Roll No: 101803186 Name: Kulpreet Singh Group: COE9

Computer Science and Engineering Department

Artificial Intelligence (UCS-521)

Lab Assignment-3

Note: As a data scientist, you have been assigned a job to solve the 8 puzzle problem. To generate the states of the search space, you need to define the rules/operators properly. As a solution, you need to print the intermediate steps of the solution as well as total number of moves used to achieve the goal state.

1. If the initial and final states are as below and H(n): number of misplaced tiles in the current state n as compared to the goal node need to be considered as the heuristic function. You need to use **Best First Search** algorithm.

Initial:

2		3
1	8	4
7	6	5

Goal:

1	2	3
8		4
7	6	5

CODE:

```
import copy

initial_arr = [[2,0,3],[1,8,4],[7,6,5]]
final_arr = [[1,2,3],[8,0,4],[7,6,5]]

#All possible moves
# up = (-1,0)
# down = (1,0)
# left = (0,-1)
# right = (0,1)

moves = [(-1,0),(1,0),(0,-1),(0,1)]
movesName = ['UP', 'DOWN', 'LEFT', 'RIGHT']

#checking valid moves
def isValidMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    if i<len(initial_arr) and i>=0 and j>=0 and j<len(initial_arr):
        return True
    return False

def performMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    temp_arr = copy.deepcopy(initial_arr)
    temp = temp_arr[i][j]
    temp_arr[i][j] = temp_arr[idx[0]][idx[1]]
    temp_arr[idx[0]][idx[1]] = temp
    return temp_arr

def findZeroIndex(initial_arr):
    for i in range(0,len(initial_arr)):
        for j in range(0,len(initial_arr[i])):
            if initial_arr[i][j] == 0:
                return i,j
```

```python
def enqueue(s,val):
    global q
    q = q + [(val,s)]


def dequeue():
    global q
    global visited

    q.sort()
    visited = visited + [q[0][1]]

    temp = q[0][1]
    del q[0]
    return (temp)

def heuristic(initial_arr, final_arr):
    count = 0
    for p in range(3):
        for q in range(3):
            if initial_arr[p][q]!=0:
                if initial_arr[p][q]!=final_arr[p][q]:
                    count = count+1
    return count

def findSol(initial_arr, final_arr):
    global visited
    global q
    enqueue(initial_arr, heuristic(initial_arr, final_arr))
    if initial_arr == final_arr:
        return
    while True:
        if len(q)>0:
            curr_state = dequeue()
        else:
            print('Not Found')
            return
        idx = findZeroIndex(curr_state)
        for move in moves:
            if isValidMove(curr_state, idx, move):
                new_arr = performMove(curr_state, idx, move)
                if new_arr == final_arr:
                    print('Solution Found!! Intermediate states are:')
                    print(visited+[new_arr])
                    return
                if new_arr not in visited:
                    h = heuristic(new_arr,final_arr)
                    enqueue(new_arr, h)


def main():
    global q
    global visited
    visited = []
    q=[]
    findSol(initial_arr, final_arr)

if __name__ == '__main__':
    main()
```

OUTPUT:

```
8-Puzzle_bestFirst
C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kulpr/PycharmProjects/OpenCVpython/8-Puzzle_bestFirst.py
Solution Found!! Intermediate states are:
[[[2, 0, 3], [1, 8, 4], [7, 6, 5]], [[0, 2, 3], [1, 8, 4], [7, 6, 5]], [[1, 2, 3], [0, 8, 4], [7, 6, 5]], [[1, 2, 3], [8, 0, 4], [7, 6, 5]]]

Process finished with exit code 0

TODO   6: Problems   Terminal   Python Console
02 expected 2 blank lines, found 1
```

2. If the initial and final states have been changed as below and approach you need to use is **Hill Climbing searching algorithm**. H(n): number of misplaced tiles in the current state n as compared to the goal node as the heuristic function for the following states.

