Lab 8: Graphs

SE2205: Data Structures and Algorithms using Java – Winter 2015

Discussion: March 26th **Due**: Sunday April 5th

Modified and reviewed by Alexandra L'Heureux (alheure2@uwo.ca) and Vasundhara Sharma (vvasundh@uwo.ca) , for more information please send an email to Alex (TA, SE2205) or Vasu (TA.SE2205)

A. Rationale and Background

In this lab we will be discussing the ADT Graph. You will be responsible for writing a small applications which will handle the creation and the traversal of a graph. In this lab, we will mimic a project management program with its main purpose of finding the critical tasks in a task network. You shall utilize your engineering skills to design the proper data structures and algorithm to solve the problem at hand.

B. Evaluation

You will get credit for your lab when you demonstrate it and get your TA's approval. Submitting your lab work online is required; failure to do so will result in a grade of 0%. Additionally, your TA will ask some questions about your coding during demonstration and you need to articulate your idea clearly. In place of questions, the TA may also ask you to modify your program in a small way - it is expected that you have sufficient understanding of the code to do so.

Your mark will be determined as follows:
Completion: Program Demonstration of Base Requirements (50%)
Understanding: Program Demonstration of Changes or Response to Questions (50%)

Please note that you may only demonstrate the code you have previously submitted to OWL.
Please ensure that the file naming conventions are being followed. You should be able to rename your classes once they are done by right clicking on them, selecting *refactor* and *rename*.

1. Submission Instructions

Number of files : 3
Files to be submitted : All Classes required
Rename these files as : your_uwo_user_name_lab08_Task.java,
your_uwo_user_name_lab08_Main.java
your_uwo_user_name_lab08_TaskNetwork.java

C. Introduction

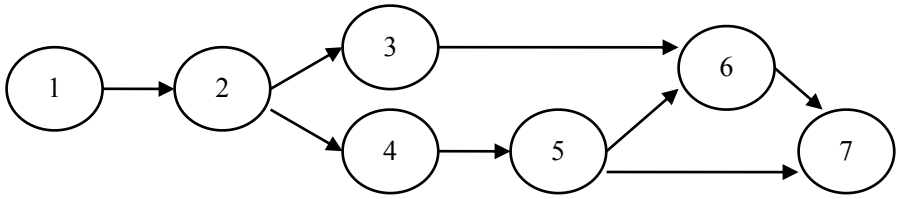
The goal of this lab is to find critical tasks in a task network.

In project management, one of the most important concerns is the delay of the project. As a manager,

you need to try your best to ensure that the project is finished according to the plan. Any delay will result in an increase of cost which is not acceptable.

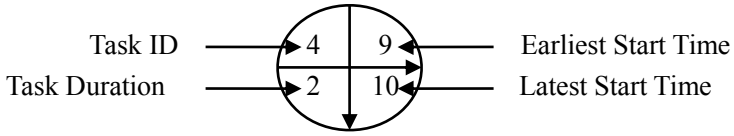
Task network is a very useful tool for project management which can help you focus on the critical tasks.

Usually a large project can be divided into multiple smaller tasks. Each task has a unique identifier (like 1, 2, 3 or A, B, C) and an execution duration. A project usually has a single starting point (the starting task) and a single ending point (the ending task). The tasks may be executed in parallel or in sequence. Following is an example of a task network:



In this figure, each circle represents a task (identified by a unique number). Task 1 is the starting point and task 7 is the ending point. An arrow from task 1 to task 2 means that task 2 is the prerequisite of task 1, that is, task 2 can start only after task 1 is successfully finished. Task 3 and task 4 can execute simultaneously after task 2 is finished. Task 4 and task 5 must be executed in a sequential order. This figure reflects a graph structure.

