Banker's Algorithm Lab

50.005 Computer System Engineering

Due date: 05 March 2020, 23:59

Natalie Agus
INFORMATION SYSTEMS TECHNOLOGY AND DESIGN

Introduction

Objective: Write a Java/C program that implements the banker's algorithm. You may choose to do in *either* Java or C. **You don't have to implement both.**

Relevant Material: *Section 7.5.3 Banker's algorithm, P331-334, Operating System Concepts with Java, Eighth Edition*

There are several customers request and release resources from the bank. The banker will grant a request only if it leaves the system in a safe state. A request is denied if it leaves the system in an unsafe state.

The bank will employ the banker's algorithm, whereby it will consider requests from n customers for m resources. The bank will keep track of the resources using the following variables:

Java:

```
private int numberOfCustomers; // the number of customers private int numberOfResources; // the number of resources

private int[] available; // the available amount of each resource

private int[][] maximum; // the maximum demand of each customer

private int[][] allocation; // the amount currently allocated

private int[][] need; // the remaining needs of each customer
```

C:

```
int numberOfCustomers; // the number of customers

int numberOfResources; // the number of resources

int *available; // the available amount of each resource

int **maximum; // the maximum demand of each customer

int **allocation; // the amount currently allocated

int **need;// the remaining needs of each customer
```

This lab is organized into three parts:

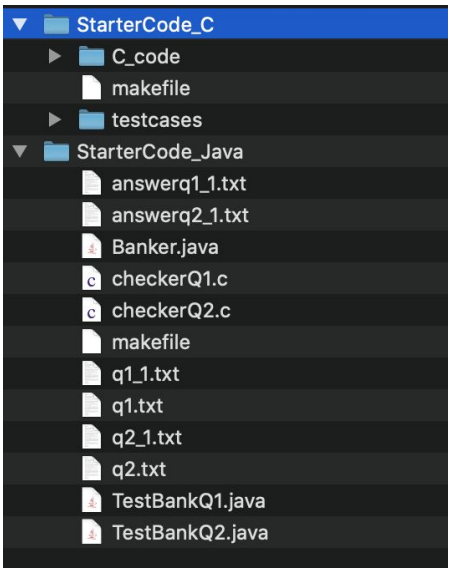
- **Q1:** Implementing a basic bank system (tested with q1_1.txt) **(20 marks)**.
- **Q2:** Implementing a safety check algorithm (tested with q2_1.txt) **(25 marks)**.
- **Q3**: Analysis of the complexity of the Banker's algorithm **(5 marks)**.

Download Materials, Compile, Run

Clone the materials:

`git clone https://github.com/natalieagus/50005Lab2.git`

You will find two types of starter code. Choose one with a language that you prefer.



If you choose C, this is your output after make, and trying to run with given test cases (you might also be met with segfault, depending):

```
natalieagus@Natalies-MacBook-Pro-2 StarterCode_C % make
gcc -o checkq1 C_code/checkerQ1.c
gcc -o q1 C_code/BankerQ1.c
gcc -o checkq2 C_code/checkerQ2.c
gcc -o q2 C_code/BankerQ2.c
natalieagus@Natalies-MacBook-Pro-2 StarterCode_C % ./q1 testcases/q1_1.txt

Current state:
Available:

Maximum:

Allocation:

Need:

natalieagus@Natalies-MacBook-Pro-2 StarterCode_C % ./q2 testcases/q1_1.txt

Current state:
Available:

Maximum:

Allocation:

Need:

natalieagus@Natalies-MacBook-Pro-2 StarterCode_C %
```

