A Star Search

By: Ethan Philpott

Running

To run, do `python3 main.py`

The program will ask you to input a filename and weight. Type them down and hit enter. For instance

```
> python3 main.py
Enter file name: ./inputs/Input3.txt
Enter weight: 1.4
```

Outputs

Output1a.txt

File: input1.txt

Weight: 1.0

```
2 0 6 4
3 10 7 9
11 5 8 1

2 10 6 4
11 3 8 9
0 7 5 1

1.0
7
22
D R D L U L D
7.0 7.0 7.0 7.0 7.0 7.0 7.0 7.0
```

Output1b.txt

File: input1.txt

Weight: 1.2

```
2 0 6 4
3 10 7 9
11 5 8 1
```

```
2 10 6 4
11 3 8 9
0 7 5 1

1.2
7
22
D R D L U L D
8.4 8.2 8.0 7.8 7.6 7.4 7.2 7.0
```

Output1c.txt

File: input1.txt

Weight: 1.4

```
2 0 6 4
3 10 7 9
11 5 8 1

2 10 6 4
11 3 8 9
0 7 5 1

1.4
7
22
D R D L U L D
9.8 9.4 9.0 8.6 8.2 7.8 7.4 7.0
```

Output2a.txt

File: input2.txt

Weight: 1.0

```
2 0 6 4
3 10 7 9
11 5 8 1

2 7 8 4
10 6 9 1
3 11 0 5

1.0
13
32
R D D L L U R U R D R D L
13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0
```

Output2b.txt

File: input2.txt

Weight: 1.2

```
2 0 6 4
3 10 7 9
11 5 8 1

2 7 8 4
10 6 9 1
3 11 0 5

1.2
13
29
R D D L L U R U R D R D L
15.6 15.4 15.2 15.0 14.8 14.6 14.4 14.2 14.0 13.8 13.6 13.4 13.2 13.0
```

Output2c.txt

File: input2.txt

Weight: 1.4

```
2 0 6 4
3 10 7 9
11 5 8 1

2 7 8 4
10 6 9 1
3 11 0 5

1.4
13
29
R D D L L U R U R D R D L
18.2 17.8 17.4 17.0 16.6 16.2 15.8 15.4 15.0 14.6 14.2 13.8 13.4 13.0
```

Output3a.txt

File: input3.txt

Weight: 1.0

```
8 7 2 4
10 6 9 1
0 11 5 3

10 6 8 4
9 7 0 2
11 5 3 1

1.0
17
169
R U R U L L D R D R R U L U L D R
13.0 13.0 15.0 15.0 15.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0 17.0
17.0 17.0 17.0
```

Output3b.txt

File: input3.txt

Weight: 1.2

```
8 7 2 4
10 6 9 1
0 11 5 3

10 6 8 4
9 7 0 2
11 5 3 1

1.2
17
125
R U R U L L D R D R R U L U L D R
15.6 15.4 17.6 17.4 17.2 19.4 19.2 19.0 18.8 18.6 18.4 18.2 18.0 17.8 17.6
17.4 17.2 17.0
```

Output3c.txt

File: input3.txt

Weight: 1.4

```
8 7 2 4
10 6 9 1
0 11 5 3

10 6 8 4
9 7 0 2
```

```
11 5 3 1

1.4
17
125
R U R U L L D R D R R U L U L D R
18.2 17.8 20.2 19.8 19.4 21.8 21.4 21.0 20.6 20.2 19.8 19.4 19.0 18.6 18.2
17.8 17.4 17.0
```

