Andrew Liu

ITCS 6150 Project 1: Solving 8-puzzle using A* search algorithm

Overview:

In this project we need to solve the 8-puzzle using the A*search algorithm. The 8-puzzle is a puzzle that takes place in a 3x3 square. Each square contains a unique number from 0-8. The 0 square is able to switch places with the squares around it, no other square has this property. The goal is to get from an initial state to a goal state. A* uses two factors to calculate how close a state is to its goal state. First factor is the g-value, this is how many nodes this state is from the initial state. Second factor is the h-value, this is the heuristic value and is dependent on the method you choose to use. For my project I am allowing the user to choose between manhattan distance and number of incorrect nodes. The g-value and h-value are then added together to create an f-value which is what will be used to calculate the best node.

For using the program the user first chooses which heuristic method they would like to choose. Then after choosing they will be prompted to enter 9 numbers for the initial state. These numbers must be 0-8 and there cannot be any repeats. Next the user will be prompted to select 9 numbers for the goal state. Just like the initial state these must be unique and 0-8.

Classes/Methods/Objects/Variables used:

Variables:

Int choice: This is to decide which heuristic to use
Int[][] Init: This is the initial state of the puzzle
Int[][] goal: This is the goal state of the puzzle
List<PuzzleBoard> visited: This is the list of nodes that have been visited
List<PuzzleBoard> generated: This is the list of nodes that have been generated
Puzzleboard best: This is the Puzzleboard with the lowest f-value.
Hashtable<String> exists: this hashtable contains a string interpretation of every puzzle generated

takeUserInput:

This function is what allows the user to enter in their initial states and the goal states. It returns a list of the integers the user has entered.

displayBoard:

This function is for displaying how the puzzle board looks like to the console. It also displays the g, h, and f value of the board. It takes a Puzzleboard as input, a String to display which state it is and returns void.

boardsEqual:

This function takes two Puzzleboards and checks if each value in the puzzles are the same. If there is a value in the two puzzles that are not equal, it will return false.

generateData:
This function generates a 2d array based on a given list.

printPuzzle:
This function is for displaying the puzzle of a PuzzleBoard. It accepts a 2d array and returns void.

generateNode:
This function is used to generate the different possible states of the board. Based on the previous best board it will generate if the 0 is moved left, right, up, or down. It will also calculate the g value and the h value based on the choice the user selected. All nodes are then added to the generated list. Before any node is added though, the function will check if the node has either been generated.

mDistance:
This function is used to calculate the manhattan distance of a given 2d array. To do this I added the i and j value of a given 2d array. Then I subtract that from the i and j value of the goal 2d array. Finally I take the absolute value of this result and add it to the result value. Once all values are checked, the result is returned.

misplacedNodes:
This function checks to see which nodes are misplaced. The result starts at 9 and each time a given 2d array and goal array have the same value in the same spot, the result decrements. After all values are checked, the result is returned.

Program Structure:

Puzzleboard Object:
Fields:
Int g: This is the how far this node is away from the initial node
Int h: This is the heuristic value
Int f: This is the combination of g and f
Int[][] puzzle: This is the puzzleboard represented as a 2d array
Puzzleboard parent: This is to keep track of the parent board.

Methods:
Getters/Setters: self explanatory
toString: self explanatory
compareTo: This method is used to compare two Puzzleboards. It first checks to see which board has a lower f value. If the f values are equal, then it will pick the one with the lowest h value.

Searching:
For checking if a puzzle has already been created, the program will check if the hashtable contains the puzzle given. Initially I had checked the Arraylists, but by changing this to use a hashtable, it has increased the program speed substantially.