2	8	3
1	5	4
7	6	

Initial State

1	2	3
8		4
7	6	5

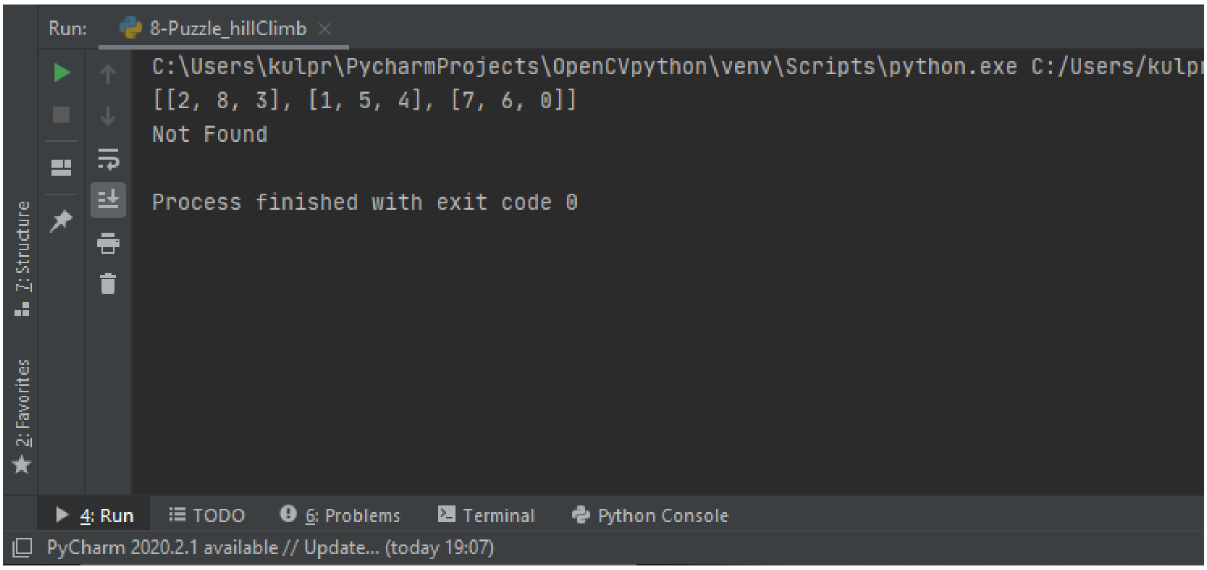
Final State

CODE:

```
import copy

initial_arr = [[2,8,3],[1,5,4],[7,6,0]]
final_arr = [[1,2,3],[8,0,4],[7,6,5]]

#All possible moves
# up = (-1,0)
# down = (1,0)
# left = (0,-1)
# right = (0,1)

moves = [(-1,0),(1,0),(0,-1),(0,1)]
movesName = ['UP', 'DOWN', 'LEFT', 'RIGHT']

#checking valid moves
def isValidMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    if i<len(initial_arr) and i>=0 and j>=0 and j<len(initial_arr):
        return True
    return False

def performMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    temp_arr = copy.deepcopy(initial_arr)
    temp = temp_arr[i][j]
    temp_arr[i][j] = temp_arr[idx[0]][idx[1]]
```

```
    temp_arr[idx[0]][idx[1]] = temp
    return temp_arr

def findZeroIndex(initial_arr):
    for i in range(0,len(initial_arr)):
        for j in range(0,len(initial_arr[i])):
            if initial_arr[i][j] == 0:
                return i,j


def heuristic(initial_arr, final_arr):
    count = 0
    for p in range(3):
        for q in range(3):
            if initial_arr[p][q]!=0:
                if initial_arr[p][q]!=final_arr[p][q]:
                    count = count+1
    return count

def findSol(initial_arr, final_arr):
    global visited
    print(initial_arr)
    visited.append(initial_arr)
    H = heuristic(initial_arr, final_arr)
    if H == 0:
        return 0
    best_h = heuristic(initial_arr, final_arr)
    best_arr = initial_arr
    bestMove = (0,0)
    idx = findZeroIndex(initial_arr)
    flag = 0
    for move in moves:
        if isValidMove(initial_arr, idx, move):
            new_arr = performMove(initial_arr, idx, move)
            if new_arr not in visited:
                h = heuristic(new_arr,final_arr)
                if h < best_h:
                    flag = 1;
                    best_h = h
                    bestMove = move
                    best_arr = new_arr

    if flag == 0:
        print('Not Found')
        return -1000000
    ind = moves.index(bestMove)
    print(f'Move = {movesName[ind]}. best_h = {best_h}')
    return findSol(best_arr, final_arr) + 1


def main():
    global visited
    visited = []
    noOfMoves = findSol(initial_arr, final_arr)
    if noOfMoves >= 0:
        print(f'Goal State!! Number of moves required = {noOfMoves}')

if __name__ == '__main__':
    main()
```

OUTPUT:



3. Apply **A* searching algorithm** by taking H(n): number of correctly placed tiles in the current state n as compared to the goal node. as the heuristic function.

