Group Members: Tracy Zhu, George Huang

## HOW TO RUN

In terminal with Python3 Installed, navigate to directory with
`15Puzzle.py`

Run by using command
`Python3 15Puzzle.py`

When "`Please enter the name of the input file:`" pumped, input the name of the input file.
When "`Please enter the name of the output file:`" pumped, input the name of the output file.

Then, press ENTER. Results will be printed and stored in the output file.

## OUTPUT FILES

***Output1***

1 5 3 13
8 0 6 4
15 10 7 9
11 14 2 12

1 5 3 13
0 7 6 4
8 10 9 2
11 15 14 12

6
32
6 5 8 1 2 3
6 6 6 6 6 6

***Output2***

9 13 7 4
12 3 0 1
2 15 5 6
14 10 11 8

9 3 7 1
13 5 4 6
12 15 0 10
14 2 11 8

11
106
7 8 2 3 4 7 5 4 7 7 1
10 11 11 11 11 11 11 11 11 11 11 11

Group Members: Tracy Zhu, George Huang

***Output3***

```
13 12 2 11
10 1 8 9
0 3 15 14
6 4 7 5

0 10 12 11
3 13 1 2
6 15 5 8
4 14 7 9

16
202
7 5 4 5 3 2 1 8 6 7 4 6 3 2 1 2
15 15 15 16 16 16 16 16 16 16 16 16 16 16 16 16
```

SOURCE CODE

***Source code***

```
import copy;

initialState = [[0 for i in range(4)] for j in range(4)];
goalState = [[0 for i in range(4)] for j in range(4)];
queue = [];
prevNodes = [];
node = 1;

#Class representing each state
class state:
    def __init__(self, array, path_prevNodes, prevActions, cost, chessboard):
        self.array = array;     #the matrix in current node
        self.path_prevNodes = path_prevNodes;       #previous node above it
        self.prevActions = prevActions;
        self.cost = cost;
        self.chessboard = chessboard;

#generate the next node
def generateChildren(expandNode, newArray, cost, action):
    newPrevNodes = expandNode.path_prevNodes;
    newPrevNodes.append(expandNode);
    newPrevActions = copy.deepcopy(expandNode.prevActions);
    newPrevActions.append(action);
    newState = state(newArray, newPrevNodes, newPrevActions, cost + 1,
sumchessboard(newArray));
    return newState;

#expand the current node
def expand(queue, node):
    expandNode = queue.pop(0);

    i, j = position(expandNode.array);

    #downleft
    if i < 3 and j > 0:
        newnode = copy.deepcopy(expandNode.array);
        newArray = downleft(i, j, newnode);
```

```
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'8');
            queue.append(newState);
            prevNodes.append(newState.array);

    #down
    if i < 3:
        newnode = copy.deepcopy(expandNode.array);
        newArray = down(i, j, newnode);
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'7');
            queue.append(newState);
            prevNodes.append(newState.array);

    #downright
    if i < 3 and j < 3:
        newnode = copy.deepcopy(expandNode.array);
        newArray = downright(i, j, newnode);
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'6');
            queue.append(newState);
            prevNodes.append(newState.array);

    #right
    if j < 3:
        newnode = copy.deepcopy(expandNode.array);
        newArray = right(i, j, newnode);
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'5');
            queue.append(newState);
            prevNodes.append(newState.array);

    #upright
    if i > 0 and j < 3:
        newnode = copy.deepcopy(expandNode.array);
        newArray = upright(i, j, newnode);
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'4');
            queue.append(newState);
            prevNodes.append(newState.array);

    #up
    if i > 0:
        newnode = copy.deepcopy(expandNode.array);
        newArray = up(i, j, newnode);
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'3');
            queue.append(newState);
            prevNodes.append(newState.array);
```

