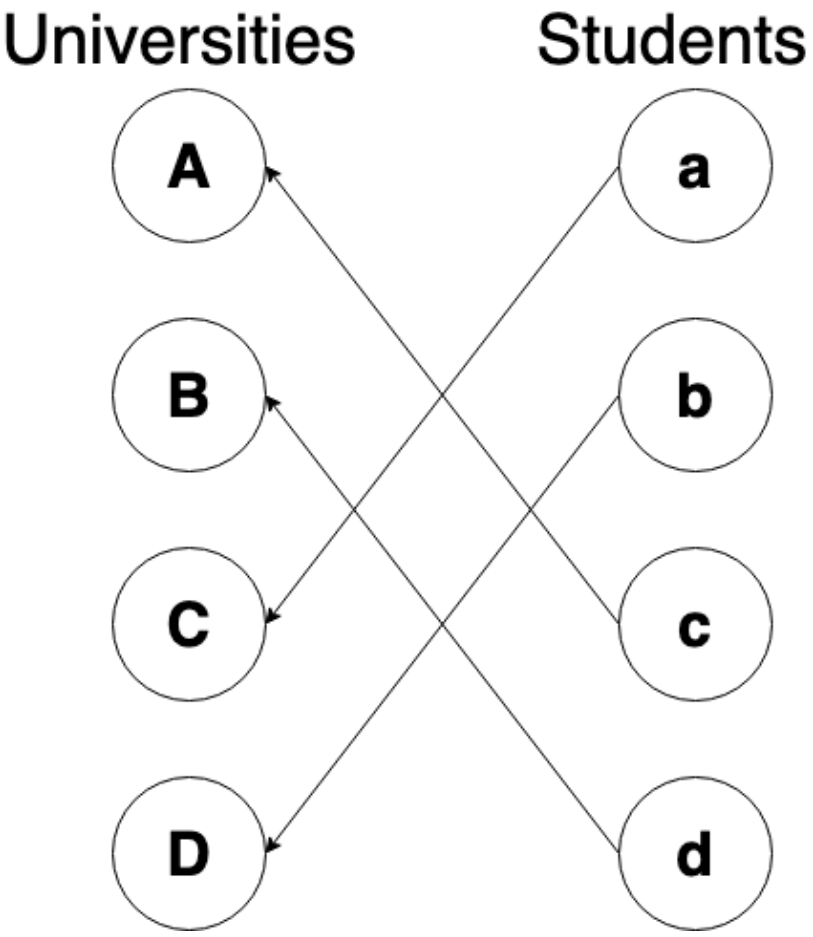
Graph Theory
Stable Mariage Algorithm



Alban Personnaz
Jérémy Theret

INP TOULOUSE ENSEEIHT

Summary

Introduction

Context

The stable marriage problem is a mathematical problem that involves finding a stable match between a number of “women” and a number of “men” based on their preference lists. The goal is to ensure that there are no unstable couples where both a man and a woman prefer each other over their current partners.

While individuals can often find a stable situation, the challenge lies in finding all possible stable solutions. To address this, researchers have developed algorithms to solve the problem using computers.

The stable marriage problem has various applications, and one well-known example is the assignment of students to their future schools. However, the problem extends beyond this specific scenario and has broader implications in different contexts.

Specifications

Here are the specifications to carry out this project:

Implement a student admission program using the stable marriage algorithm in Teams of 2.

Input :

- Student and school preferences from a file (CSV)
- User selects who does the biding (in the program)

Output :

- Student to school assignment.
- Number of rounds needed to finish.

Technical Choice

We decided to use Python as it is a user-friendly and suitable language for mathematical problems. Python offers various mathematical functionalities, along with libraries like Pandas for data manipulation. Moreover, Python supports object-oriented programming, which is convenient for our algorithm as we need to store multiple values and perform various operations related to our students and schools.

Data Structures

School and student Data structures

We have used the object-oriented programming aspect of Python to manage our schools and students. Indeed, this approach is very convenient for storing multiple data related to the same object, unlike a regular variable. Therefore, we have multiple attributes specific to each object.

```
class Entite:


    # mariage = "null"
    def __init__(self, nom, places =1, *elements):
        self.nom = nom
        self.preferences = list(elements)
        self.mariage = []
        self.places = places
        self.liste_pretendant = []
        self.list_souhait = []
        self.increment_rejet = 0
```

We have created a single 'Entity' class for our School and Student objects because in reality, they are similar except for their capacity, as a student has only 'one spot'. However, since the algorithm is reversible, it is more convenient to have a single class with the same functions.

```
self.list_souhait = []
```

Primarily, in this class, we have a list called “list_souhait” that is filled with the names of the desired entities, which are the names of students if it is a school or the names of schools if it is a student. The first element of the list represents the entity (school or student) that is most desired. Essentially, the index of the list indicates the preference order.