Test data:

Manhattan Distance test

1.

initial state 4 1 3 0 2 6 7 5 8	0 1 3 4 2 6 7 5 8 G:1 H:4 F:5	1 0 3 4 2 6 7 5 8 G:2 H:3 F:5	1 2 3 4 0 6 7 5 8 G:3 H:2 F:5
1 2 3 4 5 6 7 0 8 G:4 H:1 F:5	1 2 3 4 5 6 7 8 0 G:5 H:0 F:5	Goal state 1 2 3 4 5 6 7 8 0 Nodes Expanded: 6 Nodes Generated: 12	

2.

initial state 1 2 3 7 4 5 6 8 0	1 2 3 7 4 0 6 8 5 G:1 H:5 F:6	1 2 3 7 0 4 6 8 5 G:2 H:4 F:6	1 2 3 7 8 4 6 0 5 G:3 H:3 F:6
1 2 3 7 8 4 0 6 5 G:4 H:4 F:8	1 2 3 0 8 4 7 6 5 G:5 H:3 F:8	1 2 3 8 0 4 7 6 5 G:6 H:2 F:8	1 2 3 8 6 4 7 0 5 G:7 H:1 F:8
1 2 3 8 6 4 7 5 0 G:8 H:0 F:8	Goal state 1 2 3 8 6 4 7 5 0 Nodes Expanded: 15 Nodes Generated: 30		

3.

initial state 2 8 1 3 4 6 7 5 0	2 8 1 3 4 0 7 5 6 G:1 H:3 F:4	2 8 1 3 0 4 7 5 6 G:2 H:2 F:4	2 0 1 3 8 4 7 5 6 G:3 H:3 F:6
0 2 1 3 8 4 7 5 6 G:4 H:2 F:6	3 2 1 0 8 4 7 5 6 G:5 H:1 F:6	3 2 1 8 0 4 7 5 6 G:6 H:0 F:6	Goal state 3 2 1 8 0 4 7 5 6 Nodes Expanded: 7 Nodes Generated: 13

4.

Initial state 7 2 4 5 0 6 8 3 1	7 2 4 5 3 6 8 0 1 G:1 H:9 F:10	7 2 4 5 3 6 8 1 0 G:2 H:8 F:10	7 2 4 5 3 0 8 1 6 G:3 H:9 F:12
7 2 4 5 0 3 8 1 6 G:4 H:10 F:14	7 2 4 0 5 3 8 1 6 G:5 H:9 F:14	0 2 4 7 5 3 8 1 6 G:6 H:8 F:14	2 0 4 7 5 3 8 1 6 G:7 H:9 F:16
2 4 0 7 5 3 8 1 6 G:8 H:8 F:16	2 4 3 7 5 0 8 1 6 G:9 H:7 F:16	2 4 3 7 0 5 8 1 6 G:10 H:8 F:18	2 4 3 7 1 5 8 0 6 G:11 H:7 F:18
2 4 3 7 1 5 0 8 6 G:12 H:6 F:18	2 4 3 0 1 5 7 8 6 G:13 H:5 F:18	2 4 3 1 0 5 7 8 6 G:14 H:4 F:18	2 0 3 1 4 5 7 8 6 G:15 H:5 F:20
0 2 3 1 4 5 7 8 6 G:16 H:4 F:20	1 2 3 0 4 5 7 8 6 G:17 H:3 F:20	1 2 3 4 0 5 7 8 6 G:18 H:2 F:20	1 2 3 4 5 0 7 8 6 G:19 H:1 F:20
1 2 3 4 5 6 7 8 0 G:20 H:0 F:20	Goal state 1 2 3 4 5 6 7 8 0 Nodes Expanded: 617 Nodes Generated: 969		

5.

