INTELLIGENT SYSTEMS

Programming Project 1
Solving the 8-puzzle using A* search algorithm

Submitted by:

Dheeraj Mirashi (801151308)
Adwait More (801134139)

University of North Carolina at Charlotte

Problem Formulation :

8-Puzzle problem is consists of 8 tiles. The initial state and the goal state is provided. In the state, there is one empty tile. The puzzle is solved by using this empty tile to shift and by moving the tiles the goal state is achieved and thus problem is solved.

A problem is defined by four items viz. Initial State, Successor Function, Goal Test and Path Cost.

1	2	3
0	4	5
6	7	8

Initial State

1	2	3
4	5	6
7	8	0

Goal State

Initial State : The initial state is entered by user. In this program, the input is taken and stored in a 3x3 Array of integers. E.g. : {1 , 2 , 3 , 0 , 4 , 5 , 6 , 7 , 8}

Successor Function : S(x) = Set of Action – Pairs . The set of Action-Pairs is Up, Down, Left, Right.
In the above example, the generated successors are :
[g=1] => {0,2,3,1,4,6,7,5,8}, {1,2,3,4,0,6,7,5,8}, {1,2,3,7,4,6,0,5,8}.
[g=2] => {1,2,3,4,5,6,7,0,8}, {1,2,3,4,6,0,7,5,8}, {1,0,3,4,2,6,7,5,8}.
[g=3] => {1,2,3,4,5,6,0,7,8}, {1,2,3,4,5,6,7,8,0}

Goal Test : When the current state is equal to the goal state.

Path Cost : In this problem, the total path cost i.e from n to goal is calculated which is denoted by f(n). So the function is f(n) = g(n) + h(n) where g(n) is actual cost to reach n, h(n) is estimated cost from n to goal.
In the above example, the f(n) for initial state is 3 [g(n)=0, h(n)=3)] and total cost is 12 [g(n)=3, h(n)=0)] to reach goal.

Program Structure:

The program is implemented in JAVA and the IDE used is IntelliJ IDEA.
We have used two classes in the code : Astar & Node.
Astar class has the implementation of the A* algorithm. It is the starting point of the program. Since this is a menu-driven program, it asks user for an initial state and the goal state. We are maintaining two lists, one for storing the nodes that have been explored and the other for storing unexplored nodes. It provides functions that check whether the input state is valid or not and also calculate the total path cost i.e f(n)
Node class defines the structure of the state configuration and also provides various functions to get the children nodes, moving tiles, calculating heuristics, etc.

Following is the flow of the program :

1. Ask user for goal state
2. Ask user for initial state
3. Ask user for the desired heuristics
4. Stores the initial Node (initial state configuration) into Open node list
5. While open nodes list is not empty, find the least g(n) based on the selected heuristic and remove the node from the open list
6. Check if the node is equal to goal node, if yes goal found.
7. If goal not found, generate all of its successors.
8. Check if they exist in the Open or Closed nodes list.
9. Add all the childrens to Open node list i.e current node (Parent node) is added to Closed node list.
10. Go to step 5
11. If open node is empty and goal is not found, exit with error code Goal not found.

Variables :

There are two files in the program Astar.java and Node.java

Global Variables :

In Astar.java
Node goalState – stores user inputted goal state
String heuristic1 – stores the first type of heuristic
String heuristic2 – stores the second type of heuristic

Instance Variables :

In Node.java =>
Node parent – It has the parent node of the current node
int[][] state – It stores the current state configuration
int pathCost – It stores the path cost of the node
int fOfManhatten – It stores the heuristic value based on the manhattan distance
int fOfNTiles – It stores the heuristic value based on the number of misplaced tiles

Functions :

In Astar.java file :

Main function() => This is the starting point of the application. Here the input is taken from the user and printed on the console. It performs an iteration over the open nodes list and finds the node with least g of n and also iterates over its childrens to check for the goal state.

findNodeWithLeastGOfN => This function takes two parameters as input viz Open nodes list and the type of heuristic to be applied. Now based on the heuristic, it calls the appropriate function to get the node with least g of n.

isValidNode => This function checks whether the given inputted node is valid or not.

acceptInputState => This function takes the user input one by one digit and stores it in a 3x3 integer array.

In Node.java file :

getCoOrdinates() => This function returns the postion of the element in the node

getChildren() => This function gets the child nodes

getNode() => This function gets the node

heuristicManhattan() => This function calculates the heuristic value based on the manhattan distance

heuristicTiles() => This function calculates the number of misplaced tiles.

