Date __/__/__
Page 10

08/10/24 8-puzzle problem using DFS and Manhattan Distance

Algorithm (DFS)

1. Initialize a list which stores goal state of the puzzle [[0, 1, 2]
[3, 4, 5] (const)
[6, 7, 8]]

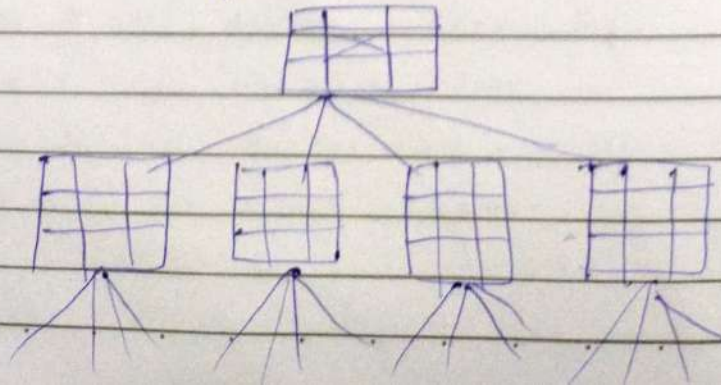
2. Take initial state of the puzzle (shuffled) as input from the user and store it in a variable.

~~3.~~ ~~Calculate Manhattan Distance:~~
~~m.d = abs(curr x - goal x) + abs(curr y - goal y)~~

4. Consider a root block to be moved at initial step.

5. Recursive function of DFS which should be called until it reaches goal state using ~~backtracking~~ brute force. each iteration

6. compare manhattan distance at each level and proceed further till goal state is reached. and manhattan distance of each block is 0.



Date __/__/__
Page _____

(Manhattan Distance)

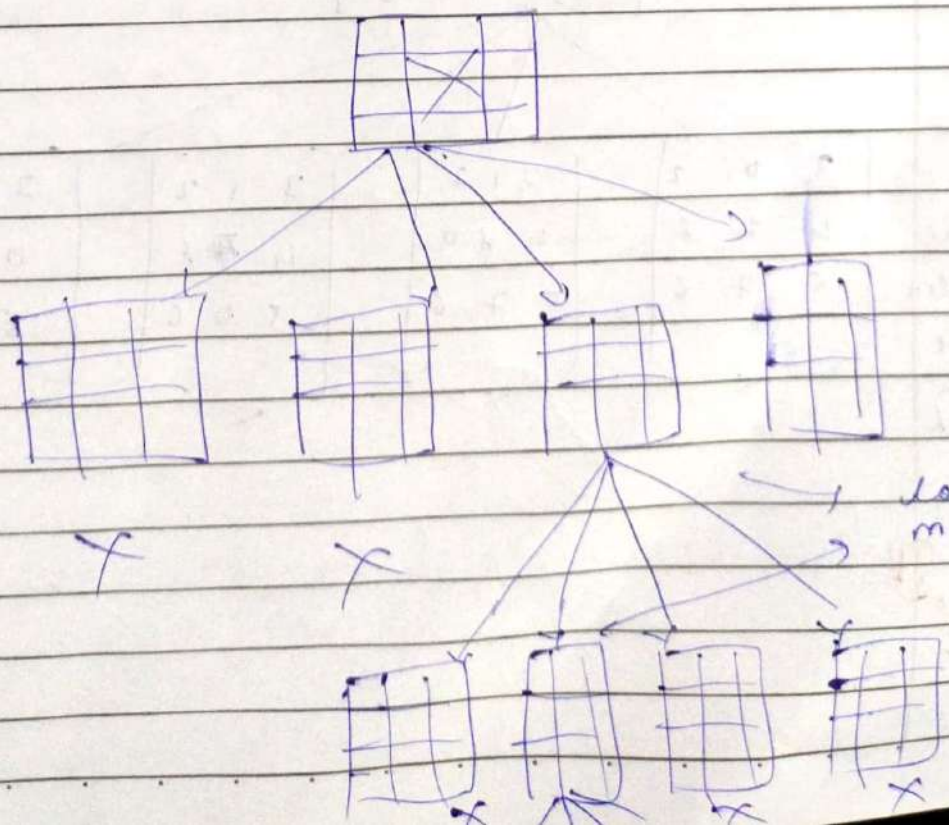
1. Initialize a list which stores goal state.

