FACULTY OF COMPUTER SCIENCE

PROJECT - DOCUMENTATION

In
The Class of

CSCI 3901: SOFTWARE DEVELOPMENT CONCEPTS

by

HARSH NARESHBHAI KATHIRIA [B00931861]

Submitted to

Prof. Michael McAllister
Department of Computer Science
Dalhousie University.

Date: 15th December 2022

Overview

This project mainly helps the power grid for the activities reporting and planning as well as optimising the rate with electrical grid to be fixed, by observing the issues and damages, in such a way that maximum people can have their power restored at the earliest. There are different power hubs with multiple postal codes in it. Based on that a repair plan is created which display the route of damage, so that employee can repair in a significant way.

Files and External Data

There are total of 11 files.

1) DamagedPostalCodes.java
2) DistributionHub.java
3) HubGraph.java
4) HubImpact.java
5) Main.java
6) Point.java
7) PostalCode.java
8) PostalCodeServe.java
9) PowerService.java
10) hubs.txt
11) postal_codes.txt

Data Structures

The data structures used in the program are as follows:

1) Set, HashSet
2) List, ArrayList
3) Map, HashMap
4) Graph

1) DamagedPostalCodes

This class is used to record the details of postal code of the damaged area.

Member Variables

private String postalCode; - indicates the postal code id.

private int numberOfRepairs; - number of repairs to be done.

Member Functions

public setPostalCode(String postalCode) - sets the postal code id.

public String getPostalCode() - returns the postal code id.

public void setNumberOfRepairs(int numberOfRepairs) - sets number of repairs.

public int getNumberOfRepairs() - returns number of repairs.

public int compareTo(DamagedPostalCodes other) - compares current damaged postal code with other, based on number of repairs.

2) class DistributionHub

This class is used to represent the distribution hub with id, location, area covered and set of postal codes served and final hours to repair.

Member Variables

private String id; - represents the distribution hub id.

private Point location; - location of the hub, which is a coordinate.

private double areaCovered; - area covered in metre squares.

private Set<PostalCode> postalCodesServed; - set of postal codes covered served by this hub.

private double hoursToRepair; - hours required to repair this hub.

Member Functions

// function to return the total area covered
private void calculateAreaCovered()

//function to return the id of the distribution hub
public String getId()

//function to set the id of the distribution hub
public void setId(String id)

//function to return the location of the distribution hub
public Point getLocation()

//function to set the location of the distribution hub

```
public void setLocation(Point location)
//function to return the area covered
public double getAreaCovered()

//function to set the aread covered
public void setAreaCovered(double areaCovered)

//function to set the postal codes served
public void setPostalCodesServed(Set<PostalCode> postalCodesServed)

//function to return the postal codes served
public Set<PostalCode> getPostalCodesServed()

//Function to add the postal code to served postal codes
public void addToServed(PostalCode code)

//Function to return the hours to repair
public double getHoursToRepair()

//Function to set the hours to repair
public void setHoursToRepair(double hoursToRepair)

//Function to compute the hashcode of the class
public int hashCode()

//Function to compare given object with current object
public boolean equals(Object obj)

//Function to return the string representation of class
public String toString()
```

3) class HubGraph

This class is used to create a graph for all the hubs in a rectangle of hubs, it contains start hub, end hub and an adjacency matrix to represent the graph. Also, it provides function to determine the optimal path in the graph, which follows the rules of monotonicity and also optimal path.