Problem Analysis and Output :

Problem 1 :
Input State : {1,3,5,4,6,0,8,7,2}
Goal State : {1,2,3,4,5,6,7,8,0}

Output:

```
Enter goal state:
1
2
3

4
5
6

7
8
0

Enter initial state:
1
3
5

4
6
0

8
7
2

Initial node is:
1 3 5
4 6 0
8 7 2

Initial node f(n)=9
Goal node is:
1 2 3
4 5 6
7 8 0

<-------------------------------------------------------------------------->
Choose a heuristic value between following:
press 1 for 'Manhatten Distance'
press 2 for 'misplaced tiles'
1
<-------------------------------------------------------------------------->
Goal found
path cost:17
path:

1 3 5
4 6 0
8 7 2
h(n) manhatten            :   9
g(n) from root            :   0
f(n) by manhatten         :   9

1 3 5
4 0 6
8 7 2
h(n) manhatten            :   8
g(n) from root            :   1
f(n) by manhatten         :   9

1 3 5
4 7 6
8 0 2
h(n) manhatten            :   9
g(n) from root            :   2
f(n) by manhatten         :   11

1 3 5
4 7 6
8 2 0
h(n) manhatten            :   8
g(n) from root            :   3
f(n) by manhatten         :   11

1 3 5
4 7 0
8 2 6
h(n) manhatten            :   9
g(n) from root            :   4
f(n) by manhatten         :   13

1 3 0
4 7 5
8 2 6
h(n) manhatten            :   8
g(n) from root            :   5
f(n) by manhatten         :   13
```

```
 1 0 3
 4 7 5
 8 2 6
 h(n) manhatten             :   7
 g(n) from root             :   6
 f(n) by manhatten          :   13

 0 1 3
 4 7 5
 8 2 6
 h(n) manhatten             :   8
 g(n) from root             :   7
 f(n) by manhatten          :   15

 4 1 3
 0 7 5
 8 2 6
 h(n) manhatten             :   9
 g(n) from root             :   8
 f(n) by manhatten          :   17

 4 1 3
 7 0 5
 8 2 6
 h(n) manhatten             :   8
 g(n) from root             :   9
 f(n) by manhatten          :   17

 4 1 3
 7 2 5
 8 0 6
 h(n) manhatten             :   7
 g(n) from root             :   10
 f(n) by manhatten          :   17

 4 1 3
 7 2 5
 0 8 6
 h(n) manhatten             :   6
 g(n) from root             :   11
 f(n) by manhatten          :   17

 4 1 3
 0 2 5
 7 8 6
 h(n) manhatten             :   5
 g(n) from root             :   12
 f(n) by manhatten          :   17

 0 1 3
 4 2 5
 7 8 6
 h(n) manhatten             :   4
 g(n) from root             :   13
 f(n) by manhatten          :   17

 1 0 3
 4 2 5
 7 8 6
 h(n) manhatten             :   3
 g(n) from root             :   14
 f(n) by manhatten          :   17

1 2 3
4 0 5
7 8 6
h(n) manhatten             :   2
g(n) from root             :   15
f(n) by manhatten          :   17

1 2 3
4 5 0
7 8 6
h(n) manhatten             :   1
g(n) from root             :   16
f(n) by manhatten          :   17

1 2 3
4 5 6
7 8 0
h(n) manhatten             :   0
g(n) from root             :   17
f(n) by manhatten          :   17

Process finished with exit code 0
```

