Louis ONILLON, Rémi PERRIER

# Project – Stable University

This project aims to implement a student admission program named Stable University (like Parcoursup) using the stable marriage algorithm. The program reads student and school preferences from input files and matches students to schools based on these preferences and the schools' capacities.

This project has been made by Louis ONILLON and Rémi PERRIER, students at ENSEEIHT.

## Module – Graph theory

June 14, 2024

## 1 ) Project installation

### 1.1. Requirements

This project requires the following libraries:

```
cutie
PrettyTable
```

To install the libraries, run the following command:

```
pip3 install -r requirements.txt
```

### 1.2. Usage

To run the program, run the following command in your terminal:

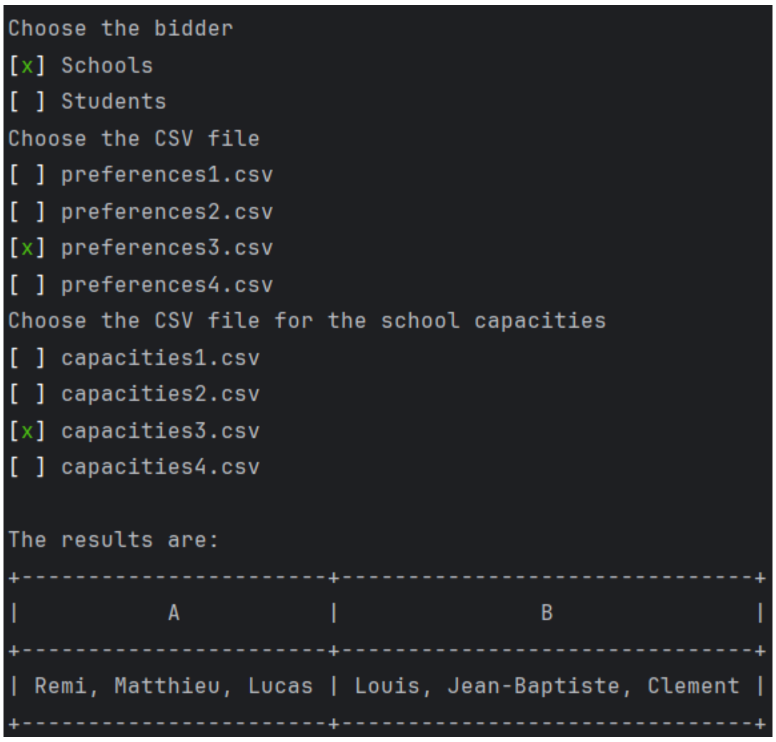
```
python3 main.py
```

You would be asked to choose some parameters in order to setup algorithm configuration:

- The bidder (student or school).
- The schools capacities CSV file.
- The preferences input CSV file.

## 2 ) Running example

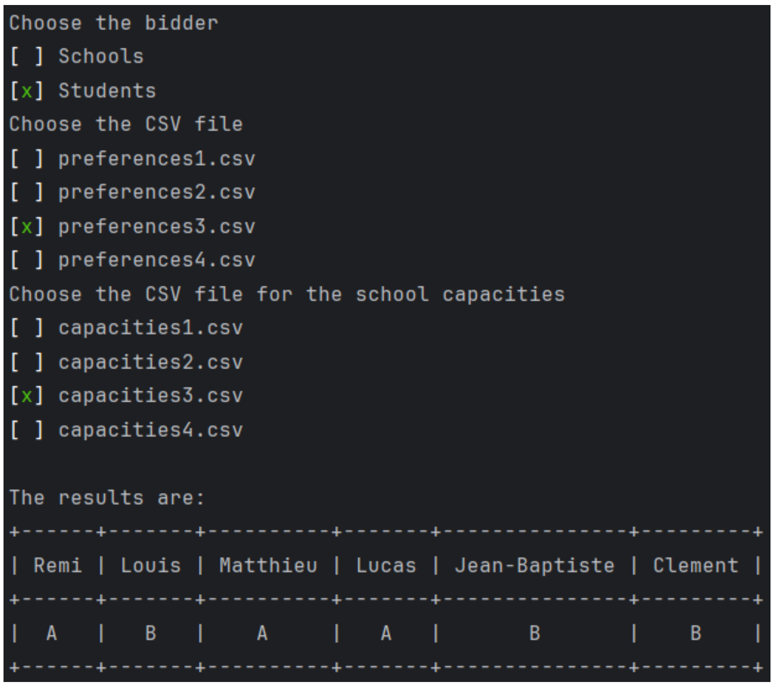
Here is two scenarios with the same dataset but different bidder (one with shcools and the other one with students).



```
Choose the bidder
[x] Schools
[ ] Students
Choose the CSV file
[ ] preferences1.csv
[ ] preferences2.csv
[x] preferences3.csv
[ ] preferences4.csv
Choose the CSV file for the school capacities
[ ] capacities1.csv
[ ] capacities2.csv
[x] capacities3.csv
[ ] capacities4.csv

The results are:
+-----------------------+-------------------------------+
|           A           |               B               |
+-----------------------+-------------------------------+
| Remi, Matthieu, Lucas | Louis, Jean-Baptiste, Clement |
+-----------------------+-------------------------------+
```

Figure 1: Scenario with schools as bidder



```
Choose the bidder
[ ] Schools
[x] Students
Choose the CSV file
[ ] preferences1.csv
[ ] preferences2.csv
[x] preferences3.csv
[ ] preferences4.csv
Choose the CSV file for the school capacities
[ ] capacities1.csv
[ ] capacities2.csv
[x] capacities3.csv
[ ] capacities4.csv

The results are:
+------+-------+----------+-------+--------------+---------+
| Remi | Louis | Matthieu | Lucas | Jean-Baptiste | Clement |
+------+-------+----------+-------+--------------+---------+
|  A   |   B   |    A     |   A   |      B       |    B    |
+------+-------+----------+-------+--------------+---------+
```

Figure 2: Scenario with students as bidder

## 3 ) File structure

The project has the following structure :

```
assets/
        capacities/
                    → this folder contains all the capacities files for the project
preferences/
                    → this folder contains all the preferences files for he project
Initializer.py      → contains all the methods to initialize the project
main.py             → contains the main algorithms
```

### 3.1. CSV file schools capacities

Input files must be located in the **assets/capacities folder.** The file must be a CSV file with the following format:

| School 1 | School 2 | … | School n |
|---|---|---|---|
| X | y | … | z |

Where x, y, …, z are integers representing the capacities of the schools.

