CSC3100 Assignment 4

Important Notes:

1. The assignment is an individual project, to be finished on one's own effort.
2. **The normal submission deadline is 6pm May 8 (Monday), 2023.**
3. **The late-/re-submission deadline is 6pm May 11 (Thursday), 2023.**
4. Please also submit your two **programs** and rename them as "StudentID_A4_ Puzzle8.java" and "StudentID_A4_ BST.java" on the blackboard. For example, a student whose Student ID is "120000001" should submit **two programs** named as "120000001_A4_Puzzle8.java" and "120000001_A4_ BST.java". **You don't need to consider the consistency of class name and file name** since we won't run the code you submitted on the blackboard. File misnaming or no submission on the blackboard will lead to **5 demerit points.**
5. Plagiarism is strictly forbidden, regardless of the role in the process. Notably, ten consecutive lines of identical codes are treated as plagiarism.
6. OJ website: http://oj.cuhk.edu.cn/. If you are off campus, please use VPN to access the OJ.
7. Access code: csc3100as4
8. Each student is only permitted to submit code to OJ **up to Twenty times for the problem**. Only the **last submission** will be used in evaluation of assignment marks.
9. Plagiarism is strictly forbidden, regardless of the role in the process. Notably, ten consecutive lines of identical codes are treated as plagiarism.

Marking Criterion:

1. For normal submission, the full score of **this assignment (Assignment 4)** is 150 marks.
2. For late-/re-submission, the full score of **this assignment (Assignment 4)** is 120 marks.
3. **Question 1** takes 66.7% weight (i.e., 100 or 80 marks in normal and late submissions respectively) with 10 test cases and 10 marks each. **Question 2** takes 33.3% weight (i.e. 50 or 40 marks in normal and late submissions respectively) with 50 test cases with 10 marks each.
4. Notice that for the cumulative score, **T marks**, of **all 4 assignments in this semester**, the **final score** for a student is the **min(400, T)**. For example, if a student A got 450 marks of all 4 assignments, student A's final score of assignments would be min(400, 450) = 400.

Running Environment:

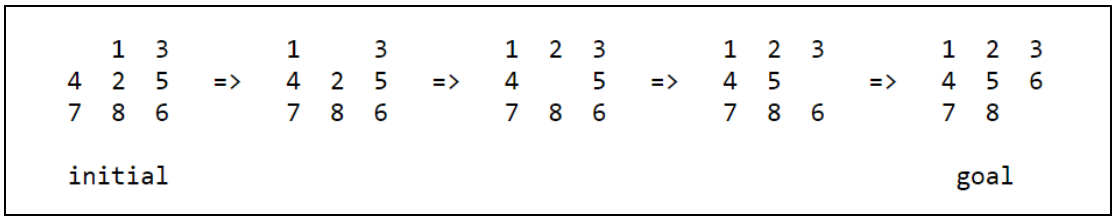
1. The submissions will be evaluated in Java environment under Linux platform.
2. The submission is acceptable if it runs in any of recent versions of Java SDK environment. These versions include from Java SE 8 to the most recent Java SE 17.
3. The submission is only allowed to import three packages of (java.lang.*; java.util.*; java.io.*) included in Java SDK. No other packages are allowed.
4. In the test, each program is required to finish within **30 seconds of time**, with no more than **128MB memory**. **This is a strict requirement measured in the server environment!**
5. All students will have an opportunity to test their programs in the OJ platform prior to the official submission.

Submission Guidelines:

1. Inconsistency with or violation from the guideline leads to marks deduction.
2. It is the students' responsibility to read this assignment document and submission guidelines carefully and in detail. No argument will be accepted on issues that have been specified clearly in these documents.

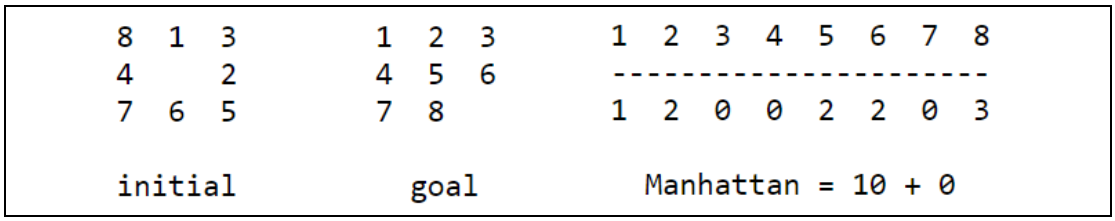
Question 1-- Problem Description:

Write a program to solve the 8-puzzle problem (and its natural generalizations) using the A* search algorithm. The 8-puzzle problem is a puzzle invented in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. You are permitted to slide blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an initial board position to the goal position.



