Beykoz University

Department of "Computer Engineering"

"Graph Theory Applications"

- Project Final Report -

- Stable Matching Problem -

Lecturer: Gizem Temalcan

Leyla Abdullayeva - 1904010038

Table of contents:

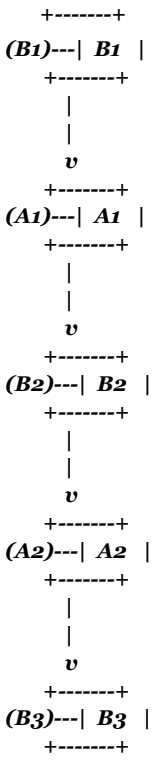
Explanation of Stable Matching:

In graph theory, a stable matching is a matching between two sets of elements that satisfies certain stability conditions. A stable matching is typically used to model relationships between two groups of people, where one group consists of men and the other group consists of women. The goal is to find a matching between the men and the women such that no man and woman would both be better off if they were paired with someone else.

There are several different algorithms that can be used to find a stable matching in a graph. One common algorithm is the Gale-Shapley algorithm, which is an iterative process in which men propose to women, and women either accept or reject the proposals. The algorithm terminates when a stable matching is found or it can be proved that no stable matching exists.

Stable matchings have a wide range of applications, including in economics, computer science, and sociology. They are often used to model relationships between employers and employees, schools and students, and hospitals and residents.

Example problem:



In this example, there are three men (B1, B2, and B3) and three women (A1, A2, and A3). Each person has a list of preferences for potential partners, with the most preferred partner listed first. For example, B1 prefers A1 to A2 to A3, while A2 prefers B2 to B3 to B1.

A stable matching is a matching in which no person would prefer to be matched with someone else rather than their current partner. In this example, one possible stable matching is B1-A1, B2-A2, and B3-A3. No person in this matching would prefer to be matched with someone other than their current partner.

Algorithm & Example Program : Gale-Shapley

One algorithm for finding a stable matching is the Gale-Shapley algorithm, which proceeds as follows:

1. Initialize the matching to be empty.
2. For each man, iterate through their preferences in order.
3. For each woman, if she is not currently matched, match her with the man.
4. If the woman is already matched, check if she prefers the current man to her current partner. If she does, match her with the current man and add her current partner to the list of free men.
5. Repeat until there are no free men left.

```
def stable_matching(men, women):
    # Initialize the matching to be empty
    matching = {}
    for man in men:
        matching[man] = None
    free_men = list(men)
    # Iterate until there are no free men left
    while free_men:
        man = free_men.pop(0)
        for women in men[man]:
            # Check if the woman is free
            if matching[woman] is None:
                matching[woman] = man
                break
            # If the woman is not free, check if she prefers the
current man to her current partner
            elif woman_prefers_man(woman, man, matching[woman], women):
                matching[woman] = man
                free_men.append(matching[woman])
                break
    return matching
def woman_prefers_man(woman, man1, man2, women):
    # Check if woman prefers man1 to man2
    preferences = women[woman]
```

men.csv

```
B1,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B2,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B3,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B4,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B5,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B6,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B7,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B8,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B9,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
B10,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10
```

"women.csv"

```
A1,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A2,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A3,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A4,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A5,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A6,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A7,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A8,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A9,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
A10,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10
```

Live Demo of Python program related to project:

- Sample preference initialization.

guyprefers is the preference profile of boys and ***galprefers*** is the preference profile of girls. In each dictionary, the keys denote the person and the values denote the strict preference list of the person the key corresponds to.