Initial state 2 8 3 1 6 4 7 0 5	2 8 3 1 0 4 7 6 5 G:1 H:2 F:3	2 0 3 1 8 4 7 6 5 G:2 H:3 F:5	0 2 3 1 8 4 7 6 5 G:3 H:2 F:5
1 2 3 0 8 4 7 6 5 G:4 H:1 F:5	1 2 3 8 0 4 7 6 5 G:5 H:0 F:5	Goal state 1 2 3 8 0 4 7 6 5 Nodes Expanded: 6 Nodes Generated: 12	

6.

initial state 0 8 3 7 4 5 6 1 2	8 0 3 7 4 5 6 1 2 G:1 H:13 F:14	8 4 3 7 0 5 6 1 2 G:2 H:12 F:14	8 4 3 7 1 5 6 0 2 G:3 H:11 F:14
8 4 3 7 1 5 0 6 2 G:4 H:10 F:14	8 4 3 0 1 5 7 6 2 G:5 H:9 F:14	0 4 3 8 1 5 7 6 2 G:6 H:8 F:14	4 0 3 8 1 5 7 6 2 G:7 H:9 F:16
4 1 3 8 0 5 7 6 2 G:8 H:8 F:16	4 1 3 0 8 5 7 6 2 G:9 H:7 F:16	0 1 3 4 8 5 7 6 2 G:10 H:6 F:16	1 0 3 4 8 5 7 6 2 G:11 H:5 F:16
1 3 0 4 8 5 7 6 2 G:12 H:6 F:18	1 3 5 4 8 0 7 6 2 G:13 H:5 F:18	1 3 5 4 8 2 7 6 0 G:14 H:4 F:18	1 3 5 4 8 2 7 0 6 G:15 H:5 F:20
1 3 5 4 0 2 7 8 6 G:16 H:4 F:20	1 3 5 4 2 0 7 8 6 G:17 H:3 F:20	1 3 0 4 2 5 7 8 6 G:18 H:4 F:22	1 0 3 4 2 5 7 8 6 G:19 H:3 F:22
1 2 3 4 0 5 7 8 6 G:20 H:2 F:22	1 2 3 4 5 0 7 8 6 G:21 H:1 F:22	1 2 3 4 5 6 7 8 0 G:22 H:0 F:22	Goal state 1 2 3 4 5 6 7 8 0 Nodes Expanded: 862 Nodes Generated: 1365

Number of incorrect nodes test:

1.

initial state 4 1 3 0 2 6 7 5 8	0 1 3 4 2 6 7 5 8 G:1 H:4 F:5	1 0 3 4 2 6 7 5 8 G:2 H:3 F:5	1 2 3 4 0 6 7 5 8 G:3 H:2 F:5
1 2 3 4 5 6 7 0 8 G:4 H:1 F:5	1 2 3 4 5 6 7 8 0 G:5 H:0 F:5	Goal state 1 2 3 4 5 6 7 8 0 Nodes Expanded: 6 Nodes Generated: 12	

2.

initial state 1 2 3 7 4 5 6 8 0	1 2 3 7 4 0 6 8 5 G:1 H:6 F:7	1 2 3 7 0 4 6 8 5 G:2 H:5 F:7	1 2 3 7 8 4 6 0 5 G:3 H:5 F:8
1 2 3 7 8 4 0 6 5 G:4 H:5 F:9	1 2 3 0 8 4 7 6 5 G:5 H:4 F:9	1 2 3 8 0 4 7 6 5 G:6 H:3 F:9	1 2 3 8 6 4 7 0 5 G:7 H:2 F:9
1 2 3 8 6 4 7 5 0 G:8 H:0 F:8	Goal state 1 2 3 8 6 4 7 5 0 Nodes Expanded: 15 Nodes Generated: 29		

3.

initial state 2 8 1 3 4 6 7 5 0	2 8 1 3 4 0 7 5 6 G:1 H:5 F:6	2 8 1 3 0 4 7 5 6 G:2 H:3 F:5	2 0 1 3 8 4 7 5 6 G:3 H:4 F:7
0 2 1 3 8 4 7 5 6 G:4 H:3 F:7	3 2 1 0 8 4 7 5 6 G:5 H:2 F:7	3 2 1 8 0 4 7 5 6 G:6 H:0 F:6	Goal state 3 2 1 8 0 4 7 5 6 Nodes Expanded: 7 Nodes Generated: 13

4.

