Lab 01: Stable Matching and the Gale-Shapley algorithm

Theory Practice (14 pts)

Question 1 (5 pts)

As a second-year student who has not yet been romantically involved with anyone, your best friend invites you to join a date with two other single individuals in hopes of sparking a potential romantic connection. You eagerly anticipate the event and prepare a list of your preferences, considering various scenarios. Each person involved in the date has their own preference list. Provide a scenario where there can be more than one set of couples that are stable over time.

Question 2 (4 pts)

With the same circumstance as **Question 1** - you are still lonely, but you now join a bigger dating event, held by a huge company. There would be many people there, let say N boys and N girls. The company choose to apply Gale-Shapley algorithm for matching couples. In the worst case, in which all the boys share the same preference list, how many iteration does it take so that everyone can find their soulmates?

Question 3 (3 pts)

Consider a bipartite graph $G = (V, E)$ with $X \cup Y = V$ and a set of weights W where w_{ab}, $a, b \in V$. A matching M is stable if there do not exist edges $(x, y), (x'y') \in M$ with $x, x' \in X$ and $y, y' \in Y$ such that $w_{ab'} > w_{ab} \wedge w_{b'a} > w_{ba}$. If M is stable, and all vertices in X shares the same set of weights to all vertices in Y, prove that M is unique.

Question 4 (2 pts)

You're hosting your own student chess league where your univerisity team competes against another university. The first stage is in round-robin format. This means each player from one university plays every other player at the opposing university once. To do this, each player has to provide a priority list for which opponent they want to be playing on this given game. As an organizer, you must make sure that every player should be able to be matched with the top choice possible for their first game. Is it possible for there to be a set of players' preferences where you can randomly generate the schedule and still get a stable matching everytime? Justify your answer.

Programming Practice (11 pts)

Problem 1 (5 pts)

Given a stable matching problem of 2 groups `A` and `B` with `n` members each. A matching `M` is generated. Write a Python program that takes as input:

- `n`: size of a stable matching problem,
- `P`: set of preferences of size `2n` × `n`,
- `M`: set of matching of size `2n`,

And print whether the matching `M` is stable under the preferences set `P`. To simplify, you can assume the entries for `P` are always in sorted order of the members' names (see example below). The matching `M` is from the `A` group perspective (a pair `X:Y` means `X` from `A` is paired to `Y` from `B`). For example, if we have a stable matching problem with `n = 3`, and the preferences:

```
A1: B1 > B2 > B3
A2: B2 > B1 > B3
A3: B1 > B2 > B3

B1: A2 > A1 > A3
B2: A1 > A2 > A3
B3: A1 > A2 > A3
```

Given the matching `M: {A1:B3, A2:B2, A3:B1}`, the program should print `False`. For this example, the input for your program will look like:

```
python V2025001_Week1_P1.py
3
B1 B2 B3
B2 B1 B3
B1 B2 B3
A2 A1 A3
A1 A2 A3
A1 A2 A3
A1 B3 A2 B2 A3 B1
False
```

Given the matching `M: {A1:B1, A2:B2, A3:B3}`, the program should print `True`. For this example, the input for your program will look like:

```
python V2025001_Week1_P1.py
3
B1 B2 B3
B2 B1 B3
B1 B2 B3
A2 A1 A3
A1 A2 A3
A1 A2 A3
A1 B1 A2 B2 A3 B3
True
```

Problem 2 (4 pts)

Now have to implement the algorithm itself based on the Pseudo code provided during the lecture. Write a Python program that takes as input:

- `n`: size of a stable matching problem,

- `P`: set of preferences of size `2n` × `n`

and print out the sceen list M represent for matching couples. M is produced by choosing the first n peoples as *proposers*, the iteration order is the same as the input order.
Format. The input format is all integer, first people sets is indexed from $0 \to (n-1)$, while the second group is from $n \to (2n-1)$. The output M should be a list of matchings, in which element $M[i] = k$ denote the matching of individual k from the first group with individual $(i+n)$ from the second group. For example, if we have a stable matching problem with `n = 4`, and the preferences:

```
0: 6 > 4 > 5 > 7
1: 6 > 5 > 7 > 4
2: 7 > 6 > 4 > 5
3: 7 > 6 > 4 > 5

4: 1 > 0 > 3 > 2
5: 2 > 1 > 0 > 3
6: 0 > 1 > 2 > 3
7: 2 > 3 > 1 > 0
```

When selecting the first n peoples as *proposers* in the order $0 \to 1 \to 2 \to 3$, the program should print `[3, 1, 0, 2]`. For this example, the input for your program will look like:

```
python V2025001_Week1_P2.py
4
6 4 5 7
6 5 7 4
7 6 4 5
7 6 4 5
1 0 3 2
2 1 0 3
0 1 2 3
2 1 3 0
[3, 1, 0, 2]
```

Problem 3 (2 pts)

Let's take a more general approach to our Algorithm Design class, where we have N students and $M(M < N)$ Teaching Assistants. The subject material may be difficult for some students, so the Professors suggest assigning each TA to up to K students based on preferences of both the TAs and students. It's important to note that not all TAs know every student and may only have preferences for a subset of students, **and the same goes for students**.
To solve this problem, we need to extend the Gale-Shapley algorithm to find the best possible matches between TAs and students. There are some additional constraints to simplify the problem:

- Since students are the central of our class, students' preferences are the first to be considered.
- Matching should be done in the order of students preference' lists submission. (follow the read input sequence during implementation).
- If a student cannot find a stable match (can be owing to the TAs they choose do not listed him/her; or are coupled with other students and reach their capacities), he/she should be assigned the value of *"unmatch"*.

For example, suppose we have the input as follow:

```
Students:
    C: V > M
    H: M
    A: V > M
    B: V > P
    J: P > M > V
TAs:
    V: A > H
    M: A > J > H > B
    P: J > C > H > A > B
K: 2
```

Then the output should be a dictionary with keys are the students and their values are their assigned TA. In the above example is:

```
{
    'C': 'unmatch',
    'H': 'M',
    'A': 'V',
    'B': 'P',
    'J': 'P'
}
```

The input for your program would be in the following format:

- N - number of students
- Next N lines are the students' preference lists, in which the 1st elements are the identities of the students themselves
- M - number of students
- Next M lines are the TAs' preference lists, in which the 1st elements are the identities of the TAs themselves
- K - TA's maximum capacity

For the above example, the input for your program will look like:

```
python V2025001_Week1_P3.py
5
C V M
H M
A V M
B V P
J P M V
3
V A H
M A J H B
P J C H A B
2
{'C': 'unmatch', 'H': 'M', 'A': 'V', 'B': 'P', 'J': 'P'}
```