An algorithmic solution to the problem is based on the classical A* search algorithm (https://en.wikipedia.org/wiki/A*_search_algorithm). We define a state of the game to be the board position, the number of moves made to reach the board position, and the previous state. First, insert the initial state (the initial board, 0 moves, and a null previous state) into a priority queue. Then, delete from the priority queue the state with the minimum priority, and insert onto the priority queue all neighboring states (those that can be reached in one move). Repeat this procedure until the state dequeued is the goal state. The success of this approach hinges on the choice of priority function for a state. We consider the following priority function:

- **Manhattan priority function.** The sum of the distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the state.



There is a key observation here: to solve the puzzle from a given state on the priority queue, the total number of moves we need to make (including those already made) is at least its priority, using the Manhattan priority function. (This is true because each block must move its Manhattan distance from its goal position. Note that we do not count the blank tile when computing the Manhattan priorities.)

Consequently, as soon as we dequeue a state, we have not only discovered a sequence of moves from the initial board to the board associated with the state, but one that makes the fewest number of moves.

Question 1-- Functional Requirement

Write a program called **Puzzle8.java** that reads the initial board from the input of three lines and outputs a sequence of board positions that solves the puzzle in the fewest number of moves.

The input has three lines and consists of the 3-by-3 initial board position. The input format uses "0" to represent the blank square and each number is separated by a space. As an example,

Input Example:
0 1 3 4 2 5 7 8 6

After running the program, the output file will be:

Output Example:
4 0 1 3 4 2 5 7 8 6 1 0 3 4 2 5 7 8 6 1 2 3 4 0 5 7 8 6 1 2 3 4 5 0 7 8 6 1 2 3 4 5 6 7 8 0

Note in the output example.

- Line 1 is **the fewest number of moves to solve the puzzle**, followed by an empty line.
- Line 3-6 is the initial board as given in the input, followed by an empty line.
- Line 7-9 is the board after the first move, followed by **an empty line**.
- …
- Line xx-yy is the objective board of the puzzle that has been solved.
- The final line **is not an empty line**.
- The 8-puzzle problem is always be solvable in this assignment.

Question 2-- Problem Description and Functional Requirement:

Write a Java program called **BST.java** that reads a sequence of **distinct input integers**, then builds a binary search tree by adding a new node to the tree with each input number one by one, and outputs the pre-order, in-order and post-order traversals of the binary search tree.

The input is a line containing a number of distinct integers separated by a space.

Consider the following input:

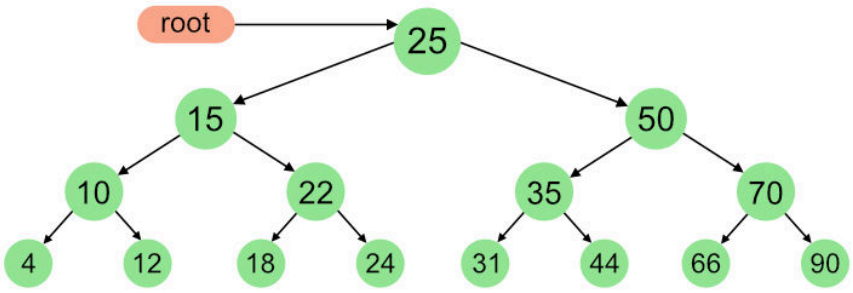
Input Example:
25 15 50 35 44 70 31 66 90 10 4 12 22 18 24

Based on the order of the input numbers, the corresponding binary search tree and the traversal orders are shown as follows.

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Therefore, your program is expected to output the following:

Output Example:
4 10 12 15 18 22 24 25 31 35 44 50 66 70 90
25 15 10 4 12 22 18 24 50 35 31 44 70 66 90
4 12 10 18 24 22 15 31 44 35 66 90 70 50 25

Note that in the output example, there are three lines in total.

- Line 1 is the in-order traversal result, with numbers **separated by a space**.
- Line 2 is the pre-order traversal result, with numbers **separated by a space.**
- Line 3 is the post-order traversal result, with numbers **separated by a space.**
- In the test, the number of input integers is between 2 and 1000.
- The input sequence may produce a binary search tree of any structure, which may not be complete or perfect as in this example.