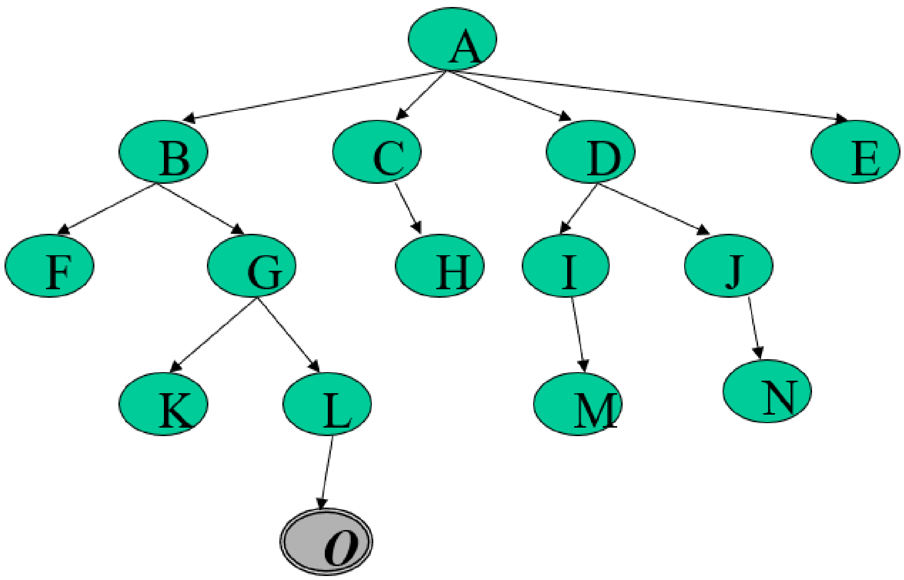
Breadth First Search

- Application1:

  Given the following state space (tree search), give the sequence of visited nodes when using BFS (assume that the node ***O*** is the goal state):



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.......+ b^d= O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

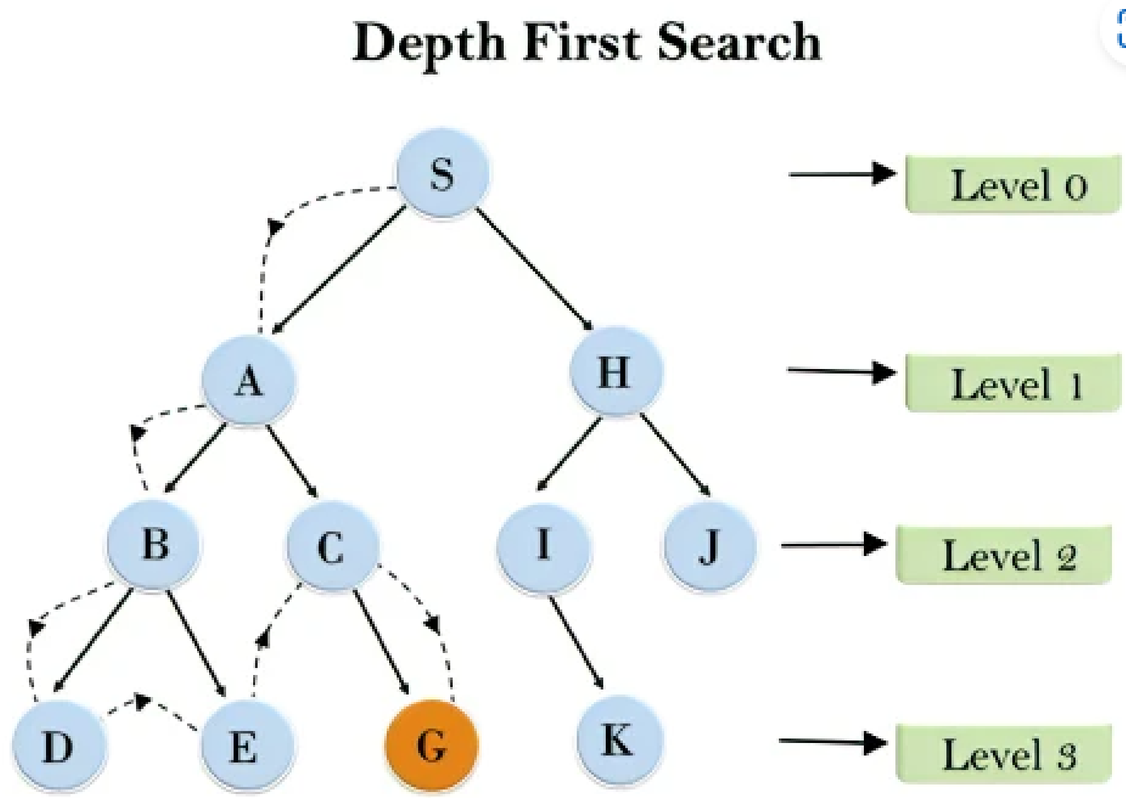
Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

