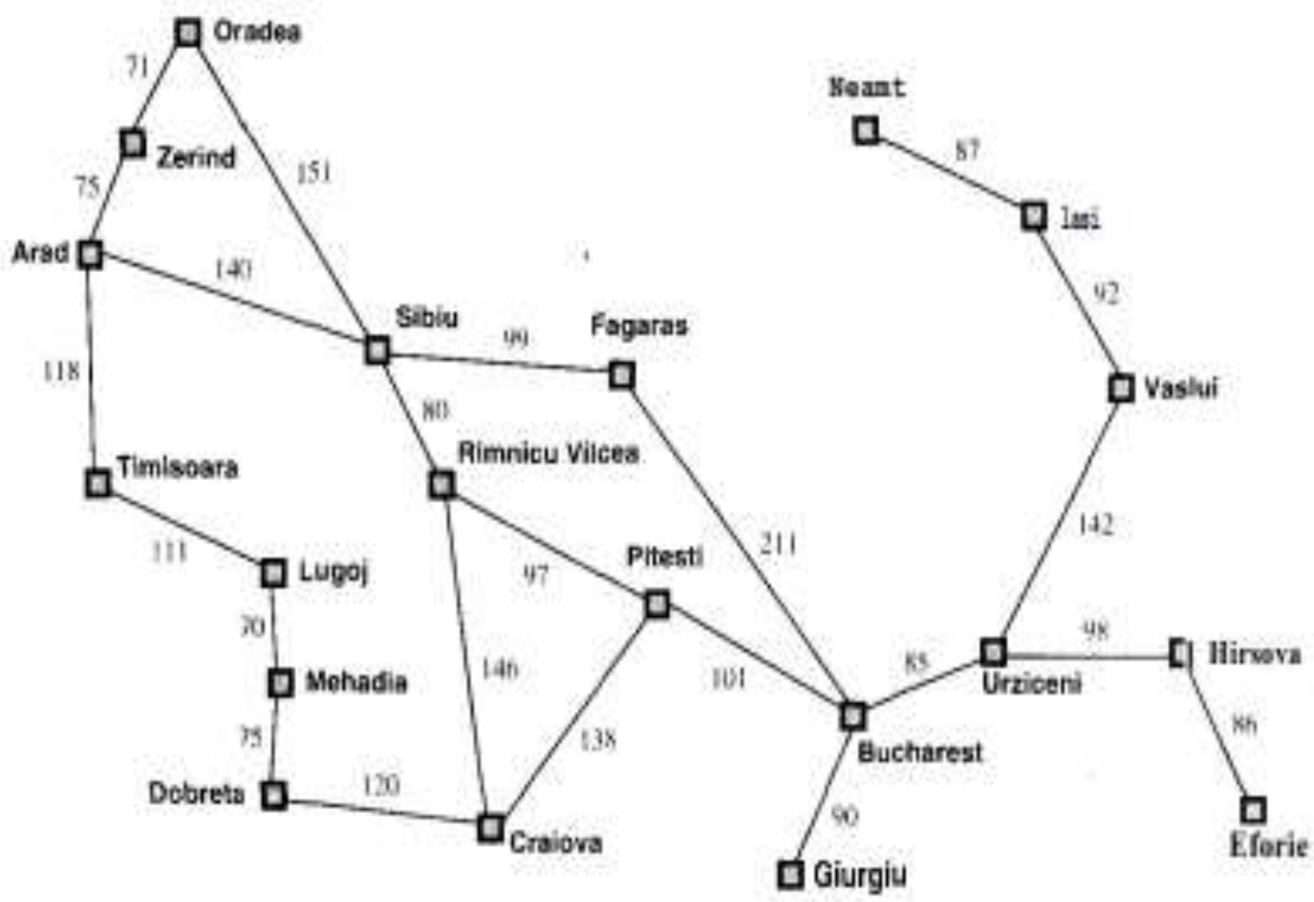
But if you follow the path:

A – C – E – H – G (cost is 41) this is less as compare to above.

So if we consider both the heuristic and cost then there is changes to get optimal solⁿ.