Member Variables

```
//list to hold the hubs in a particular rectangle area in the 2d plane
private List<DistributionHub> hubsInRect;

//map variable to hold the graph mappings of the hubs in 2d plane,
// hub id will be the key for the map and the adjacent hubs
// will be stored as list of strings
```

```
private Map<String, List<String>> adjacencyList;
//Variable to hold the starting hub in graph
private DistributionHub startHub;

//Variable to hold the ending hub in graph
private DistributionHub endHub;

//variable to hold the diagonal slop value
private double diagonalSlope;

//variable to hold the set of paths which are valid
// in the graph, and each set will have list of hubs
private Set<List<DistributionHub>> allPossiblePaths

//Variable to hold hubs in a map, where hub id will be the key
// hub object will be the value
private Map<String, DistributionHub> hubMap
```

Member Functions

```
//Function to create a new graph using the start hub and end hub and store the
// graph is adjacency list
private void createGraph()

//Function to compute the slope of the given two hubs
private double getSlope(DistributionHub hub1, DistributionHub hub2)

Function to check whether the edge can be added between two hubs
private boolean canAddEdge(DistributionHub hub1, DistributionHub hub2)

Function to add edge between given two vertices, which is hub
public void addEdge(String v1, String v2)

Function to return the set of monotonic hubs, that is the hubs which follow the
condition of monotonicity, either monotonic or y monotonic, but not both
public Set<List<DistributionHub>> getMonotonicHubs()

Function to return the set of optimal paths, that is the paths which are having
Maximum impact values
private Set<List<DistributionHub>> getOptimalPaths()

//Function to generate the possible paths between v1 and v2
private void generatePaths(String v1, String v2,
                    Map<String,Boolean> visitedMap,
                    List<String> finalPathList)
```

4) class HubImpact

This class contains mapping of hub id to its impact value.

Member Variables

```
// refers to hub id
private String hubId;
//hub impact value
private double impactValue;
```

Member Functions

```
//Setters and getters for variables
public void setHubId(String hubId)

public String getHubId()

public void setImpactValue(double impactValue)

public double getImpactValue()
```

5) class HubImpactPopulation

This class is used to map the hub impact to its total population, for example if a hub has got 20 postal areas, then the total population is sum of population in in each postal code.

Member Variables

```
HubImpact impact;
int population;
```

Member Functions

```
HubImpact getImpact()
void setImpact(HubImpact impact)
int getPopulation()
void setPopulation(int population)
int compareTo(HubImpactPopulation other)
```

6) class Point

This class is used to represent a point in the 2d plane, its used by distribution hub class to set its location.

Member Variables

private int x
private int y

Member Functions

//Setters and getters for variables

public void setX(int x)

public int getX()

public void setY(int y)

public int getY()

public String toString()

7) class PostalCode

This class is used to represent the postalcode in a hub, it contains a unique id, number of people in the postal area, and total area covered

Member Variables

private String id;
private int numPeople;
private double areaCovered;

Member Functions

//Setters and getters for variables
public String getId()

public void setId(String id)

public int getNumPeople()

public void setNumPeople(int numPeople)

public double getAreaCovered()

public void setAreaCovered(double areaCovered)

8) class PostalCodeServe implements Comparable<PostalCodeServe>

This class is used to represent the postal code with the fraction of people served among different hubs. For example, if a particular postalcode is served by different hubs, then the fraction is calculated as 1 / hub count and its stored here.

Member Variables

```
private String id;
private double servedFraction;
```

Member Functions

```
//Setters and getters for variables

public String getId()

public void setId(String id)

public double getServedFraction()

public void setServedFraction(double servedFraction)

public int compareTo(PostalCodeServe o)
```

