Universidad EAFIT

Departamento de Informática y Sistemas

Academic Scheduling Optimization

Data Structures and Algorithms II

Simón Marín Giraldo
Julián Ramírez Giraldo
June 2020

Contents

1 Introduction

EAFIT University wants to optimize their academic scheduling in order to ensure that every student has an appropiate classroom for their classes and their mobility.

Figure 1: EAFIT campus

We are asked to ensure that every student has a proper classroom such that students with mobility issues have their classes located in blocks with wheelchair accessibility. This blocks normally have elevators for ensuring that students with this issues reach their classrooms.

An additional and optional task to solve is optimizing the scheduling so that students having consecutive classes don't have to cross the whole campus, getting stressed, tired and probably arriving late to their classes.

1.1 Problem definition

Our scheduling must meet the following conditions in order to consider it a correct scheduling:

- No teacher has two different classes scheduled simultaneously, that is, at the same time on the same day.
- No classroom is allowed to have two classes scheduled simultaneously.
- All groups are assigned a teacher and a classroom.

Our scheduling must satisfy the following restriction:

- It must be ensured that all persons with mobility limitations have their classes scheduled in easily accessible classrooms.

Optionally, we should also satisfy the following restriction:

- The average time it takes for a person to move from one class to another should be minimized.

2 Input

In order to solve the problem, we must define some concepts for understanding the data.

2.1 Academic Programming

The list of all groups scheduled for the semester. Each group has an associated professor and classroom. It is delivered in a text file where each line corresponds to an hour assigned to a group and has the following fields separated by commas.

- `course:` sequence of six (6) alphanumeric symbols. The first two are letters and represent the academic department responsible for the course and four digits which are the course code.
- `group:` sequence of three (3) digits that identifies the group. The combination `course`, `group` identifies each programmed group.
- `profesor:` integer that identifies the teacher.
- `classroom:` sequence of alphanumeric symbols that identifies the classroom where the group is programmed.
- `day:` an alphabetical symbol representing the day of the week like this:

  L Monday
  M Tuesday
  W Wednesday
  J Thursday
  V Friday
  S Saturday

- `starting time`: integer table number digits representing the start time of the class in military format.
- `end time`: integer number of table digits representing the end time of the class in military format.

2.2 Enrollment

The list of all students in the university with the courses they have enrolled in. It's a text file with the values separated by commas.

- `student:` integer identifying the student.
- `course:` sequence of six (6) alphanumeric symbols. The first two are letters and represent the academic department responsible for the course and four digits which are the course code.
- `group:` sequence of three (3) digits that identifies the programmed group. The combination course, group identifies each programmed group.

2.3 Classrooms

The list of all the classrooms in the university. It's a text file with the values separated by commas.

- `classroom:` sequence of alphanumeric symbols that identifies the classroom where the group is programmed.
- `type:` integer value representing the type of classroom
- `capacity:` maximum number of students who can attend class si- multaneously in the classroom.
- `access:` an integer indicating the type of access to the classroom, like this:
  0 Inclusive access that allows access to any person
  1 Does not have inclusive access, so people with mobility limitations cannot access the classroom.

2.4 The map of the university

Is represented as a graph, where each block is a node. For simplicity, we assume that we can neglect the distance between two classrooms that are in the same block, so only the distance between blocks is provided. Two blocks are connected by an arc if there is a direct path between them that does not pass through any other blocks. This is a very similar representation to the one used by GPS systems. The information is in a text file in a comma-separated format, where each line contains three entries:

- `block_1:` sequence of alphanumeric symbols that identifies the ini- tial block.
- `block_2:` sequence of alphanumeric symbols identifying the final block.
- `Distance:` A number in floating point format representing the dis- tance between the two blocks

2.5 Persons with mobility impairments

A list containing the identifications of all persons who have mobility impairments. Each line has a single element that is an integer and represents the identity of the person who has mobility limitations.

3 Solution and complexity analysis

Now, we are going to describe the coding solution given to this particular problem.

