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: Constrained Shortest Path (50 points)

Thanos has just acquired the Power Stone on Xander and is ready to acquire the next Infinity Stone on his quest for ultimate power. He has pinpointed the exact location of the Reality Stone to be in the Collector's collection on Knowhere and his ready to head there. However, he must first decide what route to take. He wants to get there as fast as possible but is constrained by the amount of fuel in his ship.

More formally, consider an undirected graph G where every edge e has two weights: travel time $a_e \geq 0$ and travel cost (e.g. fuel) $b_e \geq 1$. You are given two nodes s and t in G (Xander and Knowhere). The goal is to find the $s-t-path$ with the smallest travel time subject to a limit B on the total travel cost. The travel time of a path p is defined as the sum of travel times of edges along the path ($\sum_{e \in p} a_e$), and the travel cost of a path p is defined as the sum of travel costs of edges along the path ($\sum_{e \in p} b_e$). In other words, you need to find the $s-t-path$ p that minimizes $\sum_{e \in p} a_e$ such that $\sum_{e \in p} b_e \leq B$. Note you only need to output a number denoting the travel time needed to get from s to t. (Hint: Let OPT[v,b] be the minimum travel time from node v to t with cost at most b.)

Your written report should include:

(a) The optimal substructure for the problem and the travel time of the optimal solution (i.e., by providing the recursive definition)

(b) Your algorithm (in psuedocode) for calculating the travel time of the optimal solution

(c) An analysis of the time and space complexity of your algorithm

Part 2: Allocating Resources Optimally (50 points)

Thanks to his efficient route planning, Thanos has successfully obtained 5 of the Infinity Stones and his headed to Wakanda where you and the Avengers are currently protecting the Mind Stone. At this point, Thanos is too powerful and will be able to obtain the last stone in a fixed amount of time. Your only chance of stopping him is to inflict as much damage as possible in this amount of time. As the Avengers, you have a list of attacks that you can use against Thanos (Hulk Smash, Iron Man Laser Beam, etc.) that do various amount of damage given a certain amount of time.

There are n attacks and Thanos will be able to take the last stone in T time.

Each attack can do a certain amount of damage $x \geq 1$. For any given attack, the damage can vary based on the amount of time spent using it. Assume you have a set of functions $\{f_0, f_1, ..., f_{n-1}\}$, where $f_i(t)$ is the damage inflicted on Thanos if you spend t amount of time using attack i. The functions f_i are nondecreasing; spending more time on an attack will not *lower* the damage done to Thanos. You can only spend integer amounts of time on each attack. Also, only one attack can be used at a time so that you do not hit another Avenger. Your goal is to divide the time across attacks such that the damage done to Thanos is maximized.

Given T time, n attacks and functions $\{f_0, f_1, ..., f_{n-1}\}$, determine the maximum damage that can be inflicted. You algorithm should be polynomial in T and n.

Your written report should include:

(a) The optimal substructure for the problem and the value of the optimal solution (i.e., by providing the recursive definition)

(b) Your algorithm (in psuedocode) for computing the value of the optimal solution

(c) Describe an algorithm for recreating the optimal solution (i.e., mapping your T time to your n attacks)

(d) An analysis of the time and space complexity of your algorithm

Provided Code

You are given the following:

- `DamageCalculator` class: This class has already been implemented for you. It takes in a text file representing the attack 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 `DamageCalculator` during grading. Treat it as a black box to access the damage 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`.

- `SpaceFlight` class: This class has already been implemented for you. It represents one edge between two planets and contains the destination planet, the travel time, and the travel cost. Use the getter methods to access these attributes of the flight.
- `PlanetPathScenario` class: This class has already been implemented for you. It takes in a text file representing the planets, fuel capacity, edges between planets, and start and end planets. 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.