If you choose Java, this is your output after `make`, and trying to run them with given test cases (you will find that exception because the variables are not yet initialized):

```
natalieagus@Natalies-MacBook-Pro-2 StarterCode_Java % make
javac Banker.java TestBankQ1.java TestBankQ2.java
gcc -o checkq1 checkerQ1.c
gcc -o checkq2 checkerQ2.c
natalieagus@Natalies-MacBook-Pro-2 StarterCode_Java % java TestBankQ1.java q1_1.txt

Current state:
Available:
null

Maximum:
Exception in thread "main" java.lang.NullPointerException
        at Banker.printState(Banker.java:54)
        at Banker.runFile(Banker.java:207)
        at Banker.main(Banker.java:223)
        at TestBankQ1.main(TestBankQ1.java:9)
natalieagus@Natalies-MacBook-Pro-2 StarterCode_Java % java TestBankQ1.java q2_1.txt

Current state:
Available:
null

Maximum:
Exception in thread "main" java.lang.NullPointerException
        at Banker.printState(Banker.java:54)
        at Banker.runFile(Banker.java:207)
        at Banker.main(Banker.java:223)
        at TestBankQ1.main(TestBankQ1.java:9)
natalieagus@Natalies-MacBook-Pro-2 StarterCode_Java %
```

As usual, you can call `make clean` to remove all results.

Q1: Implement a basic bank system (20 marks)

The first part of the project is to implement the following functions:

Java:

```
public Banker (int[] resources, int numberOfCustomers);

public void setMaximumDemand(int customerIndex, int[] maximumDemand);

public synchronized boolean requestResources(int customerIndex, int[] request);

private synchronized boolean checkSafe(int customerIndex, int[] request);

 public synchronized void releaseResources(int customerIndex, int[] release);
```

C:

```
void initBank(int *resources, int m, int n);

void setMaximumDemand(int customerIndex, int *maximumDemand);

int requestResources(int customerIndex, int *request);

void releaseResources(int customerIndex, int *release);
```

For Q1, you would not need to implement the `requestResources` to check for a safety state yet. For now, you can just let it return true, or 1.

Also, you can assume that the customerIndex will be in the valid range, and the array passed to `releaseResources` is valid.

To help you focus on implementing the algorithms, a function, `runFile`, has been provided to parse and run the test cases. For Q1, the schema of the test case is:

Test Case

To run the test case, compile your scripts first by typing `make`, and then you can pass the test files as the first argument into the program.

(for Java: java TestBankQ1 ./q1_1.txt)

(for C: ./q1 testcases/q1_1.txt)

This test case sets up the bank, initializes the maximum needs of the customers, and attempts to make a sequence of requests and releases. If we were to inspect the state of the bank, we will see that it goes through the following states.

After initialization:

```
Customers Allocation

      0  000
      1  000
      2  000
      3  000
      4  000

Maximum Need Available 000 000 1057 000 000

000 000

000 000 000 000
```

After initializing the maximum needs:

```
Customers Allocation

      0  000
      1  000
      2  000
      3  000
      4  000

Maximum Need Available 753 753 1057 322 322

902 902

222 222 433 433
```

Final state after the sequence of requests and releases:

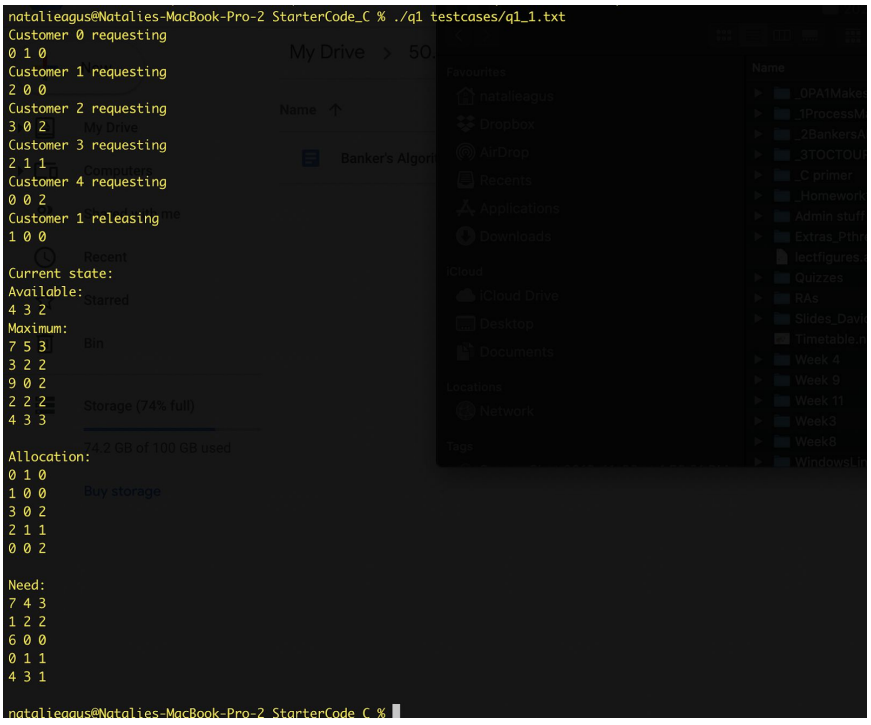
```
Customers Allocation Maximum Need Available

       0  010 753 743 432
       1  100 322 222
       2  302 902 600
       3  211 222 011
       4  002 433 431
```

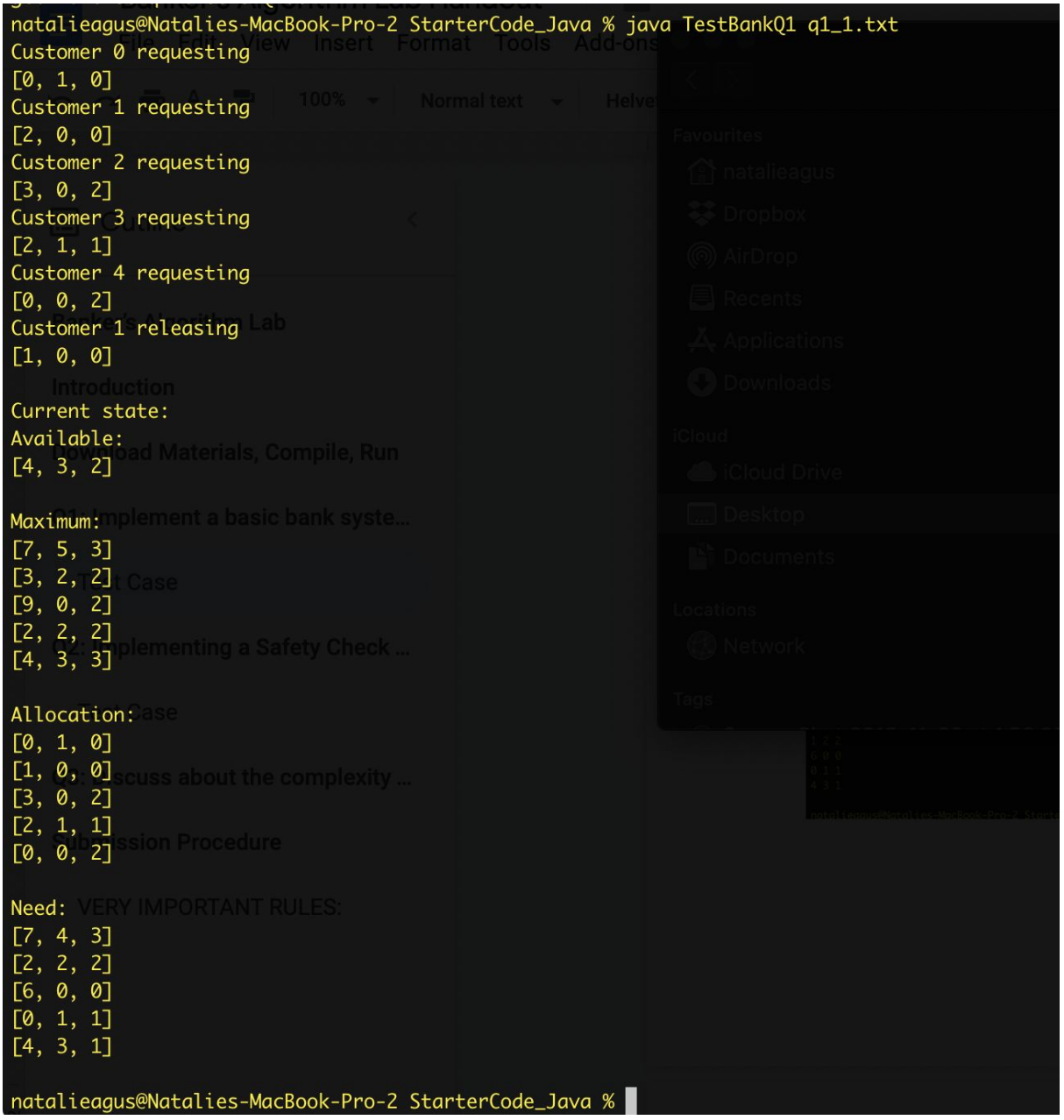
Hence, the expected output should be:

```
Customer 0 requesting 010
Customer 1 requesting 200
Customer 2 requesting 302
Customer 3 requesting 211
Customer 4 requesting 002
Customer 1 releasing 100
Current state: Available: 432
Maximum: 753 322 902 222 433
Allocation: 010
100
302
211 002
Need: 743 222 600 011 431
```

