CSC425 Assignment 3

Mir Shafayat Ahmed 1910456

In this problem you are to output the solution to the 8-puzzle problem, i.e., to find the sequence of moves from some initial configuration/state of the puzzle to a final state. As for the moves consider moving the blank tile moving UP, DOWN, LEFT, RIGHT in that order. Each move has cost = 1.

Figure 1 below shows example initial and final states.

Initial State:

1	4	2
	5	3
6	7	8

Final State:

1	2	
5	4	3
6	7	8

Figure 1: an example initial and final state

Question 1

1. [10 points] Show by hand the execution of A* search algorithm. Initially the frontier will have the initial state (name it state 0) with its f-value. Number the subsequent states and compute their f-values. In each step show: a. which path is selected, b. whether it is the optimal path, and c. the contents of the frontier at the completion of the step.

The heuristic function to be used: **h(n) = number of misplaced tiles**, where n is a state. For example, for the initial state, h(initial) = 3 (as tiles 2, 4 and 5 are not in the correct positions)

Order : L -> U -> R -> D

Goal		
1	2	_
5	4	3
6	7	8

S0	f=0+3	Initial
1	4	2
_	5	3
6	7	8

```
Frontier:
    S0 : 0+3 = 3
```

S1	f=1+4	U	S2	f=1+2	R	S3	f=1+4	D
_	4	2	1	4	2	1	4	2
1	5	3	5	_	3	6	5	3
6	7	8	6	7	8	_	7	8

```
Frontier:
    S0 -> S2 : 1+2 = 3
    S0 -> S1 : 1+3 = 4
    S0 -> S3 : 1+3 = 4
```

S1	f=1+4	U	S21	f=2+1	U	S22	f=2+3	R	S23	f=2+3	D	S3	f=1+4	D
_	4	2	1	_	2	1	4	2	1	4	2	1	4	2
1	5	3	5	4	3	5	3	_	5	7	3	6	5	3
6	7	8	6	7	8	6	7	8	6	_	8	_	7	8

```
Frontier:
    S0 -> S2 -> S21 : 2+1 = 3
    S0 -> S1 : 1+3 = 4
    S0 -> S3 : 1+3 = 4
    S0 -> S2 -> S22 : 2+3 = 5
    S0 -> S2 -> S23 : 2+3 = 5
```

S1	f=1+4	U	S22	f=2+3	R	S23	f=2+3	D	S3	f=1+4	D
_	4	2	1	4	2	1	4	2	1	4	2
1	5	3	5	3	_	5	7	3	6	5	3
6	7	8	6	7	8	6	_	8	_	7	8

S211	f=3+2	L	S212	f=3+0	R
_	1	2	1	2	_
5	4	3	5	4	3
6	7	8	6	7	8

```
Frontier:
    S0 -> S2 -> S21 -> S212 : 3+0 = 3 #Goal found. Therefore Search Stopped.
    S0 -> S1 : 1+3 = 4
    S0 -> S3 : 1+3 = 4
    S0 -> S2 -> S21 -> S211 : 3+2 = 5
    S0 -> S2 -> S22 : 2+3 = 5
    S0 -> S2 -> S23 : 2+3 = 5
```

Question 2

2. [30 points] In this problem you will implement the Depth first Branch and Bound algorithm.

a. [5 points] Write a pseudo-code of the "generic" Depth First Branch and Bound algorithm. Update the generic search algorithm presented in class for this.

b. [25 points] Write a code to implement the algorithm to solve the 8-puzzle problem.

