Programming Assignment #3

Programming assignments are to be done individually. Do not make your code publicly available (such as a Github repo) as this enables others to cheat and you will be held responsible. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

This programming assignment is due **Sunday, August 8, 2021 at 11:59pm.** If you are unable to complete the assignment by this time, you may submit the assignment late until Tuesday, August 10, 2021 at 11:59pm for a 20 point penalty.

The goals of this lab are:

- Recognize algorithms you have learned in class in new contexts.
- Ensure that you understand certain aspects of dynamic programming algorithms.

Problem Description

In this project, you will implement a dynamic programming solution and write a small report. We have provided Java code skeletons that you will fill in with your own solution. Please read through this document and the documentation in the starter code thoroughly before beginning.

You are on the planning committee of the Austin Police Department, and you are given the task of building k police stations in a town with n houses along a line. Your goal is to achieve the best response time by picking the optimal locations for these k police stations. In other words, you want to guarantee that your police officers can be on the scene in at most r minutes, in case there is an emergency in any of the houses. You want to pick police station locations such that the response time r is minimized.

More precisely, you are given a set X of n houses sorted by position along a line, i.e, House 1 is at x_1, House 2 is at x_2 and so on with $x_1 < \cdots < x_n \in \mathbb{Z}$. Each house location x_i $(1 \leq i \leq n)$ is an **even integer**. You are also given an integer k, the number of police stations $(1 \leq k < n)$. You need to find the set of police station positions, C, of k points $c_1, ..., c_k \in \mathbb{Z}$ that minimizes the response time. In case there is an emergency at House i, and the closest police station is located at c_j, the time it takes for the police officers to be on the scene is calculated by:

$$t = \frac{|x_i - c_j|}{v},$$

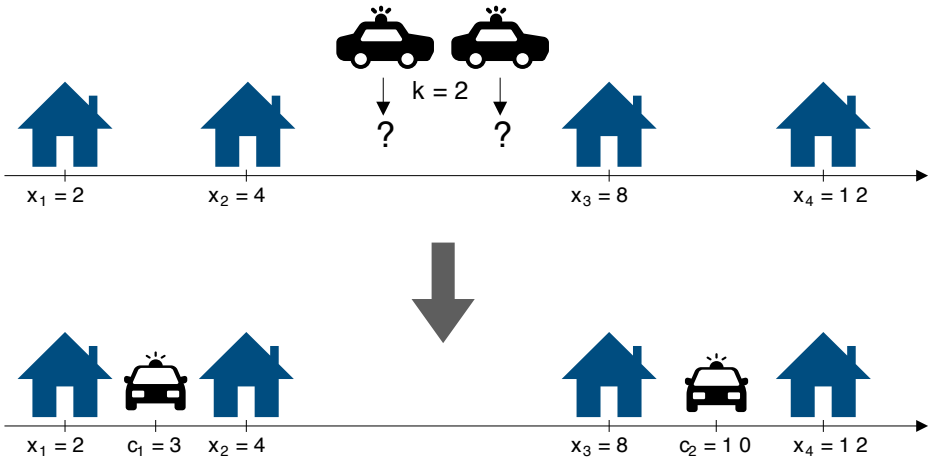
where v is the traveling speed. Without loss of generality, we assume that $v = 1$ throughout this assignment. We also assume that it is always possible to send police officers from the closest police

station to the scene. With these assumptions, the response time r that we can guarantee for any possible emergency is given by:

$$r = \max_{1 \le i \le n} \left(\min_{1 \le j \le k} |x_i - c_j| \right)$$

where $\min_{1 \le j \le k} |x_i - c_j|$ represents picking the closest police station for each house, and the maximization finds the global response time for the worst-case scenario.

For example, consider a town in Austin with 4 houses positioned at 2, 4, 8 and 12 with 2 police stations to be set up. The optimal response time for this given town plan is 2 which can be achieved by setting up the police stations at positions $c_1 = 3$ and $c_2 = 10$. This example can be better visualized with the illustration below. (Note that here the response time was determined by houses 3 and 4, as we can reach houses 1 and 2 faster.)



Hint: The problem might make you scratch your head, but it isn't as difficult as it seems! Think of the possible subproblems that could be used. You could construct a $n \times k$ table indexed $r[t, j]$ that stores the optimal response time for the subproblems and another indexed $c[t, j]$ that stores a set of police station locations for the subproblems. What can these sub-problems represent?

Part 1: Write a report [20 points]

Write a short report that includes the following information:

(a) Give the pseudocode for your implementation of a dynamic programming solution to find the above mentioned minimum response time r. Mention the base cases and recurrence equation clearly and briefly explain what it represents.

(b) Give the pseudocode for your implementation of a dynamic programming solution to find the set of police station positions, C.

(c) Justify the runtime of your algorithm (i.e. the algorithm that does both parts (a),(b) above).

Part 2: Implementation to find Optimal Response Time [50 points]
Implement your dynamic programming based solution for finding the optimal response time which you devised in the report. You are provided several files to work with. Implement the function that yields the optimal response time in `findOptimalResponseTime()` inside of `Program3.java`. You need to **only** update the data member `responseTime` of object `town` which belongs to class `TownPlan`. **Do not** update the data member `policeStationPositions` here.