3.1 Methods

- **DataFillClassroom:** This method goes through the `classrooms` dataset and adds the data to a **Classroom Linked List**. Finally it returns the **Linked List** to be added to the global classrooms **list**.

  As result in O notation we get a complexity of:

  $$O(c)$$

  Where c represents the number of classrooms in the dataset.

- **DataFillStudentsMI:** This method goes through the `persons with mobility impairments` dataset and adds the data to a **Student MI Linked List**. Finally it returns the **Linked List** to be added to the global Student MI **list**.

  As result in O notation we get a complexity of:

  $$O(m)$$

  Where m represents the number of students with **mobility impairments**.

- **DataFillStudents:** This method goes through the `students` dataset and adds the data to a **Student Linked List**. Finally it returns the **Linked List** to be added to the global Students **list**.

  As result in O notation we get a complexity of:

  $$O(s^2)$$

Where s represents the number of **students** in the dataset.

- **DataFillGroup:** This method goes through the `groups` dataset and adds the data to a **Groups Linked List**. Finally it returns the **Linked List** to be added to the global Groups **list**.

  As result in O notation we get a complexity of:

  $$O(n)$$

  Where n represents the number of **groups** in the dataset.

- **specialClassroomFill:** This method creates a ***regular expression*** with the word **"Laboratorio"** and **"Cabina"** and goes through the **groups** list and the **groups and classrooms** list. Then, the ***regular expression*** checks if any of the descriptions start with **"Laboratorio"** or **"Cabina"** and adds it to the global **special classroom** list.

  As result in O notation we get a complexity of:

  $$O(n \times t \times x)$$

  Where n represents the number of groups, t represents the number of classroom types and x represents the comparations of the regular expression.

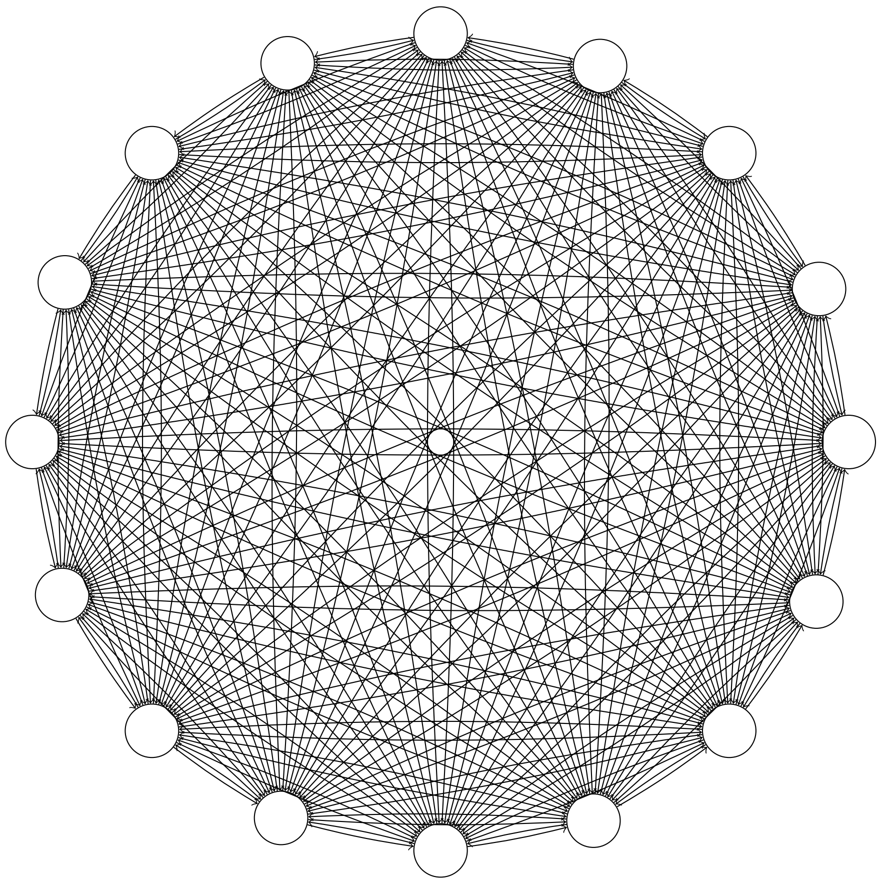
- **classType:** This method goes through the `groups` list and in every iteration it also goes through the `classrooms` list comparing both current values to know when the classrooms match. If they match, the classroom and its description are added to the **group type** list. Finally it returns the list and assigns it to the global **group type** list.