Input format: To input the instance of the problem, assume the cell numbers are as shown in figure 2. Then the initial state in figure 1 can be input by the following sequence: 1 4 2 -1 5 3 6 7 8, where the i-th cell of the puzzle contains the i-th number in the sequence and -1 represents the blank tile. The final state of the example above can be input as: 1 2 -1 5 4 3 6 7 8.

cell#1	cell#2	cell#3
cell#4	cell#5	cell#6
cell#7	cell#8	cell#9

Figure2: Cell numbering

```
create a stack
```

```
push start node into stack
find children of start node
calculate f-value for each child
push the children of the start node such that lowest
```

```python
from cgi import print_form
from dataclasses import dataclass
from importlib.resources import path
import numpy as np
from collections import deque
from bisect import insort_left

@dataclass
class PuzzleState():
    actions = ["Start"]
    def __init__(self, grid, goalState=None, cost=0, actions=None) -> None:
        self.grid = np.array(grid)
        x,y = np.where(self.grid == -1)
        self.bXY = [x[0], y[0]]
        self.costToReachState = cost

        if actions is not None:
            # print(actions)
            self.actions = actions
        else: actions = ["Start"]
        if goalState is None:
            self.h=0
        else:
            self.h = self.heuristic(goalState)
            self.goalState = goalState
        self.f = self.costToReachState + self.h

    def __lt__(self, other):
        return self.f < other.f

    def left(self):
        if self.bXY[1] == 0 or self.actions[-1]=="Right":
            return None
        else:
            newGrid = self.grid.copy()
            newGrid[self.bXY[0]][self.bXY[1]] = newGrid[self.bXY[0]][self.bXY[1]-1]
            newGrid[self.bXY[0]][self.bXY[1]-1] = -1
            newAction = self.actions.copy()
            newAction.append("Left")
            return PuzzleState(newGrid, self.goalState, self.costToReachState+1, newAction)

    def up(self):
        if self.bXY[0] == 0 or self.actions[-1]=="Down":
            return None
        else:
            newGrid = self.grid.copy()
            newGrid[self.bXY[0]][self.bXY[1]] = newGrid[self.bXY[0]-1][self.bXY[1]]
            newGrid[self.bXY[0]-1][self.bXY[1]] = -1
            newAction = self.actions.copy()
            newAction.append("Up")
            return PuzzleState(newGrid, self.goalState, self.costToReachState+1, newAction)

    def right(self):
        if self.bXY[1] == len(self.grid[0]) -1 or self.actions[-1]=="Left":
            return None
        else:
            newGrid = self.grid.copy()
```

```
            newGrid[self.bXY[0]][self.bXY[1]] = newGrid[self.bXY[0]][self.bXY[1]+1]
            newGrid[self.bXY[0]][self.bXY[1]+1] = -1
            newAction = self.actions.copy()
            newAction.append("Right")
            return PuzzleState(newGrid, self.goalState, self.costToReachState+1, newAction)

    def down(self):
        if self.bXY[0] == len(self.grid)-1 or self.actions[-1]=="Up":
            return None
        else:
            newGrid = self.grid.copy()
            newGrid[self.bXY[0]][self.bXY[1]] = newGrid[self.bXY[0]+1][self.bXY[1]]
            newGrid[self.bXY[0]+1][self.bXY[1]] = -1
            newAction = self.actions.copy()
            newAction.append("Down")
            return PuzzleState(newGrid, self.goalState, self.costToReachState+1, newAction)

    def listChildren(self):
        children = deque()
