Report

April 3, 2022

1 Data Structures used

- Node `custom made structure`
- Tree `custom made structure`
- ArrayList
- Queue
- Stack
- Priority Queue
- HashTable
- Enumeration
- ActionPath `custom made structure`

1.1 Action Enumeration

It's used to represent that action taken by the parent node to reach the child node and it contains the following

- `Up`
- `Down`
- `Left`
- `Right`

1.2 Creating the Node

The Node will contain the following data fields:

- `Parent:Node`
- `Children:ArrayList`
- `state:int[][]`
- `stringState:String` *which will be used to generate a hash code and compare it the goal state*
- `Direction:Action` *Action taken to reach this node*
- `depth:int`
- `missingTileRow:int`
- `missingTileCol:int`
- `cost:int`

It contains the following methods:

- `CreateStringBoard():String` which creates a string of the state of the node
- `addChild(Node child):void` which is a helper function when creating a child‘

- `CreateChild(int a, int b):void` Creates a child where paramaters a,b represent the position of missing tile
- `getRowCol(int value):int[]` returns an array of size two, the method is used in calculating the herustic functions
- `equals(Object obj):boolean Override` It overrides the default equals method
- `isGoal():boolean` checks if the Node state is the goal state
- `hashCode():int Override` overrides the default hashCode method so that it returns the hash code of the string state
- `toString:String Override` Overrides the default toString method and makes use of the string builder object

1.3 Creating the ActionPath class

The main goal of this class is to use backtracking the print the path from the goal node to the root node with the neccessary information. The class uses the Stack data-structure *LIFO* where we are going to use the property that the last element that enters the stack is the first element that exits.

The methods the class contains are as follows:

- `getPath:Stack`
- `printPath:void`

`getpath` functions takes both the root node and goal node as inputs and pushes the goal node parents inside it till it reaches the root node.

`printPath` function returns nothing, it is only used for printing the path from the root to the goal.

1.4 Creating the Tree

The tree class will contain the search functions as well as the expand function for the nodes.

Moreover, it contains the f_1 & f_2 Comparator objects that will be used in the priority queue.

It has only one data field : `root:Node`

The inner classes:

- `f_1` which implements Comparator used to determine whic node should enter the priorty queue first, It uses the manhattan distance herustic
- `f_2` has the same functionality as `f_1` but uses the Euclidean distance herustic

The methods:

- `expand(Node node):List<Node>` returns a list of the children of the given node i.e. it expands it
- `manhattanDistance(Node n):int` which calculates the manhattan distance
- `euclideanDistance(Node n):int` which calculates the euclidean distance
- `aStar(int i):void` which implements the A* star search, i paramter is used to determine which herustic to use

2 Breadth first Search : 1 2 5 3 4 0 6 7 8

```
Welcome to 8 puzzle Solver
Enter the puzzle : 1 2 5 3 4 0 6 7 8
Choose the Algorithm

1. BFS
2. DFS
3. A*

Enter your choice: 1
The root node
1       2       5
3       4       0
6       7       8
-----------------------
Current Node:

1       2       0
3       4       5
6       7       8

Direction Moved: Up
Depth: 1
Cost: 1
-----------------------
Current Node:

1       0       2
3       4       5
6       7       8

Direction Moved: Left
Depth: 2
Cost: 2
-----------------------
Current Node:

0       1       2
3       4       5
6       7       8

Direction Moved: Left
Depth: 3
Cost: 3
-----------------------
Time: 1.0 millie seconds
Space: 16
```

