CS340 - Registrar's Problem

Andy Hong, Alton Wiggers, Carter Langen, Russell Rivera

October 16, 2019

1 Description

Initially add all students to their preferred classes. Sort classes by popularity and rooms by size. Set the next available room for each time slot be the (next) largest room in our set of rooms. While there are still classes unassigned to rooms and times, consider the most popular class. For each time slot, we want to maximize the number of students who have enrolled in that time slot, such that there are no time conflicts between student classes, and professors can teach at that given time slot. As we select students, we are interested Assign that class to the biggest available room at that time slot and mark the enrolled students as busy at that time.

2 Pseudocode

```
Function registrar(Students, Classes, Times, Teachers, Rooms)
    Add all students to their preferred classes
    Sort Classes by most popular classes to least popular
    Sort Rooms by room size from largest to smallest
    Set the largest room to be the first available room for all times
    spotsEnrolled = 0
    while Classes is not empty do
        class = the most popular class in Classes
        chosenSlot = None
        increase = 0
        create empty set students
        foreach time in Times do
            if class.teacher.time ≠ time ∧ time.room ≠ None then
                create empty set possibleStudents
                x = 0
                foreach student in class.students do
                    if student not in time.students then
                        x+ = 1
                        add student to possibleStudents
                    end
                end
                if time.room.size < x then
                    x = time.room.size
                end
                if x > increase then
                    chosenSlot = time
                    increase = x
                    students = possibleStudents
                end
            end
        end
        class.teacher.time = chosenSlot
        class.room = chosenSlot.room
        class.time = chosenSlot
        class.students = students
        assign |increase| many students from students to chosenSlot.students
        spotsEnrolled+ = increase
        chosenSlot.room = room.next
        pop class from Classes
    end
    return spotsEnrolled
```

3 Revised Pseudocode

```
Function registrar(Students, Classes, Times, Teachers, Rooms)
    Add all students to their preferred classes
    Sort Classes by most popular classes to least popular
    Set teachers first class to be their most popular and second class to be their
     second most popular
    Sort Rooms by room size from largest to smallest
    Set the largest room to be the first available room for all times
    Divide the time slots into two groups
    spotsEnrolled = 0
    while Classes is not empty do
        class = the most popular class in Classes
        chosenSlot = None
        increase = 0
        create empty set students
        foreach time in Times do
            if (class.teacher.first = class or class.teacher.first.group
             ≠ time.group) ∧ time.room ≠ None then
                create empty set possibleStudents
                x = 0
                foreach student in class.students do
                    if student not in time.students then
                        x+ = 1
                        add student to possibleStudents
                    end
                end
                if time.room.size < x then
                    x = time.room.size
                end
                if x > increase then
                    chosenSlot = time
                    increase = x
                    students = possibleStudents
                end
            end
        end
        class.teacher.time = chosenSlot
        class.room = chosenSlot.room
        class.time = chosenSlot
        class.group = time.group
        class.students = students
        assign |increase| many students from students to chosenSlot.students
        spotsEnrolled+ = increase
        chosenSlot.room = room.next
        pop class from Classes
    end
    return spotsEnrolled
```

4 Time Analysis

Before running the algorithm all s number of students need to be assigned to their preferred classes. This takes $O(s)$. A c number of classes then need to be sorted by popularity, which takes $O(c \log c)$. An r number of rooms need to be sorted, which can be done in $O(r \log r)$. The t number of time slots must have their first available room set to the largest room, which can be done in $O(t)$. Lastly, we initialize a dictionary of dictionaries, of which the inner dictionary can be initialized in $O(s)$ time, where s is the number of students, and the outer dictionary can be produced in $O(ts)$ time (the student dictionary is produced t times). Together, these operations produce a pre-processing time of $O(ts) + O(r \log r) + O(c \log c) + O(s) + O(t) \in$ O(ts).

The while loop runs the c times. Within the while loop, the outer for-each loop is run for the t number of time slots. In the inner for-each loop, all students who want to take the given class are checked against the students busy at that time slot. Since at most s students want to take a class, this can be at most s comparisons. After the outer for-each loop but still within the while loop, a number of students are assigned to the time slot for a class, which can be at most s. So the inner for-each loop runs a total of cts times and the assignments run a total of cs times. Together these cause $cts + cs$ iterations of loops or $c(t + 1)s$ iterations. Therefore the algorithm takes $O(cts)$ time.

