l_board):
(i,j)
(initial
(true)
late (
ution)
e, visited)
5,6},
tial_board)

2	3
5	6
	0

15-10-24.

- Solving 8-puzzle problem A* and Implementing Iterative Deeping Search Algorithm

Step 1:- Initialize the problem
- Considering {3x3} grid as the Initial state
- Considering {3x3} grid as the Goal state
- Assume the empty tile as '0'

Initial state

1	2	3
8		4
7	6	5

Goal state

2	8	1
	4	3
7	6	5

Step 2:- Defining the method A*
to solve the problem we use manhatten method to find distance b/w initial state and final state
distance += abs (i-goal_i) + abs (j-goal_j)
return distance
# get neighbour state
using moves [0 1] (1 0) (-1 0) (0 -1)
find neighbour state to present state
Priority queue
implementing priority queue to select (or) choose next move
Choose the lowest state

```
if (current state == final state):
else return path
~~else~~ find lowest (distance): move to lowest sub
# using back tracking to return path
path (){
    back track the move to print the path
}
```

initial stat

1	2	3
8	0	4
7	6	5

final state

2	8	1
0	4	3
7	6	5

priority queue

priority	state	H	V	Dist
2	1	2	0	2
1	2	1	0	1
1	3	0	1	1
1	4	1	0	1
0	5	0	0	0
0	6	0	0	0
0	7	0	0	0
2	8	4	4	8

highest Distance state has highest priority
lowest priority performs first

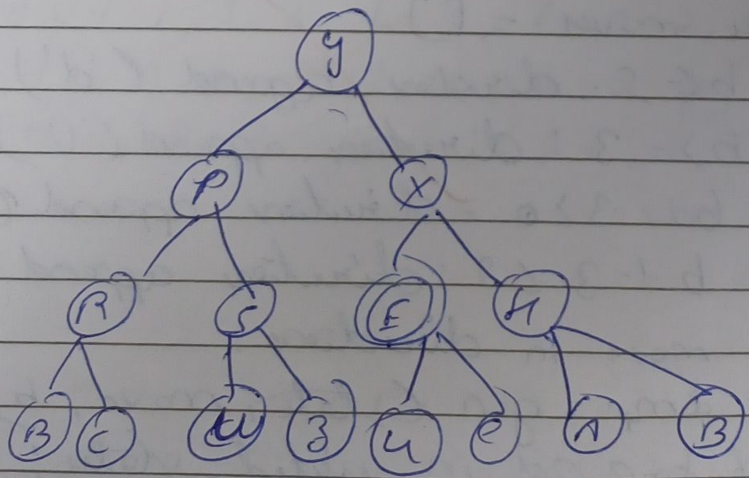
```
1 2 3      1 0 3      0 1 3      8 1 3      8 1 3
8 0 4  →   8 2 4  →   8 2 4  →   0 2 4  →   2 0 4
7 6 5      7 6 5      7 6 5      7 6 3      7 6 5

8 1 3      8 1 0      8 0 1      0 8 1      2 8 1
2 4 0  →   2 4 3  →   2 4 3  →   2 4 3  →   0 4 3
7 6 5      7 6 5      7 6 5      7 6 5      7 6 5
```

IDF

Step 1:- Initialize the tree with node and leaf node.
Mention initial node & destination node

```
# find destination node first
find(){
  using BFS method ();
    find level by level for
    destination node
  if present
        return level
    else
        go to next level
# find the parent node until reaches
  start node
find_parent(){
    Back track the path of current
    node to get parent and store it
    in the list if it is parent node
    return false
# Back track and print path Back track
destination to start node to print path
```

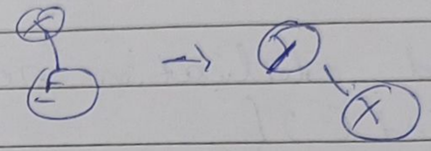


Date __/__/__
Page ____

initial (start node : Y
destination node : E

BFS :- level 1 : Y   False next level
level 2 : P, X   False next level
level 3 : R S (E)   Find E Break

Find Parent :



Back track ~~to~~ to print path
path :-

(Y) → (X) → (E)

15/10/24

```
def H_n (state, target):
    return sum (x != y for x, y in zip)
def F_D (state_with_level, target):
    state, level = state_with_level
return H_n (state, target) + level
def possible_moves (state_with_level, visited_states
state, level = state_with_level
    b = state.index(0)
    directions = []
    pos_moves = []
    if b <= 5: directions.append('d')
    if b >= 3: direction.append('U')
    if b % 3 > 0 : direction.append('l')
    if b % 3 < 2 : direction.append('r')
for move in directions:
    temp = gen (state, move, b)
  if temp not in visited_states:
  pos_moves.append (temp, level + 1))
    return pos_moves.
```

e next level
next level
E Break

e)

l, initial_state)

)
)

))

```
def gen (state, move, b):
        temp = state. copy (), if move == 'U':
temp if ~~temp~~ move == 'u': temp[a], tmp(b),
tmp[b-3] = temp[b-3, temp[b]
elif move == 'd': temp[b], temp[b+3] =
    temp[b+3], temp[b]
elif move == 'r': temp[b], temp[b+1] =
    temp[b+1], temp[b]
return tmp
Print ("No solution found")
```

Output

→ IDDFS

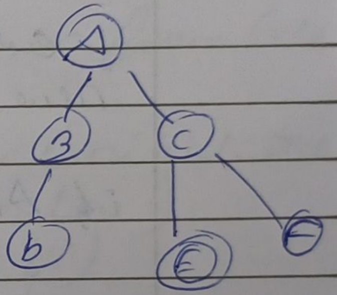
Iteration 1:

A -> B -> C ->

Iteration 2:

A -> B -> D -> C -> E ->

Target node E found



A*

Output

Start = (1, 2, 3, 4, 5, 6, 0, 7, 8)

goal = (1, 2, 3, 4, 5, 6, 7, 8, 0)

=> (1, 2, 3, 4, 5, 6, 0, 7, 8)

(1, 2, 3, 4, 5, 6, 7, 0, 8)

(1, 2, 3, 4, 5, 6, 7, 8, 0)

next level
next level
E Break

)

d, visited, start)

:)
:')

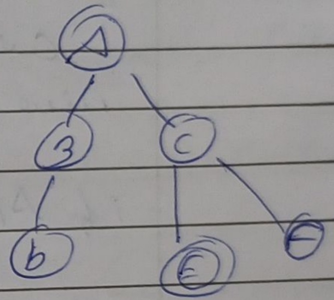
,))

```
def gen(state, move, b):
        temp = state. copy (), if move == 'U':
temp if dem move == 'u': temp[a], temp(b),
temp[b-3] ? temp[b-3, temp[b]
elif move == 'd': temp[b], temp[b+3] =
    temp[b+3], temp[b]
elif move == 'r': temp[b], temp[b+1] =
    temp[b+1], temp[b]
return temp
Print ("No solution found")
```

<u>Output</u>

<u>IDDFS</u>

Iteration 1:
A -> B -> C ->
Iteration 2:
A -> B -> D -> C -> E ->
Target node E found



A*

<u>Output</u>
Start = (1, 2, 3, 4, 5, 6, 0, 7, 8)
Goal = (1, 2, 3, 4, 5, 6, 7, 8, 0)

=> (1, 2, 3, 4, 5, 6, 0, 7, 8)
(1, 2, 3, 4, 5, 6, 7, 0, 8)
(1, 2, 3, 4, 5, 6, 7, 8, 0)