2. Initial state of shuffled puzzle as input from user and store it in a variable.

3. Calculate Manhattan Distance.
"Md = abs(curr x - goal x) + abs(curr y - goal y)

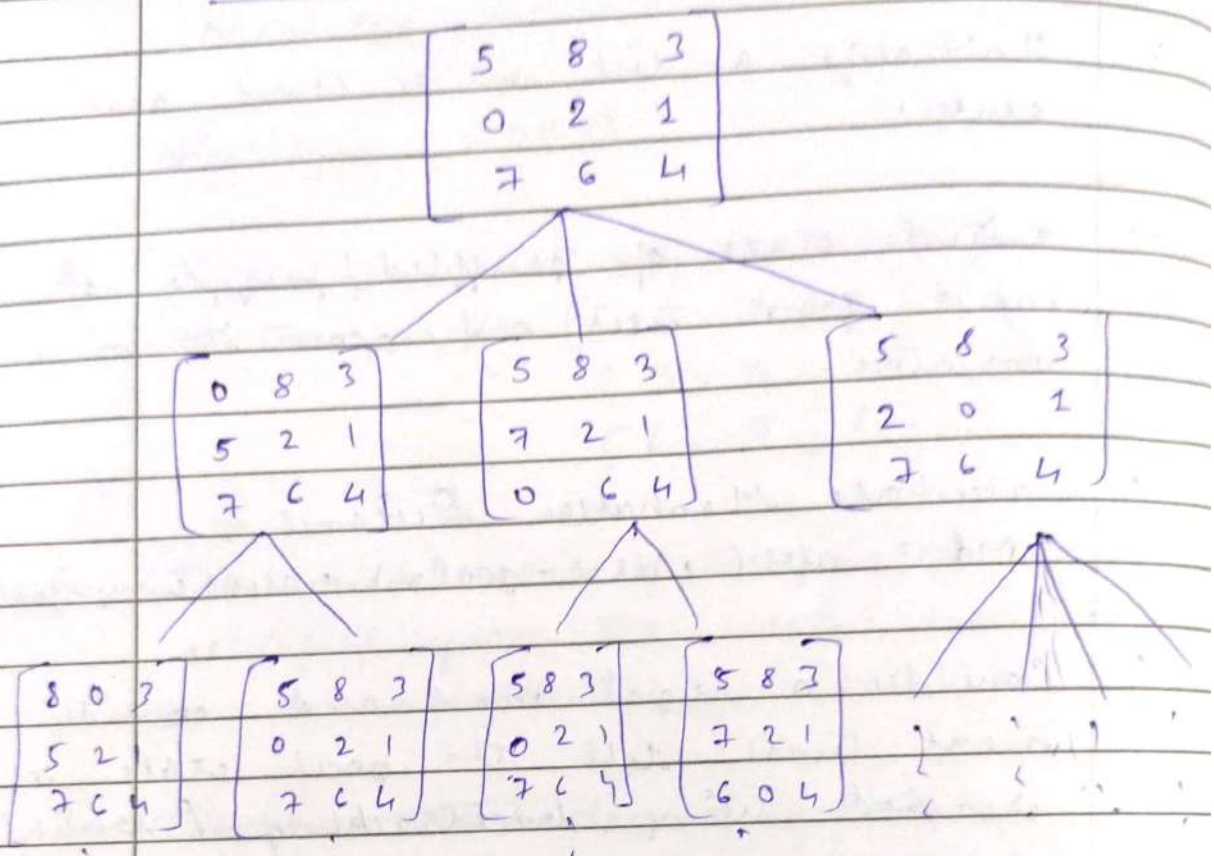
4. Consider a legal move and consider next level till the goal state is reached using backtracking / branch/ bound.

5. Using Manhattan distance and priority queue. optimize the level. at each state and move further based on minimum Manhattan distance.

