Project 3

EN.500.133 Bootcamp: Python

January 6, 2024

1 Problem A

Planet Namek has signed a comprehensive trade agreement with Earth, and Namek is undergoing a huge, capitalism-driven makeover. People are moving in from Earth and malls and Burger Kings are popping up everywhere. You, an enterprising individual, have decided to re-brand yourself as an investor in Namekian businesses.

Figure 1 shows the map of the newly constructed Piccolo Mall. It has 4 buildings A, B, C, and D. There are pedestrian walkways between buildings (arrows). Since malls tend to be crowded, these walkways are meant for one-way traffic only. The arrows indicate the direction of foot traffic. Between A and D and between C and D, there are two walkways between shops, one in each direction.

The mall builders ran out of money towards the end of construction. To cut costs, they built only one walkway between buildings C and A. Same goes for D and B.

Each building has several stores and you have decided to open your own store but have not yet decided in which building to do so. Perhaps the builders' error can work to your advantage, as a lack of walkways could make one of the buildings more populous on average. Knowing that people who make purchases at the store in one building are likely to make purchases in the other stores in that building, you would like to rank these buildings based on the purchasing behavior in each.

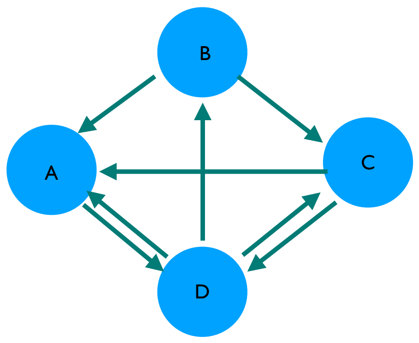

Figure 1: Map of mall buildings. An arrow indicates the presence of a walkway and the direction of foot-traffic between two buildings.

1.1 Algorithm

Given a specific (albeit simplistic) model predicting the purchasing behavior of customers during a visit to the mall, here is an algorithm for determining each building's revenue. We will assume that the mall has n visitors. For simplicity we will aggregate the stores in a building so that we only need to keep track of the total revenue and pricing schemes of the buildings as a whole. Each visitor has a budget that they plan to spend at the mall during their visit. At the start of their visit, they randomly choose one of the buildings to shop at. They make a purchase, if their budget allows, and then randomly choose another building that can be reached via the outgoing walkways from the building they are currently in. They will continue making purchases and moving from building to building until they no longer have sufficient funds to make a purchase at their current location. At this point they will cease moving around and cease making purchases. As the visitors make their way through the mall on their visits, you should keep track of the purchases made at each building so that you can determine its total revenue.

You may have realized that the map above is simply a directed graph. The technique described above to rank the network is also quite general and can be applied to say, finding the most popular poster on Tiktok or designing roads and traffic lights. It is a modified version of Google's Pagerank algorithm.

1.2 Preliminary Task and Constraints

In this project, you must write a class for working with fractions. You may not use any floating point calculations in the fraction class, nor in the main task that follows. You also may not use any external libraries other than the `random` library, as described in Section 1.3.

Below is a short code snippet that uses a module called `frac`, which assumes we have defined a class for fractions called `Frac` in a file named `frac.py`.

```
import frac
x = frac.Frac(1,2)
y = frac.Frac(1,3)

print(x+y)
print(x*y)
```

```
5/6
1/6
```

Task 1: Please write your own class for fractions in a file named `frac.py`. Using this class, one should be able to initialize a fraction object by taking in a numerator and denominator integer value as input. The `Frac` class should have two attributes, named `num` for numerator, and `den` for denominator.

The values for `num` and `den` should both be passed in as arguments to your `__init__` function (in that order). Your class must overload the $+, -, *, /$ operators for fractions. You should also define a `simplify` function which is called as part of each overridden arithmetic operation so that the fraction's internal representation is always in simplified form. For instance, in the expression `Frac(3, 12) + Frac(6, 8)`, the resulting object should be a `Frac` object with numerator and denominator both equal to 1.

You may wish to further define and use a set of methods that may be needed when working with fractions in the function `run_simulation`. For instance, you may find it useful to overload the $==, !=, >, <, >=$, and $<=$ operators for comparisons, but this is not required. It *is* required, however, that your class overload the arithmetic operators $+, -, *, /$ and implement the `simplify` function.

1.3 Main Task

You will write a script that implements the technique discussed in the intro, but for an *arbitrary directed graph* rather than just the one in Figure 1. The information on connectivity between the buildings in the arbitrary graph will be provided in a text file. You may modify the code snippet from the Project 1 spec to read in input files.

```
A  D
B  A
B  C
C  A
C  D
D  A
D  B
D  C
```

Figure 2: Contents of sample input file

The contents of the network connectivity file will be of the form shown in Figure 2. Specifically, each line represents a single-direction link between two nodes. In the example shown in the figure, the first line of the file indicates that there is a link from A to D, the second line indicates a link from B to A, and so on.

In addition to the network structure information, a second file will contain information regarding the pricing scheme that each building has instituted for the products they sell. Namekians have a rather strange pricing scheme: they charge based on the prospective buyer's budget, above a minimum accepted price. If the prospective buyer's remaining budget is greater than the minimum price, the buyer pays the business a fraction of their budget (rather than a fixed amount). If the remaining budget is less than or equal to the minimum price, then no purchase is made.

