Data Structures

There data structures used in the program are as follows:

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

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()

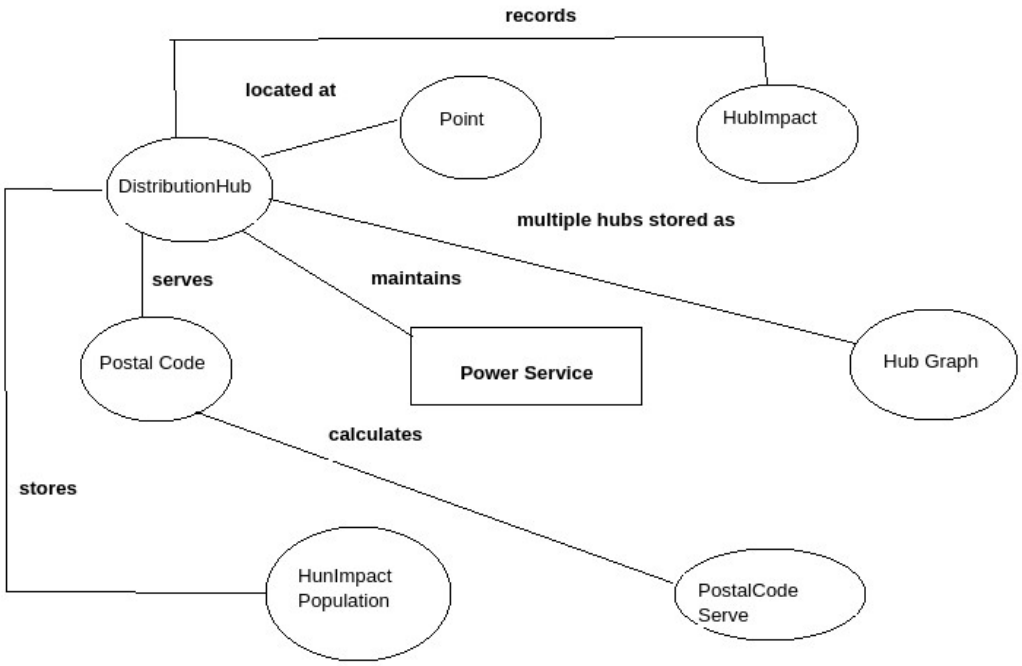
public void setId(String id)

public double getServedFraction()

public void setServedFraction(double servedFraction)

public int compareTo(PostalCodeServe o)

Code Design



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.