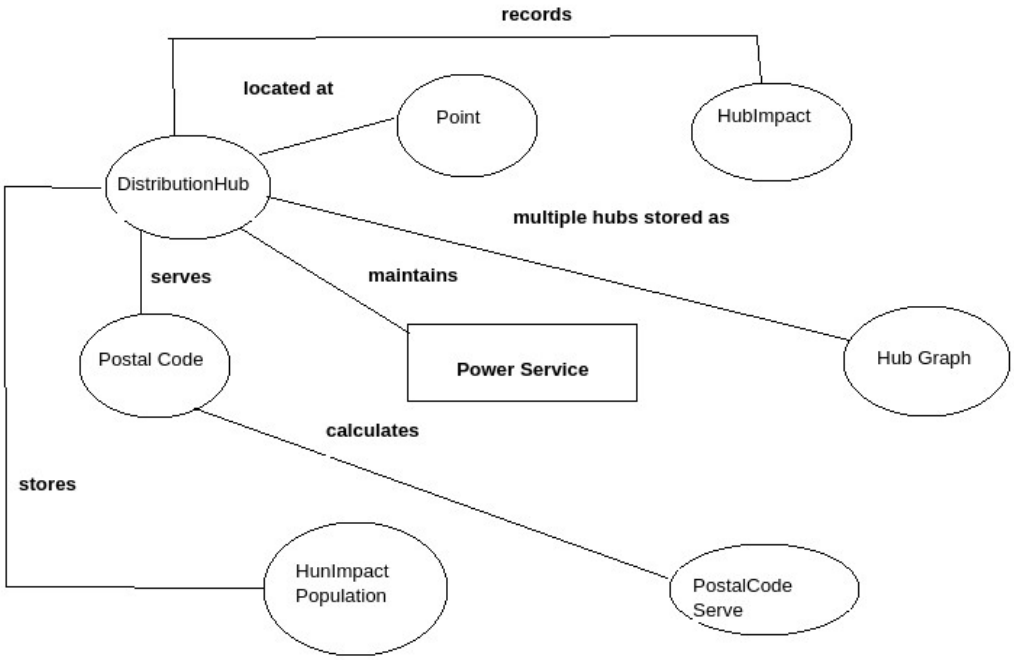
Assumptions

Some of the assumptions are as follows:

1) No postal code identifier will overlap. For example, postal codes B3J4R2 and B3J will not be provided.
2) All the postal codes are stable. They will not be deleted once they are created.

Code Design

The code design is illustrated in following figure:



Key Algorithm

1) Declare a list to hold the hub impact objects.

2) Determine the hubs in range for given maximum distance.

a) Declare a list to hold the result and call it as hubsInRange.

b) For each hub in the system do below.

if the distance between current hub and the start hub is less than or equal to given maximum distance, then add this hub to result.

3) Determine the maximum impact hub and set it as end hub.

4) Determine the which are located within starthub and end hub and call it as hubsInRect.
a) if the coordinates lie within the range of star hub and end hub, then consider the hub is in rectangle, otherwise ignore it.

5) Generate a graph using the hubs in range using below algorithm.

a) declare adjacency list to hold the vertex in graph.

b) For each hub in hubsInRect.

i) Add edge between start hub to current hub and current hub to end hub.

ii) Again, add edges to all other hubs in the system.

6)Determine the monotonic hubs in graph using below steps.

a) Iterate through each hub in graph except start hub and end hub.

b) If the slope between starthub and current hub is less than or equal to slope between starthub and end hub, then consider current hub does not violate the diagonal rule.

c) Also, if the current hub follows the rule monotonicity, then add it to result.

7) Determine the optimal path from monotonic hubs above, that is the path between start hub and end hub will have the maximum impact value.

8) Print the optimal path to console.

Test Cases

Test Case 1: Add a new postal code with valid id to the system and check the return value is true.

Sample input:

please enter id:
A1S2D3
please enter Population:
456
please enter Area covered:
2000

Expected output:

Successfully added.

Test Case 2: Updating an existing postal code with valid population and area covered and check return value is true.

Sample input:
please enter id:
A1S2D3
please enter Population:
780
please enter Area covered:

2300

Expected output:
Updated existing entry successfully.

Test Case 3: Add null value or empty postal code to the system and check return value is false.

Test case 4: Adding a new hub and its corresponding postal codes to the system with valid id, location, served areas and check the return value is true.

Sample input:

please enter id:
QWER1234
please enter coordinates (x,y):
12,3
Enter serviced postal area code:
A1S2D3

Expected output:
Added successfully.

Test Case 5: Setting the hub status to repair complete and printing the employee details.

