Banker's Algorithm Lab

50.005 Computer System Engineering

Due date: 4 Mar, 08:30 AM (Week 6)

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:

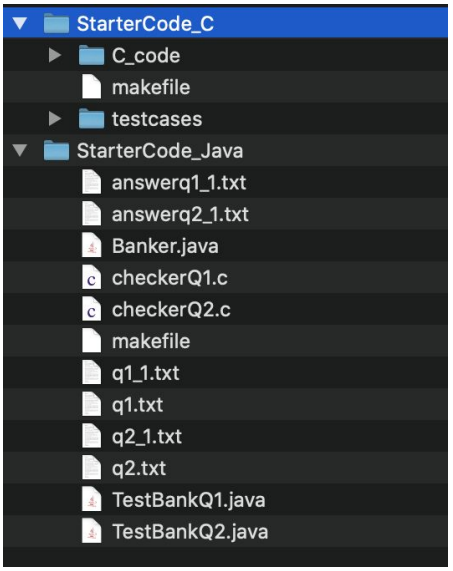
- **Q1:** Implementing a basic bank system (tested with q1_1.txt) **(20 marks)**.
- **Q2:** Implementing a safety check algorithm (tested with q2_1.txt) **(25 marks)**.
- **Q3**: Analysis of the complexity of the Banker's algorithm **(5 marks)**.

Download Materials, Compile, Run

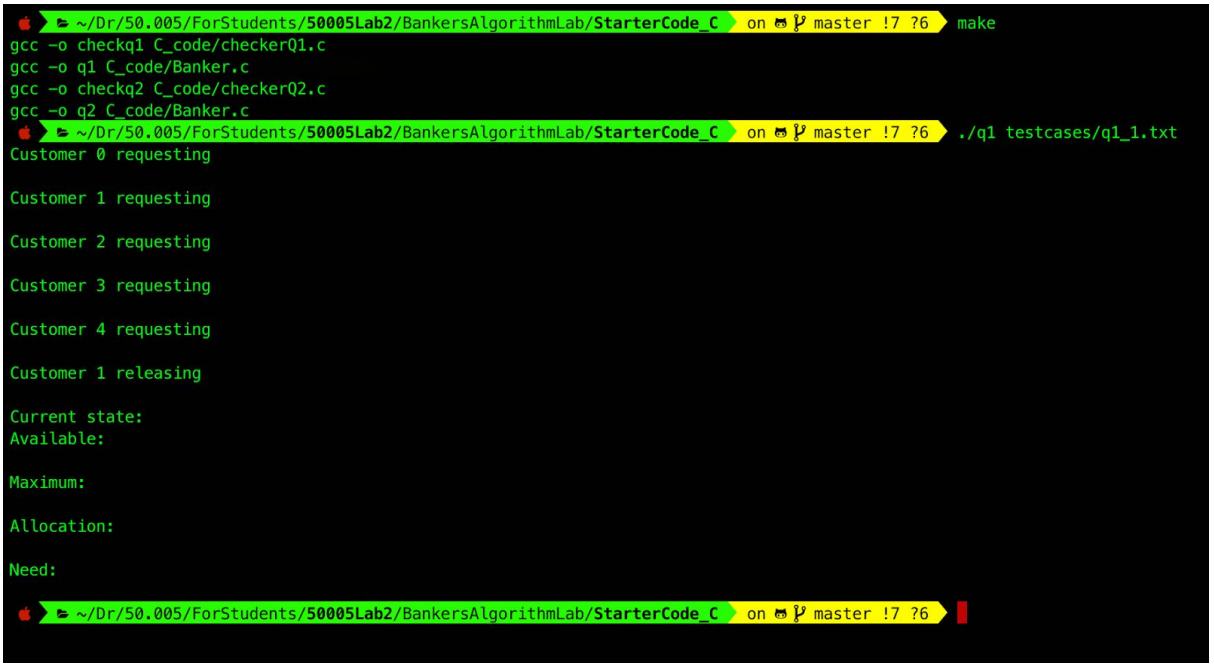
Clone the materials:

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

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

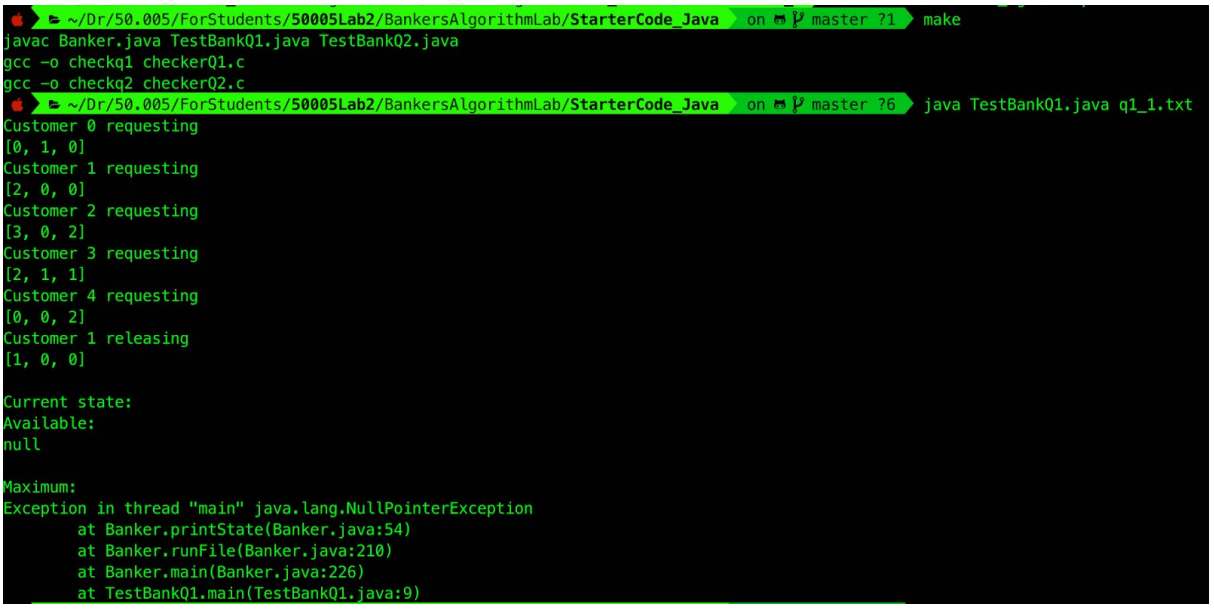


If you choose C, this is your output after make, and trying to run with given test cases (you might also be met with segfault, depending on your machine):



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):

Note: some systems might run Java as `java <compiled binary> <input args>` **without** .java extension in the binary.



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

Q1: Implement a basic bank system in Banker.java or c (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 (modify BankerQ1.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.

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`, or simply type `make testq1`)

(for C: `./q1 testcases/q1_1.txt`, or simply type `make testq1`)

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			Maximum			Need			Available		
0	0	0	0	0	0	0	0	0	0	10	5	7
1	0	0	0	0	0	0	0	0	0			
2	0	0	0	0	0	0	0	0	0			
3	0	0	0	0	0	0	0	0	0			
4	0	0	0	0	0	0	0	0	0			

After initializing the maximum needs:

Customers	Allocation			Maximum			Need			Available		
0	0	0	0	7	5	3	7	5	3	10	5	7
1	0	0	0	3	2	2	3	2	2			
2	0	0	0	9	0	2	9	0	2			
3	0	0	0	2	2	2	2	2	2			
4	0	0	0	4	3	3	4	3	3			

Final state after the sequence of requests and releases:

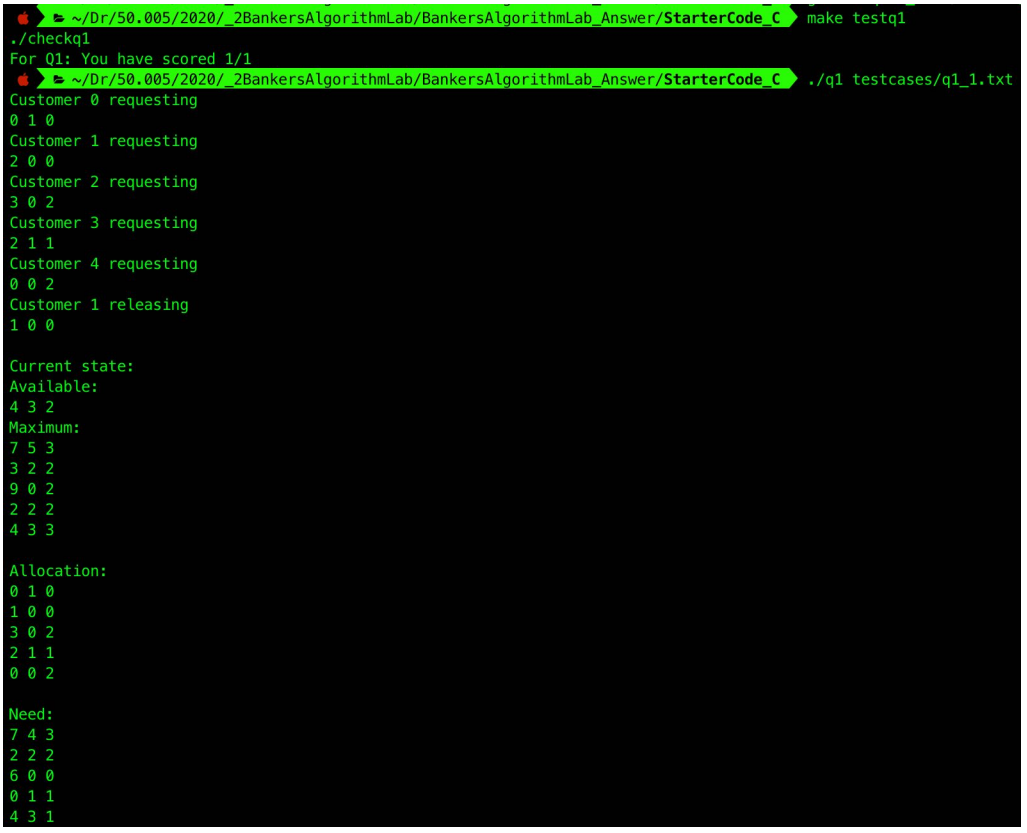
Customers	Allocation			Maximum			Need			Available		
0	0	1	0	7	5	3	7	4	3	4	3	2
1	1	0	0	3	2	2	2	2	2			
2	3	0	2	9	0	2	6	0	0			
3	2	1	1	2	2	2	0	1	1			
4	0	0	2	4	3	3	4	3	1			

