Scheduling Exams

1 Short Answers

Complete the following problems from the book and turn them in at the start of lab.

15.2, 15.10, 15.15 (consider add, remove, addEdge)

2 Lab Program

You are to write a program which will help the registrar schedule final exams so that no student has two exams at the same time. The goals of this lab are to:

- Gain experience using basic graph building and traversal operations.
- Develop a fairly sophisticated algorithm requiring several coordinated data structures.

You will use a greedy algorithm to determine an assignment of classes to exam slots such that:

- 1. No student is enrolled in two courses assigned to the same exam slot.
- 2. Any attempt to combine two slots into one would violate rule 1.

The second requirement ensures that we do not gratuitously waste exam slots (students would like to get out of here as soon as possible, after all).

2.1 Input

Input to your program will be through files containing student class information. For example:

Edmund Rucci CSCI 136 MATH 251 ENGL 201 PHIL 101 Jared Strait PSYC 212 ENGL 201 HIST 301 CSCI 136 Kristof Redei SOCI 201 CSCI 136 MATH 251 PSYC 212

For each student, there are five lines. The first is the name, and the next four are the courses for that student:

- Edmund Rucci is taking CSCI 136, MATH 251, ENGL 201, and PHIL 101;
- Jared Strait is taking PSYC 212, ENGL 201, HIST 301, and CSCI 136; and
- Kristof Redei is taking SOCI 201 CSCI 136, MATH 251, and PSYC 212.

We provide small, medium, and large input files on the handouts web page.

The output of the program should be a list of time slots with the courses whose final will be given at that slot.

2.2 Algorithm

г

The key to doing this assignment is to build a graph as you read in the file of students and their schedules.

Each node of the graph will be a course taken by at least one student in the college. An edge will be drawn between two nodes if there is at least one student taking both courses. The label of an edge could be the number of students with both classes (although we don't really need the weights for this program). Thus if there are only the three students listed above, the graph would be as given below (edges without a weight label have weight 1).



A greedy algorithm to find an exam schedule satisfying our two constraints would work as follows. Choose a course (say, PHIL 101) and stick it in the first time slot. Search for a course to which it is not connected. If you find one (e.g., HIST 301), add it to the time slot. Now try to find another which is not connected to any of those already in the time slot. If you find one (e.g., SOCI 201), add it to the time slot. Continue until all nodes in the graph are connected to at least one element in the time slot. When this happens, no more courses can be added to the time slot (why?). (By the way, the final set of elements in the time slot is said to be a *maximal independent set* in the graph.)

If there are remaining nodes in the graph, pick one and enter it in a new time slot and then try adding other courses to the same slot as before. Continue adding time slots for remaining courses until all courses are taken care of. Print the exam schedule. For the graph shown, here is one solution:

```
Slot 1: PHIL 101, HIST 301, SOCI 201
Slot 2: MATH 251
Slot 3: CSCI 136
Slot 4: ENGL 201
Slot 5: PSYC 212
```

Notice that no pair of time slots can be combined without creating a time conflict with a student. Unfortunately, this is not the minimal schedule as one can be formed with only four time slots. (See if you can find one!) Thus a greedy algorithm of this sort will give you a schedule with n slots, no two of which can be combined, but a different selection of courses in slots may result in fewer than n slots. Any schedule which satisfies are our constraints will be acceptable (although see below for extensions to compute the optimal solution).

2.3 Details

You should represent graphs as adjacency lists. (Why does that make the most sense for this application?) Vertex labels should be the course names.

Here is one possible way to find a collection of maximal independent sets from the graph. Represent each slot by some sort of a list (or, better yet, a binary search tree). To find a maximal independent set for a slot, pick any vertex of the graph and add it to the list. Cycle through all other vertices of the graph. If a vertex is not connected to any of the vertices already in the slot, throw it in. Continue until you have tried all vertices. Now delete all vertices in the slot from the graph. Fill successive slots in the same way until there are no vertices left in the graph.

2.4 Extensions

Please complete at least one interesting extension to the program. I have listed a few examples below. This is also a great opportunity for earning some extra credit by including adding more features.

- 1. Print out a final exam schedule ordered by course number.
- 2. Print out a final exam schedule for each student, in alphabetical order.
- 3. Always generate the best possible exam schedule (that is, the one with the fewest number of slots). The best known algorithms for this scheduling problem run in O(n!) time, where n is the number of classes. This is bad. For example, our medium file has 16 classes, and 16! \approx 21 trillion.

However, there are nice algorithms to handle small cases. One tactic is to try all possible orderings of the graph's nodes to see which yields the best schedule with our greedy algorithm. (This will in fact be the optimal solution.) You may find the Permutation class on the handouts page useful for this task. This iterator constructs all permutations of a vector.

Feel free to explore other approaches as well.

4. Arrange the time slots in an order which tries to minimize the number of students who must take exams in three consecutive time slots. This is trickier than the other options.

Feel free to add other features as well. Be sure sure to indicate in the heading of your program what extras you have included.

2.5 Deliverables

Turn in your well-documented program using turnin by the due date.