BRANCH AND BOUND.

→ A systematic method for solving optimization problems.

→ is used when the Greedy method and dynamic programming methods may fail.

→ Branch and bound is much slower. It often leads to exponential time complexity in the worst case. On the other hand, if applied carefully, it can lead to algorithms that run reasonably fast on average.

→ The general idea of branch and bound is, it is a BFS like search for optimal solution, but not all the nodes get expanded. Rather, a carefully selected criterion determines which node to expand and when, and another criteria tells the algorithm a optimal solution has been found.

→ Both BFS and DFS generalizes to branch and bound strategies

1. BFS is a FIFO search in terms of live nodes. List of live nodes are represented in the form of a queue.

2. DFS is a LIFO like search in terms of live nodes. List of live nodes are represented in the form of a stack.

3. Least cost search is based on the minimum cost.

Like backtracking, in branch and bound also we a bounding function to avoid searches that doesn't lead to a solution.

→ Branch and bound refers to all the state space search method in which all the children of an E-node are generated before any other live node can become the E-node.

Algorithm:

{ Live node :- A node we get generated, but its children not generated.

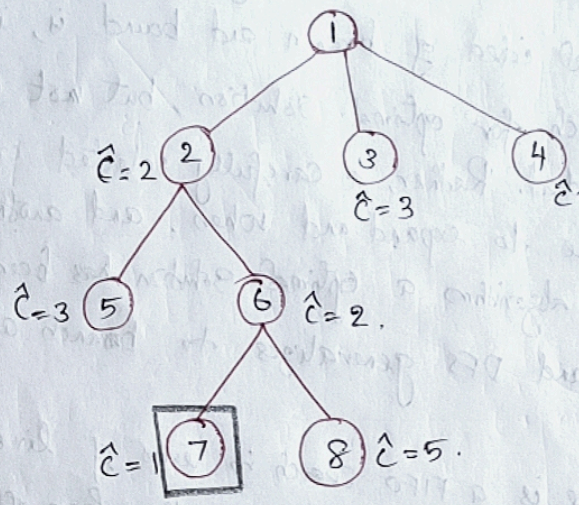
E-node :- A node is generated and also its children are generated.

Dead :- The expansion of this node has no use. So, we will kill the node ~~that~~

}

Live node
E-node
Dead node.

Eg:- State space tree.



$\hat{c}$ - Cost function.

Initially we will take node 1 as E-node. Next generate the children of node 1. The children of node 1 are 2, 3, & 4. By using ranking function, we will calculate the cost of node 2, 3, and 4 $\hat{c}(2)=2$, $\hat{c}(3)=3$, $\hat{c}(4)=4$ respectively. Now we will select a node which is minimum cost ie, node 2. Generate the children of node 2 ie, 5 & 6. Between 5 & 6, we will select node 6 since its cost is minimum. Generate the children of node 6. ie 7 & 8. We will select node 7, since the cost of 7 is minimum. ∴ Cost of Node 7 is the answer. Terminate the selection proc.

Here, E-nodes – 1, 2, 6.

Live-nodes – 3, 4, 5, 8

→ Branch and Bound method of algorithm design involves:

1. Tree organization of solution space.
2. Use of bounding functions to limit the search.

ie, to avoid the generation of subtrees that do not contain an answer node.

→ Search techniques use in BB

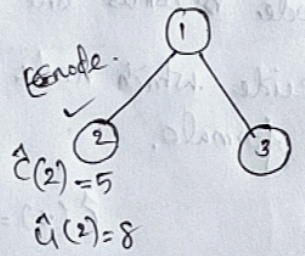
1. BFS
2. DFS
3. Least cost search (LC)

→ Two types of bounds and used in LCBB.

1. Lower bound ($\hat{c}$)
2. Upper Bound ($\hat{u}$)

→ while calculating $\hat{c}$ for a node in the state space tree, fractions are allowed.

→ while calculating $\hat{u}$ for a node, fractions are not allowed.

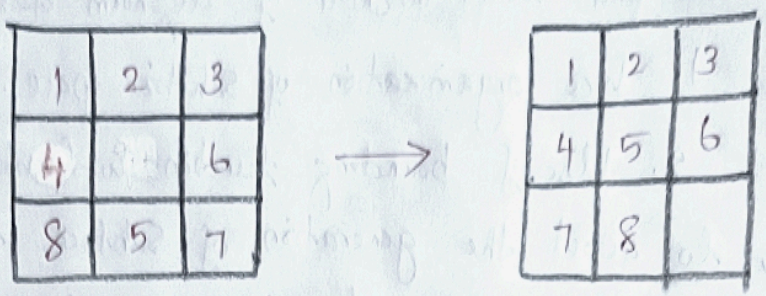


8- PUZZLE PROBLEM.

→ In 8-puzzle problem, there are 8 tiles which are numbered from 1 to 8 placed on a 9-tile capacity square frame.

→ The objective of 8-puzzle problem is to transform the arrangement of tiles from initial arrangement to a goal arrangement.

→ The initial and goal arrangement is shown in below figures.



a) Initial arrangement b) Goal arrangement.

→ There is always an empty slot in the initial arrangement.

→ Legal moves are the moves in which the tiles adjacent to empty slot are moved to either left, right, up, down.

→ The state space tree for 8-puzzle is large because, there can be 9! different arrangements

→ In state space tree, the nodes are numbered as per the level

→ Each next node is generated based on empty slot positions.