What Criteria are used to Compare different search techniques ?

As we are going to consider different techniques to search the problem space, we need to consider what criteria we will use to compare them.

- **Completeness**: Is the technique guaranteed to find an answer (if there is one).
- **Optimality/Admissibility** : does it always find a least-cost solution?
  - an admissible algorithm will find a solution with minimum cost
- **Time Complexity**: How long does it take to find a solution.
- **Space Complexity**: How much memory does it take to find a solution.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

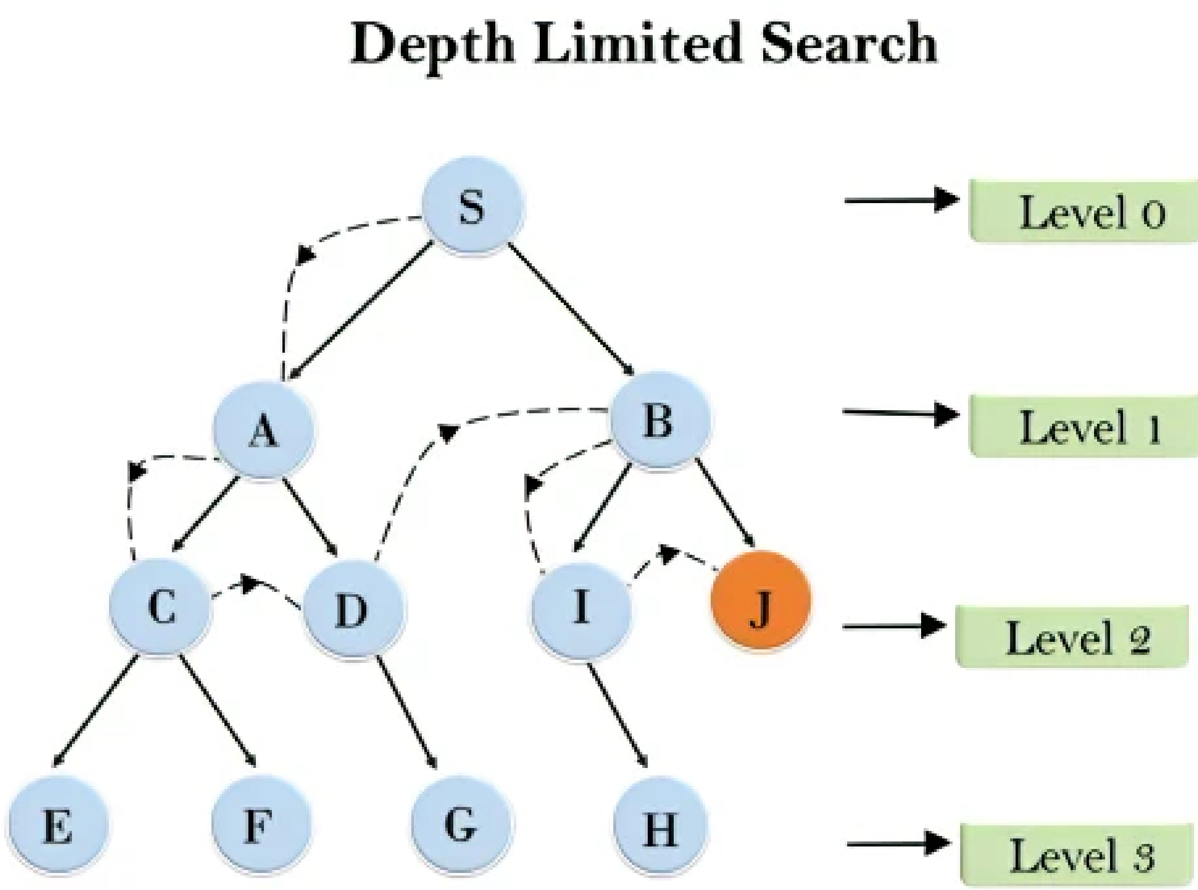
Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

