Universitetet i Oslo
Institutt for Informatikk
I. Yu, D. Karabeg

INF2220: algorithms and data structures

Mandatory assignment 2

Issued: 24. 09. 2015

Due: 15. 10. 2015

1 General description: project planner

In this assignment, you are going to develop a project planning tool. A *project* consists of *tasks*, each taking an estimated *time* and certain *manpower* to complete.

Figure 1 shows an example project, where a project with eight tasks is depicted as a directed graph. Each task is represented as a node with a unique identity, time estimate (T), and manpower requirements (M). The dependency between two tasks is represented as a directed edge.

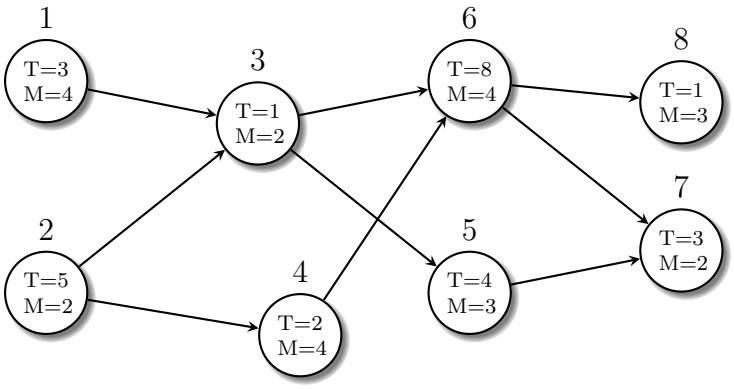

Figure 1: A sample directed graph of a project

2 Problems

Your implementation of the project planner should solve the following problems:

Realizability

Is the project realizable? Considering only the task dependencies (i.e., ignoring manpower and time information). A project is unrealizable if the dependency graph includes one or more cycles (it is not possible to realize the project if task 1 has to be completed before task 2 can be started and task 1 cannot be started until task 2 is completed).

If the project is not realizable your program should find a cycle and print it before it finally terminates. A cycle comprises of all the nodes in the path with the start and finish being the same node. All of these nodes must be printed.

NOTE: Your program does not have to find all cycles if there are more than one, just one of them.

Optimal time schedule

If the project is realizable the program should plan an ideal time schedule, so that the project is finished as early as possible. The plan should be given by describing the earliest start and finishing time of each task.

In this assignment we will assume to have infinite manpower. This means that a task can be started as soon as all of its dependencies are finished. For example, in Figure 1 task 6 cannot be started until task 3 and 4 have been completed. This also means that multiple tasks might have to be executed simultaneously in an ideal plan.

If we don't have infinite manpower, the problem is more complicated. This is given as an optional assignment (which you can find here `http://www.uio.no/studier/emner/matnat/ifi/INF2220/h15/obligatoriske-oppgaver/oblig-2/oblig2-optional.pdf`), for those of you who want a little more challenge.

NOTE: This does not mean that this is parallel programming. This assignment does not include threads. You should use one or more graph algorithms to calculate these times.

Latest start, slack and critical tasks

Because a task can depend on multiple tasks, some tasks can be *delayed* without delaying the *overall* project completion time.

In Figure 1 task 1 can be delayed at least 2 time units as task 3 also must wait for task 2 which takes longer time than task 1. But task 4 might also be delayed because task 5 can be delayed as task 6 takes very long time increasing the maximum time task 1 can be delayed. When a task can be delayed we say that it has positive *slack* (ie. task 1 has 3 slack).

Task 2, 4, 6 and 7 may not be delayed though as they represent the longest path (in time steps) through the graph. Tasks that may not be delayed *critical* and have 0 *slack*.

Your program should calculate the latest possible finishing time for a task without delaying the overall project completion time as well as slack and whether a task is critical.

3 What should be printed?

The program should print a simulation of the execution of the project. I.e the time schedule including the starting times, finishing times and the manpower used at all times.

Listing 1: Suggested output

```
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 ****
```

It should also print a list of all tasks with the following information:

- Identity number
- Name
- Time needed to finish the task
- Manpower required to complete the task
- Earliest starting time
- Latest starting time
- Slack

- A list of tasks (identities) which depend on this task

4 Guidelines & hints for the implementation

Reading input files and constructing the graph

A project is given by an input file. At the top of this file is the total number of tasks in 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
dependency edges	A sequence of identities that states which tasks must be completed before this task can start (its predecessors). This list is terminated by a 0.

NOTE: Task ID's are always in the range 1 to `num_tasks`.

To represent the tasks in a graph you can use the following class:

```
class Task {
    int          id, time, staff;
    String       name;
    int          earliestStart, latestStart;
    List<Task>   outEdges;
    int          cntPredecessors;
}
```

5 How to deliver

The assignment should be carried out individually and delivered through `https://devilry.ifi.uio.no/`.

- The implementation language is JAVA.
- Your implementation must compile on the LINUX machines at the University and run either with the command:

  ```
  java Oblig2 <projectName>.txt manpower
  ```

  where first argument of the `Oblig2` program is input file specifying the project, and the second argument `manpower` is an integer.

- Your delivery should contain
  - Compilable (and afterwards runnable) source file(s) of your implementation.
  - A file named `report.txt` in which you should state the complexity of you implementation and justify the stated complexity. If you havecdifferent 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 `README`-file which contains:
  1. How to compile your program (ie. javac *.java)
  2. Which file includes the main-method
  3. Any assumptions you have made when implementing the assignment
  4. Any peculiarities about your implementation
  5. The status of your delivery (what works and what does not)
  6. Give credit if your code is heavily influenced by some source (ie. teaching material)

Good luck!