Data Structure Algorithm

To represent the functioning of the algorithm in the code, we have chosen a data structure that simulates the act of standing under a balcony to court the person we desire.

So, we use a shared dictionary for each student and each school, where students indicate their first-choice school. If it's the schools that move, they put the students they desire the most (as many student names as the school's capacity).

```
for eleve in eleves:
    for nbchoix in range(eleve.places):
        eleve.list_souhait.append(eleve.getList()[nbchoix])
```

'eleves' *represent the list of objects courting each other, which may not necessarily be actual students(it's just a variable name). It can be the schools based on what we want.*

```
#correspondre choix aux courtisans
for eleve in eleves:
        dico_balcon[eleve] = eleve.list_souhait  # plus une valeur
```

So in the dictionary, we have a key which is an 'Entity' object (either representing a school or a student), and for each key, we have a value that is a list containing the names of the schools or students we desire. It depends on who has the role of staying or moving.

Input Data Structure

CSV stands for Comma-separated values, and is used to structure data in rows and columns, usually separated by commas.

To use CSV, we used the python library pandas. We transform the csv file into a data frame object. This data frame contains all the data required to create entities. With the number entered by the user indicating the number of students (see User Guide), we can separate schools from students.

```
def importFromCsv(df, nb_eleves) ->
```

The function that creates the entities is the "importFromCsv" function. We create the entity with the name of the column, the data in the first line represents the number of places and the rest the preferences.

If we have more students than schools (or vice versa), before adding the student's (or school's) preferences, don't forget to remove the empty data from the column using the dataframe's "dropna" method. The function will return a dictionary with "eleves" as key and a

list with all entities representing students in it as value, and a "ecoles" key with a list with all entities representing schools in it.

See more about input data structure in User Guide Part.

Output Data Structure

The program's results are displayed in the terminal where you launched the program.

```
print("EN " + str(i) + " TOURS")
for cle, valeur in dico_balcon.items():
    print(f"{cle.getNom()}: {valeur}")
```

After the algorithm is finished, we display the results of the stable matches. In this case, we can observe that it is the students who moved.

```
n7student@Laptop-app:~/python/jerem$ python3 class_entite_1_1.py -f "./test.csv" -n 9
EN 2 TOURS
Eleve 1: ['Ecole B']
Eleve 2: ['Ecole B']
Eleve 3: ['Ecole C']
Eleve 4: ['Ecole A']
Eleve 5: ['Ecole B']
Eleve 6: ['Ecole B']
Eleve 7: ['Ecole B']
Eleve 8: ['Ecole B']
Eleve 9: ['Ecole B']
```

The display in the terminal shows the number of rounds required to establish stable matches and the stable matchs. The number of rounds may differ from what you would have done manually on paper. This is because the algorithm does not wait for each entity to pass in front of each balcony before redirecting them to the next element in their preference list. As soon as an entity is not the top preference or there is no available spot for them, they move directly to the next balcony without waiting for the next round. This saves time in the rounds without altering the results of the algorithm.