3 Depth first Search : 1 2 5 3 4 0 6 7 8

```
Welcome to 8 puzzle Solver
Enter the puzzle : 1 2 5 3 4 0 6 7 8
Choose the Algorithm

1. BFS
2. DFS
3. A*

Enter your choice: 2
The root node
1       2       5
3       4       0
6       7       8
-----------------------
Current Node:

1       2       5
3       4       8
6       7       0

Direction Moved: Down
Depth: 1
Cost: 1
-----------------------
Current Node:

1       2       5
3       4       8
6       0       7

Direction Moved: Left
Depth: 2
Cost: 2
-----------------------
Current Node:

1       2       5
3       0       8
6       4       7

Direction Moved: Up
Depth: 3
Cost: 3
-----------------------
Current Node:
```

```
1       0       5
3       2       8
6       4       7

Direction Moved: Up
Depth: 4
Cost: 4
-----------------------
Current Node:

1       5       0
3       2       8
6       4       7

Direction Moved: Right
Depth: 5
Cost: 5
-----------------------
Current Node:

1       5       8
3       2       0
6       4       7

Direction Moved: Down
Depth: 6
Cost: 6
-----------------------
Current Node:

1       5       8
3       2       7
6       4       0

Direction Moved: Down
Depth: 7
Cost: 7
-----------------------
Current Node:

1       5       8
3       2       7
6       0       4

Direction Moved: Left
Depth: 8
Cost: 8
-----------------------
```

```
Current Node:

1       5       8
3       0       7
6       2       4

Direction Moved: Up
Depth: 9
Cost: 9
-----------------------
Current Node:

1       0       8
3       5       7
6       2       4

Direction Moved: Up
Depth: 10
Cost: 10
-----------------------
Current Node:

1       8       0
3       5       7
6       2       4

Direction Moved: Right
Depth: 11
Cost: 11
-----------------------
Current Node:

1       8       7
3       5       0
6       2       4

Direction Moved: Down
Depth: 12
Cost: 12
-----------------------
Current Node:

1       8       7
3       5       4
6       2       0

Direction Moved: Down
Depth: 13
```

```
Cost: 13
-----------------------
Current Node:

1       8       7
3       5       4
6       0       2

Direction Moved: Left
Depth: 14
Cost: 14
-----------------------
Current Node:

1       8       7
3       0       4
6       5       2

Direction Moved: Up
Depth: 15
Cost: 15
-----------------------
Current Node:

1       0       7
3       8       4
6       5       2

Direction Moved: Up
Depth: 16
Cost: 16
-----------------------
Current Node:

1       7       0
3       8       4
6       5       2

Direction Moved: Right
Depth: 17
Cost: 17
-----------------------
Current Node:

1       7       4
3       8       0
6       5       2
```

```
Direction Moved: Down
Depth: 18
Cost: 18
----------------------
Current Node:

1       7       4
3       8       2
6       5       0

Direction Moved: Down
Depth: 19
Cost: 19
----------------------
Current Node:

1       7       4
3       8       2
6       0       5

Direction Moved: Left
Depth: 20
Cost: 20
----------------------
Current Node:

1       7       4
3       0       2
6       8       5

Direction Moved: Up
Depth: 21
Cost: 21
----------------------
Current Node:

1       0       4
3       7       2
6       8       5

Direction Moved: Up
Depth: 22
Cost: 22
----------------------
Current Node:

1       4       0
3       7       2
```

```
6       8       5

Direction Moved: Right
Depth: 23
Cost: 23
-----------------------
Current Node:

1       4       2
3       7       0
6       8       5

Direction Moved: Down
Depth: 24
Cost: 24
-----------------------
Current Node:

1       4       2
3       7       5
6       8       0

Direction Moved: Down
Depth: 25
Cost: 25
-----------------------
Current Node:

1       4       2
3       7       5
6       0       8

Direction Moved: Left
Depth: 26
Cost: 26
-----------------------
Current Node:

1       4       2
3       0       5
6       7       8

Direction Moved: Up
Depth: 27
Cost: 27
-----------------------
Current Node:
```

```
1       0       2
3       4       5
6       7       8

Direction Moved: Up
Depth: 28
Cost: 28
-----------------------
Current Node:

0       1       2
3       4       5
6       7       8

Direction Moved: Left
Depth: 29
Cost: 29
-----------------------
Time: 1.0 millie seconds
Space: 55
```

4 A* search

4.1 Manhattan distance - puzzle : 1 2 5 3 4 0 6 7 8

The first Search -Manhatten distance- using the following puzzle : 1 2 5 3 4 0 6 7 8

```
Welcome to 8 puzzle Solver
Enter the puzzle : 1 2 5 3 4 0 6 7 8
Choose the Algorithm

1. BFS
2. DFS
3. A*

Enter your choice: 3
Choose the Heuristic function

1. Manhattan Distance
2. Euclidean Distance

Enter your choice: 1
The root node
1       2       5
3       4       0
6       7       8
-----------------------
Current Node:
```

```
1       2       0
3       4       5
6       7       8

Direction Moved: Up
Depth: 1
Cost: 1
-----------------------
Current Node:

1       0       2
3       4       5
6       7       8

Direction Moved: Left
Depth: 2
Cost: 2
-----------------------
Current Node:

0       1       2
3       4       5
6       7       8

Direction Moved: Left
Depth: 3
Cost: 3
-----------------------
Time: 0.0 millie seconds
Space: 6
```

5 Euclidean distance - puzzle : 1 2 5 3 4 0 6 7 8

The second search uses the ecludiean distance to find goal node. It uses the same puzzle as the manhattan herustic

```
Welcome to 8 puzzle Solver
Enter the puzzle : 1 2 5 3 4 0 6 7 8
Choose the Algorithm

1. BFS
2. DFS
3. A*

Enter your choice: 3
Choose the Heuristic function
```

```
1. Manhattan Distance
2. Euclidean Distance

Enter your choice: 2
The root node
1       2       5
3       4       0
6       7       8
-----------------------
Current Node:

1       2       0
3       4       5
6       7       8

Direction Moved: Up
Depth: 1
Cost: 1
-----------------------
Current Node:

1       0       2
3       4       5
6       7       8

Direction Moved: Left
Depth: 2
Cost: 2
-----------------------
Current Node:

0       1       2
3       4       5
6       7       8

Direction Moved: Left
Depth: 3
Cost: 3
-----------------------
Time: 0.0 millie seconds
Space: 6
```