EE4483 CA-Project2 Report

Source Codes

```
from copy import deepcopy
class State:
    def __init__(self,currentState, zero, parrent=None,pathLength=0):
        self.parrent = parrent
        self.zero = zero
        self.pathLength = pathLength
        self.currentState = currentState
        self.heuristic = pathLength
        for row in range(3):
            for col in range(3):
                self.heuristic += abs(row-goalCoordinate[currentState[row][col]][0])+abs(col-goal
Coordinate[currentState[row][col]][1])
goal = [[1,2,3],[8,0,4],[7,6,5]]
goalCoordinate = [[1,1],[0,0],[0,1],[0,2],[1,2],[2,2],[2,1],[2,0],[1,0]]
startState = [[],[],[]]
open=[]
print("""
Enter each row of the start state. Use integer 1-8 to represent the eight tiles and space for the
 empty grid. For example:
First row:  321
Second row: 4 8
Third row:  567

Our goal state is:
123
8 4
765
""")

firstRow = input("Enter the first row:  ")
secondRow = input("Enter the second row: ")
thirdRow = input("Enter the third row:  ")
for c in firstRow:
    startState[0].append(int(c) if c!=" " else 0)
for c in secondRow:
    startState[1].append(int(c) if c!=" " else 0)
for c in thirdRow:
    startState[2].append(int(c) if c!=" " else 0)
if startState == goal:
    print()
    print("Solution: ")
    print(firstRow)
    print(secondRow)
    print(thirdRow)
    quit()
for r in range(3):
    for c in range(3):
        if startState[r][c]==0:
            zero=[r,c]
open.append(State(startState,zero))
while True:
    state = open[0]
```

```
    open = open[1:]
    currentState = state.currentState
    zeroR = state.zero[0]
    zeroC = state.zero[1]
    pathLength = state.pathLength + 1
    if zeroR>0:
        newState = deepcopy(currentState)
        newState[zeroR][zeroC] = newState[zeroR-1][zeroC]
        newState[zeroR-1][zeroC] = 0
        newState = State(newState,[zeroR-1,zeroC], state, pathLength)
        if newState.heuristic-pathLength==0:
            lastState = newState
            break
        open.append(newState)
    if zeroR < 2:
        newState = deepcopy(currentState)
        newState[zeroR][zeroC] = newState[zeroR + 1][zeroC]
        newState[zeroR + 1][zeroC] = 0
        newState = State(newState, [zeroR + 1, zeroC], state, pathLength)
        if newState.heuristic - pathLength == 0:
            lastState = newState
            break
        open.append(newState)
    if zeroC > 0:
        newState = deepcopy(currentState)
        newState[zeroR][zeroC] = newState[zeroR][zeroC-1]
        newState[zeroR][zeroC-1] = 0
        newState = State(newState, [zeroR , zeroC -1], state, pathLength)
        if newState.heuristic - pathLength == 0:
            lastState = newState
            break
        open.append(newState)
    if zeroC < 2:
        newState = deepcopy(currentState)
        newState[zeroR][zeroC] = newState[zeroR][zeroC+1]
        newState[zeroR][zeroC+1] = 0
        newState = State(newState, [zeroR , zeroC +1], state, pathLength)
        if newState.heuristic - pathLength == 0:
            lastState = newState
            break
        open.append(newState)
    open = sorted(open, key = lambda state: state.heuristic)

state = lastState
solution = [state.currentState]
while state.parrent:
    state = state.parrent
    solution.append(state.currentState)
print("\nSolution:\n")
for i in range(len(solution)-1,-1,-1):
    state = solution[i]
    for r in range(3):
        string=""
        for c in range(3):
            string+= str(state[r][c]) if state[r][c]!=0 else " "
        print(string)
    print()
```

Question a: The program should be able to solve the 8-puzzle problem with arbitrary input. A visual interface is not necessary but the program should output each step when solving the puzzle.

In the command line, type: `python CA2.py` or run the script in any python IDE. After enter each row of the start state, the program will print out each step when solving the puzzle, if it is solvable with the input data.

Question b: Explain what heuristic measure and search strategy you use.

The heuristic measure is f(n) = g(n)+h(n), where g(n) is the length of the path (i.e. the depth of this state) and h(n) is sum of Manhattan Distance. h(n) = $\sum_{i=0}^{8} |x_i - \bar{x}| + |y_i - \bar{y}|$. And the heuristic value of each state is computed by the following code:

```
self.heuristic = pathLength
for row in range(3):
    for col in range(3):
        self.heuristic += abs(row-goalCoordinate[currentState[row][col]][0])+abs(col-goalCoordina
te[currentState[row][col][1])
```

In each iteration, the state with the minimal heuristic value in the open states list would be examined. And its child states would be generated and inserted into the open states list.

Question c: Discuss the complexity and memory cost of your search algorithm.

The time complexity in the worst case is O(4^d) where d is the length of the path, as there might be up to four possible moves in each step. And the program would store all the states that have been generated in the memory, because we need to print the path to goal state in the end. So the space complexity is also O(4^d).

Although the complexity for the worst case is the same as BFS and DFS algorithm, heuristic search enables us to skip most of the states and find the path faster in average case.

Question d: Explain in the worst case, how many steps your program need to complete the search.

As discussed above, the program needs to complete up to 4^d steps.