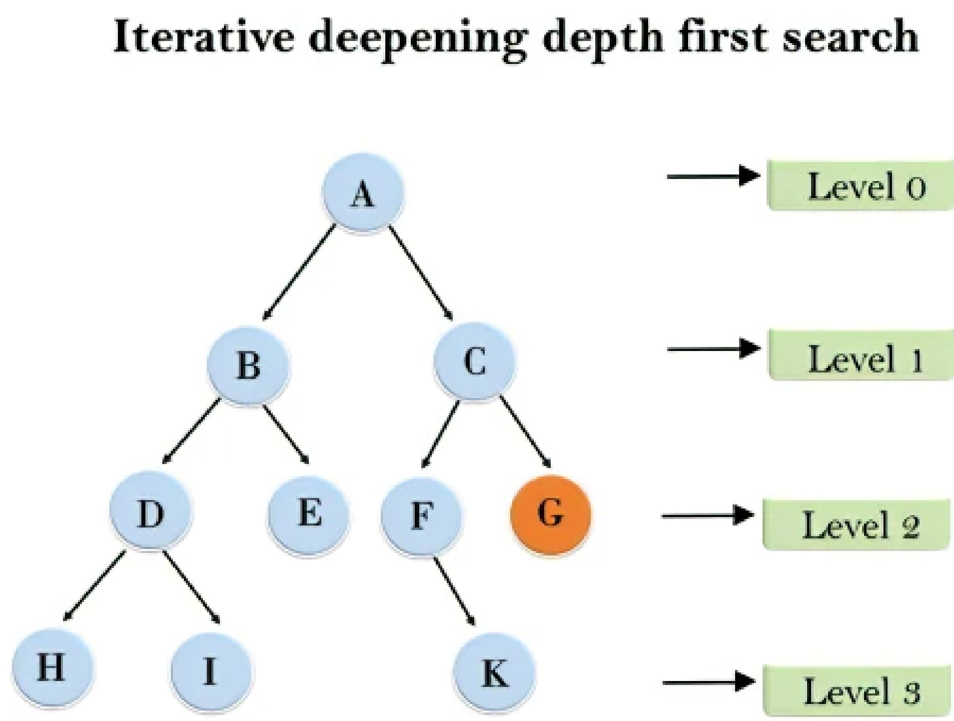


Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **O(b$^{\ell}$)**.

Space Complexity: Space complexity of DLS algorithm is O**(b×ℓ)**.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.



1'st Iteration-----> A

2'nd Iteration----> A, B, C

3'rd Iteration------>A, B, D, E, C, F, G

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

Completeness:

This algorithm is complete is ifthe branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**.

Space Complexity:

The space complexity of IDDFS will be **O(bd)**.

Optimal:

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C^* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = $C^*/\varepsilon+1$. Here we have taken +1, as we start from state 0 and end to C^*/ε.

Hence, the worst-case time complexity of Uniform-cost search is**$O(b^{1 + [C^*/\varepsilon]})$/**.