Initial:

2		3
1	8	4
7	6	5

Goal:

1	2	3
8		4
7	6	5

CODE:

```
import copy

initial_arr = [[2,0,3],[1,8,4],[7,6,5]]
final_arr = [[1,2,3],[8,0,4],[7,6,5]]

#All possible moves
# up = (-1,0)
# down = (1,0)
# left = (0,-1)
# right = (0,1)

moves = [(-1,0),(1,0),(0,-1),(0,1)]
movesName = ['UP', 'DOWN', 'LEFT', 'RIGHT']

#checking valid moves
def isValidMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    if i<len(initial_arr) and i>=0 and j>=0 and j<len(initial_arr):
        return True
    return False

def performMove(initial_arr, idx, move):
    i = idx[0] + move[0]
    j = idx[1] + move[1]
    temp_arr = copy.deepcopy(initial_arr)
    temp = temp_arr[i][j]
    temp_arr[i][j] = temp_arr[idx[0]][idx[1]]
    temp_arr[idx[0]][idx[1]] = temp
```

```
    return temp_arr

def findZeroIndex(initial_arr):
    for i in range(0,len(initial_arr)):
        for j in range(0,len(initial_arr[i])):
            if initial_arr[i][j] == 0:
                return i,j

def enqueue(s,val):
    global q
    q = q + [(val,s)]


def dequeue():
    global q
    global visited

    q.sort(reverse = True)
    visited = visited + [q[0][1]]

    temp = q[0][1]
    del q[0]
    return (temp)

def h_val(curr_state, final_arr):
    count = 0
    for p in range(3):
        for q in range(3):
            if curr_state[p][q]!=0 and curr_state[p][q] == final_arr[p][q]:
                count = count + 1
    return count

def g_val(curr_state, initial_arr):
    count = 0
    for p in range(3):
        for q in range(3):
            if curr_state[p][q]!=0:
                if curr_state[p][q] == initial_arr[p][q]:
                count = count + 1
    return count

def heuristic(initial_arr, curr_state,  final_arr):
    return (h_val(curr_state, final_arr) + g_val(curr_state, initial_arr))


def findSol(initial_arr, final_arr):
    global visited
    global q
    enqueue(initial_arr, heuristic(initial_arr, initial_arr, final_arr))
    if initial_arr == final_arr:
        return
    while True:
        if len(q)>0:
            curr_state = dequeue()
        else:
            print('Not Found')
            return
        idx = findZeroIndex(curr_state)
        for move in moves:
            if isValidMove(curr_state, idx, move):
                new_arr = performMove(curr_state, idx, move)
```

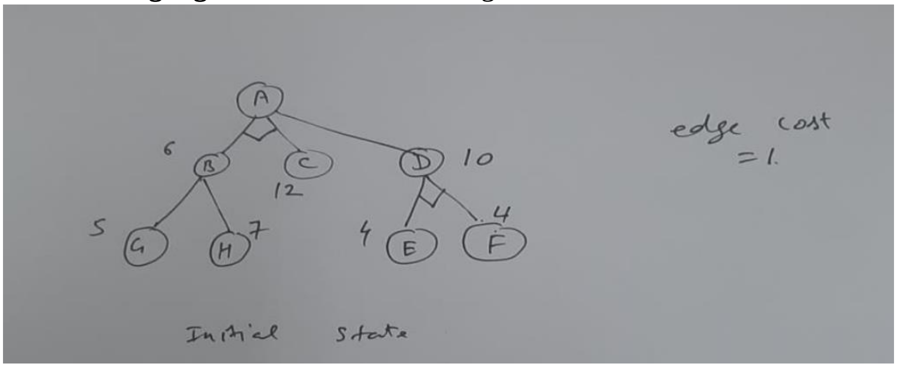
```
                if new_arr == final_arr:
                    print('Solution Found!! Intermediate states are:')
                    print(visited+[new_arr])
                    return
                if new_arr not in visited:
                    h = heuristic(initial_arr, new_arr,final_arr)
                    enqueue(new_arr, h)


def main():
    global q
    global visited
    visited = []
    q=[]
    findSol(initial_arr, final_arr)

if __name__ == '__main__':
    main()
```

OUTPUT:

```
8-puzzle_Astar
C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kulpr/PycharmProjects/OpenCVpython/8-puzzle_Astar.py
Solution Found!! Intermediate states are:
[[[2, 0, 3], [1, 8, 4], [7, 6, 5]], [[0, 2, 3], [1, 8, 4], [7, 6, 5]], [[1, 2, 3], [0, 8, 4], [7, 6, 5]], [[1, 2, 3], [8, 0, 4], [7, 6, 5]]]

Process finished with exit code 0
```

