Katholieke Universiteit Leuven

Data Structures and Algorithms

Assignment 2
8-Puzzle

Author: Pieter Verlinden
Student ID: r0369547
Study: 1Ba Informatica

Lecturer: Dr. Ir. Philip Dutré

April 5, 2013

Contents

1 Introduction

The purpose of this assignment is to study the importance of data-structures, in particular the "priority-queue", by writing a program that solves the 8-puzzle-problem (see section 2).

This puzzle-problem can be solved by implementing the A* algorithm[2]. This algorithm uses priorityfunctions such as the Manhatten priority and the Hamming priority (see section 2.1), to determine the closest path to the goalstate.

In section 2.2, some experimental data will be given based on the two used priorityfunctions. With this data, conclusions will be made and some questions will be answered in section 3.

2 8-puzzle

The 8-puzzle, also called a sliding puzzle, is a puzzle that consists of a frame of numbered tiles in random order (see Figure 1).

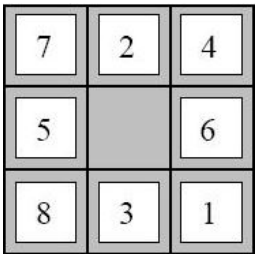

Figure 1: An unsolved 8-puzzle.

2.1 Solvemethod - A* algorithm

To solve this kind of puzzles, we use the widely known "A* algorithm"[2] developed by Peter Hart, Nils Nilsson and Bertram Raphael.
To implement this algorithm, we first define a state. This state holds the number of moves made so far, the current board position (the configuration of the board) and the previous state.
Each initial board has neighbours (a.k.a. all the boards with swapped empty places). Create a state of each neighbour and add it to a minimum priority queue. Delete and assign the state with the least priority to a temporary state, and repeat the above algorithm.

The priority of each state is the key to success of this algorithm. Two priorityfunctions are defined: the Manhattan priorityfunction and the Hamming priorityfunction.
The Manhattan priorityfunction holds the sum of the distances between the position of each element and its goalposition. The Hamming priorityfunction holds the amount of elements out of place[1].

By taking the state with the least priorityscore (a.k.a. the board with the least distance between this state an the goalstate or the amount of tiles out of place) we know that this state is closest to the goalstate.

2.2 Experimental data

puzzle	moves	Hamming (s)	Manhattan (s)
puzzle28.txt	28	2	1
puzzle30.txt	30	4	1
puzzle32.txt	32	>5min	2
puzzle34.txt	34	>5min	1
puzzle36.txt	36	>5min	4
puzzle38.txt	38	>5min	4
puzzle40.txt	40	>5min	2
puzzle42.txt	42	>5min	8

Table 1: Experimental data on the A* algorithm for solving 8-puzzles.

From this data we can conclude that the Manhattan priorityfunction is extensively more efficient than the Hamming priority. This is due to the fact that the Hamming priority gives less information about the configuration of a board than the Manhattan priority.

3 Code clarification

In this section, some of the implemented code will be clarified and explained.

Board.java first constructs a board of tiles, using a two-dimensional array. Then, it stores the indexes of the empty spot in two variables. The class also implements the Iterable interface, which makes it possible to loop through all the neighbouring boards of this board.
Neighbours are boards with a tile configuration where the empty spot is moved one tile from its initial spot, in each direction (see Figure 2).

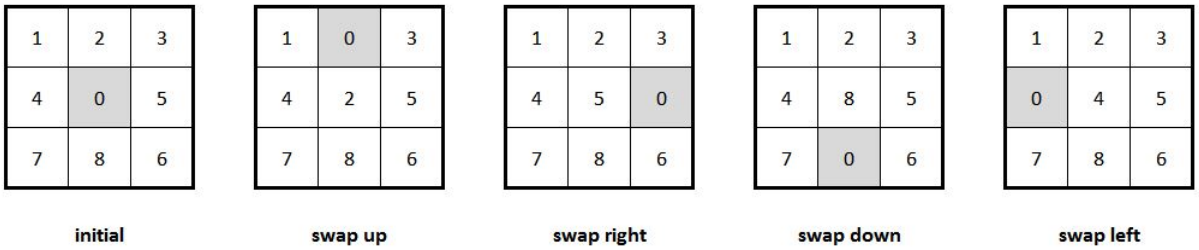

Figure 2: An initial 8-puzzle board and its neighbours.

To get these neighbouring boards, extra methods such as "swapUp" and "swapLeft" and extra checkers such as "isSwappableUp" and "isSwappableLeft" are made. These functions use a new exchange method that swaps elements in a two-dimensional array.

A state (as explained in section 2.1)is defined by implementing a new class: InterState.java. This class implements the interface Comparable, which makes it possible to put InterState objects in a priority queue. On top of that, it holds an integer "moves" which holds the current amount of moves done so far, a previous InterState, the current board and finally an integer holding the priorityscore. This score consists out of the sum of the amount of moves and an amount of a priorityfunction.

To check whether a puzzle is solvable, an isSolvable method is made in Solver.java. This method is slightly different from the given method, and is based on the method of odd and even inversions stated in the 8 puzzle assignment: cos 226 from Princeton University[1].

From the experimental data (see Table 1), we can derive that the priorityfunction is a key element to a good and fast working algorithm. The Manhattan priorityfunction is significantly faster and less memory-absorbent than the Hamming priorityfunction. So if we ask ourselves, what is the best solution for bigger and more difficult puzzles, we could answer that a thorough study on priorityfunctions is the most efficient way to fasten our algorithm. If this is still not enough, more memory could be added. The law of Moore[3] states that transistors on integrated circuits, double every eighteen months. This makes enlarging the memory a plausible solution.

4 Conclusion

In my opinion, this assignment was very instructive and informative. It shows us that a good algorithm can solve almost any problem in a significant amount of time. By implementing the famous A* algorithm, I acquired more insight in the solving of problems and more experience in debugging and optimizing my code. Also concepts like a priority queue, interfaces and generic types like Iterable and Comparable are more clear to me now.
Though this assignment costed a lot of time and work, the experience and knowledge gained was fully worth it.

References

[1] cs.princeton.edu. 8 puzzle - cos 226 programming assignment. http://www.cs.princeton.edu/courses/archive/fall12/cos226/assignments/8puzzle.html.

[2] Wikipedia. A* search algorithm. http://en.wikipedia.org/wiki/A*_search_algorithm.

[3] Wikipedia. Moore's law. http://en.wikipedia.org/wiki/Moore%27s_law.