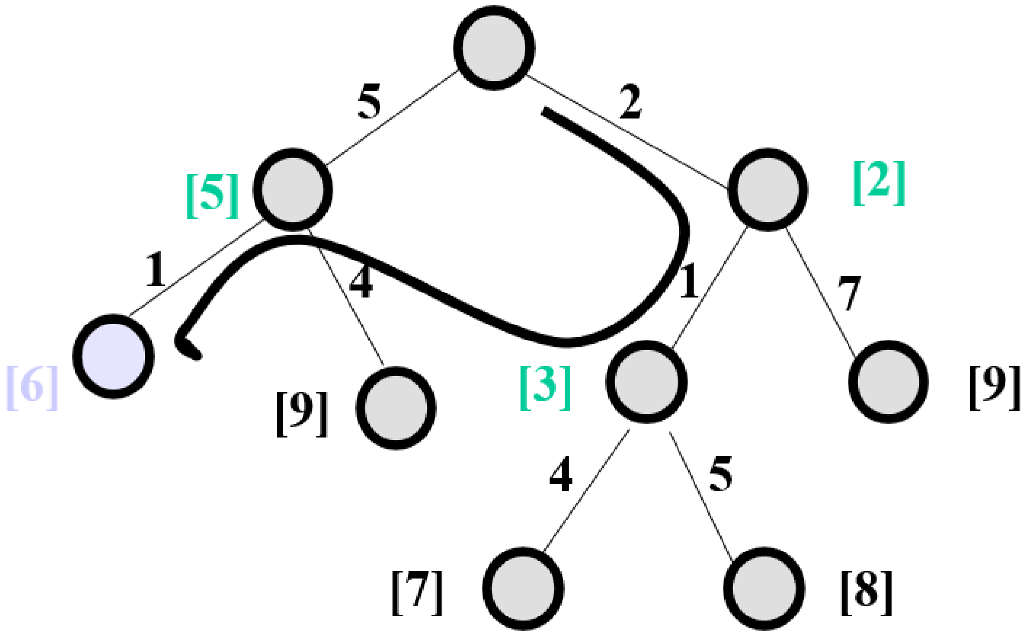
Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is **$O(b^{1 + [C^*/\varepsilon]})$**.

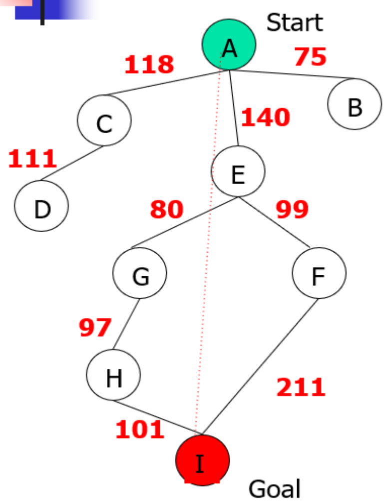
Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

Uniform Cost Search (UCS)