Part 3: Implementation to find Set of Police Station Positions [30 points]
Implement your dynamic programming based solution for finding the police station positions which correspond to the optimal response time that you found. You need to implement the function that yields the police station positions in `findOptimalPoliceStationPositions()` inside of `Program3.java`. You need to **only** update the data member `policeStationPositions` of object `town` which belongs to class `TownPlan`. The police station positions should be sorted. **Do not** update the data member `responseTime` here. The reason for having these two functions update these data members separately is to give partial credit for the two sections.

If you are using two tables to construct your solution (as mentioned in the Hint), you are allowed to repeat the code from the previous part, i.e, from `findOptimalResponseTime()`.

Of the files we have provided, please only modify `Problem3.java`, so that your solution remains compatible with ours. However, feel free to add any additional Java files (of your own authorship) as you see fit.

Instructions

- Download and import the code into your favorite development environment. We recommend you use Java 1.8 and NOT other versions of Java, as we can not guarantee that other versions of Java will be compatible with our grading scripts. **It is YOUR responsibility to ensure that your solution compiles with Java 1.8 and our grader.** If you have doubts, email a TA or post your question on Piazza.
- **Do not add any package statements to your code.** Some IDEs will make a new package for you automatically. If your IDE does this, make sure that you remove the package statements from your source files before turning in the assignment.
- There are several `.java` files, but you should only make modifications to `Program3`. You may add additional source files in your solution if you so desire. There is a lot of starter code; carefully study the code provided for you, and ensure that you understand it before starting to code your solution. The set of provided files should compile and run successfully before you modify them.

- The main data structure for a town plan is defined and documented in `TownPlan.java`. A TownPlan object includes:
  - **n**: The number of houses
  - **k**: The umber of police stations to plan
  - **housePositions**: An ArrayList of even Integer values containing the position of each house, in ascending order. The positions can be indexed in order from 0 to $n - 1$.
  - **policeStationPositions**: An ArrayList of Integer values to hold the positions of police stations for the optimal solution. This ArrayList (should) hold the position of each of the k police stations in sorted order. This field will be empty in the `TownPlan` which is passed to your functions. The results of your algorithm should be stored in this field by calling `setPoliceStationPositions(<your_solution>)`.
  - **responseTime**: The optimal response time (Integer) that needs to be found. This field will be empty in the `TownPlan` which is passed to your functions. The results of your algorithm should be stored in this field by calling `setResponseTime(<your_solution>)`.
- You must implement the methods `findOptimalResponseTime()` and `findOptimalPoliceStationPositions()` in the file `Program3.java`. You may add methods to this file if you feel it necessary or useful. You may add additional source files if you so desire.
- `Driver.java` is the main driver program. Use command line arguments to choose between either displaying the optimal response computed through your solution or the set of positions of the police stations and to specify an input file. Use -r for response, -p for position of police stations, and input file name for specified input (i.e. `java -classpath . Driver [-r] [-p] <filepath>` on a Linux machine). As a test, the 10-5.in input file should output the following for displaying both response and set of police station positions (corresponding command is `java -classpath . Driver inputs/10-5.in`):
  n = 10 k = 5
  Optimal response time = 14

  n = 10 k = 5
  Station 0 Position 11
  Station 1 Position 37
  Station 2 Position 108
  Station 3 Position 216
  Station 4 Position 254
- You can find a few test case files in the `inputs` directory of the zipped `Programming_Assignment_3` starter files given to you.
- **Input File Format:**
  The first number on the first line of file is the number of houses and the second number is the number of police stations to plan. The next line contains the position of the houses

which **are sorted in ascending order**.
For example, if the input file is as follows:
6 3
2 6 8 10 14 20
Then there are 6 houses in the town and 3 police stations are required to be planned.
Think of the houses positions to be on a number line. Then,
House 0 is at position 2
House 1 is at position 6
House 2 is at position 8
House 3 is at position 10
House 4 is at position 14
House 5 is at position 20

- To make things easier, we will **not** be testing for the cases where there are no houses or no police stations to setup.
- Make sure your program compiles on the LRC machines before you submit it.
- We will be checking programming style. A penalty of up to 10 points will be given for poor programming practices (e.g. do not name your variables foo1, foo2, int1, and int2).

NOTE: To avoid receiving a 0 for the coded portion of this assignment, you MUST ensure that your code correctly compiles with the original, unmodified `Driver.java` on Java 1.8. Do not modify the signatures of or remove existing methods of `Program3.java`. Do not add package statements. Do not add extra imports. You must zip your code using the exact format described below with no spaces. We recommend testing compilation of your code using the ECE LRC Linux machines (using "javac *.java" and "java Driver inputfile.txt") after redownloading `Driver.java` from Canvas. We will not be allowing regrades on this assignment, so please be careful and double check that your final submission is correct.

What To Submit (please read carefully)

You should submit to Canvas a single ZIP file titled `pa3_eid_lastname_firstname.zip` that contains all of your java files and any extra `.java` files you added. Do not put your `.java` files in a folder before you zip them (i.e. the files should be in the root of the ZIP archive). Your ZIP file name MUST have the exact format: `pa3_eid_lastname_firstname.zip`. Be certain that there are no spaces in your zip file name. Failure to follow this naming format will result in a penalty of up to 5 points.

Your PDF report should be typed or legibly scanned and submitted to Gradescope. Both your zipped code and PDF report must be submitted BY 11:59pm on Sunday, August 8, 2021. If you are unable to complete the assignment by this time, you may submit the assignment late until Tuesday, August 10, 2021 at 11:59pm for a 20 point penalty.