This preference profile was pulled from Rosetta Code.

```
guyprefers = {
   'abe':  ['abi', 'eve', 'cath', 'ivy', 'jan', 'dee', 'fay', 'bea', 'hope', 'gay'],
   'bob':  ['cath', 'hope', 'abi', 'dee', 'eve', 'fay', 'bea', 'jan', 'ivy', 'gay'],
   'col':  ['hope', 'eve', 'abi', 'dee', 'bea', 'fay', 'ivy', 'gay', 'cath', 'jan'],
   'dan':  ['ivy', 'fay', 'dee', 'gay', 'hope', 'eve', 'jan', 'bea', 'cath', 'abi'],
   'ed':   ['jan', 'dee', 'bea', 'cath', 'fay', 'eve', 'abi', 'ivy', 'hope', 'gay'],
   'fred': ['bea', 'abi', 'dee', 'gay', 'eve', 'ivy', 'cath', 'jan', 'hope', 'fay'],
   'gav':  ['gay', 'eve', 'ivy', 'bea', 'cath', 'abi', 'dee', 'hope', 'jan', 'fay'],
   'hal':  ['abi', 'eve', 'hope', 'fay', 'ivy', 'cath', 'jan', 'bea', 'gay', 'dee'],
   'ian':  ['hope', 'cath', 'dee', 'gay', 'bea', 'abi', 'fay', 'ivy', 'jan', 'eve'],
   'jon':  ['abi', 'fay', 'jan', 'gay', 'eve', 'bea', 'dee', 'cath', 'ivy', 'hope']
}


galprefers = {
   'abi':  ['bob', 'fred', 'jon', 'gav', 'ian', 'abe', 'dan', 'ed', 'col', 'hal'],
   'bea':  ['bob', 'abe', 'col', 'fred', 'gav', 'dan', 'ian', 'ed', 'jon', 'hal'],
   'cath': ['fred', 'bob', 'ed', 'gav', 'hal', 'col', 'ian', 'abe', 'dan', 'jon'],
   'dee':  ['fred', 'jon', 'col', 'abe', 'ian', 'hal', 'gav', 'dan', 'bob', 'ed'],
   'eve':  ['jon', 'hal', 'fred', 'dan', 'abe', 'gav', 'col', 'ed', 'ian', 'bob'],
   'fay':  ['bob', 'abe', 'ed', 'ian', 'jon', 'dan', 'fred', 'gav', 'col', 'hal'],
   'gay':  ['jon', 'gav', 'hal', 'fred', 'bob', 'abe', 'col', 'ed', 'dan', 'ian'],
   'hope': ['gav', 'jon', 'bob', 'abe', 'ian', 'dan', 'hal', 'ed', 'col', 'fred'],
   'ivy':  ['ian', 'col', 'hal', 'gav', 'fred', 'bob', 'abe', 'ed', 'jon', 'dan'],
   'jan':  ['ed', 'hal', 'gav', 'abe', 'bob', 'jon', 'col', 'ian', 'fred', 'dan']
}
```

```
model = MarriageModel(guyprefers, galprefers)
mu = model.Deferred_Acceptance()
mu
{'dan': 'fay',
 'col': 'dee',
 'abe': 'ivy',
 'jon': 'abi',
 'bob': 'cath',
 'hal': 'eve',
 'ian': 'hope',
 'ed': 'jan',
 'fred': 'bea',
 'gav': 'gay'}

galprefers['abi'] = ['bob', 'fred']

model = MarriageModel(guyprefers, galprefers)
mu_2 = model.Deferred_Acceptance()
mu_2
{'abi': None,
 'bob': 'cath',
 'fred': 'bea',
 'ed': 'jan',
 'hal': 'eve',
 'ian': 'hope',
 'gav': 'gay',
 'col': 'dee',
 'jon': 'fay',
 'abe': 'ivy',
 'dan': None}
```

```
Tentative matching after Round 1:
{'jon': 'abi', 'bob': 'cath', 'ian': 'hope', 'dan': 'ivy', 'ed': 'jan',
'fred': 'bea', 'gav': 'gay'}
Tentative matching after Round 2:
{'hal': 'eve', 'jon': 'abi', 'bob': 'cath', 'ian': 'hope', 'dan':
'ivy', 'ed': 'jan', 'fred': 'bea', 'gav': 'gay'}
Tentative matching after Round 3:
{'jon': 'abi', 'bob': 'cath', 'hal': 'eve', 'ian': 'hope', 'dan':
'ivy', 'ed': 'jan', 'fred': 'bea', 'gav': 'gay'}
Tentative matching after Round 4:
{'col': 'dee', 'abe': 'ivy', 'jon': 'abi', 'bob': 'cath', 'hal': 'eve',
'ian': 'hope', 'ed': 'jan', 'fred': 'bea', 'gav': 'gay'}
```

Source code for my project - Github Link:

https://github.com/leyviya/stable-matching-problem

References:

https://en.wikipedia.org/wiki/Gale%E2%80%93Shapley_algorithm

https://www.geeksforgeeks.org/stable-marriage-problem/

https://www.inc.com/burt-helm/gale-shapley-algorithm-innovation-nobel-prize.html#:~:text=The%20winners%20of%20the%202012,urban%20students%20with%20magnet%20schools.

https://towardsdatascience.com/gale-shapley-algorithm-simply-explained-caa344e643c2

https://towardsdatascience.com/stable-matching-as-a-game-a68c279d70b

https://en.wikipedia.org/wiki/Stable_marriage_problem