Programming Assignment #3

Programming assignments are to be done individually. 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 project has several goals:

- Ensure that you understand dynamic programming.
- Able to argue for optimal substructure of the problem and define recurrence for optimal solution.
- Able to implement and test the algorithm for the optimal solution proposed.

Part 1: Allocating Resources Optimally (50 points)

You are the Black Panther, the leader of the Kingdom of Wakanda. Your kingdom owns some Vibranium, a rare metal that has the potential to be used in a wide variety of applications. Wakanda's Council of Elders has decided use some of the Vibranium in various R&D projects to improve Wakanda's technology. They have asked you to figure out which R&D projects should be invested in. There are n projects, and the Council has assigned V Vibranium for R&D (V is a positive integer).

Each project can grow the Wakandan economy by a certain percentage that goes from 1 to $p > 1$. For any given project, this percentage can vary based on the amount of Vibranium spent on it. Assume you have a set of functions $\{f_0, f_1, ..., f_{n-1}\}$, where $f_i(v)$ is the percentage that the economy will grow if you spend v amount of Vibranium on project i. The functions f_i are nondecreasing; spending more Vibranium on a project will not *lower* the economy growth. You can only spend integer amounts of Vibranium on each project. As the Black Panther, your goal is to divide the Vibranium across projects such that the growth of the Wankadan economy is maximised.

Given V Vibranium, n projects and functions $\{f_0, f_1, ..., f_{n-1}\}$, determine the maximum potential gain in the Wakandan economy. You algorithm should be polynomial in V and n.

For full credit, in your written report, you should argue that the problem has optimal substructure, define the value of the optimal solution (i.e., by providing the recursive definition), describe an algorithm for iteratively computing the value of the optimal solution, and describe an algorithm for recreating the optimal solution (i.e., mapping your H hours to your n projects). Your implementation should provide a Java implementation of your algorithms.

Part 2: Protecting Resources (50 points)

Since Vibranium has so many uses, it is no surprise that thieves are always surrounding Vibranium mines to sell Vibranium ores in the black market. To figure out how much loss your kingdom can incur, you play devil's advocate and assume the role of a thief who has a bag that could hold atmost W kilograms and V cubic meters of Vibranium ores.

Assume that there are n pieces of Vibranium ores laying on the floor for you to steal, each with selling price in Wakandan dollars, weight in kilograms, volume in cubic meters. Note that the price, weight, and volume need not be the same for each ore. Now, given this information, you want to maximize the total selling price of the Vibranium ores you put into the bag while maintaining the weight and volume constraints of the bag.

More formally, given Vibranium ores $o_1, ..., o_n$ each with a price (p_i), weight (w_i), and volume (v_i) provided, determine the maximum total price of Vibranium ores the thief could steal by putting ores into a bag with weight capacity of W kilograms and volume capacity of V cubic meters. Note you only need to output a number denoting the total Wakandan dollar amount of Vibranium ores the thief can steal.

Provided Code

You are given the following:

- `EconomyCalculator` class: This class has already been implemented for you. It takes in a text file representing the gain functions and parses them for use in your program. Do not modify this class to make your solution work. We will be using our own version of `EconomyCalculator` during grading. Treat it as a black box to access the gain functions $\{f_0, f_1, ..., f_{n-1}\}$.
- `Driver` class: You can run this to test your code. We will be using our own Driver class during testing, so ensure that your solution is not dependent on parts of your `Driver.java` code. To test your code, use the parameters `-1` for part 1 and `-2` for part 2 followed by the appropriate file. Ex. `java Driver -1 4.in` would test part 1 of your solution with the input file `4.in`.
- `Vibranium` class: This class has already been implemented for you. It represents one Vibranium ore and contains the price, weight, and volume of the Vibranium ore. Use the getter methods to access these attributes of the Vibranium ore.
- `VibraniumOreScenario` class: This class has already been implemented for you. It takes in a text file representing the weight and volume capacities of the thief's bag in addition to the different Vibranium ores laying down for the thief to steal. Do not modify this class to make your solution work. We will be using our own version of this class during grading.
- `Program3` class: You will be implementing the solution here. We have provided the function headers that we will call when testing your program. You are allowed to create additional methods or classes to solve the problem. We will assume your `Program3.java` has a no-parameter constructor, and we will initialize `Program3` using the `initialize` method.

You may not change or augment the provided code in any way. Your solution should be provided in `Program3.java`. You may include additional class files as well.

What to Submit

Failure to follow these instructions **will** result in a deduction of points.

(a) Implement your solution in `Program3.java`. You may create any additional classes you want; be sure to submit them as well. Document your code (with comments and self-documenting code).

(b) Describe in a brief report your approach to testing your implementation. Describe (briefly) what test cases you defined, how they provided complete coverage of your implementation, and what, if any, interesting bugs you found.

(c) Include your name and UT EID at the top of any of the files you submit, including Program3.java and your report.

(d) Submit `.java` files, not `.class` files.

(e) Submit a PDF for your report, not a DOCX or any other format.

(f) Only your most recent submission to Canvas will be graded. You can submit as many times as you like, and we will only grade the most recent submission. However, this also means that if you submit your code once and then you submit your report (and not your code) later, we will only see your report, and your code will not be graded.

(g) **DO NOT USE PACKAGE STATEMENTS.** Your code will fail to compile in our grader if you do.

(h) When you create your `.zip` file as described in the submission section below, please open up your file in your archive viewer to **ensure that the files are visible from the root of the archive**, i.e., there should be no folder **inside** your zip file.

Submission

You should submit a single file titled `EID_LastName_FirstName.zip`, along with a report (`.pdf` file) titled `EID_LastName_FirstName.pdf` that contains your report. Your solution must be submitted via Canvas before the deadline.

If you name the `.zip` file or ESPECIALLY the `.pdf` file incorrectly, points will be deducted.