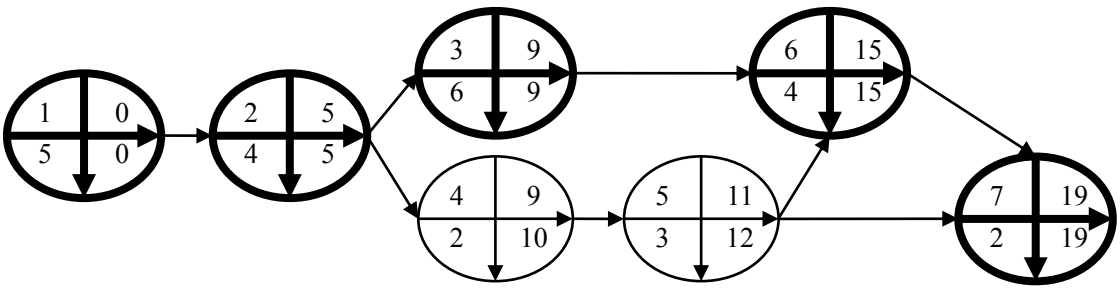
An important goal in a project management is to find the critical tasks. A critical task is a task whose delay in execution will result in a delay of the entire project execution, and therefore it cannot be delayed. A non-critical task can delay for some time without causing any delay in the project execution. To find the critical tasks in a task network, we need to calculate the earliest start time and the latest start time for each task by considering the task sequences and the corresponding duration of each task. To do this calculation, we need to represent each task as shown below:



In this representation, a task is specified by its ID, duration, earliest start time, and latest start time. Note that the ID and duration of each task are given by the user (the one who designs the task network), but the earliest start time and latest start time need to be calculated.

Note: in this lab we use a simple time system which only considers pure integer numbers instead of the actual calendar time system. For example, assuming that we have two tasks, task 1 followed by task 2, if task 1 starts at time 0 and its duration is 5 time units, then task 2 can start at time 5 since task 1 will finish at that time.

Assuming that we have finished the calculation, the above network can be elaborated to a new one:



The earliest start time and latest start time of the first task (task 1 in the example) are both 0. Since task 1 needs 5 time units to run, task 2 can start at as early as time 5. After task 2 spends 4 time units, task 3 and task 4 can start at time 9. Similarly, we know that task 5 can start at time 11. Looking at task 5 and task 6, it seems that task 6 can start at time 14, however we cannot forget task 3. Since task 6 can only start after both task 3 and task 5 are done, it can be concluded that the earliest start time should be the larger one of the earliest finish times of task 3 and task 5, which is 15. By repeating this calculation we know that task 7 should start at time 19.

To calculate the latest start times, we need to start with the ending task: task 7. The earliest start time and latest start time are also the same for the ending task. Therefore, the latest start time of task 7 is set as 19. Since task 7 must start at time 19 and task 6 needs 4 time units, task 6 cannot start later than time 15. Considering task 7 and 5, it seems that task 5 can start at as late as time 16, but we have to note that task 5 must finish before task 6, therefore the latest start time of task 5 should be 12.

After we calculate the earliest start times and latest start times of the tasks, we can see that task 1, 2, 3, 6, and 7 are critical since each of them has the same earliest start time and latest start time, which means that any delay of such a task will delay the project. They are highlighted by double-lined circles. Task 4 and 5 are non-critical. For example, task 4 can take a rest during time 9 and 10 without delaying the project (surely, it cannot start later than time 10).

As a project manager, you will not delay your project if you know which tasks are critical and pay more attention to them. You will be awarded bonus whereas other managers are experiencing annoying project delay.

D. Lab Questions

1. Design your Data Structures

In this part, you are required to think about the data structure of the tasks and a task network.

- Design a java class for a task.
- Design a java class for a task network. Note that in a task network there are multiple tasks.

Note: You need to consider how to represent the links between the tasks in a task network. You can design this structure in the task network class. If you think you need an individual class for the links, it is fine as long as your design is reasonable.

