

## Programming assignment 1

The objective is to implement various search strategies to solve the 8-puzzle, as described on page 70 of the textbook. Submit your completed code, as a single zip file of all `.java` source files, to Moodle. Please see the course webpages for a general statement on the criteria for grading code. Note that while developing a solution, you will probably need to print out plenty of debugging information. However, the submitted version must produce output in exactly the same format as the code framework provided, without any additional output.

The code framework for this assignment, available on the course webpages, contains an incomplete version of breadth-first search for the 8-puzzle. Familiarize yourself with the framework by reading the code and the comments, and ensure that you can run the main method in `EightsPuzzleMain` from the command line, using `java EightsPuzzleMain bfs tree -1 -1`. (The code will not yet produce the correct output.)

### Question 1 (30%)

Improve the existing algorithm in the following five ways:

- Fill in the missing code in the following two methods from the `EightsPuzzleWorldState` class: `getValidActions()`, and `apply()`. The program should now find a solution, but it ignores some of the command line arguments and does not compute the number of expanded and generated nodes. Specifically, the output should be:

```
Solution found.  
Expanded nodes: 0  
Generated nodes: 0
```

- At present, the algorithm does not compute the number of expanded and generated nodes correctly. Add this functionality. The output should now be:

```
Solution found.  
Expanded nodes: 43  
Generated nodes: 121
```

- At present, the algorithm ignores the value of `maxNodes` in the `ClassicalSearch` class. Ensure that the algorithm terminates with failure if it attempts to expand more than `maxNodes`. The special value `maxNodes == -1` indicates no maximum on the number of nodes expanded. (From this point on in the assignment, you will need to carefully test the behavior of your algorithm to ensure that it is correct. The expected outputs will not be provided. Creating suitable JUnit tests would be one effective way of testing your code.)
- At present, the algorithm ignores the value of `maxDepth` in the `ClassicalSearch` class. Ensure that the algorithm terminates with failure if it has explored all nodes at depths up to and including `maxDepth` without finding a solution. The special value `maxDepth == -1` indicates no maximum on the depth.

- At present, tree search is implemented but graph search is not. Ensure that when the `graph` option is specified on the command line, the algorithm never expands the same state more than once.

### Question 2 (20%)

Implement depth first search, by creating and implementing a `DepthFirstSearchNode` class that extends `SearchNode`. Also make the necessary changes to `EightsPuzzleMain` so that the `dfs` option now works from the command line.

### Question 3 (10%)

Implement A\* search with the number-of-misplaced-tiles heuristic described as  $h_1$  on page 103 of the textbook. This can be accomplished by creating and implementing an `AStarNumTiles` class that extends `SearchNode`. Also make the necessary changes to `EightsPuzzleMain` so that the `as1` option now works from the command line.

### Question 4 (15%)

Implement A\* search with the Manhattan distance heuristic described as  $h_2$  on page 103 of the textbook. This can be accomplished by creating and implementing an `AStarManhattan` class that extends `SearchNode`. Also make the necessary changes to `EightsPuzzleMain` so that the `as2` option now works from the command line.

### Question 5 (15%)

Your `AStarNumTiles` and `AStarManhattan` classes contain some shared functionality. Refactor your code to eliminate the shared functionality. Hint: one way to do this involves creating a new abstract class.

### Question 6 (10%)

Make any changes necessary so that the code works on different sizes of puzzle. For example, by redefining `START_BOARD` as `{{ 0, 1, 2, 3 }, { 4, 5, 6, 7 }, { 8, 9, 10, 11 }, { 12, 13, 14, 15 }}`, and declaring a suitable goal board, the code should now be able to solve 15-puzzles. It should also work for arbitrary rectangular puzzles (e.g. a 3x4 board). Note: if your code was written using good software development technique, no other changes should be required — or at most, changes to one or two constants. If you find that further changes are required, make note of how you could avoid such problems in future projects, by avoiding the use of hard-wired assumptions in your code.

### Suggested further work

(Sorry, no extra credit is available. Completing this further work will, however, be extremely beneficial for understanding the algorithms and increasing your maturity as a computer scientist.)

- Examine the number of nodes explored and generated for each of the algorithms implemented. Do the numbers make sense?
- The goal state provided in the code framework is a very easy goal. Rerun your algorithms on a moderately difficult goal (e.g.  $\{\{0, 3, 1\}, \{7, 6, 2\}, \{4, 8, 5\}\}$ ) and a difficult goal (e.g.  $\{\{7, 2, 4\}, \{5, 0, 6\}, \{8, 3, 1\}\}$ ). Again, do the results make sense?
- Implement iterative deepening search.

**Acknowledgment:** Several of the above questions are based on a lab created by Prof. Grant Braught in 2008, and I'm grateful for his permission to incorporate this material.