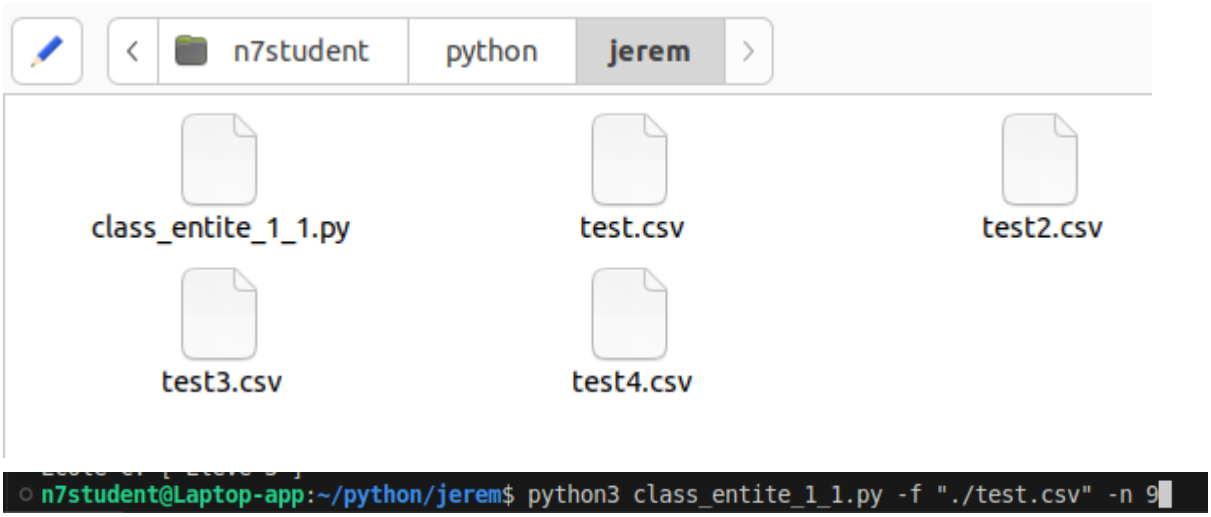
User Guide

Required (on Linux)

- Download python3 (version >= 3.10) (apt install python3) // CLI
- Download pip (sudo apt install python3-pip) // CLI
- Download pandas library (pip3 install pandas) // CLI

Run the code

Beforehand, make sure that the Python file you are going to run is in the same folder as the test files.

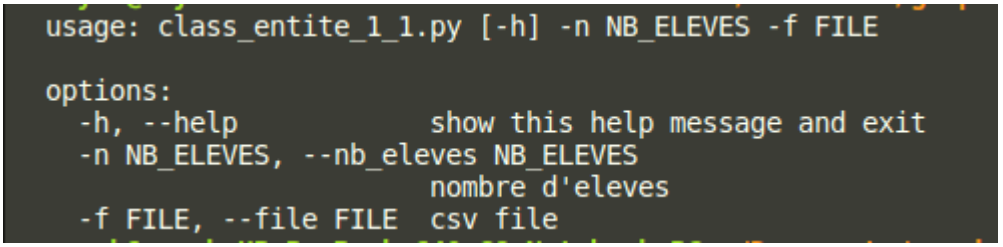


python3 class_entite_1_1.py -f "./test.csv" -n 9
// for the test file test.csv

python3 class_entite_1_1.py -f "./test2.csv" -n 3
// for the test file 1 "test2.csv"

python3 class_entite_1_1.py -f "./test3.csv" -n 4
// for the test file 1 "test3.csv"

python3 class_entite_1_1.py -f "./test4.csv" -n 4
// for the test file 1 "test4.csv"



Important Point

To change the roles of those who move and those who stay on the balcony, it is done within the code at the last line !

```
stable_mariage_alban(data["ecoles"], data["eleves"])
```

In this case, it is the schools that move. If you want to reverse it, you need to do the following."

```
stable_mariage_alban(data["eleves"], data["ecoles"])
```

Input Data Structure

	A	B	C	D	E	F	G
1	Eleve 1	Eleve 2	Eleve 3	Eleve 4	Ecole A	Ecole B	Ecole C
2	1	1	1	1	1	2	1
3	Ecole A	Ecole C	Ecole C	Ecole C	Eleve 2	Eleve 1	Eleve 3
4	Ecole B	Ecole A	Ecole B	Ecole B	Eleve 3	Eleve 2	Eleve 1
5	Ecole C	Ecole B	Ecole A	Ecole A	Eleve 1	Eleve 3	Eleve 2
6					Eleve 4	Eleve 4	Eleve 4

Example : test3.csv

In a **CSV file**. So you need to put the student's name in the first line, and in the second line the capacity (so it's one because a student can go to one school). Below these are the names of the schools in the order of preference, so for Student 1, their preferred school is School A. Please note that the names of the students and schools are crucial. If you use different names, the program will not work. Therefore, it is important that the names of the students in the school preferences and vice versa for the students are identical.

! You can use as many student numbers as you want, but be careful to match them with the school's capacity. In line 2, you can see the capacity for each school; Ecole B has 2 spots available !

Details

As you can see, we use arguments when running the program, such as #-f "./test3.csv" -n <number>#. This is because the algorithm requires a CSV file as input to execute the code. Therefore, you can use your own data as long as you adhere to the CSV file structure and the specified running command. You need to specify the student number after "-n".