Hence, the expected output should be showing these values:

```
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:



The Java counterpart should give similar output.

Q2: Implementing a Safety Check algorithm in Banker.java or c (25 marks)

Now, you will need to implement the checkSafe function.

Java:

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

C (paste over your implementation from BankerQ1.c into BankerQ2.c, and continue your implementation in BankerQ2.c):

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

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

You need to:

1. *Pretend* that the request is granted, make a **copy** of the available, need, and allocation matrix that serves this supposed request
2. Check whether it leads to the safe state
3. Returns True/False accordingly

Refer to the recorded lab session on *Deadlock Avoidance* or the summary notes to know how the algorithm works.

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, or simply type make testq2)

(for C: ./q2 testcases/q2_1.txt or simply type make testq2)

For this part, the state of the bank is as follows just before the very last request:

Customers	Allocation			Maximum			Need			Available		
0	0	1	0	7	5	3	7	4	3	2	3	0
1	3	0	2	3	2	2	0	2	0			
2	3	0	2	9	0	2	6	0	0			
3	2	1	1	2	2	2	0	1	1			
4	0	0	2	4	3	3	4	3	1			

Although there are 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.**

Hence, the expected output should be showing these values:

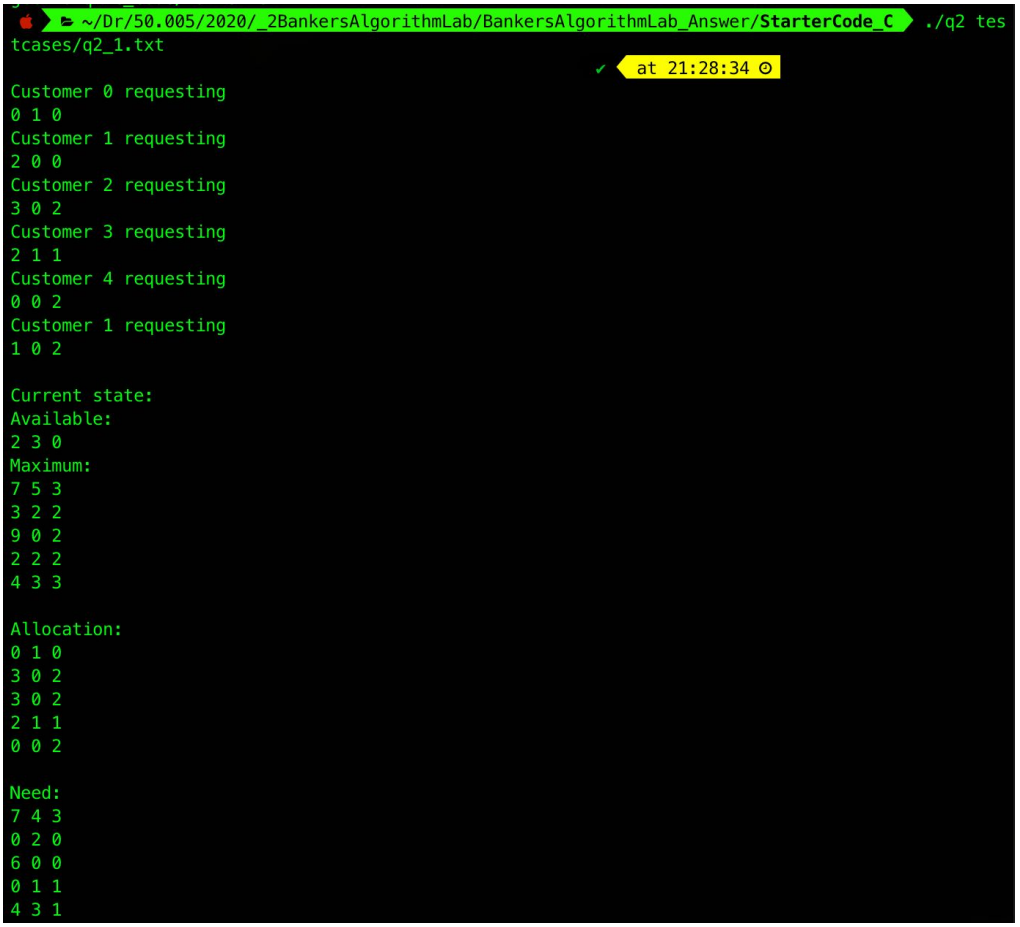
```
Customer 0 requesting 010
Customer 1 requesting 200
Customer 2 requesting 302
Customer 3 requesting 211
Customer 4 requesting 002
Customer 1 requesting 102

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 020

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 supposed output of the C code:



The Java counterpart should give similar output.

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

Write a short, 300-word report **in maximum** (no minimum, you can use screenshots if you want) on how you would obtain the **time** complexity of the complete banker's algorithm that you have implemented.

Submission Procedure

Put your result screenshots (Just Q2 -- or you can add Q1 too if you want but not necessary. 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, with THIS NAME). **If you complete Q2, then just submit your implementation of Q2. No need to submit Q1 anymore because Q2 is inclusive of Q1.**
3. **NO OTHER SOURCE CODE IS ACCEPTABLE, make sure your code is COMPILABLE and test it using `make`.**

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. If it doesn't work, open the answer text files given and just ensure they contain the same output.**

   This is **EXTREMELY** important because it checks your answer against the answer key using **string comparison.** If your system somehow doesn't pass the test case but you're **ABSOLUTELY** sure that the output of both your program and answer key is the same, then notify @natalieagus so we can check your submission manually.

No other documents are accepted beside these two.

Please **zip all (again ZIP, not RAR, or whatever compression algorithm you have)** and submit before the deadline stated in the front page of this handout:

- **Upload** to `@csesubmitbot` telegram bot using the command `/submitlab2`
- **CHECK** your submission by using the command `/checksubmission`