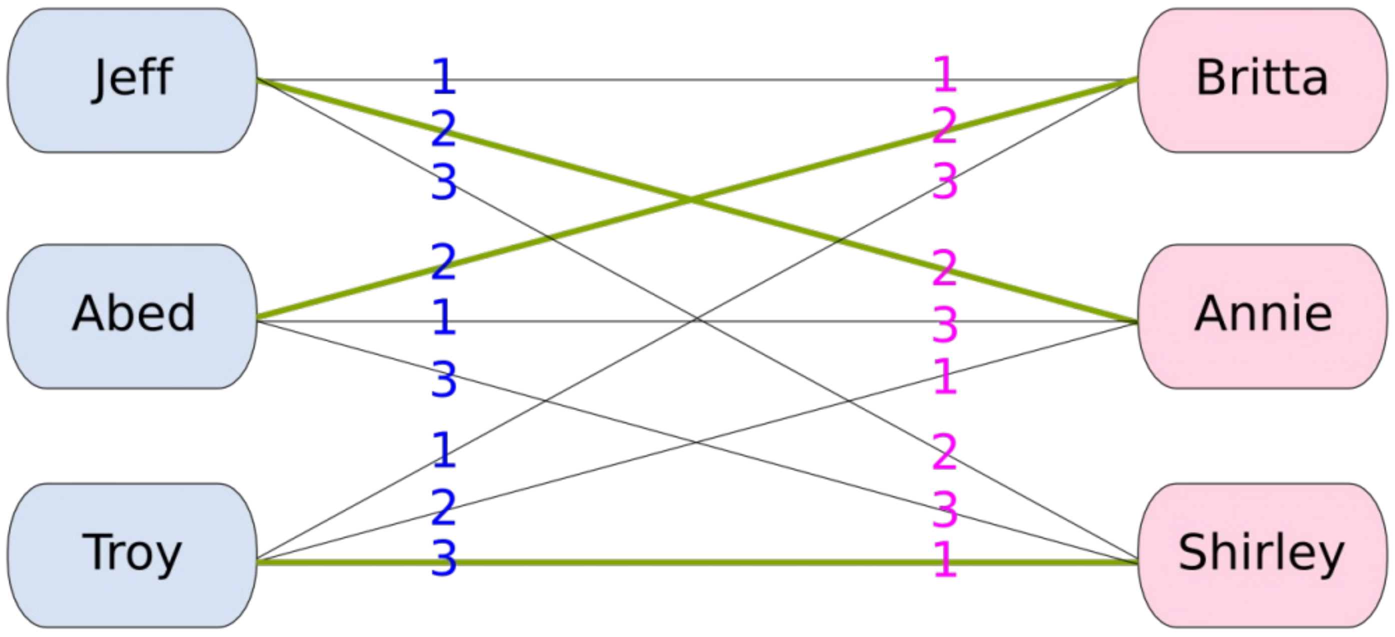
Stable Marriage Problem

The stable marriage (SMP) or pairing problem, it consists in finding a stable pair for two equally sized groups of elements where each of them has an order of preferences for the elements of the other group. A stable match is achieved when neither of the parties have a better possible match. This problem can take many forms, such as students applying for universities, job openings and applicants, assigning users to servers in a large distributed Internet service or finding the best combinations of couples in a dating platform, which is the one we will focus on. We will consider there are only males and females on the platform and that males are only attracted to females and vice versa.



Each person will be an object of the following class, they will have an assigned name, gender, list of preferences, relationship status, who theyre in a relationship with and to whom they have alredy proposed, the class also has methods to match and unmatch a couple and to check if thre's a possible better match.

In [1]:

```
class Person:

    def __init__(self, name, status, gender, preferences, engaged_to, proposed_to):
        self.name = name
        self.isFree = status
        self.gender = gender
        self.preferences = preferences
        self.engaged_to = engaged_to
        self.proposed_to = proposed_to

    def break_up(self):
        partner = self.engaged_to

        if not self.isFree:
            self.engaged_to = ""
            self.isFree = True
            print(self.name, "just unmatched   ", partner.name)

    def which_loves_more(self, person1, person2):
        for i in self.preferences:

            if i == person1.name:
                return person1
            if i == person2.name:
                return person2
    def engage(self, partner):
        self.engaged_to = partner
        self.isFree = False
        partner.engaged_to = self
        partner.isFree = False
        print(self.name, "just matched with", partner.name)
```

The male names will be set as capital letters while the females will be numbers, each person has a ranking of the elements of the other group from most prefered to least.