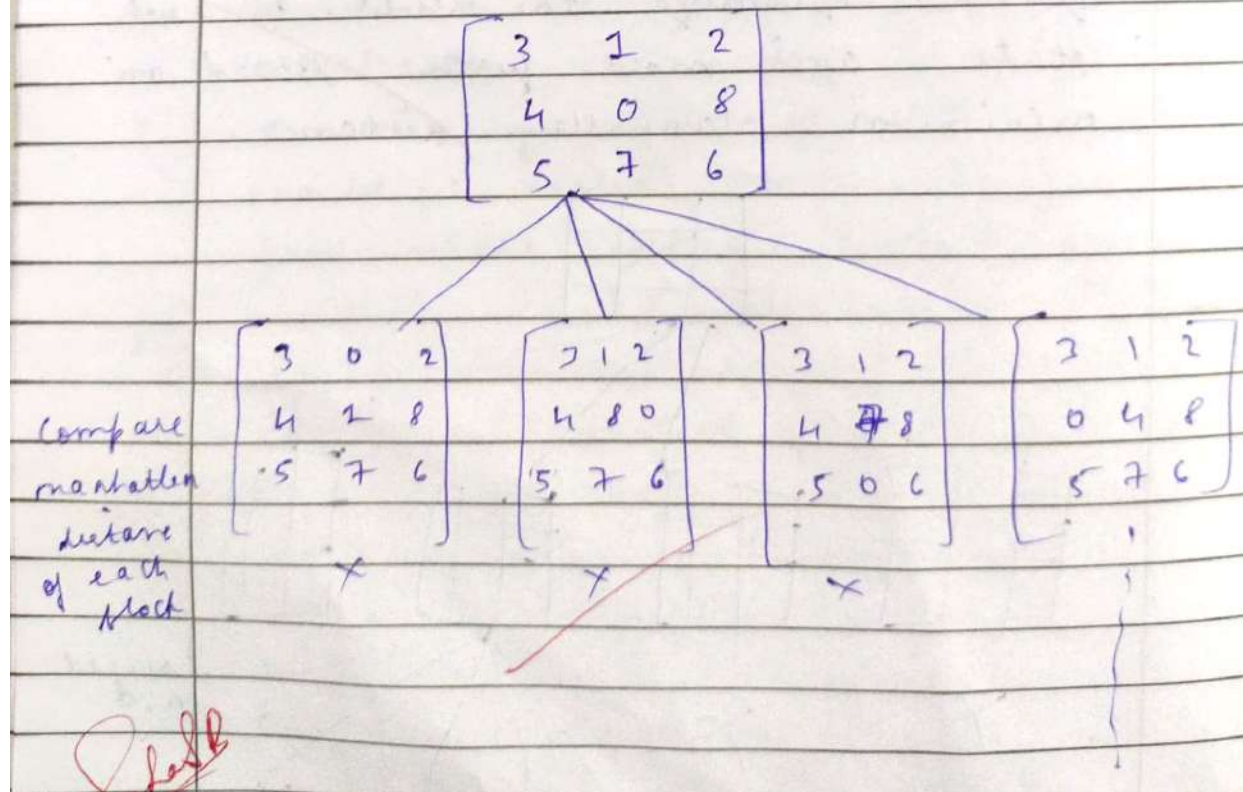


Date __/__/__
Page _____

DFS example



Manhattan Distance example



Date __/__/__
Page _____

Code:

```
import heapq
import numpy as np

goal = [[0,1,2],[3,4,5],[6,7,8]]
vis = set()
q = []
path = []

def manhattan(curr):
    ans = 0
    for i in range(3):
        for j in range(3):
            for k in range(3):
                for l in range(3):
                    if goal[i][j] == curr[k][l]:
                        ans += abs(i-k) + abs(l-j)
    return ans

def moves(curr):
    x = 0
    y = 0
    for i in range(3):
        for j in range(3):
            if curr[i][j] == 0:
                x = i
                y = j
                break
    pos = [[0,-1],[-1,0],[1,0],[0,1]]
    x1 = x
    y1 = y
    for pos in poss:
```

```
x += pos[0]
y += pos[1]
if 0 <= x < 3 and 0 <= y < 3:
    curr1 = [row.copy() for row in curr]
    curr1[x][y], curr1[x][y] =
    curr1[x][y], curr1[x][y]
    tuple_curr1 = tuple(map(tuple, curr1))
    if tuple_curr1 not in vis:
        heapq.heappush(q, (manhattan
        (curr1), curr1))
        vis.add(tuple_curr1)
x = x1
y = y1

def dfs(curr):
    if curr == goal:
        path.append(curr)
        return True
    moves(curr)
    if q:
        curr = heapq.heappop(q)[1]
        if dfs(curr):
            path.append(curr)
            return True
    return False

c = [[4,8,3],[5,0,6],[1,7,2]]
dfs(c)
print(np.array(c))
for state in reversed(path):
    print(np.array(state))
```