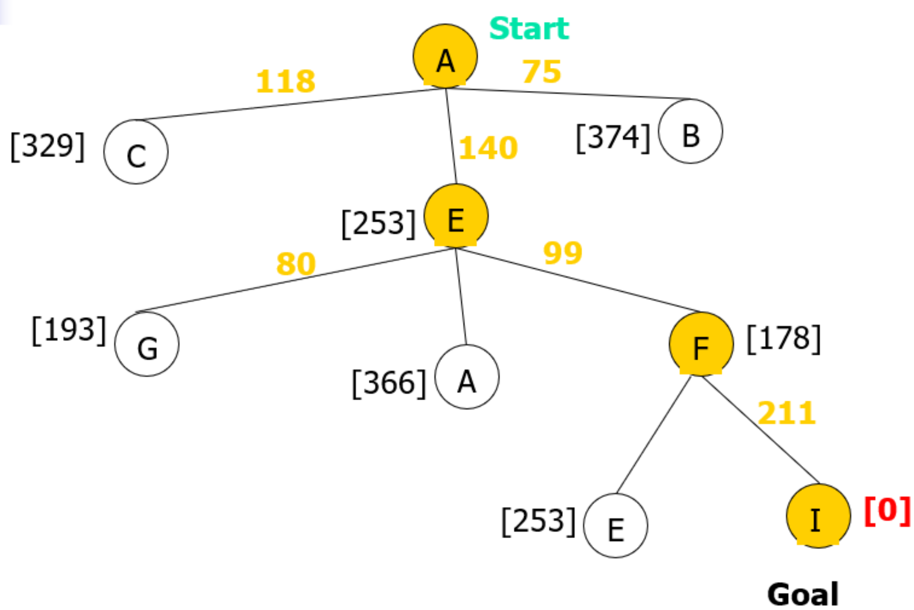
Greedy Search



State	Heuristic: h(n)
A	**366**
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I	0

***f(n) = h (n)* = straight-line distance heuristic**

Greedy Search: Tree Search



Path cost(A-E-F-I) = 253 + 178 + 0 = 431

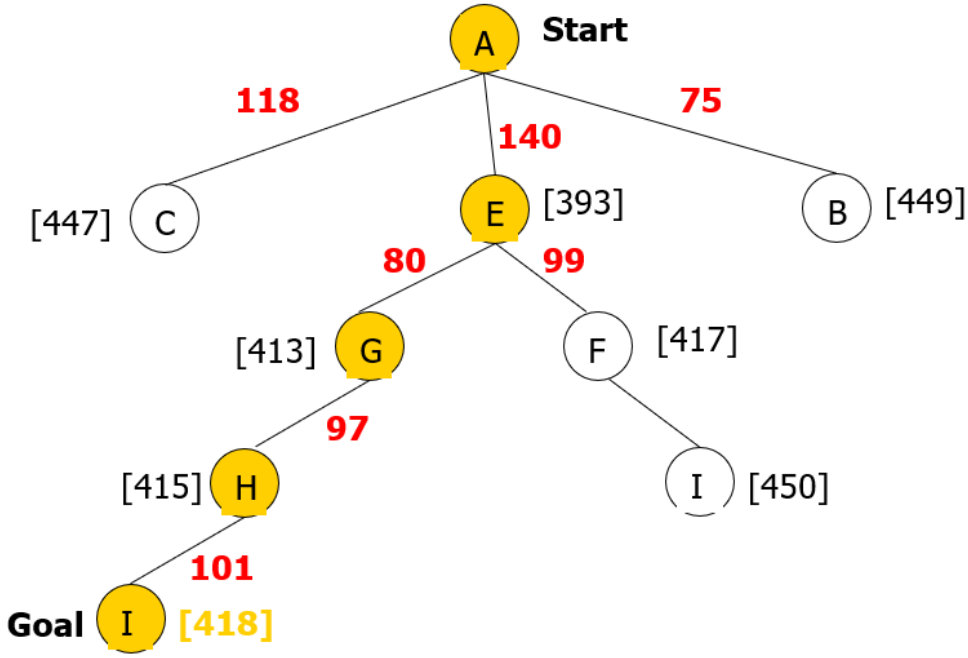
dist(A-E-F-I) = 140 + 99 + 211 = 450

Algorithm	Time Complexity	Space Complexity	Completeness	Optimality
BFS(Breadth First Search)	$O(b^{d+1})$	$O(b^{d+1})$	Yes (b is finite)	Yes (Step Cost =1)
DFS(Depth First Search)	$O(b^{m})$	O(bm)	No (infinite)	No (high cost)
DLS(Depth limited Search)	$O(b^{l})$	O(bl)	No	No
IDS (Iterative Deepening Search)	$O(b^{d})$	O(bd)	Yes (b is finite)	Yes (Step Cost =1)
Uniform Cost Search	$O(b^{[c^*/\varepsilon]})$	$O(b^{[c^*/\varepsilon]})$	Yes	Yes
Greedy Best First Search	$O(b^{m})$	$O(b^{m})$	No (infinite)	No
A* Star Search	$O(b^{d})$	$O(b^{d})$	Yes (b is finite)	Yes (h(n) admissible Heuristic)
Hill Climbing	$O(\infty)$	O(b)	No	No
Bi - Directional	$O(b^{d/2})$	$O(b^{d})$	Yes (If bfs both)	Yes

