Analysis of Algorithms

Programming Project 2 – Order Statistic Queries

One application of the order statistic query algorithm (Rand-Select) that we discussed in class is finding the locations that are nearby a given query point. So, for example, if we are working for a company that has many stores across the country, we may want to help a user find the stores that are closest to their current location (which in practice might be provided by their cell phone). This can be done by computing the distance from the user to all the stores, then performing an order statistics query to find the stores that are closest to the user. As discussed in class, the expected running time of the order statistic query algorithm is O(n), where n here would be the number of stores. This would practically be a very fast running time even if we have thousands of stores to search through.

In this project, we will perform this task from the perspective of Whataburger and Starbucks. In this project, you will read in the IDs, addresses, latitude, and longitude of each store location from a file and save it in an "array-like" data structure (a vector, ArrayList, etc. is fine as well). You will need to implement the Rand-Select() algorithm we covered in class (as well as the Rand-Partition() algorithm that is uses). There is a separate file that contains the query points, one per line. A query consists of three numbers: latitude, longitude, and the number of stores we want to find. So, a general outline of how to handle the queries is as follows:

For each query:

- Compute the distance of the query point to each of the different stores using the Haversine formula. See here for my Java implementation of this formula: https://youtu.be/Camxl1fTzaw?t=290. The project zip file provides implementations of the Haversine formula in C, Java, and Python.
- Use the order statistic query algorithm (Rand-Select()) to find the farthest store from the query that we care about. For example, if we want to report 30 stores, you should search for the 30^{th} closest store to the query point.
- Now that you know the 30^{th} closest store to the query, you can then find all the other stores that are at least this close to the query in the indices "to the left" of the store returned by the algorithm. Sort them in increasing distance from the query point and print them as I have shown here: https://youtu.be/Camxl1fTzaw?t=808

If you do not use an implementation of the randomized selection algorithm for this project, you will get a 0. It is not hard to get the correct answer with a basic algorithm. The entire point of this project is to use the algorithm from class that will obtain the answer very quickly.

How to code this project:

For the most part, you can code this project in any language you want provided that the language is supported on the Fox servers at UTSA. We want you to use a language that you are very comfortable with so that implementation issues do not prevent you from accomplishing the project. That said, we will be compiling and running your code on the Fox servers, so you need to pick a language that is already installed on those machines. See the PDFs on Blackboard in the Programming Projects folder for more information on how to connect to the Fox servers remotely (also covered in the Programming Project 1 overview lecture). You can submit this project with the file "WhataburgerData.csv" and "Queries.csv" hardcoded, so no need to do command line arguments for this project.

If you are not submitting an Eclipse project:

Since everyone is coding in their own preferred language, we are asking you to provide a bash script named *project2.sh* that will act similarly to a makefile. I covered how bash scripts work in class in the Project 1 overview lecture, and I recorded a short follow-up to this here: https://youtu.be/CaIFJWiyU_U

In short, your bash script should contain the command to compile your code, and then on a different line, it should contain the line to execute your code. So, the command to execute your code should look like this:

bash project2.sh

If submitting an Eclipse project:

You don't need the bash file, and you can upload an exported version of your Eclipse project.

Files provided in the project:

Since you are programming in different languages, we are providing no source files for your code (my implementation of Store.java is provided so you can copy/paste the Haversine formula code…you do not need to use it in your project). We have provided a blank project2.sh as well as data files in CSV format. The csv file extension stands for "comma separated values". These files can be opened both by spreadsheets as well as by text editors. The idea is that each line of the file corresponds to a row in the spreadsheet, and the columns are separated by commas. This is a common format when one "column" of data could contain spaces, such as the addresses in this project. So you should not read the data in based off of white space. You should break it up according to commas. This is very easily done in Java using functions like split() (Python has a similar function). It is a little more complicated in C, but here is a nice website

showing how it can be done: https://stdin.top/posts/csv-in-c/. We will be testing your code with a different query file, but it will have the same name so it can be hardcoded in your program.

Grading

We will grade according to the rubric provided on Blackboard. The majority of the points come from the following: did the student give a correct implementation of the selection algorithm that runs in expected O(n) time that returns the correct answers for all possible inputs? Proper documentation may help the grader understand your code and earn you partial credit in the event you have some mistakes in the code.

Violations of the UTSA Student Code of Conduct will be penalized harshly. In particular, be very careful about sending code to a student who asks how you accomplished a particular task. I've heard this story several times recently: "They said they just wanted to see how to perform part X of the project. I didn't think they would submit my exact code." If this happens, you will both be penalized for cheating. To protect yourself and to more properly help your fellow student, send pseudocode, and not actual compliable code.

Also we know about the online sites where people upload projects and have a third party complete the project for you. This is a particularly egregious form of cheating (it's in the best interest of your career to not tolerate this). If you use a solution from one of these sites or submit a minor modification (minor is at the discretion of the instructor) of a solution from one of these sites, you will receive a 0 and will be reported to the university for a violation of the UTSA Student Code of Conduct.

Submitting

Zip up your project folder (or exported Eclipse project) and submit on the dropbox on Blackboard by the due date.