**NB: the name of the school must be corresponding to the name of the school in the preferences file.**

Here is an example of what could be found in a capacity CSV file :

```
School 1, School 2, School 3
2, 3, 1
```

### 3.2. CSV file preferences input

Input files must be located in the **assets/preferences** folder. The CSV file represent the following table:

| Student | School 1 | School 2 | … | School n |
|---|---|---|---|---|
| Student 1 | prefs. of 1,1 | prefs. of 1,2 | … | prefs. of 1,n |
| Student 2 | prefs. of 2,1 | prefs. of 2,2 | … | prefs. of 2,n |
| … | … | … | … | … |
| Student m | prefs. of m,1 | prefs. of m,2 | … | prefs. of m,n |

Each preference cell must be in the following format: x,y where x is the student's preference for that school and y is the school preference for student x.

Here is an example of what could be found in a preference CSV file:

```
,School 1,School 2,School 3
Student 1, "1,3", "2,1", "3,1"
Student 2, "2,1", "1,3", "3,2"
Student 3, "3,2", "2,2", "1,3"
```

*Note that the header row must start with a comma.*

The program **will not** check the validity of the preferences file as well as the capacities file and **will crash** if the files are not correctly formatted.

## 4 ) Initializer class

**Initializer** allows to initialize the data structure for the first iteration of the marriage algorithm. To better understand the data structure, we will explain the **Initializer** class with scenario 3 (capacities3.csv and preferences3.csv) as an example. With students bidder, here is are the attributes initialized:

- **student_pref:** The students' preferences from **preferences2.csv** with the following format:

```
{
  "Remi": ["A", "B"],
  "Louis": ["B", "A"],
  "Matthieu": ["A", "B"],
  "Lucas": ["A", "B"],
  "Jean-Baptiste": ["B", "A"],
  "Clement": ["A", "B"]
}
```

- **school_pref:** The schools' preferences from **preferences2.csv** with the following format:

```
{
  "A": ["Remi", "Matthieu", "Jean-Baptiste", "Lucas", "Clement", "Louis"],
  "B": ["Remi", "Matthieu", "Lucas", "Jean-Baptiste", "Clement", "Louis"]
}
```

- **school_capacities:** The capacity for each school from **capacities2.csv** with the following format:

```
{
   "A": 3,
   "B": 3
}
```

- **romeo_dict:** The preferences for the romeos' dictionary where is equal to **student_pref** for school bidder or **school_pref** for student bidder. So in our case, it's equals to **school_pref**.
- **juliette_pref:** The juliettes' preferences where is equal to **student_pref** for student bidder or **school_pref** for school bidder. So in our case, it's equals to **student_pref**.
- **juliette_dict:** The juliettes' dictionary is the first iteration of the marriage algorithm based on romeo dictionary and school capacities.

```
{
   "Remi": [],
   "Louis": ["B"],
   "Matthieu": ["D"],
   "Lucas": ["A", "C"]
}
```

- **school_enrollments:** school enrolment at a given time T. So in our case, at the beginning of the algorithm, it's equal to:

```
{
   "A": 3,
   "B": 3
}
```

## 5 ) Main algorithm

The **stable_marriage** function implements the Stable Marriage algorithm, which is a solution concept in game theory for the problem of finding a stable matching between two equally sized sets of elements given an ordering of preferences for each element. In this case, the two sets are schools and students.

The function takes one parameter, bidder, which determines whether the schools or the students are the ones proposing in the algorithm. The algorithm operates in a series of "days" (iterations), which are represented by a while loop. On each day, the following steps are performed:

1. The function iterates over all the "juliettes" (which can be either schools or students, depending on the bidder parameter). For each juliette, it checks if the number of "romeos" (the other set) in its list is greater than its capacity. The capacity is 1 for students and the school's capacity for schools.

2. If the juliette's list is over capacity, it removes the least preferred romeo from its list. This is done by the remove_least_pref function, which finds the least preferred romeo by filtering the juliette's preference list with its current list of romeos and taking the last element.

3. After removing the least preferred romeo, the function gets the next preferred juliette for this romeo using the get_next_pref_juliette function. This function finds the next preferred juliette by getting the index of the current juliette in the romeo's preference list and returning the next element if it exists.

4. If the next preferred juliette exists, the function adds the least preferred romeo to its list. If the bidder is "Schools", this is done directly. If the bidder is "Students", the function also decreases the romeo's (school's) enrollment and continues to add the least preferred romeo to the next preferred juliettes' lists until the romeo's capacity is reached or there are no more preferred juliettes.

5. After processing a juliette, the function increments the day counter and breaks the loop to start a new day.

6. If the function goes through all the juliettes without finding any that are over capacity, it concludes that the matching is stable and exits the loop.

The function returns the number of days it took to reach a stable matching.
This algorithm ensures that the final matching is stable, meaning that there are no two elements (a school and a student in this case) who would both prefer each other over their current partners.