Problem 2 :
Input State : {1,2,3,5,0,4,6,8,7}
Goal State : {1,2,3,4,5,6,7,0,8}

Output :

```
Enter goal state:
1
2
3

4
5
6

7
8
8

Enter initial state:
1
2
3

5
8
4

6
8
7

Initial node is:
1 2 3
5 0 4
6 8 7

Initial node f(n)=9
Goal node is:
1 2 3
4 5 6
7 0 8
<--------------------------------------------------------------------------->
Choose a heuristic value between following:
press 1 for 'Manhatten Distance'
press 2 for 'misplaced tiles'
1
<--------------------------------------------------------------------------->
Goal found
path cost:17
path:

1 2 3
5 0 4
6 8 7
h(n) manhatten            :   9
g(n) from root            :   0
f(n) by manhatten         :   9

1 2 3
5 8 4
6 0 7
h(n) manhatten            :   10
g(n) from root            :   1
f(n) by manhatten         :   11

1 2 3
5 8 4
6 7 0
h(n) manhatten            :   9
g(n) from root            :   2
f(n) by manhatten         :   11

1 2 3
5 8 0
6 7 4
h(n) manhatten            :   10
g(n) from root            :   3
f(n) by manhatten         :   13
```

```
1 2 3
5 0 8
6 7 4
h(n) manhatten            :   9
g(n) from root            :   4
f(n) by manhatten         :   13

1 2 3
5 7 8
6 0 4
h(n) manhatten            :   10
g(n) from root            :   5
f(n) by manhatten         :   15

1 2 3
5 7 8
0 6 4
h(n) manhatten            :   9
g(n) from root            :   6
f(n) by manhatten         :   15

1 2 3
0 7 8
5 6 4
h(n) manhatten            :   10
g(n) from root            :   7
f(n) by manhatten         :   17

1 2 3
7 0 8
5 6 4
h(n) manhatten            :   9
g(n) from root            :   8
f(n) by manhatten         :   17

1 2 3
7 6 8
5 0 4
h(n) manhatten            :   8
g(n) from root            :   9
f(n) by manhatten         :   17

1 2 3
7 6 8
5 4 0
h(n) manhatten            :   7
g(n) from root            :   10
f(n) by manhatten         :   17

1 2 3
7 6 0
5 4 8
h(n) manhatten            :   6
g(n) from root            :   11
f(n) by manhatten         :   17

1 2 3
7 0 6
5 4 8
h(n) manhatten            :   5
g(n) from root            :   12
f(n) by manhatten         :   17

1 2 3
7 4 6
5 0 8
h(n) manhatten            :   4
g(n) from root            :   13
f(n) by manhatten         :   17

1 2 3
7 4 6
0 5 8
h(n) manhatten            :   3
g(n) from root            :   14
f(n) by manhatten         :   17

1 2 3
0 4 6
7 5 8
h(n) manhatten            :   2
g(n) from root            :   15
f(n) by manhatten         :   17

1 2 3
4 0 6
7 5 8
h(n) manhatten            :   1
g(n) from root            :   16
f(n) by manhatten         :   17

1 2 3
4 5 6
7 0 8
h(n) manhatten            :   0
g(n) from root            :   17
f(n) by manhatten         :   17

Process finished with exit code 0
```

Problem 3 : (using no of displaced tiles heuristics)
Input State : {1,3,4,2,5,6,7,0,8}
Goal State : {1,2,3,4,5,6,7,8,0}

Output :

```
Initial node is:
1 3 4
2 5 6
7 0 8

Initial node f(n)=7
Goal node is:
1 2 3
4 5 6
7 8 0

<--------------------------------------------------------------------------------->
Choose a heuristic value between following:
press 1 for 'Manhatten Distance'
press 2 for 'misplaced tiles'
2
<--------------------------------------------------------------------------------->
Goal found
path cost:17
path:

1 3 4
2 5 6
7 0 8
h(n) misplaced tiles       :   4
g(n) from root             :   0
f(n) by misplaced tiles    :   4

1 3 4
2 0 6
7 5 8
h(n) misplaced tiles       :   5
g(n) from root             :   1
f(n) by misplaced tiles    :   6

1 3 4
0 2 6
7 5 8
h(n) misplaced tiles       :   5
g(n) from root             :   2
f(n) by misplaced tiles    :   7

1 3 4
7 2 6
0 5 8
h(n) misplaced tiles       :   6
g(n) from root             :   3
f(n) by misplaced tiles    :   9

1 3 4
7 2 6
5 0 8
h(n) misplaced tiles       :   6
g(n) from root             :   4
f(n) by misplaced tiles    :   10
```

```
1 3 4
7 2 6
5 8 0
h(n) misplaced tiles       :   5
g(n) from root             :   5
f(n) by misplaced tiles    :   10

1 3 4
7 2 0
5 8 6
h(n) misplaced tiles       :   6
g(n) from root             :   6
f(n) by misplaced tiles    :   12

1 3 0
7 2 4
5 8 6
h(n) misplaced tiles       :   6
g(n) from root             :   7
f(n) by misplaced tiles    :   13

1 0 3
7 2 4
5 8 6
h(n) misplaced tiles       :   5
g(n) from root             :   8
f(n) by misplaced tiles    :   13

1 2 3
7 0 4
5 8 6
h(n) misplaced tiles       :   4
g(n) from root             :   9
f(n) by misplaced tiles    :   13

1 2 3
7 4 0
5 8 6
h(n) misplaced tiles       :   4
g(n) from root             :   10
f(n) by misplaced tiles    :   14

1 2 3
7 4 6
5 8 0
h(n) misplaced tiles       :   3
g(n) from root             :   11
f(n) by misplaced tiles    :   14

1 2 3
7 4 6
5 0 8
h(n) misplaced tiles       :   4
g(n) from root             :   12
f(n) by misplaced tiles    :   16
```

```
1 2 3
7 4 6
0 5 8
h(n) misplaced tiles       :   4
g(n) from root             :   13
f(n) by misplaced tiles    :   17

1 2 3
0 4 6
7 5 8
h(n) misplaced tiles       :   3
g(n) from root             :   14
f(n) by misplaced tiles    :   17

1 2 3
4 0 6
7 5 8
h(n) misplaced tiles       :   2
g(n) from root             :   15
f(n) by misplaced tiles    :   17

1 2 3
4 5 6
7 0 8
h(n) misplaced tiles       :   1
g(n) from root             :   16
f(n) by misplaced tiles    :   17

1 2 3
4 5 6
7 8 0
h(n) misplaced tiles       :   0
g(n) from root             :   17
f(n) by misplaced tiles    :   17

Process finished with exit code 0
```