↳ which is used by A* search

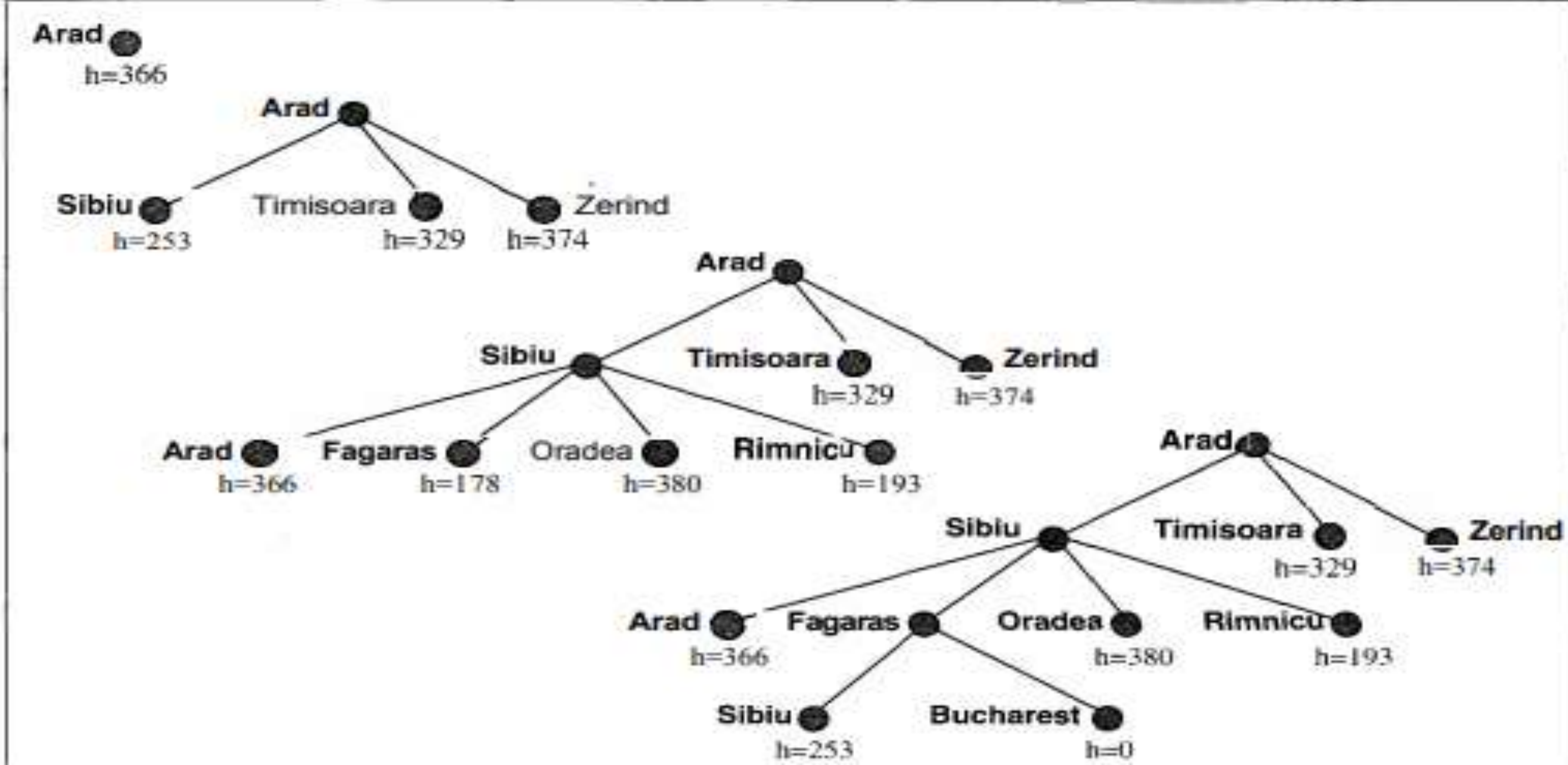
Assignment



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Map of Romania with road distances in km, and straight-line distances to Bucharest.



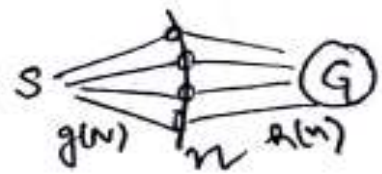
Stages in a greedy search for Bucharest, using the straight-line distance to Bucharest as the heuristic function h_{SLD}. Nodes are labelled with their h-values.

Imp

4. A* Algorithm : (Informed Searching)

A* algorithm is a combination of 'branch and bound' and 'best search' methods. It uses a heuristic or evaluation function usually denoted by $f(x)$ to determine the order in which the search visits node in the Tree. The heuristic function for a node N is defined as follows:

$$\boxed{f(N) = g(N) + h(N)}$$



function g is a measure of cost of getting from start node to current node N.

function h is an estimate of additional cost of getting from current node N to goal node.

