Project #6: Closest Starbucks

Points: 100

Key objectives of this project are:

- Working with different Data Structures
- Designing a new solution using effective Data Structures
- Considering Time Complexities in designing solution
- Practice using GIT and committing after every major change
- Review and adhere to CSE department Style guide

Submission Instructions

The GitLab link of this project must be turned-in electronically via Canvas.

Ensure your program compiles successfully, without any warnings or errors.

Ensure your class and methods are commented properly.

Ensure you have tested operations of your program as indicated.

Once you have tested everything, do the last push and submit the **clone with HTTPS link of your project6 folder** on Canvas.

The program submitted for this homework **must compile and run in order to qualify for earning any score at all**. Programs that do not meet basic requirements or just skeleton code will be **assigned zero** score, even if 99.99% of the code is already finished!

- **NOTE:** Violating of CSE programming style guidelines is an error! Your program should not have any style errors.
- **Bonus points and Delayed submission:** The gamified grading scale will be used. Refer to the syllabus to learn more about gamified grading scale.
- **Formatting & Style Guide:** Full points are reserved for programs that have:
  - shorts methods, not longer than **35 lines**
  - the code is formatted nicely. You can use the Red Hot tool on VS Code to format your code. Here is a video that show you how you can use that tool:
    - **https://youtu.be/xAGbpoTYxXM?si=ZKPT_nYE7gOSIqef**
  - your class and all methods are commented properly

Set up your Project and Initiate a GIT repository

Open the VScode and add a project inside the **cse274** folder (which is already connected to your local repository) and call it **project6** (no space, all lowercase). Download the following **four** files from the project#6 folder on canvas:

1. Locations.java (DO NOT CHANGE)
2. Sturbucks.java (DO NOT CHANGE)
3. StudentSturbucks.java
4. Controller.java (DO NOT CHANGE)

and, add them to your project under the src folder. Here are some tips as you are initiating the project:

- If you haven't before, watch this video to learn how to create a default project in VSCode: **https://youtu.be/CK3C4KXVXdk**
- Ensure that your IDE is not automatically adding any packages. There should be no extra folder under the src folder, meaning all your files must be placed directly inside the src folder, with no extra subfolders.
- If you drag and drop your files under the src folder and you are getting errors regarding it can't see/find some of your files, you can do the following:
  - Remove all the files you dropped under the **src** folder.
  - Right-click on the **src** folder, select **"New" > "Java File" > "Class…"**.
  - Create a new class and call this one **Locations**.
  - Then, copy the content of the Locations.java that you downloaded from canvas, and paste it inside the new Locations class that you just added under the src folder.
  - The goal is to create the files manually first, then copy and paste **their content**, instead of dragging and dropping them under the src folder. So, repeat the same process for the rest of the files: first, add files manually. Then, copy and add the **contents**.
- You also need to work with the provided CSV file. This csv file provides all the data that you need to work with. Download the **Starbucks.csv** and drag and drop it directly into the project6 folder **(not inside the src folder nor any other subfolders). The csv file must be placed directly in the project6 folder)**
- Now that all files are added to your project, this should be a great time to do your first: *git add, git commit -m "some comments",* and *git push* to save everything on your local and remote repositories.

Find the nearest Starbucks location:

Imagine that you need to find the closest Starbucks store to your current location. Google Maps makes this very easy, simply searching for “Starbucks” along with your current address will quickly show you the nearest options: (example).

This kind of problem is called the **nearest neighbor problem**. In the nearest neighbor problem, we start with a large but seldom changing list of locations (that is, we get the latitude and longitude of all Starbucks locations), and **build a data structure**. This data structure’s goal is to make it very efficient to **find the nearest Starbucks** to a given pair of (longitude, latitude) coordinates.

Finding the nearest neighbor is a very common task not just for coffee shops, but across many fields. For example, delivery services need to find the closest driver to a customer, navigation systems search for nearby gas stations, and recommendation systems suggest items based on the “closest” match to a user’s preferences. Although the problem can sometimes seem simple, solving it quickly and accurately becomes more challenging as the number of locations grows larger.

