Program 5: Prerequisite Graphs

Owen Kaser

November 20, 2015

Due: 8 December, end of class. Accepted late with 10% penalty between then and 11:59pm that evening. Not accepted after that.

1 Introduction

The prerequisite relationship between courses forms a directed graph without cycles, as noted in the textbook. In this program, you will read information that lets you construct a prerequisite graph that is somewhat related to UNBSJ courses, and then you will use it to solve three somewhat useful problems, viz.:

• Determine the entire set of courses that one must take to be able to have completed a specified set of courses. (Eg, Fred really wants to take Math 1013 and CS1083. But to complete them, he will have to have completed CS1073, CS1083, Math1003 and Math 1013.)

• Determine whether a given list of courses can be taken in the order specified, one at a time, without breaking any prerequisite rules. (Eg, Fred lists CS1083, CS1073, Math 1003, Math1013 and the answer is “no”. If Fred were to swap the order of the two CS courses, the answer would be “yes”. Or if Fred listed just CS1083, the answer would be “no”.)

• Given
  1. a list of courses that Fred wants to complete,
  2. the maximum number of courses that Fred is willing to take during the term,

you produce a schedule for Fred that

  1. obeys the prerequisites,

  2. does not make Fred take more than the maximum number of courses during any term and

  3. has Fred finish in not-too-many terms more than the best possible schedule. (Ie, if Fred could take 5 courses per semester, don’t produce a 15 year schedule where he takes one course each term, even though more could be taken.)
Call your program Prog5. It does not need to have a main() method, as you will produce methods called question1, question2 and question3.

There is an interface (with the imaginative name of Prog5Interface) for the three questions you need to answer. Do not modify it. If you skip a part, just make a dummy method in your Prog5, so that your program can compile with the original Prog5Interface.

There are also some JUnit tests in Prog5Test.java. (commented out, which you need to enable).

Since the end of semester is a crazy time, the first (easiest) activities are worth the most. The final question is partly a bonus.

## 2 Details

Your program should build its graph(s) by reading from the URL http://pizza.unbsj.ca/~owen/backup/courses/2383-2015/courses.txt

The format of this data is as follows: Each line starts with the name of a course. Call this course C. Next, after some whitespace, we have the name of the first prerequisite for C (or the value ‘none’). If there are several prerequisites, they appear (after some whitespace). The final entry on a line is a one-letter code that indicates when course C is offered. We won’t use this information for this program.

Your graph should be represented using a net.datastructures.Graph<Course, Void>. (Vertices are associated with Strings (the course names), but edges are associated with ... what?? Actually, we don’t have any meaningful information to store along with edges, but there is a type parameter for the edge information, and we need to fill it in with something, to keep the compiler happy. Type Void is just the ticket. The only value of this type is null.)

### 2.1 First Question

To answer the first question, you need to list the entire collection of courses that you’d have to take, before getting credit for completing a desired set of courses.

You will need to do some sort of (directed) graph traversal, that proceeds from a course to its prerequisites, to their prerequisites, and so forth.

Your method question1 will produce a java.util.Set with the names of the courses.
2.2 Second Question

The second question is related to the topological-sorting problem in Section 14.5.1, but is perhaps easier. I do not think you will be able to borrow much code from Fragment 14.11, but the overall idea can be borrowed: maintain a count of the in-degree of each node, and reducing these indegrees when courses have been successfully taken. (Basically, working on paper, you’d cross off the prerequisites left before you are able to take CS4567. Any time after all prerequisites have been crossed off (the in-degree has reached zero), CS4567 can be appear in the proposed schedule.)

Your method question2 will return a boolean.

2.3 Third Question

The third question is hardest, and should not be started until you have solved the first problem. (Fred may just list a single course, such as CS1083, as his desired set of courses. However, the schedule will have to show him taking CS1073 also. Thus, you need to essentially use the solution to the first question as part of the solution to this third question.) A reasonable approach to scheduling might be to modify the topologicalSort method. Assuming that Fred is willing to take up to 3 courses in a semester, you would modify line 15. Rather than looking for 1 course, you would see whether you can find 3 courses that are left to be taken, whose prerequisites are all met. If you find 3 or fewer such courses, you choose them all. If you find 4 or more, you arbitrarily choose 3 of them to be taken now.

Some arbitrary choices are better than others, so the schedule that you get will probably not be optimal. You can worry about how you might do better than this, when you learn advanced techniques in CS3913.

If you modify topologicalSort in the textbook, note that ready should probably be something other than a stack.

Given a java.util.Set<String> containing the names of the courses that Fred definitely wants, your method question3 will produce a net.datastructures.List<ScheduleEntry> where ScheduleEntry is a small provided class that has the name of a course and an integer representing the semester in which Fred is to take the course. (Any course that Fred takes in the winter of his second year should have semester=4.)

Simplifications: If a course C has several prerequisites listed, we require that all of them must be taken before C is taken. While this is very common, UNB has other sorts of prerequisites that we cannot model. For instance, if one of a set of alternative courses must be taken, we cannot handle this. We do not handle co-requisites. Nor do we handle
prerequisites such as “70ch completed toward BScCS”. These would have to be handled, before your assignment could be a useful tool at UNB.

3 Hints

Remember that a JAR file is just zipped archive. Inside of the JAR file for the net.datastructures package, you can find GraphExamples.java, which builds a graph. You can also look inside of GraphAlgorithms.java, which includes their implementation of topological sort (Fragment 14.11) as well as some graph searches. Feel free to borrow/hack their code (with acknowledgement), or just simply call their code from your program.

4 Grading

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building the prerequisite graph(s)</td>
<td>40</td>
</tr>
<tr>
<td>Better of first and second question</td>
<td>30</td>
</tr>
<tr>
<td>Worse of first and second question</td>
<td>20</td>
</tr>
<tr>
<td>Third question</td>
<td>10+10 bonus</td>
</tr>
</tbody>
</table>