A* algorithm is called OR graph / tree search algo.

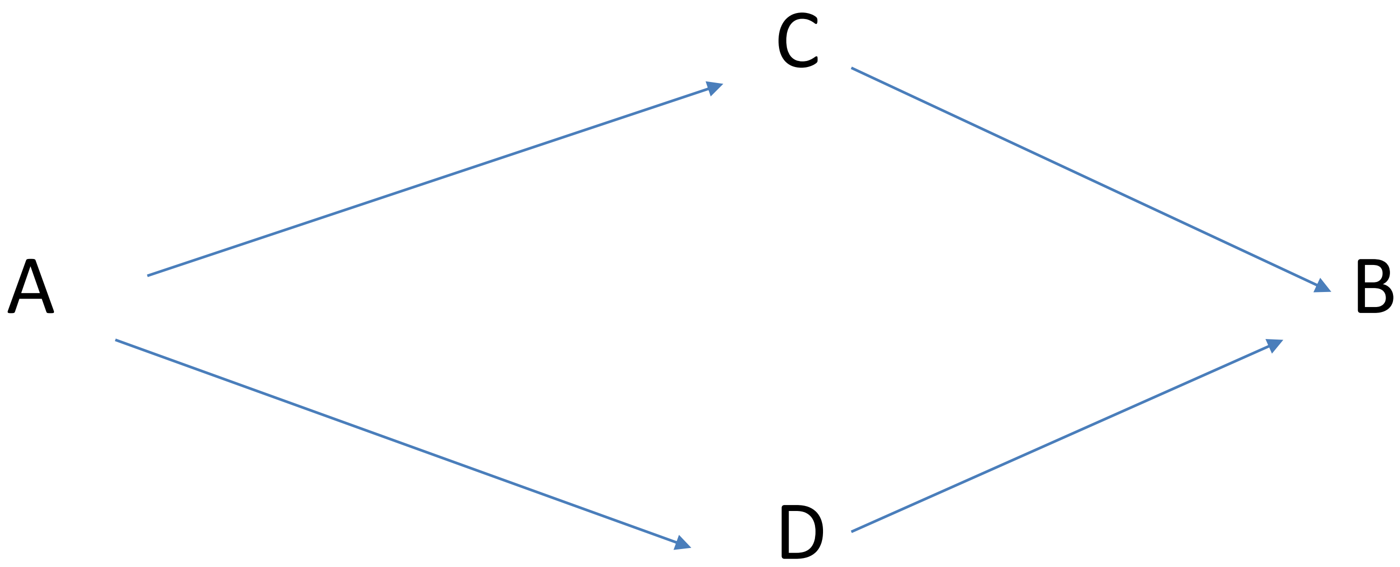
A* algo incrementally searches all the routes starting from the start node until it finds the shortest path to a goal.

Algorithm (A*)

```
Input: START and GOAL states
Local Variables: OPEN, CLOSED, Best_Node, SUCCs, OLD, FOUND;
Output: Yes or No
Method:
• initialization OPEN list with start node; CLOSED= φ; g = 0, f = h,
  FOUND = false;
• while (OPEN ≠ φ and Found = false) do
  {
  • remove the node with the lowest value of f from OPEN list and store
    it in CLOSED list. Call it as a Best_Node;
  • if (Best_Node = Goal state) then FOUND = true else
    {
    • generate the SUCCs of the Best_Node;
    • for each SUCC do
      {
      • establish parent link of SUCC;  /* This link will help to
        recover path once the solution is found */
      • compute g(SUCC) = g(Best_Node) + cost of getting from Best_Node
        to SUCC;
      • if SUCC ∈ OPEN then /* already being generated but not
        processed */
        {
        • call the matched node as OLD and add it in the successor list
          of the Best_Node;
        • ignore the SUCC node and change the parent of OLD, if required
          as follows:
          • if g(SUCC) < g(OLD) then make parent of OLD to be Best_Node
            and change the values of g and f for OLD else ignore;
        }
      • If SUCC ∈ CLOSED then /* already processed */
        {
        • call the matched node as OLD and add it in the list of the
          Best_Node successors;
        • ignore the SUCC node and change the parent of OLD, if required
          as follows:
          • if g(SUCC) < g(OLD) then make parent of OLD to be
            Best_Node and change the values of g and f for OLD and
            propogate the change to OLD's children using depth first
            search else ignore;
        }
      • If SUCC ∉ OPEN or CLOSED
        {
        • add it to the list of Best_Node's successors;
        • compute f(SUCC) = g(SUCC) + h(SUCC);
        • put SUCC on OPEN list with its f value
        }
      }
    }
  }  /* End while */
• if FOUND = true then return Yes otherwise return No.
• Stop
```