Here is the screenshot for the C-code:



Here is the screenshot for the Java code:



Q2: Implementing a Safety Check algorithm (25 marks)

Now, you will need to implement the checkSafe function.

Java:

```
private synchronized boolean checkSafe(int customerIndex, int[] request);
```

C:

```
int checkSafe(int customerIndex, int *request);
```

This function implements the safety algorithm for the Banker's algorithm.

The pseudocode is as follows:

```
boolean checkSafe(customerNumber, request)
{
    temp_avail = available - request;
    temp_need(customerNumber) = need - request;
    temp_allocation(customerNumber) = allocation + request;
    work = temp_avail;
    finish(all) = false;
    possible = true;
    while (possible)
    {
        possible = false;
        for (customer Ci = 1 : n)
        {
               // 1. perform appropriate checking
               // 2. set possible = true if you can find such i that fulfils (1)
                if (finish(Ci) == false && temp_need(Ci) <= work)
               {
                      possible = true;
                      work += temp_allocation(Ci);
                      finish(Ci) = true;
               }
        }
    }

    return (finish(all) == true);
}
```

Test Case

To run the test case, compile your scripts first by typing `make`, and then you can pass the test files as the first argument into the program.

(for Java: java TestBankQ2 ./q2_1.txt)

(for C: ./q2 testcases/q2_1.txt)

For this part, the schema of the test case is as follows:

For the very last request, the state of the bank is:

```
Customers Allocation Maximum

        0  010 753
        1  302 322 020
        2  302 902 600
        3  211 222 011
        4  002 433 431
```