Group Members: Tracy Zhu, George Huang

```
    #upleft
    if i > 0 and j > 0:
        newnode = copy.deepcopy(expandNode.array);
        newArray = upleft(i, j, newnode);
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'2');
            queue.append(newState);
            prevNodes.append(newState.array);

    #left
    if j > 0:
        newnode = copy.deepcopy(expandNode.array);
        newArray = left(i, j, newnode);
        if newArray not in prevNodes:
            copy_expandNode = copy.deepcopy(expandNode);
            newState = generateChildren(copy_expandNode, newArray, expandNode.cost,
'1');
            queue.append(newState);
            prevNodes.append(newState.array);

    node += 1;
    return node;


#Find the position of 0
def position(array):
    for i in range(4):
        for j in range(4):
            if array[i][j] == '0':
                return i, j;

#move up
def up(i, j, array):
    array[i][j] = array[i - 1][j];
    array[i - 1][j] = '0';
    return array;

#move down
def down(i, j, array):
    array[i][j] = array[i + 1][j];
    array[i + 1][j] = '0';
    return array;

#move left
def left(i, j, array):
    array[i][j] = array[i][j - 1];
    array[i][j - 1] = '0';
    return array;

#move right
def right(i, j, array):
    array[i][j] = array[i][j + 1];
    array[i][j + 1] = '0';
    return array;

#move down and right
def downright(i, j, array):
    array[i][j] = array[i + 1][j + 1];
```

```
    array[i + 1][j + 1] = '0';
    return array;

#move down and left
def downleft(i, j, array):
    array[i][j] = array[i + 1][j - 1];
    array[i + 1][j - 1] = '0';
    return array;

#move up and right
def upright(i, j, array):
    array[i][j] = array[i - 1][j + 1];
    array[i - 1][j + 1] = '0';
    return array;

#move up and left
def upleft(i, j, array):
    array[i][j] = array[i - 1][j - 1];
    array[i - 1][j - 1] = '0';
    return array;


#Read the file, store the 4x4 intial and goal board into initialState and goalState
def readfile(file_name):
    f = open(file_name);
    index = 0;
    while index < 9:
        flines = f.readline();
        nums = flines.split();
        if index < 4:
            initialState[index] = [nums[0], nums[1], nums[2], nums[3]];
        elif index > 4:
            goalState[index-5] = [nums[0], nums[1], nums[2], nums[3]];
        index += 1;


#Calculate the sum of chessboard distance
def sumchessboard(currentArray):
    init_i = goal_m = goal_n = sum = 0;
    while init_i < 4:
        init_j = 0;
        while init_j < 4:
            currentnode = currentArray[init_i][init_j];
            if currentnode != '0':
                for goal_m in range(4):
                    for goal_n in range(4):
                        if currentnode == goalState[goal_m][goal_n]:
                            if (abs(init_i - goal_m) >= abs(init_j - goal_n)):
                                sum += abs(init_i - goal_m);
                            else:
                                sum += abs(init_j - goal_n);
            init_j += 1;
        init_i += 1;
    return sum;


def main():
    print("Please enter the name of the input file:");
    file_name = input();
    print("Please enter the name of the output file:");
    outputFileName = input();
```

```
    readfile(file_name);
    outputFile = open(outputFileName, "w");

    for i in initialState:
        outputFile.write(' '.join(str(x) for x in i));
        print(' '.join(i));
        outputFile.write("\n");
    outputFile.write("\n");
    print();
    for i in goalState:
        outputFile.write(' '.join(str(x) for x in i));
        print(' '.join(i));
        outputFile.write("\n");
    outputFile.write("\n");
    print();

    currentArray = copy.deepcopy(initialState);
    currentState = state(currentArray, [], [], 0, sumchessboard(currentArray));
    queue.append(currentState);

    numNodes = expand(queue, node);
    queue.sort(key = lambda x: x.cost + x.chessboard);

    while queue[0].array != goalState:
        numNodes = expand(queue, numNodes);
        queue.sort(key = lambda x: x.cost + x.chessboard);

    resNode = queue[0];
    outputFile.write(str(resNode.cost));
    outputFile.write("\n");
    outputFile.write(str(len(prevNodes)));
    outputFile.write("\n");
    outputFile.write(' '.join(resNode.prevActions));
    print(resNode.cost);
    print(len(prevNodes));
    print(' '.join(resNode.prevActions));
    lst = [];
    for i in resNode.path_prevNodes:
        lst.append(str(i.chessboard + i.cost));
    lst.append(str(resNode.chessboard + resNode.cost))
    outputFile.write("\n");
    outputFile.write(' '.join(lst))
    outputFile.close();

    print(' '.join(lst));

main();
```