b = branching factor , d = depth of shallowest solution , l = limit of level

m = maximum depth of node or path length

A* Search: Tree Search



Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

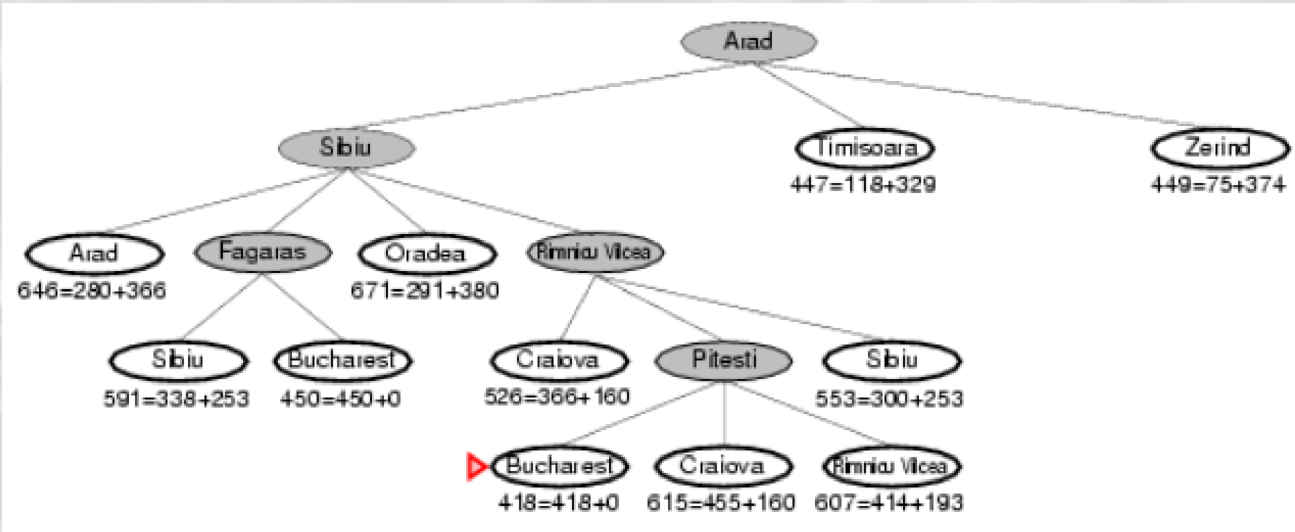
- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

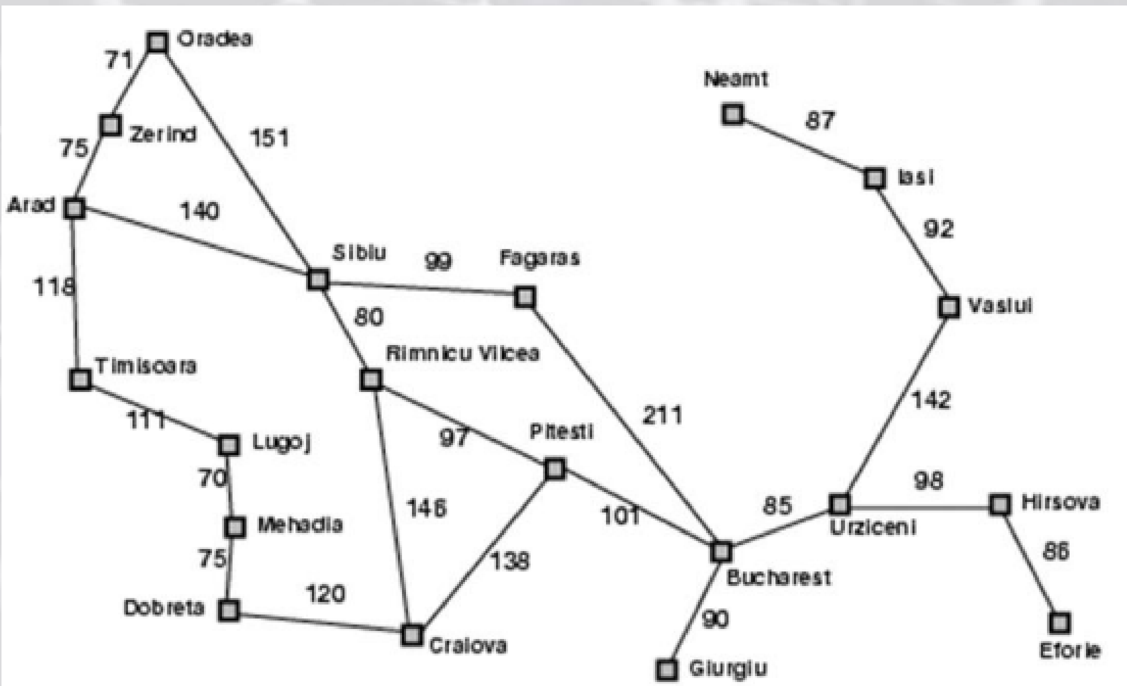
If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