TODO 6: Problems Terminal Python Console
riable 'visited' is undefined at the module level

4. Apply **AO* searching algorithm** on the following search tree.

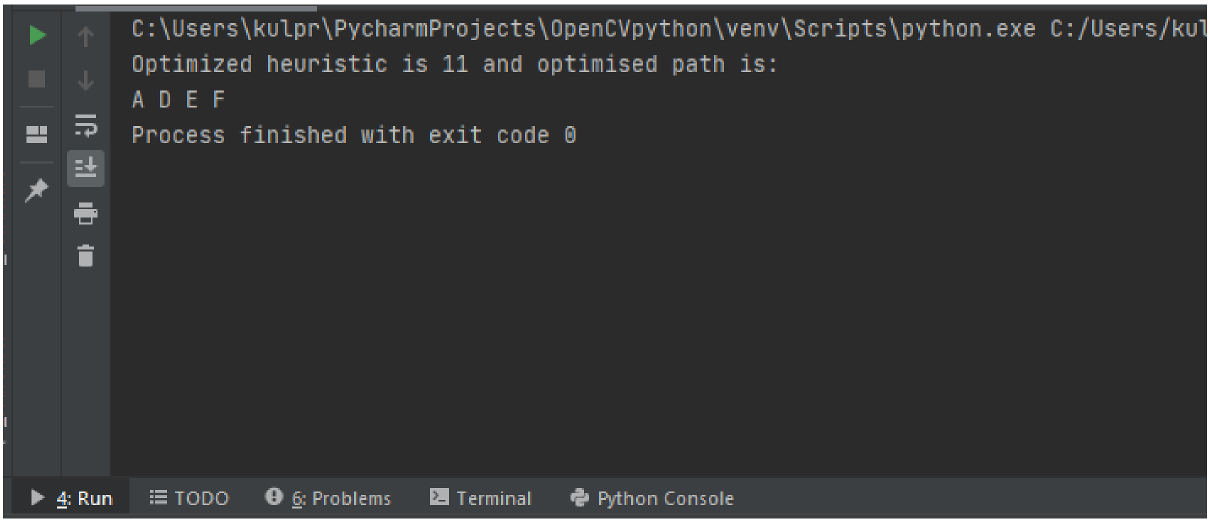


CODE:

```
def optimizePath(root):
    global visited
    global graph
    q = []
    # enqueue(q,[root],graph[root][1])
    for children in graph[root][2]:
        heu = 0
        for child in children:
            heu = heu + graph[child][1]
        q = q + [(heu,children)]
```

```
    if len(q)==0:
        return graph[root][1]
    graph[root][1] = 100
    while True:
        if len(q)>0:
            q.sort()
            visited.append(q[0][1])
            curr_children = q[0][1]
            del q[0]
        else:
            return graph[root][1]
        curr_heu = 0
        for curr_child in curr_children:
            curr_heu = curr_heu + 1 + optimizePath(curr_child)
        if (curr_heu < graph[root][1]):
            graph[root][1] = curr_heu

def getPath(root):
    global graph
    q = []
    for children in graph[root][2]:
        heu = 0
        for child in children:
            heu = heu + graph[child][1]
        q = q + [(heu, children)]
    print(graph[root][0], end=' ')
    if len(q) > 0:
        q.sort()
        curr_children = q[0][1]
        del q[0]
        for curr_child in curr_children:
            getPath(curr_child)

def AOstar(root):
    global graph
    minHue = optimizePath(root)
    print(f'Optimized heuristic is {minHue} and optimised path is: ')
    getPath(root)

def main():
    global visited
    visited = []
    global graph
    # graph element: idx, heu, child
    graph = [['A',100,[[1,2],[3]]],
             ['B',6,[[6],[7]]],
             ['C',12,[]],
             ['D',10,[[4,5]]],
             ['E',4,[]],
             ['F',4,[]],
             ['G',5,[]],
             ['H',7,[]]]
    root = 0
    visited.append([root])
    AOstar(root)

if __name__ == '__main__':
    main()
```

OUTPUT:


C:\Users\kulpr\PycharmProjects\OpenCVpython\venv\Scripts\python.exe C:/Users/kul
Optimized heuristic is 11 and optimised path is:
A D E F
Process finished with exit code 0
4: Run
TODO
6: Problems
Terminal
Python Console