COMP 250: Introduction to Computer Science
Assignment 5

Posted Thursday April 3, 2014
Due Friday April 11, 2014
No penalties until Monday April 28, 2014

1 Introduction

In this homework you will write a search program for solving a sliding tile puzzle, also know as an 8-puzzle. The sliding puzzle consists of a 3×3 board with 8 numbered tiles and a blank space, denoted by a zero. A tile adjacent to the blank space can slide into the space. You can see and solve such puzzles yourselves at: http://www.tilepuzzles.com/default.asp?p=12

The objective is to figure out the steps needed to get from one configuration (which is is an arbitrary arrangement of the tiles) to a goal configuration. For example, one might start with:

1	2	3		1	2	3
4	0	5	and aim to reach the goal configuration:	4	5	6
7	8	6		7	8	0

Finding an optimal solution (i.e. a solution in a minimal number of moves) for a puzzle of size $n \times n$ is NP-complete, but nonetheless, solving particular puzzles is quite feasible. In this homework, you will implement a version of the A^* algorithm, which is a breadth-first-search-style algorithm that works well for this problem.

2 Description of the algorithm

Your program will take an initial board position, and then try to find a sequence of moves from the initial position to the goal position. Moves are represented by moving the blank space left, right, up, or down. For example, suppose we wanted to solve the puzzle shown above. This could be accomplished by the following sequence of moves (we represent a move as the direction the empty space moved):

1	2	3		1	2	3		1	2	3
4	0	5	R	4	5	0	U	4	5	6
7	8	6		7	8	6		7	8	0

where R=right, L=left, U=up, D=down. So we can solve this puzzle with the sequence of moves ”RD”.

In the code provided, TileBoard.java represents a board using a string of tiles. For example, the goal state above is encoded by the String ”123456780”. It also stores a string representing a sequence of moves that can be used to obtain this board. E.g. in the case above, it would store the string ”RU”. You will need to design and implement a functionality for manipulating boards. Try to be efficient in both memory and time. Make sure to comment your design decisions!

The A^* search algorithm is based on the idea of prioritizing boards that are close to the goal configuration. More precisely, consider a board, which we will also call a *state*, s. Let $L(s)$ be the number of moves made to reach this board. Let $H(s)$ be a *heuristic* or guess about the number of moves remaining until a goal can be reached. The algorithm will use a priority queue (use the Java pre-defined PriorityQueue class in your code), in which boards are ordered increasingly by the value:

$$L(s) + H(s)$$

Note that if H is always 0, we obtain breadth-first search based on the cost of the solution. In general, if H is *optimistic,* i.e. it under-estimates the cost to the goal, this approach is guaranteed to yield the optimal solution. Such heuristics are typically obtained from a *relaxation* of the problem, i.e. we think of a problem which is easier to solve, and compute the cost of the optimal solution in the relaxed problem.

It now remains to specify a heuristic that would work well. In this case, we will use the *Manhattan distance*, which comes from a relaxation of the 8-puzzle in which we are allowed to move a tile on top of another tile. More precisely, the Manhattan distance of two boards is the sum of the distances of each tile from where it belongs, completely ignoring all the other tiles. For example, the Manhattan distance between ”213540678” and ”123456780” is 9, because tile 3 is in its place, tile 6 would require 3 moves to be put in the right place, and all other tiles are 1 move away from their goal place; summing these up gives 9. Note that we do *not* consider the empty space in this calculation, as it is not really a tile.

3 What you need to do

For the assignment, you should fill out the methods indicated in the files TileBoard.java and Sliding-Solver.java.

In class TileBoard.java, you need to make a constructor, write a method that generates the possible next boards given the current one (by swapping the space with an adjacent tile), and the method that computes the Manhattan distance between a board passed as argument and the board in the current object. You should feel free to add both attributes and method to this class if you need them.

In SlidingSolver.java, the solvePuzzleAStar will contain the code for performing the search. For best performance, you should check before inserting a board in the priority queue if it already exists. If so, you should keep only one copy, which corresponds to the lowest value of $L + H$. Most of the code that you need will be in this method. You also need to make a constructor and a comparator method, which will be used by the priority queue, in order to decide how to enqueue boards. Please cut off the search and return ”no solution” after a maximum number of moves has been reached (this is specified in a static variable in the code, and set to 20). You should feel free to add attributes and methods to this class if you need them.

The SlidingSolution.java class contains some helper methods and you should not need to modify it. The file SlidingSolverDriver.java contains a main method to test the code. You can put in different boards here to test your code. Please do not add any classes beyond the ones provided.

You need to turn in your SlidingSolver.java and TileBoard.java files, as well as a table with 3 columns. In the first column you will show an initial board, in the second column the number of moves it takes to solve the problem, and in the last column the solution string (containing the moves). You should show at least 5 initial boards, with different lengths of optimal solution.

4 Grading scheme

- A^* algorithm - 40 points
- Manhattan distance - 20 points
- Move generator in TileBoard.java - 10 points
- Constructor and comparator in SlidingSolver.java and constructor in TileBoard.java - 15 points
- General coding style, comments - 10 points
- Testing table - 5 points