Programming assignment 1

Getting started

All your submissions should be implemented in Java. If you do not have a Java development environment, we recommend that you either use the Visual Studio Code Java extensions or Jetbrains IntelliJ. If you use IntelliJ, do not forget to register as a student, so you get access to the full versions. If you run Linux, macOS, or WSL, and want more control over your JDK, we recommend SDKMAN!

Note that you must solve all of the programming problems in Java. You are allowed to use Python (or any other suitable tool) to analyse the results, plot graphs, and so on. If you have discussed an alternative language with Morgan, it's fine to ignore the Java requirement and use, e.g., C++ instead.

Problems

Problem 1

Implement the first variant of UnionFind (quick find).

Problem 2

Implement one of the faster variants of UnionFind: quick union, weighted, or path compression.

Problem 3

Create a Java class (or set of classes) to measure the running time of a method. Feel free to use the Python `timeit` as inspiration. The class should be general enough that you can use it in future assignments. You are of course allowed to modify it in the future, but try to create an as general design as possible.

Problem 4

Run your two implementations of UnionFind for various array sizes and unions. Measure the time it takes and use these to empirically determine how they grow. Does it match your expectations? Why/why not?

Problem 5

Implement the $O(N^3)$ brute-force variant of 3sum.

Problem 6

Use one of the discussed techniques to design and implement a version of 3sum with an upper bound that is lower than $O(N^3)$. Note that the "smart iteration' ' version is $O(N^3)$, so it does not qualify here.

Problem 7

Run your two implementations of 3sum for various array sizes and measure the time it takes. Use the measures to empirically determine how they grow. Does it match your expectations? Why/why not?

Problem 8 (optional, for higher grade)

Percolation refers to the movement and filtering of fluids through porous materials. The most common example is probably Coffee percolation, but it can also describe the cracking of trees or spread of diseases. In this problem, you will study the probability that a system percolates.

You should model a percolation system as a $n \times n$ grid of sites. These sites can either be *blocked* or *open*. A site is *full* if it is open and can be connected to the top row via a chain of neighboring (left, right, above, below) open sites. The system percolates if there is a path of open sites from the top row to the bottom row, i.e., if there is a full site in the bottom row. You can think of this as a path through which water can flow, from the top to the bottom (if you draw the grid and mark the closed sites).

Your task is to use your model to determine with what probability a system percolates if sites are set to open with probability p. A system with $p = 0$ will never percolate, since no sites are open, and a system with $p = 1$ will percolate, since all sites are open. For sufficiently large n values, there is a threshold p^* such that when $p < p^*$, a random $n \times n$ grid almost never percolates and when $p > p^*$ it almost always percolates. Write a computer program that estimate p^*.

You can use Monte Carlo simulation to estimate p^*. Start by initializing all sites as blocked. Then randomly open sites until the system percolates. The fraction of sites that are opened is an estimate of p^*. For example, if you run on a 10×10 grid and the system percolates after you have opened 53 sites, then the percolation threshold is $53/100 = 0.53$. Repeat the experiment a reasonable number of times, and then compute the average threshold to get an estimate of p^*. You should also compute the standard deviation to get an idea of how sharp your threshold is.

As you may have figured out, you can use *UnionFind* to determine whether a system percolates or not. Open neighboring sites form the connected components. When a site becomes open, it is connected to its adjecent open sites: left, right, above, and below. This is equal to at most four calls to union. A system percolates if there is a connected component from the top to the bottom row. To determine if such a component exist, we use to *virtual sites*: a top and a bottom site. The top site is connected to all open sites in the top row and the bottom site is connected to all open sites in the bottom row. If there is a connection between the top and bottom sites, the system percolates. Section 8.7 in the book describes how UnionFind can be used to solve a maze. This is a different problem, but study the chapter for inspiration.

Submission guidelines

Submit your solutions as a single zip-file via Moodle no later than 17:00 on September 15, 2023 (cutoff 08:00 September 18). This is a group assignment that can be done in groups of one or two students. Your submission should contain well-structured and organized Java code for the problems with a README.txt (or .md) file that describes how to compile and run the Java programs. You should also include a report that describes your experiment setup and findings from problems 4 and 7. If you solve Problem 8, the report should also contain your experiment setup and findings from that problem. The report must be submitted in PDF format. It is sufficient if one person in the group submits the work to Moodle, but make sure that the submissions contains all your names.