  As result in O notation we get a complexity of:

  $$O(n^2 \times c^2(n + c))$$

  Where n represents the number of groups and c represents the number of classrooms.

- **Graph:** The **graph** (conformed by **vertexes** and **edges**) stores information about each block in the university with the distances between them.



This is a complete graph, as the one we have in the implementation with the datasets because for each block we are given the distance to all the blocks. It can be represented with a complete graph.

As result in O notation we get a complexity of:

$$O(v + e)$$

Where v represents the vertexes and e represents the edges in the graph.

- **nearestAccesibleBlock:** In this method we receive the **classroom** list as a parameter in order to search for the block with the minimun distance given a block, also it must be wheelchair accessible. It is based in the method used for looking for the minimum number in an integer array, but applied to the weights of each edge in the edge list. It finally returns the nearest accessible block as a **Vertex** object.

As result in O notation we get a complexity of:

$$O(\theta^3)$$

Where θ represents the number of **edges** in the **graph**.

- **nearestBlock:** It works like the **nearestAccessibleBlock** method, the only difference is that we don't have the condition of wheelchair accessibility.

  As result in O notation we get a complexity of:

  $$O(\theta^2)$$

  Where θ represents the number of **edges** in the **graph**.

- **studentGroupSpecials:** This method goes through the **groups** list and in every iteration it goes through the **special classrooms** list in order to get the list of classrooms which courses can only be done there. They are special classrooms like laboratories.

  As result in O notation we get a complexity of:

  $$O(n \times p \times c)$$

  Where n represents the number of existing groups, p represents the number of special groups (such as laboratories or some language center booths) and e represents the comparison with the `.equals()` method.

- **nearestChange:** For this method we need to use the **graph** we previously created and an auxiliary variable which represents a block in the university. This method goes through different lists: `schedule` which is the list with the schedules of all the students. It goes through a 7-position array which represents each week day to know the schedule of the student.

It goes through the busy houurs list on each day and it also goes through all the **vertexes**. This is done in order to compare the block in which a student is located and the nearest block. Automatically, when it finds it, it calls the `setBlock()` method to assign the block obtained with the `nearestAccessibleBlock()` or `nearestBlock()` method, depending on the student's mobility condition. It then prints the average walked distance by all the students in the university.

As result in O notation we get a complexity of:

$$O(s^4 \times k^2 \times I + (sK + \theta^3 + \lambda^2))$$

Where s represents the size of the schedules per student, k represents the number of classes each student has per day, I represents the assignments made with the `nearestChange()` method, θ represents the complexity of the `nearestAccessibleBlock()` method and λ represents another assignation inside the method.

- **scheduleFill:** This method goes through the lists in **studentCourseGroup** and the **groups** list, having an auxiliar 7-position array which depending on the weekday, it adds to the day's respective position, storing the schedule for each student in each day.

  As result in O notation we get a complexity of:

  $$O(s^3 \times k \times c^2 \times n)$$

  Where s represents the number of students, k represents the number of schedules in the schedule list, c represents the number of students in the course group list and n represents the number of groups.

3.2 Final complexity

After solving the expression, we get a final asymptotic complexity of:

$$O(c + m + s^2 + n + n \times t \times x + n^2 \times c^2 \times (n + c) + (v + e) + \theta^3 + \theta^2 + n \times p \times c + [s^4 \times k^2 \times I + (K + \theta^3 + \lambda^2])$$