A* search example





Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

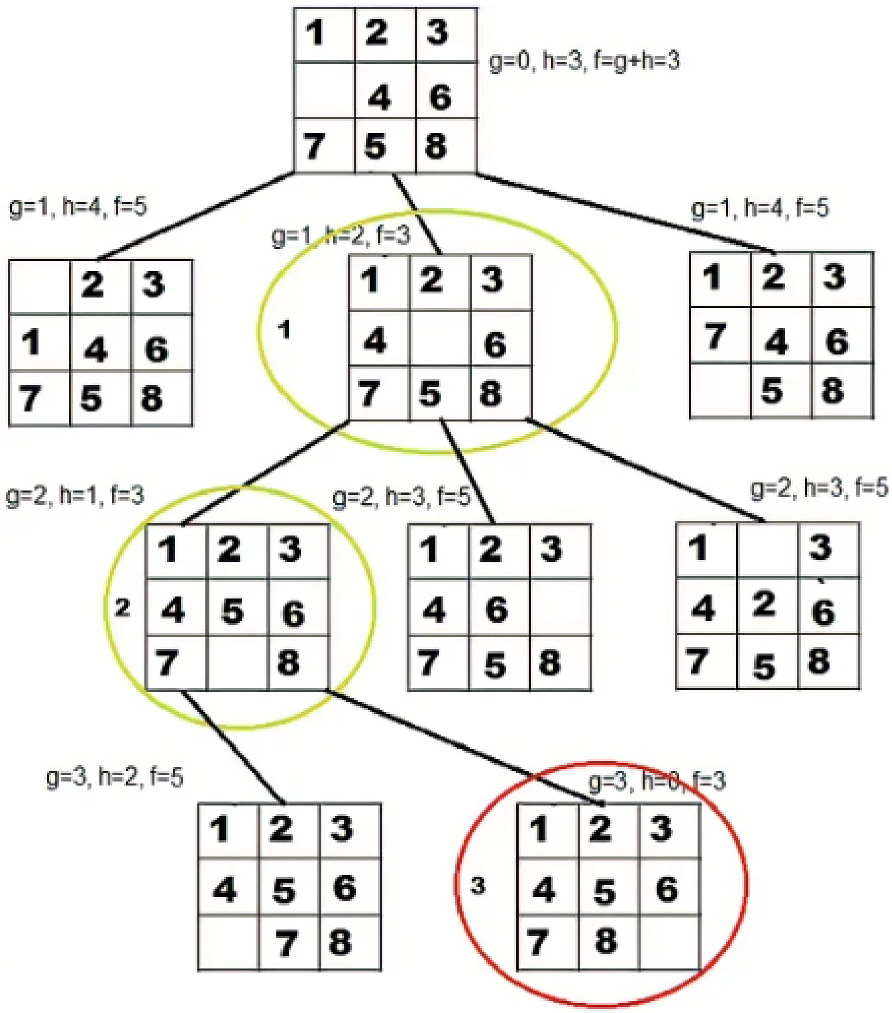


Fig 2. A* algorithm solves 8-puzzle

In our 8-Puzzle problem, we can define the **h-score** as the number of misplaced tiles by comparing the current state and the goal state or summation of the Manhattan distance between misplaced nodes.
g-score will remain as the number of nodes traversed from a start node to get to the current node.

From Fig 1, we can calculate the **h-score** by comparing the initial(current) state and goal state and counting the number of misplaced tiles.
Thus, **h-score** = 5 and **g-score** = 0 as the number of nodes traversed from the start node to the current node is 0.