Cost fn g(n) -Distance A-C-B – 5 kms (5mns)

H(n)-Traffic-5 mns ----10mns



Cost fn g(n) Distance A-D-B – 7 kms(7mns)

H(n)-Traffic – 1mnt -----   8mns

Let us consider an example of eight puzzle again and solve it by using A* algorithm. The simple evaluation function f(x) is defined as follows:

$f(X) = g(X) + h(X)$, where

$h(X)$ = the number of tiles not in their goal position in a given state X

$g(X)$ = depth of node X in the search tree

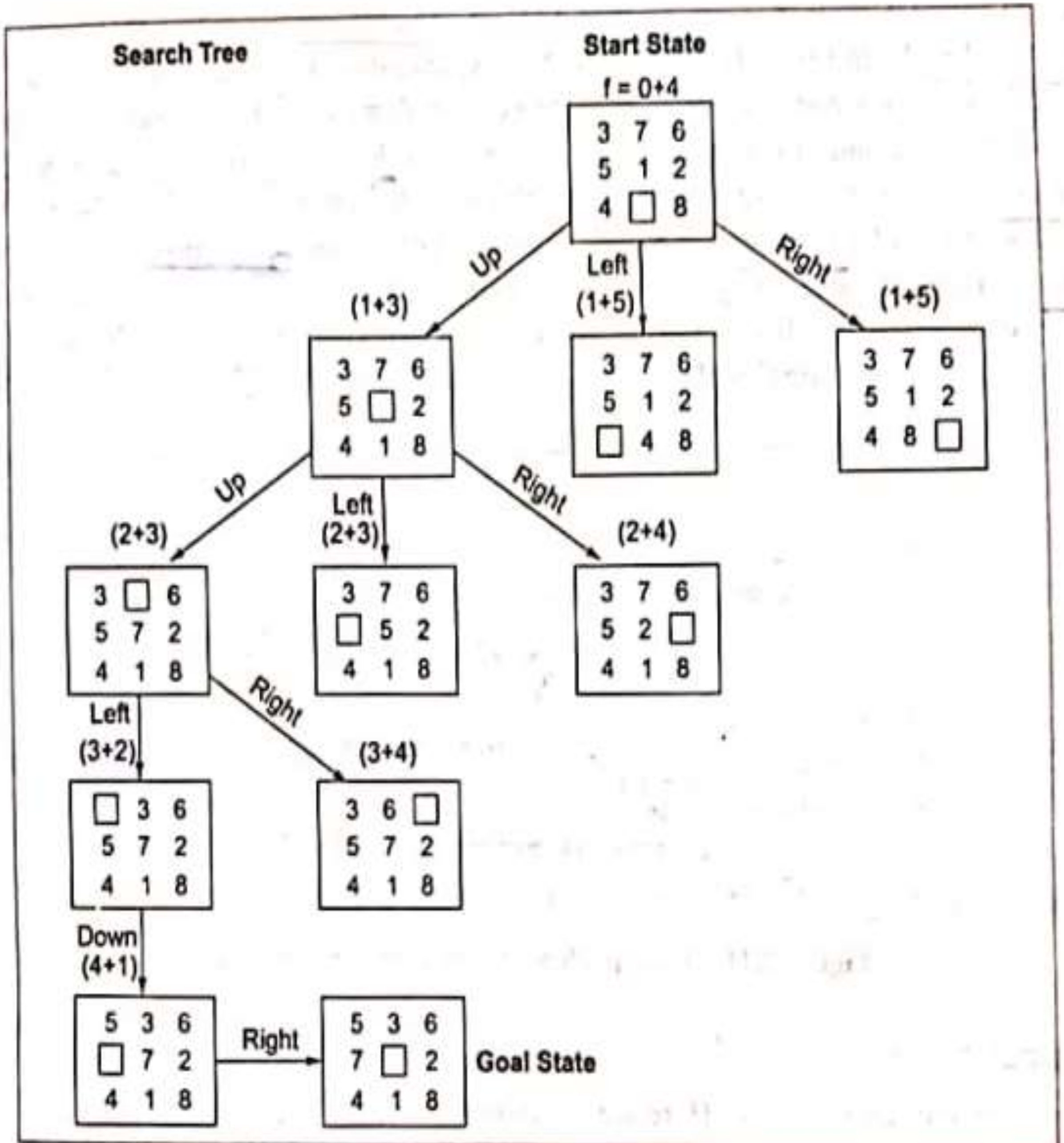
Given

Start State

3	7	6
5	1	2
4	□	8

Goal State

5	3	6
7	□	2
4	1	8


Search Tree
Start State
f = 0+4
Up
Left
Right
(1+3)
(1+5)
(1+5)
Up
Left
Right
(2+3)
(2+3)
(2+4)
Left
Right
(3+2)
(3+4)
Down
(4+1)
Right
Goal State