Initial state 7 2 4 5 0 6 8 3 1	7 2 4 5 3 6 8 0 1 G:1 H:7 F:8	7 2 4 5 3 6 8 1 0 G:2 H:6 F:8	7 2 4 5 3 0 8 1 6 G:3 H:8 F:11
7 2 4 5 0 3 8 1 6 G:4 H:8 F:12	7 2 4 0 5 3 8 1 6 G:5 H:7 F:12	0 2 4 7 5 3 8 1 6 G:6 H:7 F:13	2 0 4 7 5 3 8 1 6 G:7 H:8 F:15
2 4 0 7 5 3 8 1 6 G:8 H:8 F:16	2 4 3 7 5 0 8 1 6 G:9 H:7 F:16	2 4 3 7 0 5 8 1 6 G:10 H:8 F:18	2 4 3 7 1 5 8 0 6 G:11 H:8 F:19
2 4 3 7 1 5 0 8 6 G:12 H:7 F:19	2 4 3 0 1 5 7 8 6 G:13 H:6 F:19	2 4 3 1 0 5 7 8 6 G:14 H:6 F:20	2 4 3 1 0 5 7 8 6 G:14 H:6 F:20
0 2 3 1 4 5 7 8 6 G:16 H:5 F:21	1 2 3 0 4 5 7 8 6 G:17 H:4 F:21	1 2 3 4 0 5 7 8 6 G:18 H:3 F:21	1 2 3 4 5 0 7 8 6 G:19 H:2 F:21
1 2 3 4 5 6 7 8 0 G:20 H:0 F:20	Goal state 1 2 3 4 5 6 7 8 0 Nodes Expanded: 2468 Nodes Generated: 3845		

5.

initial state 2 8 3 1 6 4 7 0 5	2 8 3 1 0 4 7 6 5 G:1 H:3 F:4	2 0 3 1 8 4 7 6 5 G:2 H:4 F:6	0 2 3 1 8 4 7 6 5 G:3 H:3 F:6
1 2 3 0 8 4 7 6 5 G:4 H:2 F:6	1 2 3 8 0 4 7 6 5 G:5 H:0 F:5	1 2 3 8 0 4 7 6 5 Nodes Expanded: 6 Nodes Generated: 12	

6.

initial state 0 8 3 7 4 5 6 1 2	8 0 3 7 4 5 6 1 2 G:1 H:8 F:9	8 4 3 7 0 5 6 1 2 G:2 H:8 F:10	8 4 3 7 1 5 6 0 2 G:3 H:8 F:11
8 4 3 7 1 5 0 6 2 G:4 H:8 F:12	8 4 3 0 1 5 7 6 2 G:5 H:7 F:12	8 4 3 1 0 5 7 6 2 G:6 H:7 F:13	8 0 3 1 4 5 7 6 2 G:7 H:7 F:14
0 8 3 1 4 5 7 6 2 G:8 H:7 F:15	1 8 3 0 4 5 7 6 2 G:9 H:6 F:15	1 8 3 4 0 5 7 6 2 G:10 H:5 F:15	1 0 3 4 8 5 7 6 2 G:11 H:5 F:16
1 3 0 4 8 5 7 6 2 G:12 H:6 F:18	1 3 5 4 8 0 7 6 2 G:13 H:6 F:19	1 3 5 4 8 2 7 6 0 G:14 H:5 F:19	1 3 5 4 8 2 7 0 6 G:15 H:6 F:21
1 3 5 4 0 2 7 8 6 G:16 H:5 F:21	1 3 5 4 2 0 7 8 6 G:17 H:5 F:22	1 3 0 4 2 5 7 8 6 G:18 H:5 F:23	1 0 3 4 2 5 7 8 6 G:19 H:4 F:23
1 2 3 4 0 5 7 8 6 G:20 H:3 F:23	1 2 3 4 5 0 7 8 6 G:21 H:2 F:23	1 2 3 4 5 6 7 8 0 G:22 H:0 F:22	Goal state 1 2 3 4 5 6 7 8 0 Nodes Expanded: 4875 Nodes Generated: 7622