CSCI 3901 Final Project

Due date: 11:59pm Thursday, December 15, 2022 in git.cs.dal.ca. The repository is at https://git.cs.dal.ca/courses/2022-fall/csci3901/project/xxx.git where "xxx" should be replaced by your timberlea login id. The repository is already created for you, so you should just need to clone it.

Goal

The course project is your opportunity to demonstrate all of the concepts from the course in one body of work.

Project Structure

The final project will have you apply your problem solving and development skills. The solution will require you to bring together

- Abstract data types and data structures
- Java implementation
- Basic algorithms
- Software development techniques including version control, testing, debugging, and defensive programming
- Good software program design
- Database design and use

While we provide a recommended problem for the course, you have the option of proposing a different problem for the project. A project proposal must include a non-trivial example of all the concepts mentioned above.

The project milestones do not change if you choose to propose your own problem.

This project is _not_ expected to include a user interface component or software that directly accesses a device's hardware.

Recommended Problem

Hurricane Fiona descended on Atlantic Canada in the fall of 2022. It created a lot of damage to the province and left the Nova Scotia electrical grid with ongoing outages.

The goal of this problem is to provide reporting and planning activities for the power grid to help optimize the rate with which the electrical grid is fixed so that the most people have their electrical power returned as soon as possible.

We simplify the problem by only looking at damage and issues with primary power distribution hubs in the province. We will not worry about trees taking down power lines to one individual house.

The following information informs the project:

- Some notion of the population and population density of the province. We use postal code information for this information.
- The set of power distribution hubs in the province and which parts of the province are served by each hub. Each hub will serve one or more postal code areas.
- Estimates of the repair time for damage at power distribution hubs.
- Repair reports to the power distribution hubs.

The project

Create a class called "PowerService" that lets us manage information about the power service and reports on that power service. The class can use internal data structures, databases, or files to store information.

The information managed by the class should survive between execution of programs that use the PowerService class. In a deployment environment (beyond the scope of this project), the central power utility would record postal code information and distribution hub information as these are more static pieces of information. Employees in the field would have an app on their phones that would report damage or repairs to hubs to the central power utility to immediately be included in any reporting or planning by the PowerService class.

Here is the minimum information that you will need for the project.

Postal codes describe the province to the class. Each postal code has a unique string identifier. In Nova Scotia, that identifier is a 6 character string that alternates letters and digits and begins with a letter. The postal code system is hierarchical, meaning that if we truncate the postal code to fewer characters then we are capturing the idea of a grouping of smaller postal codes into one larger area. For any postal code that we enter into the system, we will want to have

- The postal code identifier
- The number of people living in the postal code (from the last government census)
- The area covered by the postal code.

Each power distribution hub is located in a community in the province. For each hub, we will know:

- An alphanumeric identifier for the hub
- The coordinate location of the hub in the world in UTM coordinates. UTM coordinates look like regular (x, y) coordinates where each unit is 1 metre and we imaging that the local world is a flat plane, so we don't worry about the curvature of the earth, which is fine for the province.
- The set of postal codes that are served by the hub. One hub can serve multiple postal codes and one postal code can be served by multiple hubs.
- The estimate of the number of hours needed to repair the current hub, if there is any damage.

As a hub is repaired, we want a log of when the repair was done, over how many hours and by which trained employee in the company. In theory, repairs can be partial repairs, and your system should be ready to handle that.

Ultimately, we want you to answer the following questions for the power company:

- How many people are without service?
- What is the most damaged region?
- Which hubs should we fix first to have the greatest impact in getting people back into service?
- Under perfect conditions, at what rate do people get service back?
- Provide a plan for one vehicle to service hubs.
- Which areas are most underserviced by the company for which we need to improve?

Your task is to create classes that will gather the information for this system and resolve these queries.

Support classes

- Point
    - A container for a 2-d (x, y) coordinate point. Include getters and setters for the coordinates.
- HubImpact
    - A container for a hub identifier and for the floating point impact value of the damage at the hub. Include getters (getHubID and getImpact) and setters.
- DamagedPostalCodes
    - A container for a postal code and the amount of repairs needed for the hubs that service the postal code. Include getters (getPostalCode and getRepairEstimate) and setters.

Methods for PowerService

boolean addPostalCode (String postalCode, int population, int area)

Add a postal code to our system. The population comes from the government census, which happens every 4 years, and the area is in square metres. Return true if the given postal code is a new addition to our set of postal codes and is now recorded in the system.

boolean addDistributionHub (String hubIdentifier, Point location, Set<String> servicedAreas)

Add a new distribution hub to our system. The UTM coordinates for the hub are given by "location" and the postal codes serviced by the hub are given by servicedAreas. Return true if the hub is a new addition to our system and is now recorded in the system.

Void hubDamage (String hubIdentifier, float repairEstimate)

Record that the given hub is damaged and that the repairs are estimated to take the given value in hours.

Void hubRepair(String hubIdentifier, String employeeId, float repairTime, boolean inService)

Report that the given employee has done repairTime hours of repairs to the given hub. If the hub is now fully repaired and ready to go then inService is true. If the hub isn't in service then the employee can update the estimated damage with a separate call to hubDamage().

Company policy is that the employee would report this repair via their cell phone app before they leave the power distribution hub.

int peopleOutOfService ()

Report the number of individuals who are out of service. Given a power distribution hub that has damage, all individuals not served by the hub are listed as out of service until the hub is fully repaired. A hub serves all of the people in the postal codes attached to the hub. If more than one hub serves a postal code then the hub servers a fraction of the people in the postal code; the fraction is 1/(number of hubs serving the postal code).

