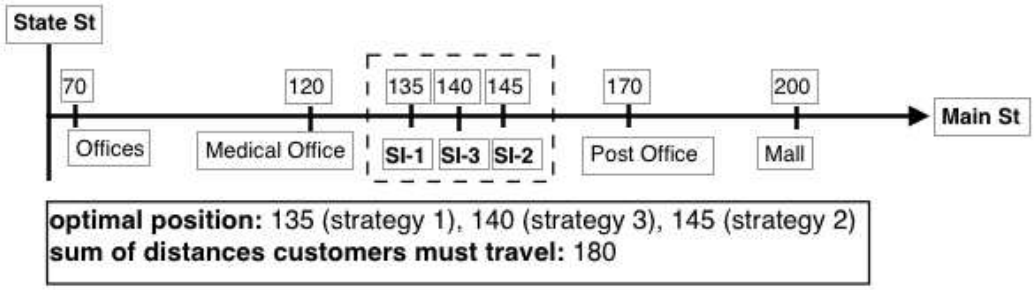
Computer Science I — CSCI-141
Donut Delite — Lab 5

10/13/2020

1 Problem

You have been hired to help determine the placement for a new Donut Delite store. There are several store fronts available along Main Street. Developers are trying to determine the best placement of a store based on the distance to several large office buildings. They want to locate the store so that it minimizes sum of the distances from the store to the offices on the street. Additionally, they would like to use the application to determine the location of any new store. (Ignore the fact that each building could have a different number of people.) The diagram below shows locations chosen by several strategies. Numbers indicate distances to the intersection of Main and State. The "Sl-N" text means store locations identified by the numbered strategies.



You were asked to consider the following strategies to determine the best location of the new store (sometimes there might be more than one best location).

1. **Midpoint Distance Strategy**
   Let x be the distance of the business closest to State Street. Let y be the distance of the business farthest from State Street. Locate the new store at the mid-point between x and y; that is, at location $(x+y)/2$. For the above example, this strategy produces the location 135 and the sum of the distances to this location is $65 + 65 + 15 + 35 = 180$.
2. **Median Distance Strategy**
   Sort the distances and choose the location of the middle element. If the number of elements is even, choose the midpoint between the two middle elements. For the above example, this strategy produces 145, and the sum of the distances to this location is $75 + 55 + 25 + 25 = 180$.
3. **Average Distance Strategy**
   Locate the store at the average of the distances. For this example, this strategy produces 140, and the sum of the distances to this location is $70+60+20+30 = 180$.

The problem is that only one of these strategies *always* provides the *best* possible sum of distances.

2 Problem-solving Session - (15%)

In a team of students, do the following:

1. Create a test case of 5 business office locations between 0 and 10 miles from State St. The test case should demonstrate that only one of the strategies is always optimal. Work through each strategy and indicate the calculated store location and the sum of the distances to each business, versus the optimal choice.
   Hint: If you think of the offices as distributed on a see-saw, how would you make it unbalanced?
   For each strategy your team should work in parallel to:
   (a) identify the optimal new store location;
   (b) compute the sum of the distances from this location to each office.
   Combine your answers, make a table of the results, and **identify the optimal strategy**. Team members should collect their answers, compare results, and list the results for each strategy.

2. Develop code that implements the optimal strategy. Given a list of the unsorted locations, write a function that returns the best location. (Assume that there is an already-written list sort function, and call it.)

3. Write another function that, given a list of locations and the best location, calculates and returns the sum of the distances between the best location and each other location.

4. Study the **quick_select** algorithm on the next page.
   Run the algorithm for `k := 3` on the following list.

   ```
   [5, 2, 9, 22, 1, 3, 17, 12, 4, 13]
   ```

   Show all the intermediate steps and the final solution.

2.1 Quickselect Algorithm

There is an algorithm, known as quickselect, that runs fast without sorting.

The quickselect approach is more efficient than sorting techniques seen so far. Quickselect finds the **kth smallest number** in an *unsorted* list of numbers. Study the algorithm below and consider how you could use quickselect to compute the median and solve the store location problem.

```
Function quick_select( a_list, k ):
    If a_list is not empty:
        pivot <- Choose the element at position ( len( a_list ) // 2 )
        smaller_list <- All elements of a_list smaller than pivot
        larger_list <- All elements of a_list larger than pivot
        count <- the number of occurrences of pivot value in a_list
        m <- the size of smaller_list
        If k >= m and k < m + count then:
            return pivot
        If m > k:
            return quick_select( smaller_list, k )
        Otherwise:
            return quick_select( larger_list, k - m - count )
```

It is a pre-condition that `k` is in the range `0` to `len(a_list)-1`.

3 Post-PSS - (10%)

For the Post-PSS complete the following tasks on *your own*.

The goal is to complete the first program that solves the problem.

1. Create a project for this assignment. There will be multiple source files.

2. Extract and copy the quicksort module from the lecture's zip file. Here is the link to the lecture materials:

   `https://www.cs.rit.edu/~csci141/Lectures/08/`

3. Create a Python file named `tools.py`. This module will contain functions that will be shared by other program modules.

4. Implement these reusable functions in `tools.py`:
   - A function to read the file and return the list of business locations; and
   - The function from problem-solving that, given a list of locations and the best location, calculates and returns the sum of the distances between the best location and each other location. The `abs()` function is useful to get the absolute value of the distance between two locations.

   An example of input file content is shown below. Return only the list of numbers.

   ```
   Offices 70
   MedicalOffice 120
   PostOffice 170
   Mall 200
   ```

5. Create the main program file named `store_location.py`. This module will contain a main and a function to compute the optimal location using quicksort. **You may not use either the builtin `list.sort()` function or the `sorted()` function**. Import the `quick_sort.py` module into `store_location.py`(do NOT copy the content). Then import the `tools.py` module into `store_location.py`, and write the function to compute the optimal location using quicksort. Finally, write the main function so that it reads the file, computes the optimal location, calculates the sum of differences, and prints the results as shown in the example below. The `store_location.py` must use the functions you wrote in the `tools.py` file.

Here is example of the output for the post-PSS:

```
Enter data file: test_data_writeup.txt
Optimum new store location:  145.0
Sum of distances to the new store:  180.0
```

Zip `store_location.py` module and the `tools.py` module into `postpss5.zip` and submit the zip to the post-PSS dropbox in MyCourses by the deadline. Read the rest of this assignment carefully to understand and then implement the remaining requirements.