WealthTransfer:

Avraham Leff

November 24, 2022

1 Assignment-Specific Packaging

The general packaging is unchanged from the basic "Homework Requirements" (see slides from first lecture and "Homework Policies for COM 2545" on Piazza).

This assignment's "DIR" **must be named** *WealthTransfer*, and your application's Java class **must be named** *WealthTransfer.java*. Your write-up file **must be named** *WealthTransfer.pdf*.

2 Motivation

The assignment is intended to give you more experience applying (and extending) concepts learned in lecture to solve new problems, and more practice explaining an algorithm's design to others.

3 Background

A person (we'll refer to her a "person #1") wishes to transfer wealth to a set of her descendants. Current tax law implies that wealth can't be transferred directly to a descendant: wealth can only be transferred via a recipient's parent. (Note that transferring directly to a child is OK.)

Several important issues further complicate the wealth transfer:

- Wealth transfers are expressed in terms of *percentages*, not actual amounts.
- If a person makes a wealth transfer to anyone (i.e., is a "donor"), she must transfer all of her wealth. That is: the percentages of the transfers must sum to 100.

- A person need not make a transfer at all: instead, the person is only a *recipient*.
- Some wealth transfers can have their magnitude *squared* ("complicated inheritance tax law for a given tax bracket").

  Example: a person has $50 and wishes to transfer 50%. When the wealth transfer is designated to be a "squared transfer", the recipient will receive $\$625$ rather than $\$25$.

3.1 The API

The lifecycle of this problem is the following:

1. The constructor defines the size of the population, implicitly assigning ids to persons.
2. `intendToTransferWealth` specifies the percentage of wealth that one person wishes to transfer to another. It also specifies whether or not the transfer is a "squared" wealth transfer or not.
3. `setRequiredWealth` specifies the amount (not the percentage!) of wealth that the designated person is supposed to receive.
4. `solveIt` determines the ***minimum*** *amount of wealth* that "person 1" provides to the system such that:
   - All `setRequiredWealth` specifications are met
   - Wealth flows from "person #1" to the recipients according to the rules specified by previous invocations of the previous two methods.

3.2 The Problem

Keep in mind: by supplying a "gazzilion" dollars to the system, all recipients will receive the wealth that they're supposed to receive. The problem's difficulty is to have `solveIt` return the minimum amount of money to "get the job done".

3.3 Observation

It may seem to you that this problem "must be" a *graph problem*. This is not the case!

To be more specific: since we've only begun the topic of graphs, I would not assign a problem that requires a graph-based solution. I assure you that you already possess the algorithms and data-structures knowledge required to solve the problem.

4 Requirements

> Aside from using the JDK, for this assignment, you may **only use code written by yourself or copied from the textbook**!

The assignment consists of both "programming" and "non-programming" components.

4.1 Non-Programming

This portion of the assignment is worth 30%. No more than a **page and a half**!

There are many ways to design a solution to this problem. In this section, you must explain your approach in its entirety.

Note: I plan to grade this part based on: *"with no prior knowledge of this problem, can an CS person understood how you solved the problem?"*. Yes, I always grade based on your *"communication skills"*: clarity, correctness, succinctness, and "convincingness". This assignment will place an even greater weight on your success in this area.

As usual, pseudo-code is the right level of detail. If appropriate, consider incorporating a diagram into your discussion.

You may "explain by reference" to material covered in lecture or textbook or to "well-known" CS-knowledge. If you choose to use a "reference to covered material", you **must offer a convincing explanation** that this material is relevant to solving our problem.

4.2 Programming

This portion of the assignment is worth 70%, and will be graded for "correctness" and "speed" (see below).

> Please review the general requirements for a programming assignment! I've tried to reduce the chances of "mistakes" occurring through the use of a "skeleton interface", but ultimately, **you are responsible** for ensuring that I can compile and test your code without incident.

It's difficult to specify an "order-of-growth" performance requirement for this problem because the problem's difficulty depends on factors other than the "size of the population". The performance requirement is "only" that your code respond very, very fast to test invocations.

To give a sense of what I mean: my implementation solves the test suite in *375 ms*. You will be graded (in addition to "correctness criteria") based on whether your code takes longer than *450 ms*.

1. Begin by downloading a skeleton `WealthTransfer.java` class from this git repository.
2. Then, implement the stubbed methods of *WealthTransfer*.