AI 30.3.23

Start : h = -4

2	8	3
1	6	4
7		5

Goal h = 0

1	2	3
8		4
7	6	5

2	8	3
1		4
7	6	5

h = -3

2	8	3
1	6	4
	7	5

h = -5

2	8	3
1	6	4
7	5	

h = -5

2	~~8~~	3
1	8	4
7	6	5

h = -3

2	8	3
	1	4
7	6	5

h = -3

2	8	3
1	4	
7	6	5

h = -4

	2	3
1	8	4
7	6	5

h = -2

2	3	
1	8	4
7	6	5

h = -4

1	2	3
	8	4
7	6	5

h = -1

1	2	3
8		4
7	6	5

h = 0

7	2	4
5		6
8	3	1

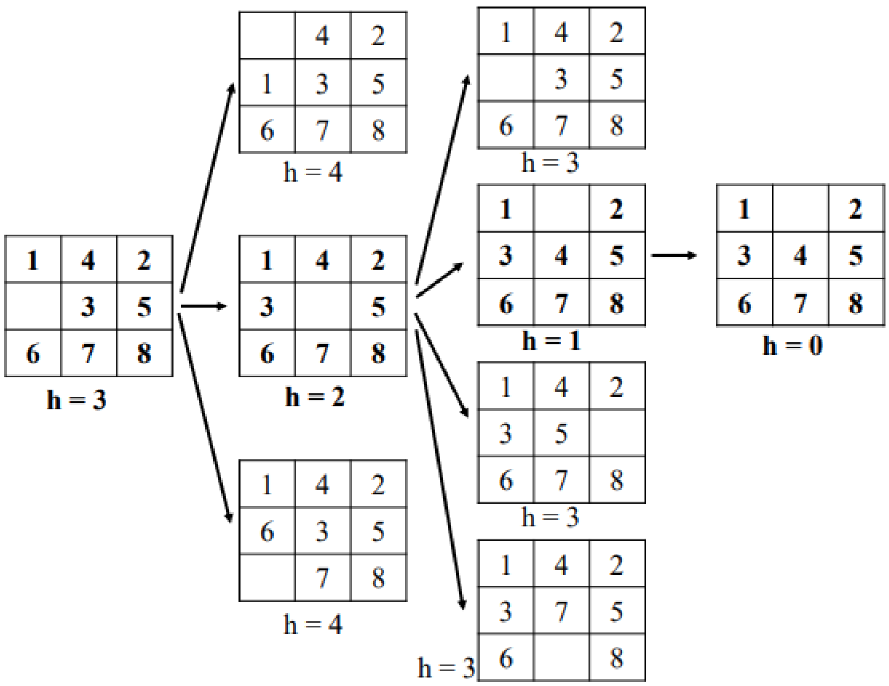
	1	2
3	4	5
6	7	8

7 is a ... → 3
2 - 1 ... → 1
5 → 2
4 → 2
6 → 3
8 → 2
3 → 2
1 → 3

$3+1+2+2+3+2+2+3$
$= 18 = h2$

Hill Climbing Example

8-puzzle: a solution case



Heuristic function is Manhattan Distance