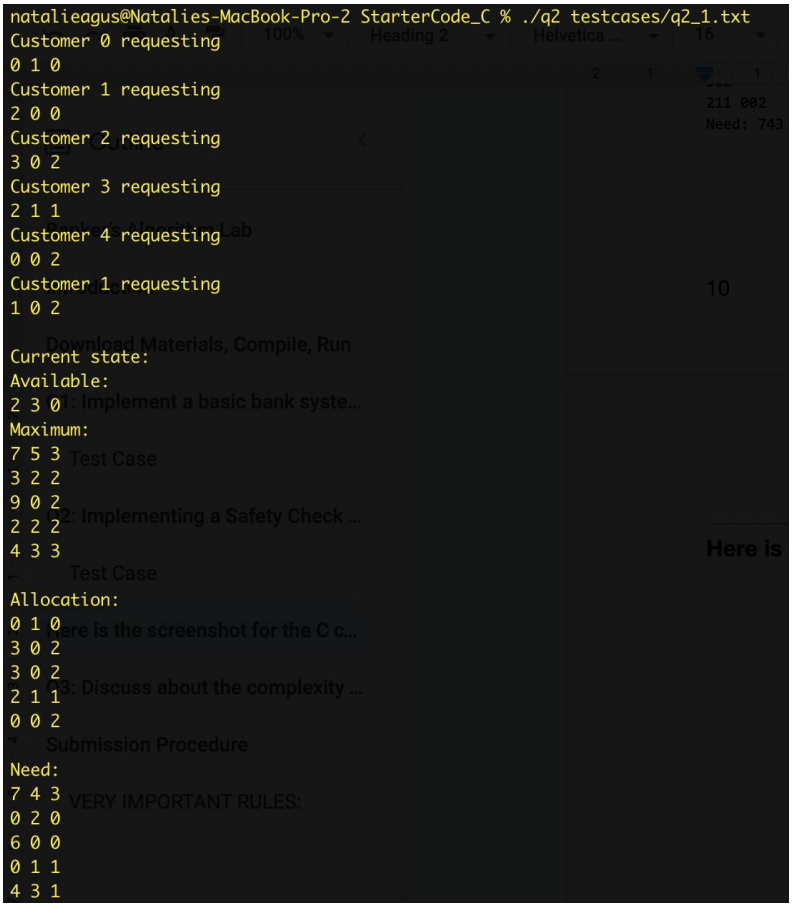
Although there is enough available resources to loan out "0 2 0" to customer 0, it will **leave the bank in an unsafe state.** Hence, this loan should not be approved and the bank state should **not change.**

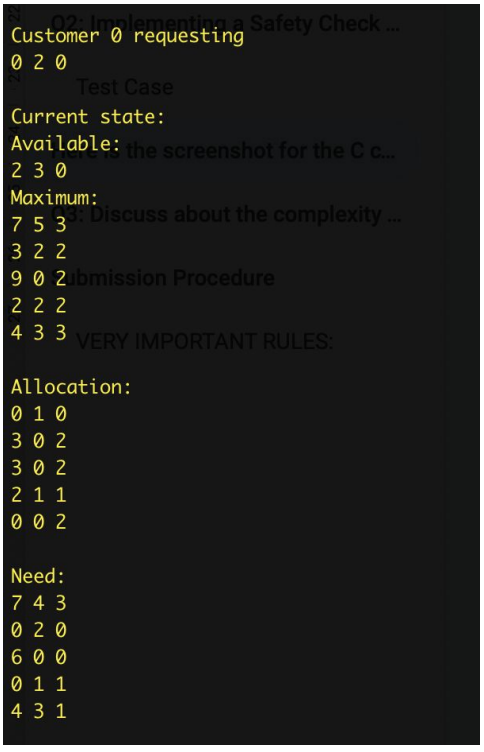
```
Need Available 743 230
```

The expected output of Q2 should be:

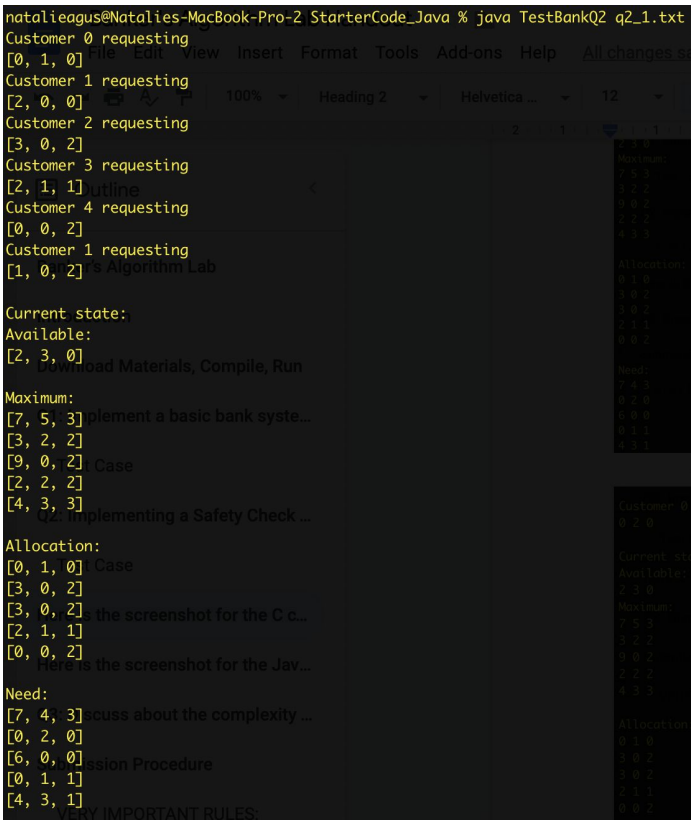
```
Customer 0 requesting 010
Customer 1 requesting 200
Customer 2 requesting 302
Customer 3 requesting 211
Customer 4 requesting 002
Customer 1 requesting 102
Current state: Available: 230
Maximum: 753 322 902 222 433
Allocation: 010
302
302
211 002
Need: 743 020 600 011 431
Customer 0 requesting 0 2 0
Current state: Available: 230
Maximum: 753 322 902 222 433
Allocation: 010
302
302
211 002
Need: 743 020 600 011 431
```

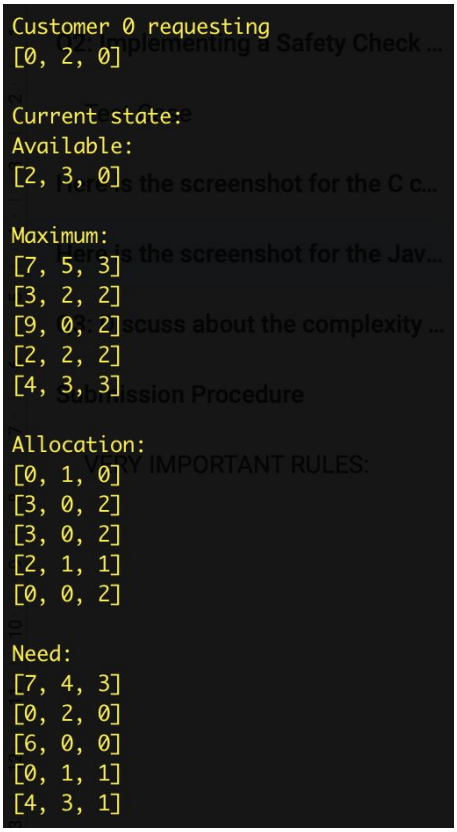
Here is the screenshot for the C code:





Here is the screenshot for the Java code:





Q3: Discuss about the complexity of Banker's algorithm (5 marks)

Write a short, 300-word report **in maximum** on how would you obtain the complexity of the complete banker's algorithm that you have implemented.

Submission Procedure

Put your result screenshots (Q1 and Q2, run it by yourself and screenshot the printed output) and Q3 analysis into a pdf file. Your submission should contain:

1. One pdf report;
2. One source code (Banker.java OR Banker.c, NOT both).
3. **NO OTHER SOURCE CODE IS ACCEPTABLE, make sure your code is COMPILABLE and test it using `make` and `make test` (see below)**

VERY IMPORTANT RULES

1. **As usual, DO NOT change any part of the given functions, not the names, not the arguments. ONLY *implement them* as instructed.**
2. **REMOVE any printouts that ARE NOT PART OF THE ORIGINAL PRINTOUTS AS SHOWN IN SCREENSHOT. If you are in doubt, test it with the checker file: `make test`**
3. **Part 2 is EXTREMELY important because it checks your answer against the answer key using string parsing.**

No other documents are accepted beside these two.

Please **zip them up** and submit before the deadline stated in the front page of this handout.