CS2710 - Programming and Data Structures Lab

Lab 14, Homework Assignment (graded)

Out: Nov 2, 2024 (**Due: 8th Nov, 2024, 23:59 hrs**)

Instructions

- You are expected to solve ALL the problems as a **homework** assignment, using a system (computer, C++ language, and g++ compiler) that is compatible with the DCF lab system (since we will be using the lab system to evaluate your code). (Bonus problems can fetch 5% capped bonus.)
- You should submit your code to the course **moodle** on time (due date and time mentioned above; moodle submission link to be made available soon).
- You must strictly adhere to the following naming convention for your .cpp files and the single .zip file submission. For example, for a Lab Assignment 14 consisting of 2 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_LAB14_Q1.cpp** to **CS23B000_LAB14_Q3.cpp**
  - Put these .cpp files in a directory named **CS23B000_LAB14**
  - Zip this directory into a file named **CS23B000_LAB14.zip**
  - Submit only this single .zip file to moodle.
- If you need assistance, ask your TA, not your classmate.
- You have to write code on your own – you can refer to codes provided in course moodle and refer to slides or Weiss book; but otherwise, you cannot use any other sources such as the internet, or take help of your friends or fellow students. Plagiarism checks will be strictly implemented both among all students and any source codes available in the internet.
- We will make available in course Moodle the public test cases and the evaluation script, which you can use to test your programs.
- You are expected to use classes/structs in this lab and **implement the requested functionalities as member functions of these classes, using any data structure(s) taught** in this course (and these data structures could be either from C++/STL or your own implementations). Please clarify with the TAs if you have any questions about which data structures to use or avoid.

Problem Statement

You are tasked with developing a Course Registration System for IITM. This system must facilitate the enrollment of students in various courses and manage student and course records efficiently. You are expected to use an Object Oriented Approach using Classes and/or Structs.

Student and Course Records:

- Each student is uniquely identified by a **student ID** and has associated information, including:
  - **Name**: The student's name.
  - **Year of Study**: The year the student is currently in.
  - **Completed Courses**: A list of course codes representing courses that the student has successfully completed (passed) in the past.
  - **Enrolled Courses**: A list of course codes representing courses the student is currently enrolled in.
- Each course is uniquely identified by a **course code** and includes:
  - **Course Name**: The name of the course.
  - **Number of Credits**: The credits associated with the course.
  - **Capacity**: The maximum number of students that can enroll in the course.
  - **Prerequisites**: A list of course codes representing the courses that must be completed before a student can enroll in this course.
  - **Time Slot**: The designated time slot for the course (e.g., Slot A, Slot B).

In your class definitions, you can maintain additional member functions or variables that you feel is needed to achieve operations efficiently.

General Notes:

1. To get full credit, besides uploading the code for each question below, you should **also submit your answer sheet as a PDF** document (also stored inside the same single zip file uploaded to moodle). In the answer sheet, before you start coding, write down the data structures (DS) you would use to implement each requested operation, along with a brief pseudocode and running time complexity of the operation.
2. For all questions (parts), it is guaranteed that:
   - $1 \le Q \le 10^4$, where Q is the number of operations.
   - $P \le 10$, where P is number of prerequisites for any course.
   - $1 \le C \le 100$, where C is the max capacity of any course.
   - $1 \le CC \le 10$, where CC is the number of completed courses for any student.
3. In terms of **Expected Time Complexity**, if $T = T_S + T_C$ is the total number of students and courses in the system, you are expected to achieve $O(log(T))$ complexity per operation (or even better). For Part 3, you are allowed $O(T)$ for each call to `minCoursesToEligible`, but $O(log(T))$ or better for the rest.
4. Any operations that would lead to errors (e.g., adding courses before adding its prerequisites – details below) should be **rejected silently without error messages**.

Specific Questions

1. [PART 1: SETTING UP THE BASICS] Implement the following operations in the Course Registration System.

   - **Add Student**: Using the input format mentioned below.
   - **Add Course**: When adding a course, the system must check that all prerequisite courses exist before adding the course. If any prerequisite does not exist, the course addition should be rejected.
   - **Enroll Student**: When enrolling a student to a course, the system must check:
     - If the student or the course does not exist, reject the enrollment.
     - If the student has completed all prerequisites for the course.
     - If the course has available capacity.
     - If there are no scheduling conflicts with the student's currently enrolled courses.

     Enrollment is done if all above criteria are met. If a student has already completed the course, they are allowed to enroll in it (provided all the above criteria are met). If the student is already enrolled in the course, reject the enrollment.
   - **Print Course**: List enrolled students in the output format given below.

   Input Format

   - The first line will contain an integer **Q**, representing the number of operations to be performed.
   - For each operation:
     - If it is `add_student`, the next lines will include:
       - Student ID, Name, and Year of Study separated by spaces.
       - The number of completed courses
       - A list of completed course codes separated by spaces.
     - If it is `add_course`, the next lines will include:
       - Course Code, Course Name, Number of Credits, Capacity, Time Slot.
       - The number of prerequisites.
       - A list of prerequisite course codes separated by spaces.
     - If it is `enroll`, the next line will include:
       - Student ID and Course Code separated by a space.
     - If it is `print`, the next line will include the course code.

   To simplify parsing of input, you can assume that student and course names, time slots, Student ID, Course Code, etc., do not each have spaces within them.