        if leftState := self.left(): insort_left(children, leftState)
        if upState := self.up(): insort_left(children, upState)
        if rightState := self.right(): insort_left(children, rightState)
        if downState := self.down(): insort_left(children, downState)
        return children

    def heuristic(self, goalState, method="misplaced") -> int:
        if method=="misplaced":
            x, y = goalState.bXY[0], goalState.bXY[1]
            sub1 = False if goalState.grid[x][y] == self.grid[x][y] else True
            diff = goalState.grid == self.grid
            return 9-np.sum(diff)-sub1

class EightPuzzle:
    def __init__(self, startState, goalState, actionCost = 1) -> None:
        self.goalState = PuzzleState(goalState)
        self.startState = PuzzleState(startState, self.goalState)
        self.actionCost = actionCost

    def isGoal(self, state):
        return np.array_equal(self.goalState.grid, state.grid)

    def solve(self, method = "DFBBS", verbose = False, maxIterations = 10000):
        stack = deque()
        upperbound = np.inf
        stack.append(self.startState)
        solution = None

        while(stack and maxIterations>0):
            maxIterations-=1
            topOfStack = stack.pop()

            if verbose:
                print("=====================================================")
                print("====================== CHOSING ======================")
                print(f"{topOfStack.costToReachState} + {topOfStack.h} = {topOfStack.f}\n{topOfStack.actions}\n{


            if self.isGoal(topOfStack) and topOfStack.f < upperbound:
                upperbound = topOfStack.f

                if verbose:  print(f"A Goal Has Been Found, Upperbound = {upperbound}")
```

```python
                solution = topOfStack.actions
                while stack:
                    candidate = stack.pop()
                    if candidate.f<upperbound:
                        topOfStack = candidate
                        break
                if not stack:
                    if verbose:  print("Optimal Path Found")
                    return solution

            children = topOfStack.listChildren()

            while(children):
                stack.append(children.pop())


            if verbose:
                print("=====================================================")
                print("==================== ALL OPTIONS ====================")
                for _ in reversed(stack):
                    print(f"{_.costToReachState} + {_.h} = {_.f}\n{_.actions}\n{_.grid}\n")
# goalState = PuzzleState([
#              [1, 2,-1],
#              [5, 4, 3],
#              [6, 7, 8]
#          ]
#     )
# state1 = PuzzleState(
#     [
#         [1, 4, 2],
#         [5,-1, 3],
#         [6, 7, 8]
#     ],
#     goalState
# )
# print(state1.actions)
# print(state1.grid)
# print()
# print()

# children = state1.listChildren()
# for i in children:
#     print(f"{i.costToReachState} + {i.h} = {i.f}")
#     print(i.actions)
#     print(i.grid)
#     print()
#     children2 = i.listChildren()
#     for j in children2:
#         print(f"{j.costToReachState} + {j.h} = {j.f}")
#         print(j.actions)
#         print(j.grid)
#         print()
#     print()
#     print()

goalState = [
        [1, 2,-1],
        [5, 4, 3],
        [6, 7, 8]
    ]
```

```python
puzzle1 = EightPuzzle(
    startState = [
        [1, 4, 2],
        [-1,5, 3],