Problem 4 :
Input State : {1,3,5,4,6,0,8,7,2}
Goal State : {1,2,3,4,5,0,7,8,6}

Output:

```
Enter goal state:
1
2
3

4
5
0

7
8
6

Enter initial state:
1
3
5

4
6
0

8
7
2

Initial node is:
1 3 5
4 6 0
8 7 2

Initial node f(n)=10
Goal node is:
1 2 3
4 5 0
7 8 6
<--------------------------------------------------------------------------->
Choose a heuristic value between following:
press 1 for 'Manhatten Distance'
press 2 for 'misplaced tiles'
1
<--------------------------------------------------------------------------->
Goal found
path cost:16
path:

1 3 5
4 6 0
8 7 2
h(n) manhatten             :   10
g(n) from root             :   0
f(n) by manhatten          :   10

1 3 5
4 0 6
8 7 2
h(n) manhatten             :   9
g(n) from root             :   1
f(n) by manhatten          :   10

1 3 5
4 7 6
8 0 2
h(n) manhatten             :   10
g(n) from root             :   2
f(n) by manhatten          :   12

1 3 5
4 7 6
8 2 0
h(n) manhatten             :   9
g(n) from root             :   3
f(n) by manhatten          :   12
```

```
1 3 5
4 7 0
8 2 6
h(n) manhatten            :   8
g(n) from root            :   4
f(n) by manhatten         :   12

1 3 0
4 7 5
8 2 6
h(n) manhatten            :   7
g(n) from root            :   5
f(n) by manhatten         :   12

1 0 3
4 7 5
8 2 6
h(n) manhatten            :   6
g(n) from root            :   6
f(n) by manhatten         :   12

0 1 3
4 7 5
8 2 6
h(n) manhatten            :   7
g(n) from root            :   7
f(n) by manhatten         :   14

4 1 3
0 7 5
8 2 6
h(n) manhatten            :   8
g(n) from root            :   8
f(n) by manhatten         :   16

4 1 3
7 0 5
8 2 6
h(n) manhatten            :   7
g(n) from root            :   9
f(n) by manhatten         :   16

4 1 3
7 2 5
8 0 6
h(n) manhatten            :   6
g(n) from root            :   10
f(n) by manhatten         :   16

4 1 3
7 2 5
0 8 6
h(n) manhatten            :   5
g(n) from root            :   11
f(n) by manhatten         :   16

4 1 3
0 2 5
7 8 6
h(n) manhatten            :   4
g(n) from root            :   12
f(n) by manhatten         :   16

0 1 3
4 2 5
7 8 6
h(n) manhatten            :   3
g(n) from root            :   13
f(n) by manhatten         :   16

1 0 3
4 2 5
7 8 6
h(n) manhatten            :   2
g(n) from root            :   14
f(n) by manhatten         :   16

1 2 3
4 0 5
7 8 6
h(n) manhatten            :   1
g(n) from root            :   15
f(n) by manhatten         :   16

1 2 3
4 5 0
7 8 6
h(n) manhatten            :   0
g(n) from root            :   16
f(n) by manhatten         :   16

Process finished with exit code 0
```