Source Code

```python
# Ethan Philpott
# Project 1
# AI
# November 11, 2021
# Professor Wong
# main.py

import copy

# Get file name from command line
filename = input("Enter file name: ")

# Open file
f = open(filename, 'r')

# Get weight from command line
weight = input("Enter weight: ")
# Convert weight to float
weight = float(weight)

# Holds all the old puzzles
past_puzzles = []

# Takes a list of text and converts it into the correct format for the 11
Puzzle problem
# Takes:
#   text: list of strings representing the puzzle
# Returns:
#   text: a 2d list of strings representing the puzzle
def convert_text(text):
    # Strip lines
    text = [line.strip() for line in text]
    # Split lines into list
    text = [line.split() for line in text]
    # Convert list to int
    text = [[int(x) for x in line] for line in text]
    return text

# Calculate manhattan distance
# Takes:
```

```
#   puzzle: 2d list of strings for the puzzle
#   goal_puzzle: 2d list of strings for the goal puzzle
#   i: int y coordinate of the current item
#   j: int x coordinate of the current item
# Returns:
#   an int for manhattan distance
def calculate_manhattan(puzzle, goal_puzzle, i, j):
    # Gets item
    item = puzzle[i][j]
    # If item is a blank space then we return 0
    if item == 0:
        return 0
    # Search 2d array for item
    for y in range(len(goal_puzzle)):
        for x in range(len(goal_puzzle[y])):
            # If item found and is not a blank space
            if item == goal_puzzle[y][x] and item != 0:
                # Calculate manhattan distance
                return abs(y - i) + abs(x - j)

# Calculate heuristic for the entire 11 puzzle state
# Takes:
#   puzzle: 2d list of strings for the puzzle
#   final_puzzle: 2d list of strings for the goal puzzle
# Returns:
#   an int for the sum of manhattan distances
def calculate_heuristic(puzzle, final_puzzle):
    # Go through state and calculate sum of manhattan distances
    sum = 0
    for i in range(len(puzzle)):
        for j in range(len(puzzle[i])):
            sum += calculate_manhattan(puzzle, final_puzzle, i, j)
    return sum

# Generates a new node for the 11 puzzle problem
# Takes:
#   state: 2d list of strings for the puzzle
#   final_puzzle: 2d list of strings for the goal puzzle
#   i: int y coordinate of the current item
#   j: int x coordinate of the current item
#   swap_i: int y coordinate of the item to swap with
#   swap_j: int x coordinate of the item to swap with
#   action: string for the action taken
# Returns:
#   int representing if the node was added or not (1 if added, 0 if not)
def generate_node(state, final_puzzle, i, j, swap_i, swap_j, action):
     # Copy state
    new_state = copy.deepcopy(state)
    # Swap blank with item above
    puzzle = new_state['puzzle']
    puzzle[i][j], puzzle[swap_i][swap_j] = puzzle[swap_i][swap_j],
puzzle[i][j]
    # If this is a duplicate state we just return 0 because we don't count
duplicate states
```

```
    if (puzzle in past_puzzles):
        return 0
    # Set up new h value
    new_state['h'] = calculate_heuristic(puzzle, final_puzzle)
    # Increment g value
    new_state['g'] += 1
    # Add action
    new_state['a'].append(action)
    # Add f value
    new_state['f'].append(new_state['g'] + new_state['h'] * weight)
    # Add to lists
    state_list.append(new_state)
    past_puzzles.append(new_state['puzzle'])
    # Returns 1 to increment counter
    return 1

# Creates new state for the 11 puzzle problem
# Takes:
#   index: int location of current state
#   state_list: list of dictionaries representing the current unexplored
states
#   final_puzzle: 2d list of strings for the goal puzzle
#   counter: int for the number of states generated
# Returns:
#   None
def create_states(index, state_list, final_puzzle, count):
    # Get puzzle
    puzzle = state_list[index]['puzzle']
    # Find blank space
    for i in range(len(puzzle)):
        for j in range(len(puzzle[i])):
            # Found a blank space
            if puzzle[i][j] == 0:
                # If we have an item above the blank
                if(i - 1 >= 0):
                    count += generate_node(state_list[index],
final_puzzle, i, j, i - 1, j, "U")
                # If we have an item below the blank