        [6, 7, 8]
    ],
    goalState = goalState
)
puzzle2 = EightPuzzle(
    startState = [
        [1, 4, 2],
        [6, 5, 3],
        [-1,7, 8]
    ],
    goalState = goalState
)
puzzle3 = EightPuzzle(
    startState = [
        [1, 2, 3],
        [-1,5, 8],
        [6, 4, 7]
    ],
    goalState = goalState
)
puzzle4 = EightPuzzle(
    startState = [
        [1, 2,-1],
        [5, 4, 3],
        [8, 7, 6]
    ],
    goalState = goalState
)
puzzle5 = EightPuzzle(
    startState = [
        [1, 4, 2],
        [6, 5, 3],
        [-1,7, 8]
    ],
    goalState = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8,-1]
    ]
)
```

```python
print(puzzle1.solve(verbose=True))
```

```
=====================================================
====================== CHOSING ======================
0 + 3 = 3
['Start']
[[ 1  4  2]
 [-1  5  3]
 [ 6  7  8]]

=====================================================
==================== ALL OPTIONS ====================
1 + 2 = 3
['Start', 'Right']
[[ 1  4  2]
 [ 5 -1  3]
```

```
 [ 6  7  8]]

1 + 4 = 5
['Start', 'Down']
[[ 1  4  2]
 [ 6  5  3]
 [-1  7  8]]

1 + 4 = 5
['Start', 'Up']
[[-1  4  2]
 [ 1  5  3]
 [ 6  7  8]]

=======================================================
====================== CHOSING ======================
1 + 2 = 3
['Start', 'Right']
[[ 1  4  2]
 [ 5 -1  3]
 [ 6  7  8]]

=======================================================
==================== ALL OPTIONS ====================
2 + 1 = 3
['Start', 'Right', 'Up']
[[ 1 -1  2]
 [ 5  4  3]
 [ 6  7  8]]

2 + 3 = 5
['Start', 'Right', 'Down']
[[ 1  4  2]
 [ 5  7  3]
 [ 6 -1  8]]

2 + 3 = 5
['Start', 'Right', 'Right']
[[ 1  4  2]
 [ 5  3 -1]
 [ 6  7  8]]

1 + 4 = 5
['Start', 'Down']
[[ 1  4  2]
 [ 6  5  3]
 [-1  7  8]]

1 + 4 = 5
['Start', 'Up']
[[-1  4  2]
 [ 1  5  3]
 [ 6  7  8]]

=======================================================
====================== CHOSING ======================
2 + 1 = 3
['Start', 'Right', 'Up']
[[ 1 -1  2]
 [ 5  4  3]
 [ 6  7  8]]
```

```
=====================================================
==================== ALL OPTIONS ====================
3 + 0 = 3
['Start', 'Right', 'Up', 'Right']
[[ 1  2 -1]
 [ 5  4  3]
 [ 6  7  8]]

3 + 2 = 5
['Start', 'Right', 'Up', 'Left']
[[-1  1  2]
 [ 5  4  3]
 [ 6  7  8]]

2 + 3 = 5
['Start', 'Right', 'Down']
[[ 1  4  2]
 [ 5  7  3]
 [ 6 -1  8]]

2 + 3 = 5
['Start', 'Right', 'Right']
[[ 1  4  2]
 [ 5  3 -1]
 [ 6  7  8]]

1 + 4 = 5
['Start', 'Down']
[[ 1  4  2]
 [ 6  5  3]
 [-1  7  8]]

1 + 4 = 5
['Start', 'Up']
[[-1  4  2]
 [ 1  5  3]
 [ 6  7  8]]

=====================================================
====================== CHOSING ======================
3 + 0 = 3
['Start', 'Right', 'Up', 'Right']
[[ 1  2 -1]
 [ 5  4  3]
 [ 6  7  8]]

A Goal Has Been Found, Upperbound = 3
Optimal Path Found
['Start', 'Right', 'Up', 'Right']
```

```
print(puzzle2.solve())
```

```
['Start', 'Up', 'Right', 'Up', 'Right']
```

```
print(puzzle3.solve(verbose=True))
```

```
=====================================================
====================== CHOSING ======================
0 + 5 = 5
['Start']
[[ 1  2  3]
 [-1  5  8]
```

```
 [ 6  4  7]]

======================================================
==================== ALL OPTIONS =====================
1 + 4 = 5
['Start', 'Right']
[[ 1  2  3]
 [ 5 -1  8]
 [ 6  4  7]]

1 + 6 = 7
['Start', 'Down']
[[ 1  2  3]
 [ 6  5  8]
 [-1  4  7]]

1 + 6 = 7
['Start', 'Up']
[[-1  2  3]
 [ 1  5  8]
 [ 6  4  7]]