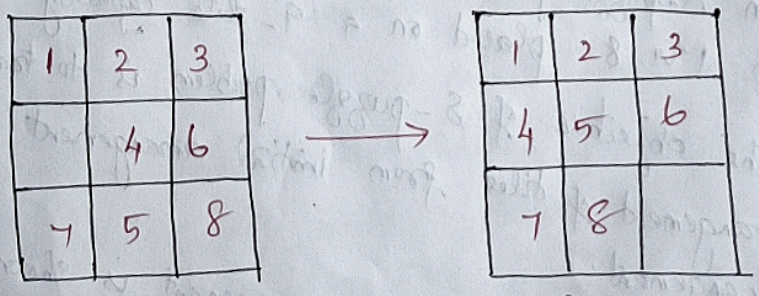
→ Edges are labelled according to the direction in which empty space moves.

→ The root node becomes the E-node.

→ We can decide which node to become an E-node based on estimation formula.

$$\hat{c}(x) = f(x) + \hat{g}(x).$$

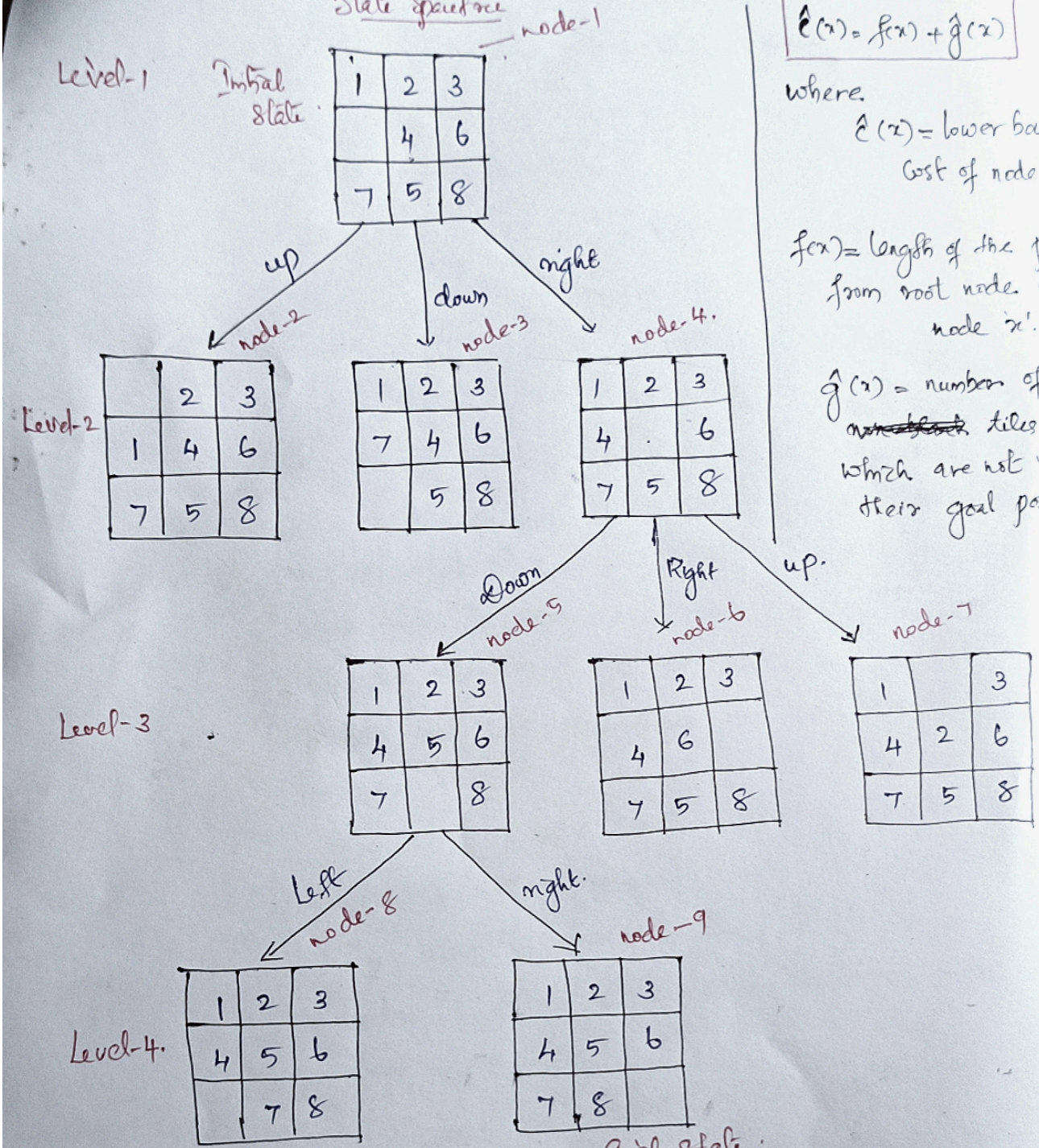
Consider the given example



Initial state Goal state.

8-Puzzle Problem.

State Space tree



Goal state.

$$\hat{c}(x) = f(x) + \hat{g}(x)$$

where,

$\hat{c}(x)$ = lower bound cost of node 'x'

$f(x)$ = length of the path from root node to node 'x'.

$\hat{g}(x)$ = number of ~~misplaced~~ tiles which are not in their goal position.

$\hat{c}(1) = 0 + 3 = 3$

$\hat{c}(2) = 1 + 4 = 5$

$\hat{c}(3) = 1 + 4 = 5$

$\hat{c}(4) = 1 + 2 = 3.$

$\hat{c}(5) = 2 + 1 = 3$

$\hat{c}(6) = 2 + 3 = 5$

$\hat{c}(7) = 2 + 3 = 5$

$\hat{c}(8) = 3 + 2 = 5$ ~~$3 + 5 = 8$~~

$\hat{c}(9) = 3 + 0 = 3$ ~~$3 + 4 = 7$~~