Solving nearest neighbor problems efficiently is an important part of making real-world applications fast and responsive. In this project, you will focus on building a system that can search through a large number of locations and quickly find the one that is nearest to a specific point.

Implement StudentStarbucks.java

You will need to implement these methods:

- void build(Locations[] allLocations)

  This method is responsible for building your internal data structure. It should add all given locations while making sure that no duplicates are added. Two locations are considered duplicates if:

  both |x1 - x2| <= 0.00001 and |y1 - y2| <= 0.00001. If a duplicate is detected, you should skip adding the new location. Only unique locations should be stored.

- Locations getNearest(double lng, double lat)

  This method finds and returns a **deep copy** of the Starbucks location nearest to the given coordinates. You must search your data structure for the location that is closest to the provided (longitude, latitude). You must return a new Locations object with the same data as the nearest location (not the original object).

Project Rules & Considerations:

- You are encouraged to add as many additional **classes** and **methods** as needed to organize your solution. Try to keep each method short and focused on small tasks. No single method should exceed **35 lines** in total. This includes all comments, brackets, and empty lines. Cramping code together or removing necessary spacing just to stay under 35 lines will be considered poor style and will result in a deduction of points.

- The running time of your **build** method is not critical, as long as it finishes in a reasonable amount of time. This timing will be measured by Controler.java file. If your build method takes longer than **10 seconds** to complete, you must clearly warn us for grading purposes.

- One easy way to implement your **getNearest** method would be to copy the array of locations and reuse my bruteGetNearest method from Controller.java. In this method I calculate the distance from my location to all Starbuckses and find the location with minimum distance. In fact, it is recommended to start this way to ensure you have a working solution that can be tested for correctness and timing. However, you must understand that the size of the Starbucks dataset (starbucks.csv) has been reduced by more than 50%, which means even with a brute force solution, your code will run reasonably. **You are allowed to submit it, but you will lose significant points.**

- Your real challenge is to create a data structure that is **significantly faster** than brute force. Alternatively, you may find a way to optimize brute force enough to show clear speed improvement compared to the provided code.

- Other possible strategies to improve performance include:

    - Using a **grid of buckets**, with each bucket containing a small number of Starbucks locations. (Choosing a good bucket size and handling edge cases like empty buckets or boundary searches will be critical.)
    - Using a **k-d tree** data structure which is using an efficient spatial partitioning method.
    - Combining multiple approaches creatively. You can actually combine the ones you have learned in this course and achieve proper timing.

- You are allowed to use any data structure that you have learned so far. You are expected to make thoughtful design decisions that balance accuracy, speed, and clarity of your code.

Test your code:

- Run the **Controller.java** file (has the main method in it) to check the accuracy and speed of your solution.

Submission:

You will be submitting the **'Clone with HTTPS'** link of the **'project6'** folder on canvas. The only file that we grade and Must be in your GitLab in order to get full points is:

- **StudentStarbucks.java**

My Results

I have three solutions so far. Except the brute force one, <u>I will not post my other solutions for copyright reasons and future use:</u>

- Brute force: 0.018 ms per search
- kd-tree: 0.0028 ms per search
- grid-based: 0.000024 ms per search, **36.6667 <u>seconds</u> to build**

Rubric:

Description	**Points**
Code is pushed on Git and the Correct GitLab link is submitted	**5**
Speed Faster than 20x — 45 pts 5x - 20x faster than brute force solution — 39 pts 2x - 5x faster than brute force solution — 23 pts Less than 2x faster than brute force solution — 0 pts	**45**
Accuracy **(if you get 0 for Speed, you will get only 13 points for accuracy)** Error = 0% — 45 pts Error: 0% > Error >= 20% — 33 pts Error: 20% > Error >= 100% — 25 pts Error: 100% > Error >= 500% — 13 pts Error > 500% — 0 pts	**45**

Your code is clean, methods are not more than 35 lines including: comments, brackets, and empty lines.	**4**
You have your name at the top of StudentStarbucks.java	**1**
Bonus Points and Penalties	**0**