In order to run the algorithm in $O(cts)$ the following operations need to be done in $O(1)$:

1. get and remove the most popular class
2. check the time a teacher is teaching
3. check and update the next available room at a time
4. check and update student availability at a given time
5. add a student to possibleStudents
6. check the size of a room

5 Data Structures

We use the following objects along with our data structures:
Teacher:
class1
class2
timeslot1 = null
timeslot2 = null

Student:
preference list

TimeSlot:
a pointer to the linked list of rooms

Room:
size
nextRoom

Class:
size/popularity
list of students
room
timeslot

Additionally, we have a dictionary with time slot keys, with dictionary values. The sub dictionary has students as keys and booleans as values, which are true if the student is busy at that time, and false otherwise. PossibleStudents is a linked list.

All class objects are stored in a sorted linked list. This takes $O(S + c\log(c))$ to make in prepossessing, where c is the number of classes, and S is the total number of students, but allows us to do (1) in constant time. We can check if a teacher is teaching at a time in constant time, because we store this as a class attribute.
We can check the time a teacher is teaching constant time in because it is stored as a class attribute.
The list of rooms can be a linked list of dictionaries. The list *Rooms* holds a pointer to *Rooms.head*. Each *room* in *Rooms* has key values for its size and the next element in the list. We can check the size of a room by calling *room.size*. This allows (6) to be done in $O(1)$ and the linked list takes $O(r)$ time to create where r is the number of rooms. A time slot can hold a pointer *time.room* to its next available room. The room can be updated by doing $time.room = time.room.next$. This allows (3) to be done in $O(1)$ with the original setup of each *time.room* done in the pre-processing.

For checking and updating student availability, we maintain a dictionary of dictionaries. The outer dictionary is keyed by time slot objects, and each value is a dictionary of students, keyed by student object, and valued by a Boolean value. For example:

dct[timeslot][student] gives a Boolean value that describes whether student has already been assigned to timeslot, i.e., whether that student is available at that timeslot or not. Because this structure is a dictionary of dictionaries, read, write, and lookup operations all happen in constant time. Producing this structure requires creating a student dictionary initialized to all False values ($O(s)$) for t different time slots. This results in a pre-processing time of $O(ts)$. through the use of this dictionary, we can complete (4) in constant time. We can also update possibleStudents (5) in constant time, since adding to the head of an unsorted linked list can be done in constant time.

In sum, since pre-processing takes $O(ts)$ time and the algorithm runtime is to the order of $O(cts)$, the complete runtime can be bound at $O(2 * cts) \in O(cts)$.

6 Proof of Correctness

Assumption: The product of the number of rooms and the number of time slots is larger than the number of classes.
Note: we recognize that this assumption might not be allowed. If there is exactly the needed number of rooms and times so that no room will be empty at any time, it is possible classes will not be assigned. If necessary we will later implement a work around for this that will ensure all classes are scheduled, trading off total enrollment.

Proof. Proof of Termination:
The algorithm terminates because one class is popped out of the Classes list for each iteration of the while loop, so that the size of the Classes list decreases by 1 with each iteration. Since the length of the Classes list is a non-negative integer, the Classes list will eventually become empty, by then the condition for the while loop will not meet. So the algorithm terminates.

Proof of Validity:
This proof will be done in 3 parts:

1. Each room has only one class at a given time slot
2. Each class has only one room, one teacher, and a set of students(which could be empty)
3. Each teacher teaches two classes at different time slots

Proof of Condition 1:
Each time slot has a pointer that points to the head of the room linked list and the algorithm initially sorts room size from the largest to the small and sets the largest room to be the first available room for all times.

In the algorithm, we iterate through all the time slots for each class and find one such time slot that gives the maximum number of students without conflicts for taking a class. We then assign this optimal time slot for a class to "chosenSlot" (chosenSlot is an instance of time slots object) and assign the class' room to its chosenSlot's room(which is a pointer that points to the largest available room) and its time to chosenSlot's time. Afterwards, we move the pointer for chosenSlot.room to the next room in the "room" linked list. In this way, we make sure each room has only one class at a given time slot.

Suppose that the algorithm assigns two classes to a room at a given time slot. Then it would mean that after assigning one class to a room at a time slot, the algorithm did not remove the assigned room as the invalid room. However, in the pseudocode, the algorithm moves the pointer for chosenSlot.room to the next largest available room for the chosen time slot making the assigned room not available to be assigned any additional courses by doing "chosenSlot.room = room.next". Thus, this is a contradiction because algorithm will never allow two classes to a room at any given time slot.

Proof of Condition 2:
The teacher for the class is given. The while loop runs for each class only once, as the most popular class is necessarily removed from the set of classes until all classes have been removed. In the while loop, each class is assigned a single room for a time and a single set of students. Therefore, each class has only one room, one teacher, and a set of students.

Proof of Condition 3:
Suppose that a teacher has been assigned a class at time slot T_t, and the only other time slot available for the current room size is T_t. Then, by our assumption that their are extra rooms, there will be a next room in which we can schedule the class at any time slot other than T_t. Thus, the teacher only teaches one class at a time, and can teach both classes.

□