UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science & Applied Mathematics

Advanced Analysis of Algorithms
Lab 2: Uninformed to Informed Search

1 Introduction

In this lab, we'll be extending your implementation of the breadth first search by changing the cost function and adding in heuristic functions. If you are submitting to the Python version of the marker, you can just upload your `.py` file as is. If you are uploading the Java version, please ensure your file is called `Program.java` and upload a `.zip` of this file.

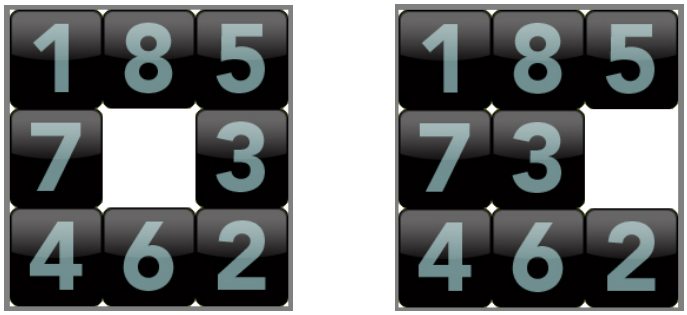
Due Date: 15 Septmber 2025

2 8-Puzzle

The 8-Puzzle is a game in which a player is required to slide tiles around on a 3×3 grid in order to reach a goal configuration. If you have never played before, I recommend trying it out here: `http://www.artbylogic.com/puzzles/numSlider/numberShuffle.htm?rows=3&cols=3&sqr=1`.

There is one empty space that an adjacent tile can be slid into. An alternative (and simpler) view is that we are simply moving the blank square around the grid. Note that not all actions can be used at all times. For example, the blank tile cannot be shifted up if it is already on the top row!

The transition below illustrates the blank tile being moved to the `right`.



(a) (b)

Figure 1: Starting in the configuration (a), the agent executes the action `right`. This moves the blank tile to the right, exchanging its position with 3, as seen in (b).

Submission 1: Uniform-cost search (10 marks)

Lets begin by modifying our code for the BFS to implement a UCS. First, modify the problem so that all actions have cost 1, except for `UP`, which now has a cost of 5 (do not remove the ability to go back to uniform costs later on). Then, write a Python or Java program that accepts an initial and goal state, and returns the cost of the shortest path between the two according to UCS.

Input

Input consists of two lines. The first line is a representation of the initial state, and the second line is the goal state.

Output

Output the cost of the optimal plan from the start to the goal state.

Example Input-Output Pairs

Sample Input #1	Sample Input #2	Sample Input #3
78651#432 12345678#	1857#3462 1#5783462	1857#3462 78651432#
Sample Output #1	**Sample Output #2**	**Sample Output #3**
49	2	36

Submission 2: Greedy Best-First Search (20 marks)

Now lets move away from uninformed searches and begin to consider informed searches. To begin with we will drop the cost counting entirely and just use a heuristic function. Return the problem to the setting where all actions have cost 1 (including for `UP`). Then, write a Python or Java program that accepts an initial and goal state, and returns the cost of the shortest path between the two states (path cost is still counted in terms of number of moves to the white square to reach the goal state).

The heuristic function you should use is the sum of the manhattan distance for each tile from its final destination in the 3×3 grid. Reminder that the manhattan distance in $\mathbb{R}^N$ is:

$$d(x, y) = \sum_{i=o}^{N} |x_i - y_i|$$

Input

Input consists of two lines. The first line is a representation of the initial state, and the second line is the goal state.

Output

Output the cost of the optimal plan from the start to the goal state.

Example Input-Output Pairs

Sample Input #1	Sample Input #2	Sample Input #3
`5#1742638` `524316#87`	`1738546#2` `13648572#`	`3547#1628` `71246#835`
Sample Output #1	**Sample Output #2**	**Sample Output #3**
`31`	`49`	`63`

Submission 3: A* search (20 marks)

Now let's combine our implementations of BFS and greedy-BFS to implement A*. Once again, use the version of the problem where all actions have cost 1 and the manhattan distance as your heuristic function. Then write a Python or Java program that accepts an initial and goal state, and returns the cost of the shortest path between the two according to A*.

Input

Input consists of two lines. The first line is a representation of the initial state, and the second line is the goal state.

Output

Output the cost of the optimal plan from the start to the goal state.

Example Input-Output Pairs

Sample Input #1	Sample Input #2	Sample Input #3
`5#1742638` `524316#87`	`1738546#2` `13648572#`	`3547#1628` `71246#835`
Sample Output #1	**Sample Output #2**	**Sample Output #3**
`19`	`25`	`21`

Submission 4: Written Submission (50 marks)

For this lab we will be experimenting with the time complexity of the informed search algorithms with different heuristic functions. Similar to Lab 1 we need to control the complexity of the problem we ask the algorithms to solve. Use your program that generates game boards which require k moves to solve. Once the desired number of moves have been made, pass the resulting game state into your greedy-BFS and A* functions and set the goal state to be $12345678\#$ (the solved puzzle).

Repeat this process for variable solution depths k and plot this relationship with the time complexity of your algorithms. Measure time complexity using the wall-clock time from the start to the end of the search algorithm. For a given value of k repeat the algorithm 5 times and plot both the mean and 2 standard deviations for both complexity measures. Finally, compare these results to what happens when the euclidean distance is used as the heuristic function instead. Reminder that the euclidean distance in $\mathbb{R}^N$ is:

$$d(x, y) = \sqrt{\sum_{i=o}^{N} x_i - y_i}$$

Input

There is no input as you will begin with $12345678\#$ as the game state and choose a value of k to run the experiment.

Submission

Submit a short (1 to 2 page) report in pdf format which depicts the following:

- The plots of the time and space complexity of the BFS for varying k. (10 Marks)
- Short paragraphs touching on the following:
    1. The optimality of the greedy-BFS and A* algorithms. (10 marks)
    2. The time complexity of the two algorithms using the manhattan distance as the heuristic function. (10 Marks)
    3. The effect of changing the heuristic function on the optimality of the algorithms. (10 Marks)
    4. The effect of changing the heuristic function on the time complexity of the algorithms. (10 marks)

This is an individual submission and each student is requires to do all of the work. You are encouraged to discuss the lab with each other and your tutors, but sharing of code or solutions is not permitted.