Each line in the pricing input file will contain 5 fields. The first will be a string, corresponding to the ID of one of the buildings in the network. The next two will correspond to the numerator and denominator of the minimum price accepted by the building. The final two will correspond to the numerator and denominator of the fractional pricing scheme.

As an example, if the prospective buyer's remaining budget is $\frac{7}{5}$ and building A's pricing scheme is $\frac{2}{5}$ and $\frac{1}{5}$ then a purchase will happen, since $\frac{7}{5} > \frac{2}{5}$, and the buyer will pay $\frac{7}{25}$ ($\frac{1}{5}$th of $\frac{7}{5}$), leaving her with a remaining budget of $\frac{28}{25}$. If, however, the prospective buyer's budget were $\frac{1}{10}$ then no purchase would happen, since $\frac{1}{10} < \frac{2}{5}$.

1.3.1 Node Class

Task 2: First, write a class called `Node` corresponding to each of the buildings. Each node has the following attributes: (i) `id` (a string), an ID to uniquely identify it, (ii) `connected_nodes`, a list of Node IDs to which this Node has outgoing edges, (iii) and (iv) `minimum_price` and `fractional_price` as described above (Frac objects), and (v) the building's revenue for the day `revenue` (also a Frac object). The values for the ID, `connected_nodes`, and price variables should be passed in as arguments to your `__init__` function in that order. You should also set the value of `revenue` to zero in the `__init__` function.

You are not required to write any other methods for this class but you may write additional class functions (such as getter and setter functions) as you see fit.

1.3.2 Buyer Class

Task 3: Next, write a class called `Buyer`. Each buyer has an ID that indicates which node the buyer currently occupies, which is stored in an attribute named `current_node_id`, and a `remaining_budget` attribute which is the amount of money the Buyer has left to spend. The initial values for `current_node_id` and `remaining_budget` should be passed in as a string argument and Frac argument to your `__init__` function, in that order.

The starting value for the Buyer's budget will be specified by a third input file. The input file will be in the form of a single line with several integers. Each integer will correspond to the budget of one Buyer. For instance, if the file contains 10 integers, there will be 10 Buyers in the program. Again, it is not required that you specify any member functions for this class beyond the constructor but you nevertheless may add additional member functions or attributes to help you out with the other tasks, as you see fit.

1.3.3 Simulation

Task 4: Outside the `Node` and `Buyer` class definitions, write a function called `run_simulation` that takes as input (i) a filename for the connectivities, (ii) a filename for the building pricing schemes, and (iii) a filename for the Buyer budgets, in that order. In this function, create a graph (a list of Nodes) using the connectivity information in the first file and the pricing information in the second file. You may use other data structures as well. Then create a list of `Buyer` objects with budgets as specified by the budet input file. Simulate their movement and purchasing on the graph. Recall that Buyers can move only from the building they are in to the buildings their current building is connected to by an outgoing edge and that Buyers choose the next building to visit at random among these outgoing connections. Additionally, once the Buyer has encountered a building in which they are unable to make a purchase, they stop moving among buildings and 'leave' the system.

Keep track of the revenue obtained by each building. **Your function should return two objects: the total revenue of all buildings and a dictionary with Node names (that is, the `id` names from Node objects) as the keys and the *fraction* of the total revenue that each building earned as values.** That is, if the total revenue among all buildings is 100 and buildings A, B, C, and D earned 25 each, you should return a `Frac(100,1)` object and a dictionary:

```
{'A': Frac(1, 4), 'B': Frac(1, 4), 'C': Frac(1, 4), 'D': Frac(1,4)}
```

2 Instructions for submission

- Submit two files (i) `mall_buyers.py` and (ii)`frac.py`. The second file contains the definition for the `Frac` class.
- We will test if your code works on general input files which will differ in values (e.g. number of nodes) from those described here, but will have formats consistent with what has been described. Note that although the number of nodes described in the example input file was 4, the number described by an input file in general may be different than 4. Additionally, node names need not be single letters; in general, they may be strings consisting of multiple non-space characters.
- We will import `frac.py` as a module, to test out a few operations using your `Frac` objects.
- Include significant comments in your code. Your code will be evaluated not only on correctness, but also on style and good programming techniques. Write docstrings for all functions and classes.
- Do not import any modules in `mall_buyers.py` except `random` and your `frac` module.
- If your solution is not working perfectly, turn it in as-is with detailed comments describing which parts are complete, and which still need work. Be sure that what you turn in runs, to make it possible to receive feedback and partial credit.
- You are strongly encouraged to adopt an incremental coding style, making small changes to your code one at a time, so that you always have a version of your program which runs, meaning you will always have something to turn in, even if it is not 100% complete. You are also reminded to exercise discipline in backing up your work.If you have trouble or need extra help, don't hesitate to attend office hours or contact a member of the course staff through Piazza.
- You are permitted to discuss the instructions with other students for clarification. However, you may not discuss any substance of your solution with anyone except

course staff, whether that be algorithms, test cases, component design, or actual code.