Project report : Graph theory

Team : Charly Ginevra & Théo Pirouelle

language python

Introduction

In our course of graph theory, we, Théo Pirouelle and Charly Ginevra, realized a project consisting of the implementation of the stable marriage algorithm.

The stable marriage algorithm

According to Wikipedia : *"The stable marriage problem (also stable matching problem or SMP) is the problem of finding a stable matching between two equally sized sets of elements given an ordering of preferences for each element."*

At the end of the execution, each element of the first set is associated with one element of the other set. Multiple answers are possible, but there will be no rogue couple (cf. vocabularies).

Theme of the project

Our idea was to create a second version of ParcourSup with this algorithm. Because this algorithm is used by ParcourSup, we decided to make "a joke" and named our project *"ParcourSupV2"*. At the end of the execution, every student will have a school unlike school which cannot have any student.

Vocabularies

To understand this algorithm we have to clarify few terms :

- **Rogue couple**: A man and a woman not married to each other and who like each other more than their spouses.
- **Stable matching**: A matching with no rogue couples.

Choices made

Project gestion

In terms of project gestion, we decided to use Github. It allowed us to work separately and to merge our programs during the class time.

For the repartition of the work, Théo was in charge of programming the user interface and how to get back information. Charly was in charge of programming the algorithm and creating the data structures.

Technology used

We chose Python as the programming language. It seems obvious for us to use it for multiple reasons. The first one would be that Python has a very big community with a lot of packages already developed. The second one would be that Python gives us the capacity to use OOP (Object Oriented Programming). It is very helpful, especially with the data structures because you can associate your data structure with its method. The last reason is because Python is a non typed programming language. Contrary to typed programming language, you can easily and quickly arrive to a solution with Python. With Java for example, it will be more time consuming and strict.

We also used PyCharm as IDE because we are familiar with this one.

For the input file we chose the `.csv` format. All actors and their preferences were stored in an array. The `.csv` is the best way to get something similar and packages are already developed to make the manipulation easier.

	Name1	Name2	Name3
School1	PrefSchool,PrefStudent	PrefSchool,PrefStudent	PrefSchool,PrefStudent
School2	PrefSchool,PrefStudent	PrefSchool,PrefStudent	PrefSchool,PrefStudent
School3	PrefSchool,PrefStudent	PrefSchool,PrefStudent	PrefSchool,PrefStudent

	School1	School2	School3
Name1	PrefStudent,PrefSchool	PrefStudent,PrefSchool	PrefStudent,PrefSchool
Name2	PrefStudent,PrefSchool	PrefStudent,PrefSchool	PrefStudent,PrefSchool
Name3	PrefStudent,PrefSchool	PrefStudent,PrefSchool	PrefStudent,PrefSchool

Data structures

Because our theme is ParcourSup, the two obvious actors are the students and the schools.

Student

We can describe him with his features (a name, preferences) and what he do (remove the school of his list when he is not accepted).

To reference attributes and methods, we created a class named student. The name is just a simple string and the preferences is a list of schools where the first item is the favourite.

```python
class Student:
    # ...

    def __init__(self, name: str):
        self.name = name
        self.preferences = list()

    def setPreferences(self, preferences: list):
        self.preferences = preferences

    def getFirstChoice(self):
        return self.preferences[0]

    def removeFirstChoice(self) -> None:
        self.preferences.pop(0)

    # ...
```

School

As the students, schools can be described by their name and preferences but also by their capacity and their candidates.

They can be overloaded. It means they have more candidates than they can receive. That's why they can decline candidates based on their preferences.

```python
class School:

def __init__(self, name:str):
    self.name = name
    self.preferences = list()
    self.capacity = 0
    self.candidate = list()

def setPreferences(self, preferences: list):
    self.preferences = preferences

def setCapacity(self, capacity: int):
    self.capacity = capacity

def isCapacityExceeded(self) -> bool:
    if len(self.candidate) > self.capacity:
        return True
    else:
        return False

    def declineStudent(self) -> list:
        declinedStudentList = list()
        # ...
        while len(self.candidate) > self.capacity:
            declinedStudentList.append(getLessFavorite())
        return declinedStudentList
```

The algorithm

The execution of the algorithm is in three parts :

1. The students without school candidate for their favourite school.
2. The schools verify if their capacity is not exceeded. If so, the school removes the least favourite student from the candidate list until the number of candidates is less than the capacity. Otherwise nothing happens.
3. The students without school remove their first choice from their list of preferences. Their second choice is now their first one.

Those parts represent an execution round. At the end, every student has a school.

Without capacities

```
// name : marriage_algorithm
// semantic: execute the stable marriage algorithm
// parameters:
// schools : In/Out list<school>; -- list of all schools
// students : In/Out list<student>; -- list of all students
// nbRound : Out Integer; -- number of round
// pre-condition: sum(schools) = sum(students)
// post-condition:
//      -- A school accepts only one student
//      -- Every student has a school
function marriage_algorithm(students: list<student>; schools: list<school>) :
Integer
    favSchool: school;  // favourite school
    not_assigned_student: list<student>;    // all students without school
    nbRound: Integer;   // number of rounds
begin
    nbRound := 0
    not_assigned_student := students
    while (not not_assigned_student.isEmpty()) do
        // Firstly : Student who doesn't have a school enrols to his favorite
one
        for student in not_assigned_student student
            favSchool := etudiant.getFirstChoice()
            favSchool.addCandidate(student)
        end for
        not_assigned_student := list()  // erase the list, all students have a
school
```

```
        // Secondly : Schools check if they are not overloaded. If so, they
delete their least favorite student until they are not overloaded anymore
        for school in schools do
            if school.isCapacityExceeded() do   // conflict
                for student in school.declineStudent() do
                    not_assigned_student.append(student)
                end for
            end if
        end for

        // Thirdly : Declined students remove their first choice and their
second one become their first one
        for student in not_assigned_student do
            student.removeFirstChoice()
        end for

        nbRound := nbRound + 1
    end while

    return nbRound
end
```

With capacities

In this part, the capacity of each school can be different, and at the end, students still need to have a school. We conclude that the number of schools and students can be different but **the sum of each school's capacity must be higher than or equal to the number of students**. Only pre-condition and post-condition change here :

```
// pre-condition: sum(school.capacity) >= sum(students)
// post-condition:
//      -- Students are register as candidates to a school
//      -- Every student has one school
```

Conclusion

Due to lack of time, we have not been able to develop this application to its full potential and there are still various areas for improvement to be explored. For example, we could allow the user to insert the capacities of each school in the file while allowing the possibility of not putting them; the application would ask the user if the capacities are included in the file or not. Another point to improve would be to check the header to know if it is empty or not. If so, the row or column without header would not be recovered.

We found this project interesting, especially putting an abstract mathematical formula into a functional application.