1.IMPLEMENTATION OF 8-PUZZLE

```
import numpy as np
import heapq

class PuzzleState:
    def __init__(self, board, moves=0):
        self.board = board
        self.blank_pos = self.find_blank(board)
        self.moves = moves

    def find_blank(self, board):
        return tuple(map(tuple, np.where(board == 0)))

    def get_neighbors(self):
        neighbors = []
        x, y = self.blank_pos
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Up, Down, Left, Right

        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < 3 and 0 <= ny < 3:
                new_board = self.board.copy()
                new_board[x, y], new_board[nx, ny] = new_board[nx, ny], new_board[x, y]
                neighbors.append(PuzzleState(new_board, self.moves + 1))
        return neighbors

    def is_goal(self):
        return np.array_equal(self.board, np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]]))

    def heuristic(self):
        return np.sum(np.abs(np.argwhere(self.board) - np.argwhere(np.array([[1, 2, 3], [4, 5, 6],
[7, 8, 0]]).flatten()))).sum(axis=1))

    def __lt__(self, other):
        return (self.moves + self.heuristic()) < (other.moves + other.heuristic())

def a_star_solver(initial_board):
    initial_state = PuzzleState(initial_board)
    open_set = []
    heapq.heappush(open_set, initial_state)
    visited = set()

    while open_set:
        current = heapq.heappop(open_set)
```

```
        if current.is_goal():
            return current.moves

        visited.add(tuple(map(tuple, current.board)))

        for neighbor in current.get_neighbors():
            if tuple(map(tuple, neighbor.board)) not in visited:
                heapq.heappush(open_set, neighbor)

    return -1  # No solution found

if __name__ == "__main__":
    # Initial configuration (0 represents the blank space)
    initial_board = np.array([[1, 2, 3],
                              [4, 0, 5],
                              [7, 8, 6]])

    result = a_star_solver(initial_board)
    if result != -1:
        print(f"Solved in {result} moves!")
    else:
        print("No solution exists.")
```

8/10/24
Tuesday

Lab 2

③ Solve 8 puzzle problems, Implement Iterative deepening Search algorithm.

1. Initialization:
   - Represent the puzzle as a 3x3 grid with 0 as the empty space.
   - Define the goal state where tiles are ordered from 1 to 8 with 0 in the bottom-right corner.
   - Define valid moves for the empty space: up, down, left, right.
2. Check goal:
   - Compare the current puzzle state with the goal state.
3. Get Empty position:
   - Locate the position of the empty space (0).
4. Move the Empty Space:
   - Try moving the empty space in all directions and generate new valid state.
5. Depth - limited Search (DLS):
   - Explore states up to a given depth limit.
   - If the goal isn't reach with the limit backtrace and try new paths.
6. Iterative Deepening Search (IDS)
   - Start with depth 0 and incrementally increase the depth limit.
   - perform DLS for each limit until the goal state is found.
7. Print Solution:
   - Once the goal is found, print the sequence of steps reaching from

the initial state to the Goal State

output:
Solution found in 14 steps.
Steps to reach the goal:

Step 0:

1	2	3
4	0	5
6	7	8

Step 2:

1	2	3
4	5	0
6	7	8

Step 3:

1	2	3
4	5	8
6	0	7

Step 4:

1	2	3
4	5	8
0	6	7

Step 5:

1	2	3
0	5	8
4	6	7

Step 6:

1	2	3
5	0	8
4	6	7

step 6:

1	2	3
5	0	8
4	6	7

Step 7:

1	2	3
5	6	8
4	6	7

step 8:

1	2	3
5	6	8
4	7	0

Step 9:

1	2	3
5	6	0
4	7	8

Step 10:

1	2	3
5	0	6
4	7	8

Step 11:

1	2	3
0	5	6
4	7	8

Step 12:

1	2	3
4	5	6
0	7	8

Step 13:

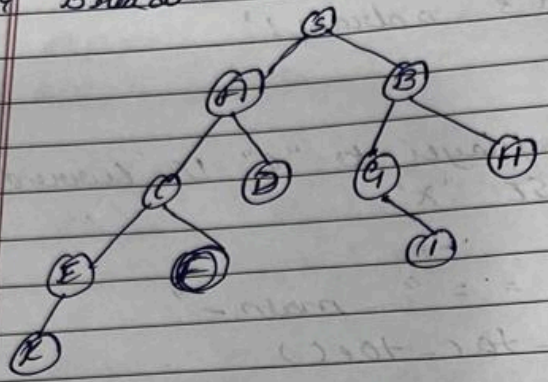
1	2	3
4	5	6
7	0	8

step 14:

1	2	3
4	5	6
7	8	0

Lab - 2

1) Breadth First Search



Frontier:

S ← [S]
A ← [A | B | C]
B ← [B | C | D | G]
C ← [C | D | G | H]
D ← [D | G | H | E]
G ← [G | H | E | F]
H ← [H | E | F | I]
E ← [E | F | I]
F ← [F | I | K]

