CSC 225 SUMMER 2016
ALGORITHMS AND DATA STRUCTURES I
ASSIGNMENT 4 - PROGRAMMING
UNIVERSITY OF VICTORIA

1 Programming Assignment

The 9-puzzle consists of a square grid containing eight tiles, marked with the numbers 1 through 8. One of the spaces in the grid is empty. The initial state of the puzzle is the configuration below:

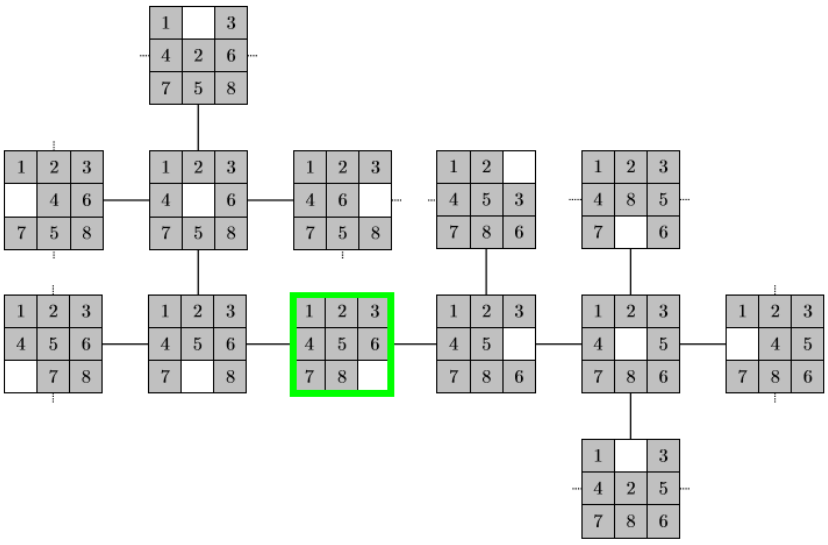
1	2	3
4	5	6
7	8	

This is considered to be the ‘solved’ state of the puzzle and is normally called the ‘goal state’. The tiles on the board can be moved horizontally or vertically into the empty space to scramble the ordering of the board, as in the configuration below:

4	1	3
	2	6
7	5	8

The programming assignment is to implement a solver for the 9-puzzle. Your submission will read a sequence of boards and, for each board, output a sequence of moves which solves the board.

The different configurations of the 9-puzzle and their relationships can be modeled with a graph. Each vertex corresponds to a possible configuration of the board. Edges represent the transformations possible by making one (legal) move. That is, if two different board configurations are connected by an edge, there is a way to obtain one configuration from the other by making a single move. To solve a given board, tiles are moved until the goal state is reached. The diagram below shows a small snapshot of the graph, with the goal state framed in green.



The set of moves needed to solve a given board is represented by a path in the graph from the board to the goal state. Therefore, a board can be solved by performing one of the two fundamental graph traversals–DFS or BFS–on the graph and searching for a path to the goal state. Some possible board configurations cannot be solved, such as the following:

1	2	3
4	5	6
8	7	

Your task is to write class `SolveNinePuzzle`. The main method in your code should help you test your implementation by entering test data or reading it from a file.

2 Input Format

9-puzzle boards are input as 3×3 tables, with the character '0' representing the empty square. For example, the board

4	1	3
	2	6
7	5	8

would be represented by the input sequence

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

3 Suggested Algorithm

If v is the vertex corresponding to the input board and u is the vertex corresponding to the goal state, then v is solvable if and only if a traversal (DFS or BFS) rooted at v encounters u (or vice-versa in fact). If u is never encountered, then `SolveNinePuzzle` should return `false`. If u is encountered, then the traversal tree computed by DFS or BFS can be used to find a path from v to u in the graph (specifically, by starting at u and tracing upward through the tree until v is reached). The implementation should print every board encountered along this path.

The exact implementation of the `SolveNinePuzzle` program is up to you. However, the algorithm outlined below is suggested for simplicity:

- Create the vertex v of G corresponding to the input board B.
- Create the vertex u of G corresponding to the goal state.
- Run DFS or BFS on G starting at v.
- If u is encountered by the traversal, print each board on the $v - u$ path and return `true`.
- Otherwise, return `false`.

In practice, puzzles like the 9-puzzle and 16-puzzle are solved with more advanced artificial intelligence algorithms, which are beyond the scope of this course.

An example pseudocode algorithm for solving this problem using DFS is as follows:

Algorithm ninePuzzleDFS(Board B, Board G, HashTable H, Path P):
    Input: Current board B as a 3×3 integer array, the goal board G as a 3×3 integer array, hash table H and some data structure holding the path so far, P.
    Output: Boolean **true** is a path exists, and **false** if one does not exist.

    if $B = G$ **then** \\ G is the goal board
        print P
        return true

    if B is NOT is H **then** \\Vertex B is unexplored
        add B to H
        for each adjacent board, B_i \\ $i = 2, 3$ or 4 of them
            add B_i to P
            ninePuzzleDFS(B_i,G,H,P)
            remove B_i from P
    return false
end

Each vertex of the underlying graph corresponds to a possible 9-puzzle board, but it can be helpful to have vertices represented by strings or integers instead of 3×3 arrays to facilitate indexing into a hash table. To enable this, you may use the `Arrays.deepToString(int[][])` to convert the array into a string. This will allow you to insert the vertices into the java hashing class `HashSet`. Note, there is no need to actually create the graph here. We can perform graph algorithms on the vertices even though we are storing them in a hash table because we know that every vertex has at most 4 adjacent vertices which can be determined when needed using the given board. So, you can create an abstract version of the graph on the fly.

4 Test Datasets

A collection of randomly generated solvable and unsolvable boards has been uploaded to conneX. Your assignment will be tested on boards similar but not identical to the uploaded boards. You may want to create your own collection of test boards.

5 Sample Run

The output of a model solution on the board specified above is given in the listing below. Note that there may be multiple move sequences that solve a given board. Console input is shown in blue.

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

Attempting to solve board 1...
4 1 3
0 2 6
7 5 8

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

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

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Board 1: Solvable.
Processed 1 board.
Average Time (seconds): 0.00
```

6 Evaluation Criteria

The programming assignment will be marked out of 25, based on a combination of automated testing and human inspection. The running time of the implemented algorithm should be at most $O(n^2)$, where n is the number of vertices in the constructed graph G. To receive full marks, the implementation should solve each board with the smallest number of moves possible. This can be achieved by using BFS instead of DFS.

Score (/25)	**Description**
0 - 5	Submission does not compile or does not conform to the provided description.
5 - 15	The implemented algorithm is not $O(n^2)$ or is substantially inaccurate on the tested inputs.
15 - 20	The implemented algorithm is $O(n^2)$ and accurate on all the tested inputs.
20 - 25	The implemented algorithm is $O(n^2)$, accurate on all the tested inputs, and the sequence of moves for each tested board is the shortest possible length.

To be properly tested, every submission must compile correctly as submitted. **If your submission does not compile for any reason (even trivial mistakes like typos), it will receive at most 5 out of 25**. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor before the due date.