Mandatory assignment 2

INF 2220, Fall 2011

Due: 20.10.2011

Updated Date: 10.10.2011

In this assignment, you are going to develop a project planning tool. A *project* usually consists of multiple *tasks*. Each task in a project takes some estimated time and certain manpower to complete. After a task is completed, the resources (manpower) will be released and accessible to other tasks. In order not to delay the project, each task in the project should start as soon as possible. While some tasks can start as soon as the project starts, the rest of the tasks have to wait for one or more other tasks to complete before they can start.

Figure 1 gives an example of a project. In this figure, a project with eight tasks is depicted as a directed graph. Each task is represented as a node. Each node contains a task's unique identity, name, time estimate and manpower requirements. The dependency between two tasks is represented as a directed edge.

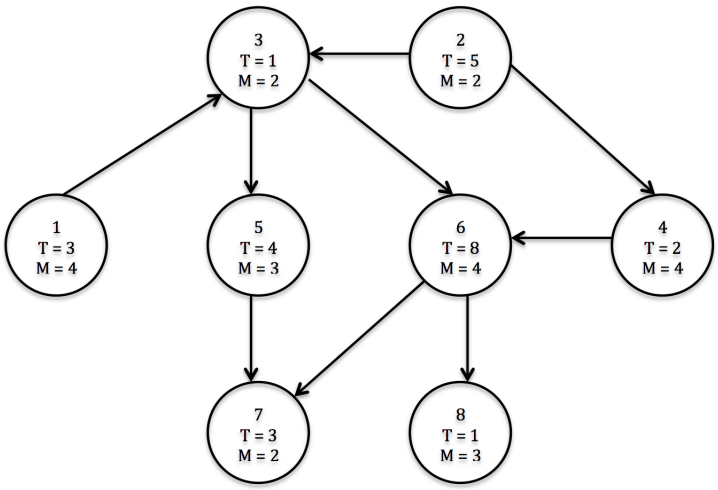

Figure 1: An example directed graph of a project

From this figure, we can see that task 1 and task 2 can start as soon as the project starts. The rest of the tasks will only start until all of its predecessors are finished. For example, task 3 has to wait until both task 1 and task 2 are completed before it starts.

Sometimes, tasks that others depend on can be *delayed* some estimated time because other dependencies will take longer to finish. For instance, from figure 1, we see that task 1

can wait for 2 units of time without delaying the project because task 3 still has to wait for 2. That is, task 1 can start anytime between `Time = 0` and `Time = 2` (`Time` is referring to the project time). Tasks which can not be delayed are considered to be *critical* for the project to complete on time.

A project is completed when *all of the comprising tasks are finished*. Therefore, a task can not be a predecessor, as well as a sucessor, of another task in a project. Otherwise, the project is not possible to complete because two tasks are waiting for each other to finish. The project in figure 1 can be completed because no task is found as the predecessor of another other task as well as a successor of the same task in the graph.

Your tasks

In this assignment, you have to implement this project planning tool in JAVA. You are asked to use a *graph* as the data structure for the implementation. The complexity of your implementation should be *better* than $O(N^2)$.

Given a project, your program should be able to answer the following questions.

1. Can the project be completed? Is there any circular dependency where tasks wait for each other? (task 1 waits for 2 to finish while task 2 waits for 1 to finish)
2. If it is possible to finish the project, how can the project be accomplished in the shortest possible amount of time, given that we have infinite manpower. How long will the whole project take to complete? How much manpower will be used at any given time? (See page 3-4 for an example of how the answers to these questions should be printed out.)
3. Which tasks are critical in terms of completing within their expected deadline, for the whole project not to be delayed?

   Extra task
4. If manpower is limited, how can this project be accomplished as fast as possible?

Guidelines for the implementation

Reading input files and constructing the graph

A project is described with an input file. At the top of this file is the total number of tasks comprising the project. The data format of the input file will be a list of task definitions, where each of them is a sequence of the following data:

- Identity of this task (integer)
- Name of this task (a string)
- Time estimate for this task (integer)
- Manpower requirements (integer)
- A sequence of identities that states which tasks must be completed before this task can start (its dependencies). This list is terminated by a 0.

To represent the graph internally we can use the following class:

```
class Task {
    int      id, time, staff;
    String   name;
    int      earliestStart, latestStart;
    Edge     outEdges;
    int      cntPredecessors;

    Task(int n) {
        number = n;
    }
}
```

Note that this is just a skeleton of the classes. Additional fields may be needed according to your implementation.