You may use basic java utilities such as LinkedList, Arrays and Iterators as you seem fit. However, you are not to use any Graph classes that may be available.

2. Designing the Algorithm

In this part, you are required to design and implement the algorithm to find the critical tasks in a task network (also known as “planning a project”). This can be done in a method in the task network class. The basic processes are: 1) calculate the earliest start times and latest start times of all the tasks, 2) check the tasks one by one and mark the critical ones.

Hint: when calculating earliest start time, we start from the starting task and traverse the network forward. When calculating latest start time, we start from the ending task and traverse the network backward. You may want to consider how the different traversal check for different critical conditions.

Note: two principles apply: (i) both earliest start time and latest start time of the starting task are 0; (ii) earliest start time and latest start time of the ending task are the same.

3. Testing Client

Write a Java client to test the algorithm. The client should be able to construct a task network, plan a task network, and print the critical tasks. Following are the details:

(1) Ask the user to input the number of tasks.

(2) Ask the user to input duration and task sequences. The sample screen output is (the tasks are indentified by numbers):

```
Please input duration for task 1: 5 (enter)
…
…
      Please input duration for task 7: 2 (enter)



      Please input link 1: 1   2 (enter)

      Please input link 2: 2   3 (enter)
…
…
      Please input link 8: 6   7 (enter)

     Please input link 9: 0   0 (enter)  // 0 0 means the previous link is the last one
```

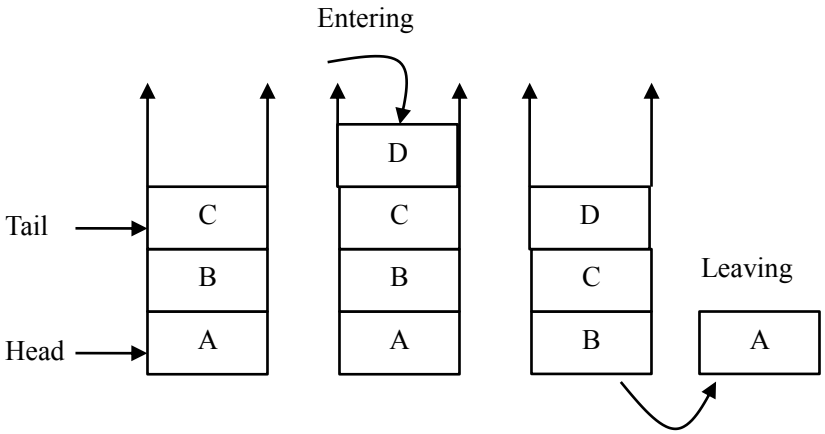
(3) Print result on the screen. The printed information should be in the following format (for each task):

```
  Task ID: 1; Duration: 5; Earliest Start Time: 0; Latest Start Time: 0; It is a critical task
or
    Task ID: 4; Duration: 2; Earliest Start Time: 9; Latest Start Time: 10; It is not a critical task
…
```

4. Hints

We need a data space to keep the intermediate results when we are exploring a network. For instance, when you are processing task 2, you need to keep its subsequent tasks (task 3 and task 4) to calculate their earliest start times; when you are processing task 7, you need to keep its preceding tasks (task 5 and task 6) to calculate their latest start times.

We can use a queue structure to do this. We keep two important positions for the queue: head and tail. It follows a "first come first leave" principle: any new element that needs to enter the queue is always appended to the tail, and when we need to get an element from the queue we always get one from the head.



For example, when you are working on task 2, you can find all its subsequent tasks, say, task 3 and 4, and put them in the array. After you finish task 2, you can continue to work on task 3 (of course, when working on task 3 all its subsequent tasks should also be put in the array) and task 4 one by one. This process repeats until you are working on a specific task which has no subsequent task—it is the ending task of the project.

E. Learning Checkpoint

This lab should have introduced you to these concepts of Java:

1. Introduction to Graph ADT.
2. Design of an ADT.
3. Graph Traversals.