Banks in Financial Distress

Banks usually lend money to each other. A bank may not be able to repay the loan if it goes bankrupt during hard times economically. The current balance plus any loans made to other banks make up a bank's total assets. Figure 1.1 illustrates five banks. The current balances of the banks are 25, 125, 175, 75, and 181 million dollars respectively. The directed edge from node 1 to node 2 shows that bank 1 lends bank 2 a sum of 40 million dollars.

A bank is considered unsafe or risky if the sum of its total assets falls below a specific limit. The money it borrowed cannot be returned to the lender, and the lender cannot count the loan in its total assets. Consequently, the lender may also be unsafe, if its total assets are under the limit.

As shown in Table 1.1, the first number in the line is the bank’s balance, the second number indicates the number of banks that borrowed money from the bank, and the rest are pairs of two numbers. Each pair describes a borrower. The first number in the pair is the borrower’s ID and the second is the amount borrowed. For example, the input for the five banks in Figure 1.1 is as follows (note that the limit is 201):

Table 1.1: Analysis of asset and loans of banks

5 201 -> n, limit

25 2 1 100.5 4 320.5 -> bank 0's balance, number of loans, loan to bank 1, $100.5, loan to bank 4, $320.5

125 2 2 40 3 85

175 2 0 125 3 75

75 1 0 125

181 1 2 125

The total assets of bank 3 are (75 + 125), which is under 201, so bank 3 is unsafe. After bank 3 becomes unsafe, the total assets of bank 1 fall below (125 + 40). Thus, bank 1 is also unsafe.

The output of the program should be

Unsafe banks are 3 1

(Hint: Use a two-dimensional array borrowers to represent loans.

borrowers[i][j] indicates the loan that bank i loans to bank j. Once bank j becomes unsafe, borrowers[i][j] should be set to 0.)

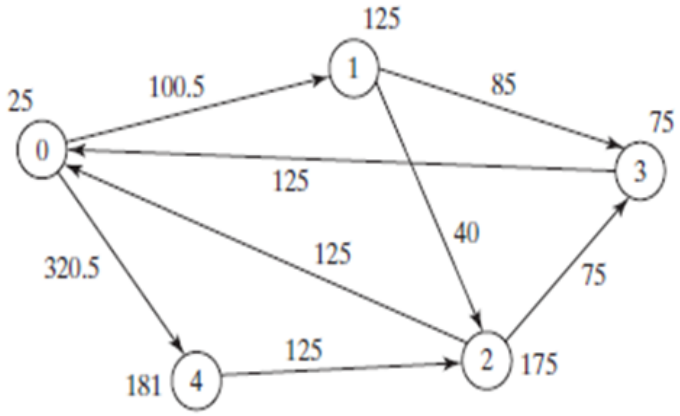

Figure 1.1 Lending of money between banks.

Source: Obaro A (2023)

1.1. Write a program to find all the unsafe banks. Your program reads the input as follows. It first reads two integers n and limit, where n indicates the number of banks and limit is the minimum total assets for keeping a bank safe. It then reads n lines that describe the information for n banks with IDs from 0 to n – 1.

(15 Marks)

1.2 Create an evaluateSafety class that evaluate the safety of banks. The class checks each bank's balance, remove loans to unsafe banks and re-checks/removes all banks with new unsafe bank values.

(10 Marks)

1.3 Create a handleUnsafeBank method which set all loans by each bank to zero (0.0) and returns borrowers bank, so that the lender cannot count the loan in its total assets.

(5 Marks)

1.4 Create a getBorrowers method get the details of the borrowers (id or index of banks that borrowed from the bank(s) in the borrower array).

(10 Marks)

A CLIENT-SERVER SYSTEM APPLICATION

Queues are used to implement the First-in-First-out (FIFO) protocol commonly used in a client-server application. For example, when cars on a toll road arrive at a toll plaza, the cars are the clients, and the toll booths are the servers. If the rate at which the cars pass through the toll booths is slower than their arrival rate, then a waiting-line builds up. That is a queue.

The concept *object-oriented programming* (OOP) in which Java objects are instantiated are utilized to represent all the interacting clients and servers. Usually, clients arrive for service at random times and services have random durations. Each client will have an arrival time, a time when service starts, and a time when it ends. All time values will be integers.

Each Server object also stores the time when it will stop serving its current client. That time is computed by adding its service time (a positive random integer) to the time when it begins serving that client. The random number generator used to generate those service times is stored as a random field in the Server object. A server's actual service time varies with each client. But the server's average service time is a fixed property of the server, initialized when the Server object is constructed.

For this system to be realistic, it must use randomly generated numbers to simulate the natural uncertainty of the real word. Those random numbers should have the same distribution as the natural uncertainties that they represent. Service times and time between client arrivals both tend to be distributed exponentially. Thus, the system needs to instantiates two random exponentially distributed number generators and separate arrays for the Server and Client objects.

The actual system is performed by the main class which sets four constants for the simulation, consisting of the number of servers, the number of clients arriving for service, the mean service time among the servers, and the mean time between arrivals for the clients. A Queue is further employed to hold the clients that have arrived for service and are waiting for an unoccupied server.

The output of your program should be displayed as the format below:

```
Number of servers:              3
Number of clients:             12
Mean service time:             25
Mean interarrival time:         4
Mean service time for Server A: 17.2
Mean service time for Server B: 51.7
Mean service time for Server C: 24.5
#1 arrived at time 0.
        Client queue: [#1]
        Client queue: []
Server A started serving client #1 at time 0.
#2 arrived at time 2.
        Client queue: [#2]
        Client queue: []
Server B started serving client #2 at time 2.
#3 arrived at time 4.
        Client queue: [#3]
        Client queue: []
```

Source: Obaro A. (2023)

To implement the described system:

2.1 Write a Client System program that iterates once for each clock tick t, and continues until all the clients have arrived. If it is time for a new client to arrive, set the next arrival time, create a new Client object, add the new client to the queue and update the Server object.

(15 Marks)

2.2 Create Client Class and its Client constructor which set and print the arrival time for each client.

(10 Marks)

2.3

a. Create a Server Class that serves at most one client at time in such a way that the Server class has a client field that references that server's client, or is null when the server is idle.

(10 Marks)

b. In the Server class, create a startServing() method which assigns a new client to the server, stores the start time in the Client object, computes and stores the stop time in its own stopTime field, and prints a report of those actions.

(5 Marks)

c. Within the Server class, create a stopServing() method that stores the stop time in the Client object and prints another report. If it is time for an active server to finish serving its client, then its stopserving() method is invoked. If a server is idle and there are clients waiting in the queue, then the next client in the queue is removed from the queue and that server begins serving it.

(10 Marks)

[Sub Total 25 Marks]

2.4 Create the two random number generators and a separate array for the Server and Clients objects.

(10 Marks)

End of Question 2