======================================================
====================== CHOSING =======================
1 + 4 = 5
['Start', 'Right']
[[ 1  2  3]
 [ 5 -1  8]
 [ 6  4  7]]

======================================================
==================== ALL OPTIONS =====================
2 + 3 = 5
['Start', 'Right', 'Down']
[[ 1  2  3]
 [ 5  4  8]
 [ 6 -1  7]]

2 + 4 = 6
['Start', 'Right', 'Right']
[[ 1  2  3]
 [ 5  8 -1]
 [ 6  4  7]]

2 + 5 = 7
['Start', 'Right', 'Up']
[[ 1 -1  3]
 [ 5  2  8]
 [ 6  4  7]]

1 + 6 = 7
['Start', 'Down']
[[ 1  2  3]
 [ 6  5  8]
 [-1  4  7]]

1 + 6 = 7
['Start', 'Up']
[[-1  2  3]
 [ 1  5  8]
 [ 6  4  7]]
```

```
=======================================================
====================== CHOSING =======================
2 + 3 = 5
['Start', 'Right', 'Down']
[[ 1  2  3]
 [ 5  4  8]
 [ 6 -1  7]]

=======================================================
==================== ALL OPTIONS ====================
3 + 2 = 5
['Start', 'Right', 'Down', 'Right']
[[ 1  2  3]
 [ 5  4  8]
 [ 6  7 -1]]

3 + 4 = 7
['Start', 'Right', 'Down', 'Left']
[[ 1  2  3]
 [ 5  4  8]
 [-1  6  7]]

2 + 4 = 6
['Start', 'Right', 'Right']
[[ 1  2  3]
 [ 5  8 -1]
 [ 6  4  7]]

2 + 5 = 7
['Start', 'Right', 'Up']
[[ 1 -1  3]
 [ 5  2  8]
 [ 6  4  7]]

1 + 6 = 7
['Start', 'Down']
[[ 1  2  3]
 [ 6  5  8]
 [-1  4  7]]

1 + 6 = 7
['Start', 'Up']
[[-1  2  3]
 [ 1  5  8]
 [ 6  4  7]]

=======================================================
====================== CHOSING =======================
3 + 2 = 5
['Start', 'Right', 'Down', 'Right']
[[ 1  2  3]
 [ 5  4  8]
 [ 6  7 -1]]

=======================================================
==================== ALL OPTIONS ====================
4 + 1 = 5
['Start', 'Right', 'Down', 'Right', 'Up']
[[ 1  2  3]
 [ 5  4 -1]
```

```
 [ 6  7  8]]

3 + 4 = 7
['Start', 'Right', 'Down', 'Left']
[[ 1  2  3]
 [ 5  4  8]
 [-1  6  7]]

2 + 4 = 6
['Start', 'Right', 'Right']
[[ 1  2  3]
 [ 5  8 -1]
 [ 6  4  7]]

2 + 5 = 7
['Start', 'Right', 'Up']
[[ 1 -1  3]
 [ 5  2  8]
 [ 6  4  7]]

1 + 6 = 7
['Start', 'Down']
[[ 1  2  3]
 [ 6  5  8]
 [-1  4  7]]

1 + 6 = 7
['Start', 'Up']
[[-1  2  3]
 [ 1  5  8]
 [ 6  4  7]]

=====================================================
====================== CHOSING ======================
4 + 1 = 5
['Start', 'Right', 'Down', 'Right', 'Up']
[[ 1  2  3]
 [ 5  4 -1]
 [ 6  7  8]]

=====================================================
==================== ALL OPTIONS ====================
5 + 0 = 5
['Start', 'Right', 'Down', 'Right', 'Up', 'Up']
[[ 1  2 -1]
 [ 5  4  3]
 [ 6  7  8]]

5 + 2 = 7
['Start', 'Right', 'Down', 'Right', 'Up', 'Left']
[[ 1  2  3]
 [ 5 -1  4]
 [ 6  7  8]]

3 + 4 = 7
['Start', 'Right', 'Down', 'Left']
[[ 1  2  3]
 [ 5  4  8]
 [-1  6  7]]

2 + 4 = 6
```

```
['Start', 'Right', 'Right']
[[ 1  2  3]
 [ 5  8 -1]
 [ 6  4  7]]

2 + 5 = 7
['Start', 'Right', 'Up']
[[ 1 -1  3]
 [ 5  2  8]
 [ 6  4  7]]

1 + 6 = 7
['Start', 'Down']
[[ 1  2  3]
 [ 6  5  8]
 [-1  4  7]]

1 + 6 = 7
['Start', 'Up']
[[-1  2  3]
 [ 1  5  8]
 [ 6  4  7]]

=====================================================
====================== CHOSING ======================
5 + 0 = 5
['Start', 'Right', 'Down', 'Right', 'Up', 'Up']
[[ 1  2 -1]
 [ 5  4  3]
 [ 6  7  8]]

A Goal Has Been Found, Upperbound = 5
Optimal Path Found
['Start', 'Right', 'Down', 'Right', 'Up', 'Up']
```

```
print(puzzle4.solve(maxIterations=25000))
```

```
None
```

```
print(puzzle5.solve(maxIterations=25000))
```