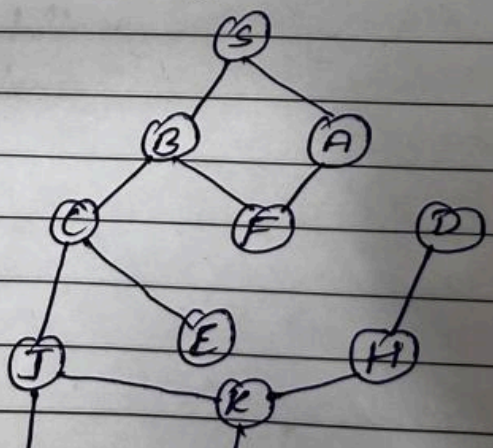
Explored:

[S]
[S | A]
[S | A | B]
[S | A | B | C | D]
[S | A | B | C | D | G]
[S | A | B | C | D | G | H]
[S | A | B | C | D | G | H]
[S | A | B | C | D | G | H | K]

Output:

S → A → B → C → D → G → H → K

2) Perform DFS



Frontier

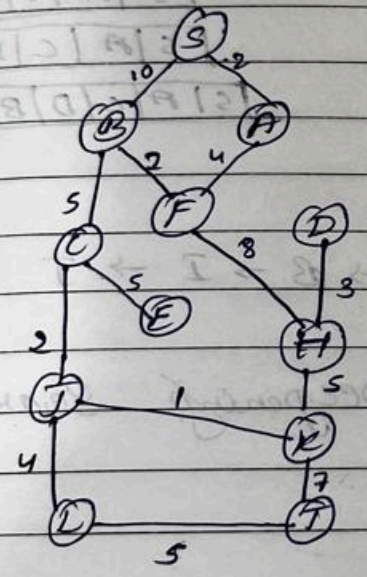
S
B	A
B	F
B	H
B	D	K
B	D	J	I

Explored:

S
S	A
S	A	F
B	D	H
B	D	H	E

output: S → B → F → H → K → T

(2) Perform uniform cost search



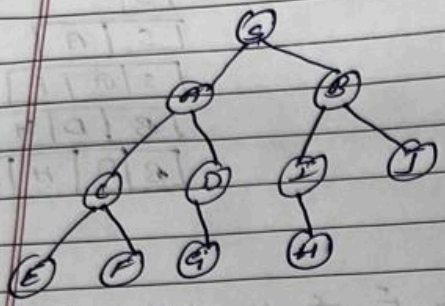
S(0)	A(2)	B(10)	F(6)	H(14)	C(15)
K(19)	D(17)	J(17)	E(20)	L(21)	T(26)

S(0)	A(2)	F(6)	B(10)	H(14)	C(15)
J(17)	K(19)	T(26)			

S → A → F → H → K → T =

2 + 4 + 8 + 5 + 7 = 26

(4) Depth Limited Search



Frontier

S

A | B

C | D | B

B

F | J

S

Explored :

S

S | A | C

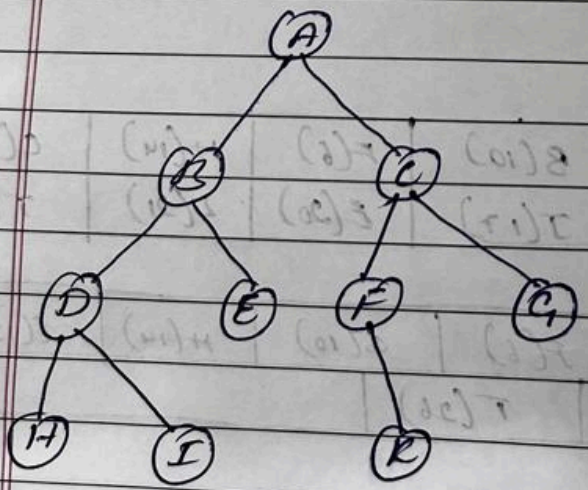
S | A | C | D

S | A | C | D | B

S | A | C | D | B | I | J

S → A → C → D → B → I → J

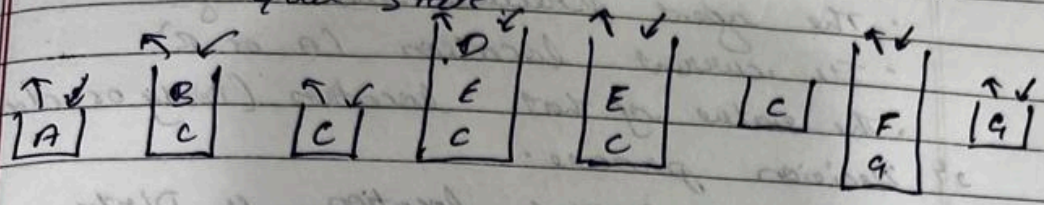
(5) Perform iterative Deepening Search



Iterative 0 : A → if it not Goal State : go to level 1

Iterative 1 : A → B → C it not goal state : go to level 2

Iterative 2 : A → B → C → D → E → F → G → Goal state



O/P : A → B → D → E → C → F → G.