In [2]:

```
males = [Person("A",True,"male",["1", "2", "3", "4"],"",[]),
         Person("B",True,"male",["1", "4", "3", "2"],"",[]),
         Person("C",True,"male",["2", "1", "3", "4"],"",[]),
         Person("D",True,"male",["4", "2", "3", "1"],"",[])
        ]

females = [Person("1",True,"female",["D", "C", "A", "B"],"",[]),
           Person("2",True,"female",["B", "D", "A", "C"],"",[]),
           Person("3",True,"female",["D", "A", "B", "C"],"",[]),
           Person("4",True,"female",["C", "B", "A", "D"],"",[])
          ]
```

Now we can iterate trough the dataframes (we chose the males as reference) and check if each person is single, has a parter and if theyre a beNow we can iterate trough the dataframes (we chose the males as reference) and check if each person is single, has a partner and if they’re a better match with someone else.

In [3]:

```
def main(males, females):
    while (True):
        possibleCouples = len(males)
        couples = 1
        for m in males:

            if (m.isFree == False) and (len(m.proposed_to) != possibleCouples):
                couples += 1
                if couples == possibleCouples:
                    print("\nFinished!")
                    return
            else:
                couples -=1

            for x in range(0, possibleCouples):
                if m.engaged_to == "":
                    if x not in m.proposed_to:
                        m.proposed_to.append(x)

                        woman_name = m.preferences[x]
                        for w in females:
                            if woman_name == w.name:
                                woman = w

                        if woman.engaged_to != "":
                            current = woman.engaged_to
                            likes_more = woman.which_loves_more(current, m)

                            likes_more.engage(woman)

                            if likes_more != current:
                                current.break_up()
                        else:
                            m.engage(woman)
```

Testing

Finally let's test the program the groups used will be the males and females presented above and the best pairs should be:

- **A - 3**
- **B - 4**
- **C - 1**
- **D - 2**

In [4]:

```
main(males, females)

for m in males:
    guy = m.name
    girl = (m.engaged_to).name
    print(guy," <---> ", girl)
```

```
A just matched with 1
A just matched with 1
B just matched with 4
C just matched with 2
B just matched with 4
D just matched with 2
C just unmatched    2
C just matched with 1
A just unmatched    1
D just matched with 2
A just matched with 3

Finished!
A  <--->  3
B  <--->  4
C  <--->  1
D  <--->  2
```

We can now test the algorithm again with larger groups to see if the results stay true.

In [5]:

```
males = [Person("A",True,"male",["1", "2", "3", "4", "5", "6"],"",[]),
         Person("B",True,"male",["2", "3", "5", "1", "4", "6"],"",[]),
         Person("C",True,"male",["2", "3", "1", "5", "4", "6"],"",[]),
         Person("D",True,"male",["2", "3", "1", "5", "4", "6"],"",[]),
         Person("E",True,"male",["1", "3", "2", "5", "4", "6"],"",[]),
         Person("F",True,"male",["2", "3", "1", "5", "4", "6"],"",[]),
        ]

females = [Person("1",True,"female",["A", "B", "C", "E", "D", "F"],"",[]),
           Person("2",True,"female",["C", "A", "B", "D", "E", "F"],"",[]),
           Person("3",True,"female",["C", "A", "B", "D", "E", "F"],"",[]),
           Person("4",True,"female",["A", "B", "C", "E", "D", "F"],"",[]),
           Person("5",True,"female",["A", "B", "C", "E", "D", "F"],"",[]),
           Person("6",True,"female",["B", "A", "C", "D", "F", "E"],"",[]),
          ]
```

The best matches should be:

- **A - 1**
- **B - 3**
- **C - 2**
- **D - 4**
- **E - 5**
- **F - 6**

```
In [6]:
```

```python
main(males, females)

for m in males:
    guy = m.name
    girl = (m.engaged_to).name
    print(guy," <---> ", girl)
```

```
A just matched with 1
B just matched with 2
C just matched with 2
B just unmatched    2
C just matched with 2
D just matched with 3
A just matched with 1
D just matched with 3
C just matched with 2
E just matched with 5
C just matched with 2
D just matched with 3
A just matched with 1
E just matched with 5
F just matched with 4
B just matched with 3
D just unmatched    3
A just matched with 1
E just matched with 5
D just matched with 4
F just unmatched    4
F just matched with 6

Finished!
A  <--->  1
B  <--->  3
C  <--->  2
D  <--->  4
E  <--->  5
F  <--->  6
```