List<DamagedPostalCodes> mostDamagedPostalCodes (int limit)

Report the "limit" postal codes that require the most repairs before everyone in the postal code has power restored. Return the list of postal codes in descending order of repair time needed.

List<HubImpact> fixOrder (int limit)

Report the "limit" most significant hubs to fix. Order the hubs in descending order of importance.

The significance of a hub to fix is the number of people who regain service per hour of repair.

List<Integer> rateOfServiceRestoration (float increment)

Produce an estimate of the rate with which people are restored to power, under ideal conditions where there is no transit time to move between hubs and where we service the hubs in decreasing order of impact (as set in the fixOrder method).

The list that is returned is a list of hours of repair effort. List entry x reports the number of hours of repairs needed across the whole system before (x * increment) percent of province's population has power. For example, if increment is 0.05 (so 5%), then entry 0 is the time to get 0% of the population with power (and presumably takes 0 hours), entry 1 is the repair time to have 5% of the population with power, entry 2 is the repair time to get 10% o the population with power, and so on.

List<HubImpact> repairPlan (String startHub, int maxDistance, float maxTime)

Return a list of hubs (and their impacts) that an employee should follow to do repairs to hubs.

The employee will start at the given “startHub” location. That location should be the first location in the returned list. The last location in the returned list should be the hub that is within “maxDistance” metres of the start hub and that has the largest repair impact among all hubs within that “maxDistance” range. The idea is that the employee will start by fixing startHub and will then want to travel to that next impactful hub to repair it. For the rest of this description we will call this next impactful hub the “endHub”.

As the employee drives from startHub to the endHub, they will pass other hubs that they could fix along the way. The list of the hubs along the way that they should fix are given, in order of fixing, by the returned list.

The path given by the returned value must satisfy the following conditions:

- The path will have horizontal and vertical segments. No diagonal segments.
- The path from startHub to endHub must remain within the rectangle defined with the line from startHub to endHub as the diagonal.
- The path from startHub to endHub can only cross the diagonal of the rectangle once
- The repair time needed at the intermediate hubs cannot exceed maxTime
- The path from startHub to endHub must be monotonic in either x or y direction. This means that either the x-coordinates of the hubs visited in the path must always increase as we go from start to end (assuming that the start x-coordinate is less than the ending x-coordinate) or the y-coordinate must always increase (assuming the start y-coordinate is less than the ending y-coordinate)
- The set of intermediate hubs serviced must, as a collection, have the most impact at being repaired over all the possible paths.

For example, consider the hub configuration in Figure 1. We start at position A and the max distance outlines the outer circle. We consider the impact of all hubs within that circle. Here, we determine that hub B has the greatest impact, so we’ll do a path from A to B. Only hubs E, J, and K are considered as stopping points along that path since they’re within the rectangle defined by A and B.

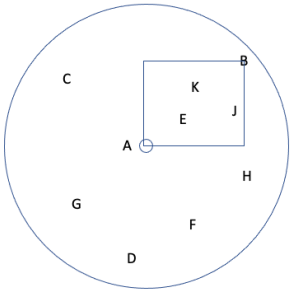

Figure 1 Finding the ending hub within a given distance.

Figure 2 then shows what paths can look like. The x-monotone path only changes the x-coordinate as ever increasing so that path fits the monotone property. However, it fails on the property of crossing the diagonal from A to B twice, with the assumption that K is above the diagonal and both E and J are below the diagonal. Similarly, the y-monotone path only has the Y value increase, so it fits the monotone property. It actually also fits an x-monotone property. The third path, not monotone, has the x coordinate on the path both increase (A -> J) and decrease (J -> K) so it isn’t x-monotone. The y coordinate also increases (A -> J) and decreases (K -> E) so it isn’t y-monotone either.

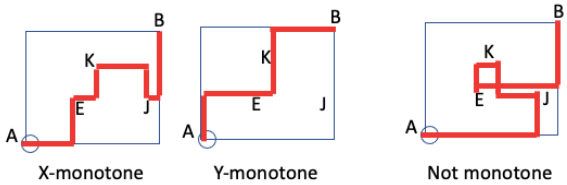

Figure 2 Monotone path examples.

List<String> underservedPostalByPopulation (int limit)
List<String> underservedPostalByArea (int limit)

Return a list of the "limit" most underserved postal codes. Sort the list by descending order of service needed, so the postal code that is in most need of more service is the first item in the list.

Given postal codes A and B, postal code A is less served than B by population if the average number of hubs per person in A is lower than in B. Postal code A is less served than B by area if A has fewer hubs per square metre in the postal code than B.

Assumptions

- You will not be provided with postal code identifiers that overlap. For example, you will not be provided with postal codes B3W1H5 and B3W.
- The postal codes are stable. Once created, we won't delete a postal code.

Notes

- Include the type of exception handling that you feel is most appropriate for your circumstances. Document that exception handling mode and be consistent with it.

Milestones

Only the final submission is graded. The intermediate milestones appear as points to get feedback on the project:

- November 16: High-level breakdown analysis of the problem
- November 23: Blackbox tests and plan+timeline of feature development
- December 7: External documentation of data structures, code design, key algorithms. Implementation report in relation to earlier plan; git submission showing some progress on implementation
- December 15: Project due

All milestone material is due in the course git repository.

Marking scheme

- Meeting intermediate milestones (10%)
- Documentation (10%)
- Overall design and coding style (15%)
    - Design, including the quality and consistency of the decisions being made in the design 10%
    - Coding style 5%
- Working implementation, including efficient processing (50%)
    - Data entry Methods 5%
    - Reporting methods 15%
    - Planning methods (considerable weight on repairPlan) 20%
    - Robustness and error reporting 5%

        - Efficiency 5%
    - Test plan (15%)