Project 2: Fire station location program

Contents

New concepts: Arrays, databases, tabular output

1 Program description

An important question in planning cities is fire station location. Each fire station can service houses within a given radius, and every house must be serviced by at least one fire station. Your task is to write a program to help city planners determine where best to locate a fire station, given up to 20 house locations.

1.1 The facility location problem

The question of where to locate a fire station is a variation of the classical *Facility Location Problem* (FLP). In a basic FLP, there is a set of potential facility sites where a facility can be opened, and a set of demand points that must be serviced. The goal is to pick a subset of facilities to open, to minimize the sum of distances from each demand point to its nearest facility, possibly plus the sum of opening costs of the facilities (or maybe the costs are constrained to be within a certain budget). There are numerous variations of FLP, some of which take into account capacity constraints at each facility, others require certain coverage levels, etc.

FLP is actually a famously difficult problem to solve, so to make things easy, we will not even attempt to solve the problem. Instead, our program will allow the city planner to input possible fire station locations and then calculate a score to indicate how good that fire station location is. We will make the simplifying assumption that only one fire station needs to be placed, and costs are not a concern.

We can say that all house and fire station locations lie on a grid. The score of a fire station will be the sum of the Euclidean distances of all the un-serviced homes from the fire station. If (x^*, y^*) is the location of the fire station and (x_i, y_i) is the location of house i, the Euclidean distance between the fire station and house i is

$$d_i(x^*, y^*) = \sqrt{(x_i - x^*)^2 + (y_i - y^*)^2}$$

Define function $\varphi_i(x^*, y^*)$ to let us know whether house i is outside of the serviceable radius r of the fire station:

$$\varphi_i(x^*, y^*) = \begin{cases} d_i(x^*, y^*) & \text{if } d_i(x^*, y^*) > r \\ 0 & \text{otherwise} \end{cases}$$

Now, we just sum up $\varphi_i(x^*, y^*)$ for all n houses to get the score of a fire station at location (x^*, y^*):

$$f(x^*, y^*) = \sum_{i=1}^{n} \varphi_i(x^*, y^*)$$

Note that lower scores indicate better locations, and a location with a score of 0 is optimal (meaning that all houses are serviced).

1.2 A database of house locations

Your program will keep a database of house locations. A database is simply an organized collection of data, and does not imply a *database management system*, e.g., MySQL, PostgreSQL, etc. In fact, a database can be as simple as an array! In this project, you will keep an array of all house coordinates. The array will have to be 2D in order to simultaneously store the x- and y-coordinates of each house.

The user will be able to enter houses into the database (which just adds to the existing set of houses) and clear the database. A more full-featured program would allow the user to edit and delete houses, but we will omit those features for simplicity.

2 Menu options

- `0 - Quit`
  Quit the program.
- `1 - Enter location of houses`
  The user enters the x- and y-coordinates of each house into the database.
- `2 - Display house locations`
  A table with the x- and y-coordinates of each house is displayed.
- `3 - Set fire station radius`
  The users enters the radius size that the fire station can service.
- `4 - Assess fire station locations`
  The user enters any number of potential fire station coordinates and the score of a fire station at those coordinates is presented. The best tested coordinates and score are displayed.
- `5 - Clear house locations`
  Clear the database of house locations.

3 Error messages

- `ERROR: No houses entered!`
  when a user tries to display house locations or assess fire station locations without any houses in the database.
- `ERROR: Invalid menu choice!`
  when the user enters an invalid menu choice.
- `ERROR: Input must be an integer in [<LB>, <UB>]!`
  where `LB` and `UB` are lower and upper bounds. If UB is equal to the maximum storable value of a `double`, then "`infinity`" is displayed instead of the actual UB value. Similarly, if LB is equal to the negative of the maximum storable value of a `double`, then "`-infinity`" is displayed instead of the actual LB value.

- `ERROR: Input must be a real number in [<LB>, <UB>]!`
  where `LB` and `UB` are lower and upper bounds. If UB is equal to the maximum storable value of an `int`, then "`infinity`" is displayed instead of the actual UB value. Similarly, if LB is equal to the negative of the maximum storable value of an `int`, then "`-infinity`" is displayed instead of the actual LB value.

4 Required elements

Your project must use the each of the elements described in this section. You can have more functions, variables, and objects if you want, but you must at least use the elements described in this section.

4.1 Variables

- Global `public static BufferedReader` as **the only** `BufferedReader` object in the entire program for reading user input
  - Although your programs will compile and run just fine on your own computer with multiple `BufferedReader` objects, they will not run correctly on a web server with multiple `BufferedReader` objects. Since your programs will be graded on a web server, please don't have more than one `BufferedReader` for reading input from the console.
- One 2D array to store all house coordinates

4.2 Functions

Function prototypes and short descriptions are provided below. It is your job to determine exactly what should happen inside each function, though it should be clear from the function and argument names and function descriptions.

- `public static void displayMenu()`
  Display the menu.
- `public static int getHouseCoords(int [][] houseCoords, int nHouses)`
  Get the house coordinates from the user and put them into the array of house coordinates.
- `public static void displayHouses(int [][] houseCoords, int nHouses)`
  Display the house coordinates in a tabular fashion.
- `public static void assessFireStationLocations(int r, int [][] houseCoords, int nHouses)`
  Get fire station locations from the user and score them. The best location and its scored will be displayed.
- `public static int getInteger(String prompt, int LB, int UB)`
  Get an integer in the range [LB, UB] from the user. Prompt the user repeatedly until a valid value is entered.

5 Helpful hints

- What is really meant by "clearing" the database? Do you really need to empty all the values in the array or deallocate and re-allocate the array?