Search Tree

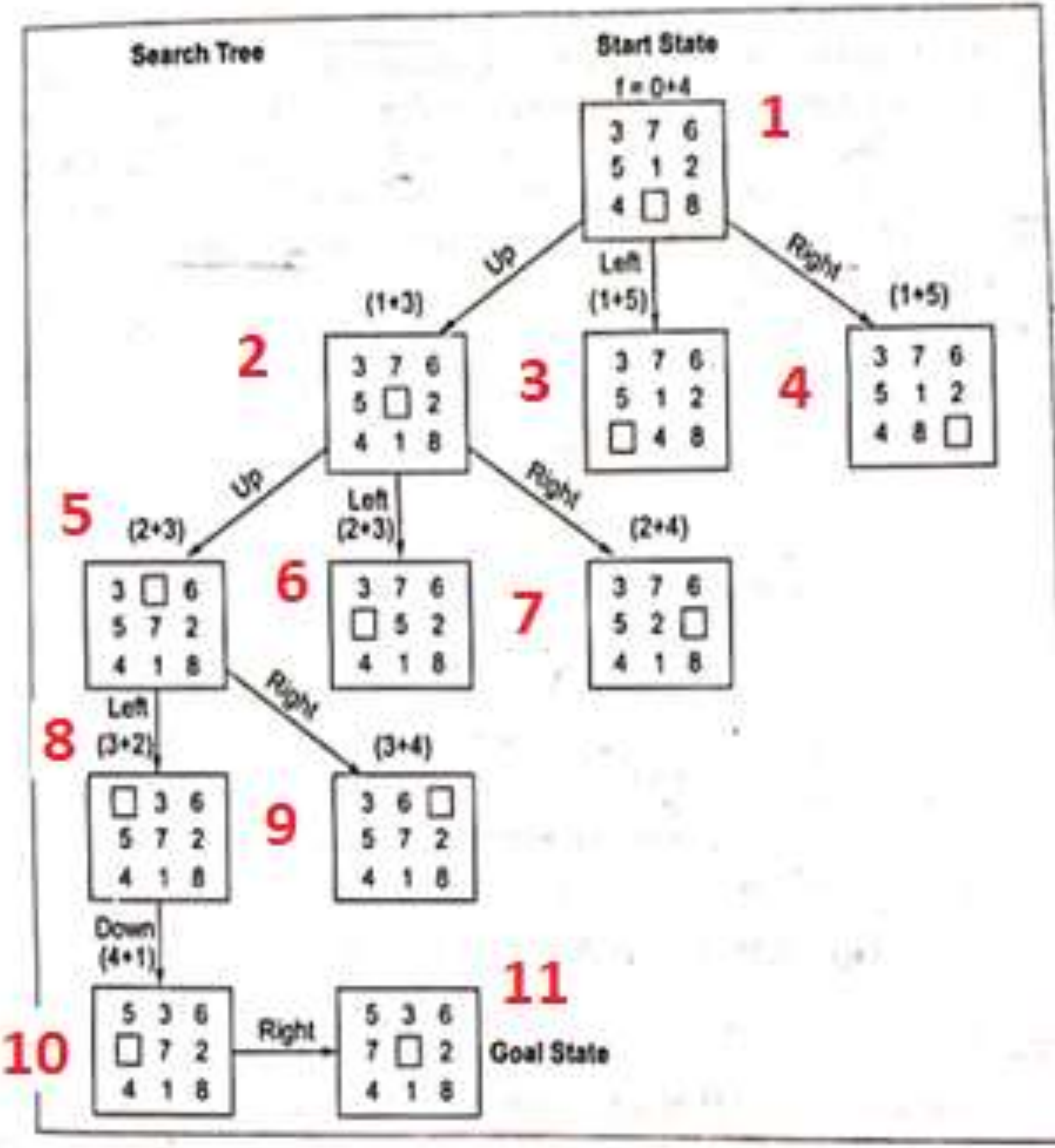
Open List-3-1(6),4-1(6),6-2(5),7-2(6),9-5(7), 12-10(7)

