CSCI 4401/5401
Principles of Operating Systems
Fall Semester 2022
Assignment 4 (11/8)

Due Wednesday, November 30th @ 11:59pm

Submission Guidelines:
1. This assignment is worth 100 points for all students.

2. It is your responsibility to make sure all files are readable and submitted on time.

Submission:
- A single Java file named Deadlock.java

- Two output files named output01.txt and output02.txt

Introduction

There are a variety of approaches to deal with resource deadlocks. For this assignment you'll investigate deadlock detection. The approach to deadlock detection that we discussed in class is to model resource allocations and requests with a Resource Allocation Graph (RAG). Deadlock detection is then accomplished by finding a cycle in the graph.

First, you will need to create graph package. The operations (AddNode, AddEdge, RemoveEdge, etc.) will be required. You can start writing this as you become acquainted with the rest of the assignment. Remember that you will need directed edges to be supported by your graph package.

You will definitely want to think about this assignment a lot before you start writing code.

Details

Your task is to construct the RAG and detect any cycles that arise. Your program will track resource requests, allocations, and releases. The ordering of resource requests and releases will be provided as input to your program in the format given below. When a request is made for a resource, it will be allocated if it is available. Otherwise, the requesting process will be blocked until the resource is released by the process that holds it.

Your program should check for the presence of deadlock (a cycle) after every resource request or allocation. In a real system, this check would only be done periodically because it is too expensive to check after every request or allocation. However, your program is required to indicate the deadlock as soon as it occurs. Note that it is not necessary to check for deadlock after a resource release, because if deadlock didn't exist before the release then it won't exist afterward.

When your program detects a deadlock, it will report the deadlock along with the resources and processes involved in the cycle and then terminate. In a real system employing deadlock detection some attempt would be made to resolve the deadlock. In this case, however, we are only interested in detecting it. If your program performs all of the resource allocations and releases without encountering a deadlock, then it will report that no deadlock was encountered and terminate.

Input

Input to your program will consist of lines like the following, read from a file:

```
1    W    1
2    W    2
3    W    6
2    W    6
4    W    3
5    W    1
1    R    1
3    W    3
5    R    1
4    W    2
1    W    4
1    W    5
1    R    4
```

Each line of input contains an integer followed by 'W' or 'R' and then another integer. The first integer identifies the process, the second integer identifies the resource, and 'W' indicates "wants" or 'R' indicates "releases". So the first line indicates that process 1 wants resource 1. The last line indicates that process 1 releases resource 4. All process and resource identifiers are unique, so the process 1 that is referred to five times in the input is the same process. Also, **each resource has only one instance**. Hence the resource can be allocated to only one process at a time. If a process wants a resource that is already allocated, then it will have to wait until the resource is released by whichever process is holding it. When a resource is released, it should be allocated to the process that has been waiting for it the longest. If there are no processes waiting for it then the resource is now free. Note that all input will be consistent. That is, I will never specify in the input file that process 7 Releases resource 3 if process 7 didn't hold resource 3.

Output

Your program will write information to standard output that tracks the requests, allocations, and releases. The following format is required:

Process 1 wants resource 1 – Resource 1 is allocated to process 1.
Process 2 wants resource 2 – Resource 2 is allocated to process 2.
Process 3 wants resource 6 – Resource 6 is allocated to process 3.
Process 2 wants resource 6 – Process 2 must wait.
Process 4 wants resource 3 – Resource 3 is allocated to process 4.
Process 5 wants resource 1 – Process 5 must wait.
Process 1 releases resource 1 – Resource 1 is allocated to process 5.
Process 3 wants resource 3 – Process 3 must wait.
Process 5 releases resource 1 – Resource 1 is now free.
Process 4 wants resource 2 – Process 4 must wait.
DEADLOCK DETECTED: Processes 2, 3, 4, and Resources 2, 3, 6, are found in a cycle.

If your program processes all of the input and finds no deadlock, then it should display the line: "**EXECUTION COMPLETED: No deadlock encountered.**" as the last line of the output.

Logistics

You should not assume any bound on the number of processes or resources. The integer identifier can range freely, and you should expect larger numbers to test your program. That is, don't read the identifier as if it's a character, read it in as an integer (or a string if you want to be more general and allow names for your processes and resources) so your program handles multiple digit identifiers.

Submission/Grading

You will submit all your source code in a single java* file called Deadlock.java. You may develop your code using separate files (e.g. Deadlock.java, RAG.java, etc.) but you will need to concatenate your source code together into a single compilable file for submission. The file Deadlock.java should begin with a public class Deadlock that contains the main program which takes the input file name as a command line argument and then performs the simulation of the resource allocations in the input file. The public class Deadlock should be followed by your other classes that comprise your system. Recall that Java will only allow one class within a source file to be declared public, so your Deadlock.java source file will look something like this:

```
public class Deadlock {
        public static void main( String [] args ) {
                … here lies code to process the input file and simulate resource allocations …
        }
}

class RAG {
        … here lies code to implement your Resource Allocation Graph data structure class …
}

class Node {
        … here lies the code for a node object, …
}

etc.
```

Run your program and use input01.txt and input02.txt:

```
$ java Deadlock input01.txt

$ java Deadlock input02.txt
```

Save the output in output01.txt and output02.txt. Put all these files (i.e., code related files plus the input and output files) in a folder, compress, and upload via Moodle under *Assignment # 4*.