   Output Format

   The output of the program will consist of the results of the `print` operations. Each print operation should output either the (space-separated) list of student IDs enrolled in the specified course (in the order in which they were enrolled) or `Invalid Course <CourseCode>` if the course does not exist.

Example

Sample Input:

```
7
add_student
S001 Ayon 2 2
C101 C102
add_course
C101 IntroToProgramming 3 30 D 0

add_course
C201 DataStructures 3 30 A 1
C101
add_course
C202 Algorithms 3 25 B 1
C201
enroll
S001 C201
print
C201
print
C203
```

Sample Output:

```
Enrolled students in C201:
S001
Invalid Course C203
```

2. [PART 2: ENHANCED FEATURES] Building on the Course Registration System developed in Part 1, you will enhance the system to:

   (a) **Maintain course waitlist** when enrolling a student to a course at full capacity (i.e., augment the enroll operation to automatically place the student on a waitlist for the course, if the course is at full capacity). Please note that the waitlist functions on a first come first serve (FCFS) basis. A student can be on multiple courses' waitlists at a time.

   (b) **Enable dropping a student from a course, followed by automatic processing of waitlist** (by adding a new operation called drop which removes a course from a student's list of enrolled courses and does waitlist processing as detailed next). When a student successfully drops a course, automatically enroll the first student on the waitlist for that course if he/she meets all enrollment criteria from Part 1. Otherwise if enrollment fails, remove from the waitlist and attempt the next student for enrollment. Repeat this process until the next eligible student on the waitlist is found or if the waitlist becomes empty.

   (c) **Check for cyclic dependency in course prerequisites** (if needed). Can a cyclic dependency be introduced when adding a course (via add_course operation along with its checks as listed in Part 1)? Say Yes/No in the answer sheet, and justify briefly. If you answered Yes, change the `add_course` code to check and reject any `add_course` operation that introduces cyclic dependencies.

All other operations and their functionalities of the system remain the same as in Part 1.

Input Format

- The first line will contain an integer Q, representing the number of operations.
- For the operations from Part 1, follow the same input format as in Part 1. For the new operation `drop`, the next lines will include:
  - Student ID and Course Code.
  - If either the student or course does not exist, the system will reject the drop request and no waitlist processing is done.

Output Format

Same as in Part 1, i.e., the output will consist of the results of the `print` operation.

Example

Sample Input:

```
10
add_student
S001 Ayon 2 2
C101 C102
add_student
S002 Shankar 2 1
C101
add_course
C101 DataStructures 3 1 A 0

add_course
C201 Algorithms 3 2 B 1
C101
```

```
enroll
S001 C101
enroll
S002 C101
enroll
S002 C201
drop
S001 C101
print
C101
print
C201
```

Sample Output:

```
Enrolled students in C101:
S002
Enrolled students in C201:
S002
```

3. [(BONUS) PART 3: ELIGIBLE PATHWAY BETWEEN COURSES] Building on Parts 1 and 2, we now want to analyze the courses in the system to determine the shortest eligible path between two courses of interest, C_s and C_t. Consider the **prerequisite graph (**G**)** formed by course prerequisites - the nodes in this graph represent all added courses in the system, and the directed edges represent all pairwise prerequisite relations (with $(C_i \to C_j)$ edge indicating that course C_i is one of the prerequisites of course C_j). A simple directed path from C_s to C_t of length k in G, denoted $(C_1 := C_s \to C_2 \to C_3 \to \ldots \to C_k := C_t)$, is considered **eligible** if NO two courses in this path is scheduled in the same time slot.

   We would like to add a new feature/operation `minEligiblePathlen` to the system to calculate the length of the shortest eligible path from a source course C_s to a target course C_t in the prerequisite graph G. To implement this operation efficiently, you may want to exploit the fact that the maximum number of time slots P is small ($P \leq 10$), but that the number of courses and prerequiste relations is much larger.

   Input Format

   - The first line will contain an integer Q, representing the number of operations.
   - For the operations from Part 1 and 2, follow the same input format as in Part 1 and 2. For the new operation, the format is:
     - "`minEligiblePathlen <courseCode1> <courseCode2>`" outputs length of the shortest eligible path from `courseCode1` to `courseCode2` as described above.
     - Assume that both courses exist in the system.

   Output Format

   In addition to output from Part 1 or 2 (i.e., results of the `print` operations), we also have output from the new `minEligiblePathlen` operation. This new operation will output the number of courses in the shortest eligible path between two specified courses (counting both specified courses). If no such path exists, output -1.

   Example

   Sample Input:

```
10
add_course
C101 DataStructures 3 30 A 0

add_course
C201 Algorithms 3 25 B 1
C101
add_course
C202 AdvancedAlgorithms 3 20 E 1
C201
add_course
C302 LinearAlgebra 3 20 C 1
C201
add_course
C303 LinearAndNonlinearOptimization 3 20 D 1
C302
add_course
C401 MachineLearning 3 20 E 2
```

```
C202 C303
minEligiblePathlen C101 C401
minEligiblePathlen C401 C101
minEligiblePathlen C202 C302
minEligiblePathlen C201 C202
```

Sample Output:

```
5
-1
-1
2
```

Explanation: From course $C101$ to course $C401$, even though the path $C101 \to C201 \to C202 \to C401$ is shorter with length 4, it is not eligible due to a time slot conflict. So the shortest eligible path from $C101$ to $C401$ is $C101 \to C201 \to C302 \to C303 \to C401$ of length 5.