Sample input:

Please enter hub id:
PWYG6926
Please enter employee id:
john
Please enter repair time:
2
Is repair over? (y/n):
y

Expected output:

Employee john has done repair for 2.0 in hub id PWYG6926.

Test Case 6: Add existing hub id, with valid location and served areas and check return the return value is true.

Test Case 7: Add null or empty hub id, with valid location and served areas and check return the return value is false.

Test Case 8: Test for setting hub status, update the repair time for a valid hub id, employee id and in service as false, check whether the mentioned hub id is added to damaged hubs.

Test Case 9: Test for setting hub status, update the repair time for a valid hub id, employee id and in service as true, check whether the message is displayed with employee finished the repair in estimated time.

Test Case 10: Test for setting hub status, update the repair time for an invalid hub id, check whether an error message is displayed.

Test Case 11: Displaying the most significant hubs are in great need to be repaired based on the impact value.

Sample Input:

Please enter the limit to show:
5

Expected output:
Most significant hubs to be fixed

FMZH3476
FFYL6995
OLEA3757
WHXJ0461
JPKI8034

Test Case 12: Test for most significant hubs, for a valid limit, display the most significant hubs and validate it.

Test Case 13: Test for most significant hubs, for a negative or zero limit, check error message is displayed and significant hubs not displayed.

Test Case 14: For a valid hub id, check whether the people out of service is displayed correctly.

Test Case 15: For an invalid hub id, check error message is displayed and people out of service is not displayed.

Test Case 16: For a valid hub id, check whether the most damaged postal codes are displayed.

Test Case 17: For an invalid hub id, check whether the error message is displayed and not most damaged postal codes.

Test Case 18: Displaying the people out of service for a particular hub, if the hub is damaged, then all the postal codes under the hub are considered to be damaged. Hence the people out of service will be the sum of people in all postal codes.

Sample Input:

Please enter hub id:
OLEA3757

Expected output:

Number of people out of service are 3007.

Test Case 19: Displaying the most damaged postal codes under all hubs, a postal code is considered to be most damaged if its number of hours to repair is higher.

Sample Input:

Please enter the limit:
5

Expected Output:

```
-----------------------------------
PostalCode | HoursToRepair
-----------------------------------
   Q7C2O6    1
```

Test Case 20: Test for repair plan, for a valid hub id, maximum distance to be covered and maximum repair time, check whether starting hub, intermediate hubs to be traversed and ending hubs are displayed.

Test Case 21: Test for repair plan, for an invalid hub id or maximum distance to be covered or maximum repair time, check whether error message is displayed and not the repair plan.

Test Case 22: Scheduling a repair plan for damaged hubs. The schedule should be optimal, only monotonic hubs are considered to be in schedule plan, as well as the hub route should have maximum impact value.

Sample Input:

please enter the start hub id:
PWYG6926
please enter maximum distance to be covered:
200
please enter maximum time:
200

Expected output:

Repair plan

PWYG6926 > OLEA3757

Test case 23: Test for checking service rates, for a valid increment percentage, check whether the service restoration per hour is displayed.

Test case 24: Test for underserved postal codes by population. For a valid limit, check whether the underserved postal codes are displayed.

Test case 25: Test for underserved postal codes by population. For an invalid limit, check whether the error message is displayed and not the postal codes.

Test case 26: Test for underserved postal codes by area. For a valid limit, check whether the underserved postal codes are displayed.

Test case 27: Test for underserved postal codes by area. For an invalid limit, check whether the error message is displayed and not the postal codes.

Test case 28: Displaying the underserved postal codes by population.

Sample Input:

please enter the limit:
5

Expected Output:

Under served by Population
---
1. L4H6R2
2. Z3A9B0
3. E8A6G7
4. U3B1H3
5. T6T5E0

Test case 29: Displaying the underserved postal codes by area.

Sample Input:

please enter the limit:
3

Expected output:

```
--------------------------
Under served by Area
--------------------------
1. E6Y8Q2
2. T4G5P8
3. W9M8D8
-----------------
```