Closed List- 1(4), 2-1(4), 5-2(5), 8-5(5), 10-8(5)

BestNode- 11-10(5)

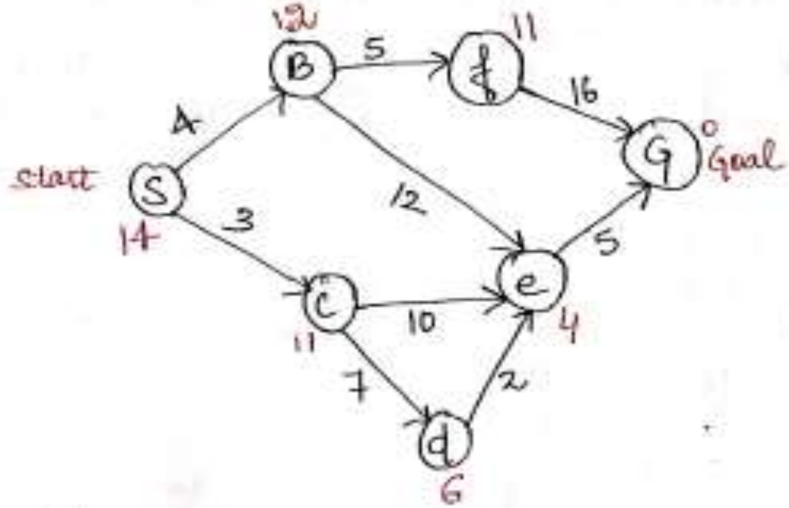
SUCCs-

Old-



Search Tree

2.



1 $f(S) = 0 + 14 = 14$

2 $S \to B$
$= 4 + 12$
$= \underline{16}$

$S \to C$
$= 3 + 11$
$= \underline{14}$

$SB \to f$
$4 + 5 + 11$
$= 20$

$SB \to e$
$4 + 12 + 4$
$= 20$

3 $SC \to e$
$= 3 + 10 + 4$
$= 17$

$SC \to d$
$3 + 7 + 6$
$= \underline{16}$

4. $Scd \to e$
$3 + 7 + 2 + 4 = \underline{16}$

5. $\boxed{Scde \to G \quad 3 + 7 + 2 + 5 + 0 = 17}$

HOW TO MAKE A* ADMISSIBLE :

↳ acceptable

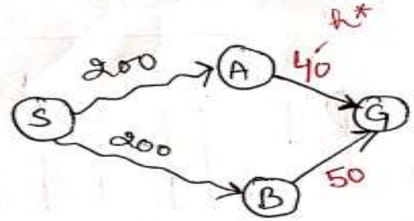
→ A* star search algorithm is admissible, if for any graph it always terminates in an optimal path from start state to goal state, if path exists.

→ If heuristic function 'h' underestimates the actual value from current state to goal state, then it bounds to give an optimal solution & hence is called admissible function.

→ So, A* always terminates with optimal path in case h is an admissible heuristic function

Underestimation $h(n) \le h^*(n)$

estimated — actual / optimal



$h(A) = 30$
$h(B) = 20$

$A^* \rightarrow f(n) = g(n) + h(n)$

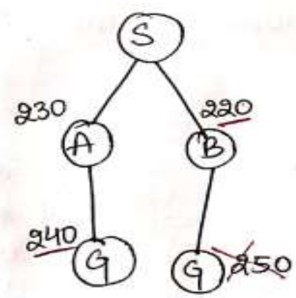
$f(A) = g(A) + h(A) = 200 + 30 = 230$

$f(B) = g(B) + h(B) = 200 + 20 = 220$

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

$= 200 + 50 + 0$

$= 250$



After reaching goal you observe that A has value 230 which is less than G = 250, so now explore A

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

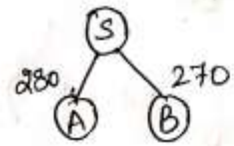
$= 200 + 40 + 0$

$= 240$ —— value is less as compared to SBG path

So if heuristic function 'h' underestimates the actual value from current state to goal state, then it bounds to give an optimal solution

OVERESTIMATION $h(n) \geq h^*(n)$

$h(A) = 80$
$h(B) = 70$



$f(A) = g(A) + h(A)$
$= 200 + 80 = 280$

$f(B) = g(B) + h(B)$
$= 200 + 70 = 270$

$f(G) = g(G) + h(G)$
$= 250 + 0 = 250$