Each task should be an object of the class "`Task`". After the input file has been read, it should contain the right values for: id, name, time, staff and cntPredecessors. You can take it for granted that all task numbers lie in the interval `1` to `maxnr` where `maxnr`, which is the total number of tasks comprising a project, is given at the top of the input file.

For each task, there should be a list of `Edge`-objects which represents the outgoing edges from this task (i.e., the tasks that cannot start up before this task is finished). Order is of no importance in this list.

As the input file is read, these data structures are constructed along the way. One has to remember to instantiate a new `Task`-object the first time a task is mentioned in the input file, which can very well be in a predecessor-list, before the definition of the task. Just leave the data-fields of this `task`-object blank until its definition occurs.

Handling circular dependencies

Question 1 requires checking if any circular dependency exists in the graph of a project or not. If any such circular dependency is detected, we should print out the path in which the dependency exists. Then, the program should exit and tell the user that this project can not be completed because of the dependency detected. This should be checked as soon as the graph is constructed.

Start tasks as soon as possible

If no circular dependency is detected, we should proceed to investigate question 2. One should start up every task as soon as possible, that is, after all tasks it depends upon are completed. Tasks without dependency should be started right away. During the simulated project execution, feedback should be provided from your system by printing out important information. This will typically be when tasks start up, and finish. Your system should also print out current working staff at these moments in time. For the project illustrated in figure 1, feedback from the system should be something as follows:

```
Time: 0     Starting: 1
            Starting: 2
       Current staff: 6

Time: 3     Finished: 1
       Current staff: 2
```

```
Time: 5
            Finished: 2
            Starting: 3
            Starting: 4
       Current staff: 6

Time: 6     Finished: 3
            Starting: 5
       Current staff: 7

Time: 7     Finished: 4
            Starting: 6
       Current staff: 7

Time: 10    Finished: 5
       Current staff: 4

Time: 15    Finished: 6
            Starting: 7
            Starting: 8
       Current staff: 5

Time: 16    Finished: 8
       Current staff: 2

Time: 18    Finished: 7

**** Shortest possible project execution is 18 ****
```

Finding critical tasks

To find out whether or not a task is critical for a project to complete on time, we have to check if there is any "*slack*" for the task or not, that is, whether there is certain amount of time that the completion of the task can be delayed without delaying the whole project.

To achieve that, we have to know when a task can be started at the *earliest*, and when it has to be started for at the *latest*. The difference between these two time points is the *slack* of a task. A task that does not have any room for delay, i.e., a task that does not have slack, is considered to be *critical*. Your program should show the user a list of all the tasks (sorted by their identifying number) which includes these properties:

- Identity number
- Name
- Time needed to finish the task
- Manpower required to complete the task
- Slack
- Latest starting time
- A list of tasks (identities) which depend on this task

Deliver to

The assignment should be carried out individually and delivered to the teaching assistant responsible through `https://devilry.ifi.uio.no/`.

How to deliver

- You have to implement the assignment in JAVA.
- Your implementation should compile on the LINUX machines in the University, and run either with the command: `java Oblig2 projectName.txt manpower` , or you have to specify in the `README` file exactly what commands should be used to compile and run the program.
- In this assignment your program should not expect any user interaction while running. It should take its input from the command line (file name and manpower argument), and print out the answers to the assignment questions on the command window.
- Note that the value of the parameter `manpower` would be `999` for the first 3 questions, which should be considered as indicating unlimited manpower. If you want to try the extra question, you can use `8` as the value of `manpower` for project `buildhouse1` and `buildhouse2`, and `100` for project `buildrail`. The corresponding input file of the mentioned projects can be found from the course webpage.
- The delivery must only be in one file: either a `.tgz` or `.zip` archive (with one of those two file extensions). Follow the steps below to genenrate the archive:

  1. `cp AssignmentFolder myusername`
  2. `find myusername -name *.class -delete`
  3. `tar cvzf myusername.tgz myusername`

  That is, first you copy the contents of `AssignmentFolder` to a folder with the name `myusername`, then delete all compiled files from the `myusername` folder and pack the folder in a gzipped tar. Of course, you have to create the directory `myusername` before step 1 from above. For a `zip`-archive, replace the `tar`-command in step 3 with: `zip -r myusername.zip myusername`
- Your archive should contain
  - The source file(s) of your implementation.
  - A report in which you should state the complexity of your implementation and justify the stated complexity. If you have different complexity for each question, discuss them separately.
  - A file named `output.txt` in which you put the feedback (print out) from system during execution of project `buildhouse1`, `buildhouse2` and `buildrail`.
  - A file named `README` in which you should describe any peculiarities of your solution, for instance, things that may be missing, or assumptions made, or questions for the group teacher. If everything runs fine, put it down in the `README` file.

Good luck!