                if(i + 1 < len(puzzle)):
                    # Copy state
                    count += generate_node(state_list[index],
final_puzzle, i, j, i + 1, j, "D")
                # If we have an item to the left of the blank
                if(j - 1 >= 0):
                    count += generate_node(state_list[index],
final_puzzle, i, j, i, j - 1, "L")
                # If we have an item to the right of the blank
                if(j + 1 < len(puzzle[i])):
                    count += generate_node(state_list[index],
final_puzzle, i, j, i, j + 1, "R")
                # Remove node we just explored
                state_list.pop(index)
    # Choose the next state
    choose_state(state_list, final_puzzle, count)
```

```python
# Checks if the puzzles are equal
# Takes:
#   puzzle1: 2d list of strings for the puzzle
#   puzzle2: 2d list of strings for the puzzle
# Returns:
#   True if equal, False if not
def equal_puzzles(puzzle1, puzzle2):
    # Check if puzzles are equal
    if puzzle1 == puzzle2:
        return True
    return False

# Prints the output to the terminal and file
# Takes:
#   initial: 2d list of strings for the puzzle
#   state: a dictionary representing the current state
#   count: int for the number of states generated
# Returns:
#   None
def print_output(initial, state, count):
    with open('./outputs/output.txt', 'w') as f:
        # Prints 2d list for initial puzzle
        for i in range(len(initial)):
            for j in range(len(initial[i])):
                print(initial[i][j], end=" ")
                f.write(str(initial[i][j]) + " ")
            print()
            f.write("\n")

        # Add newline
        print()
        f.write("\n")

        # Prints 2d list for current state
        for i in range(len(state['puzzle'])):
            for j in range(len(state['puzzle'][i])):
                print(state['puzzle'][i][j], end=" ")
                f.write(str(state['puzzle'][i][j]) + " ")
            print()
            f.write("\n")

        # Add newline
        print()
        f.write("\n")

        # Print weight
        print(float(weight))
        f.write(str(float(weight)) + "\n")

        # Print depth
        print(state['g'])
        f.write(str(state['g']) + "\n")
```

```
        # Print count
        print(count)
        f.write(str(count) + "\n")

        # Print actions
        print(" ".join(state['a']))
        f.write(" ".join(state['a']) + "\n")

        # Print f values
        print(" ".join(str(round(x, 4)) for x in state['f']))
        f.write(" ".join(str(round(x, 4)) for x in state['f']))

# Insert in order of lowest f value to greatest
# Takes:
#   state_list: list of dictionaries representing the current unexplored
states
#   final_puzzle: 2d list of strings for the goal puzzle
#   count: int for the number of states generated
# Returns:
#   None
def choose_state(state_list, final_puzzle, count):
    # Find lowest f value
    lowest_f = state_list[0]['f'][-1]
    lowest_index = 0
    for i in range(len(state_list)):
        # If we find a lower f value then update the lowest_f value and
lowest_index
        if state_list[i]['f'][-1] < lowest_f:
            lowest_f = state_list[i]['f'][-1]
            lowest_index = i
    # If we found the solution then print, else keep going
    if (not equal_puzzles(state_list[lowest_index]['puzzle'],
final_puzzle)):
        create_states(lowest_index, state_list, final_puzzle, count)
    else:
        print_output(initial_puzzle, state_list[lowest_index], count)

# Open file
f = open(filename, 'r')

# Read file lines 1 to 3
initial_puzzle = f.readlines()[0:3]
initial_puzzle = convert_text(initial_puzzle)

# Reset readline to beginning
f.seek(0)

# Read file lines 5 to 7
final_puzzle = f.readlines()[4:7]
final_puzzle = convert_text(final_puzzle)

# Get the initial heuristic value
initial_heuristic = calculate_heuristic(initial_puzzle, final_puzzle)
# Holds all the current puzzles, we add the initial state first
```

```
state_list = [
    {
        "g": 0,
        "h": initial_heuristic,
        "a": [],
        "f": [initial_heuristic * weight],
        "puzzle": initial_puzzle
    }
]
past_puzzles.append(state_list[0]['puzzle'])

# Start the program, include 1 for root node
choose_state(state_list, final_puzzle, 1)
```