Problem 1:

→ Since the goal is to find a stable solution for a matching between M and W, and that all of $m \in M$ preference lists are identical this means that the stable matching between M and W and how each m and w is matched depends on the position of m's in the preference lists of w.

→ So as we traverse the preference lists of each m in M, because each m has the same preference list $[w_1, w_2, \dots, w_n]$ each m_i would first propose to w_1, then w_2, ... until w_n in that exact order.

w_1 recieves proposals from $\{m_1, \dots, m_n\}$ and is matched with the m that is highest on their preference list that is also not engaged

w_2 recieves proposals from $\{m_1, \dots, m_{n-1}\}$ and is matched with the m that is highest on their preference list that is also not engaged

⋮

Continues until all of $m \in M$ is matched

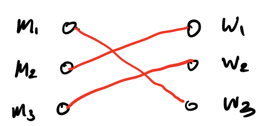
→ This means that our final solution would be that each w would get their highest m matched in order meaning w_1 will get the highest m on their list, and w_2 would get the highest m excluding the m that was previosly matched with w_1. So we can say that each w will get matched with the highest m on their list excluding the previously engaged m's. The reason why this is the only stable solution is because each m and w is matded and all both prefer eachother.

So as an example:

$m_1: [w_1, w_2, w_3]$ $\quad w_1: [m_2, m_1, m_3]$

$m_2: [w_1, w_2, w_3]$ $\quad w_2: [m_3, m_2, m_1]$

$m_3: [w_1, w_2, w_3]$ $\quad w_3: [m_3, m_1, m_2]$



- w_1 is matched first with m_2
- w_2 is matched with m_3
- w_3 is matched with m_1, because w_2 recieves a proposal from m_3 and is matched first

↳ This is because each m has identical preference lists so each w is matched in order from $1 \dots n$

Problem 2:

M

m_1 : [w_1, w_2, w_3]

m_2 : [w_2, w_1, w_3]

m_3 : [w_1, w_2, w_3]

O *matched with

W Truth

w_1 : [m_2, m_1, m_3]

w_2 : [m_1, m_2, m_3]

w_3 : [m_1, m_2, m_3]

swap

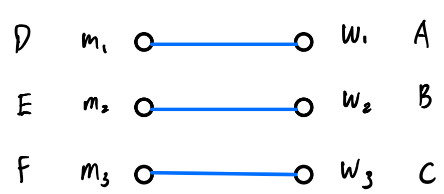
W Lie

w_1 : [m_2, m_3, m_1]

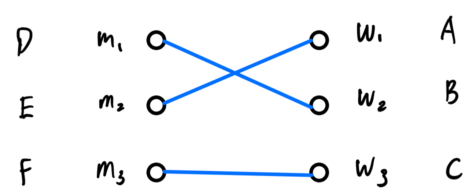
w_2 : [m_1, m_2, m_3]

w_3 : [m_1, m_2, m_3]

Truth :



Lie :



→ For this example if we have make a preference list M and two preference lists W where w_1 will be the female candidate that lies. We could run the G-S algorithm on both to see if w_1 achieves a matching with a m_i of higher rank. In the example above the w_1 with its true list has the ranking [m_2, m_1, m_3] and the w_2 with the fake list has the ranking [m_2, m_3, m_1]

Steps for $I = (M, W_{truth})$:

1. m_1 proposes to w_1 → w_1 accepts
2. m_2 proposes to w_2 → w_2 accepts
3. m_3 proposes to w_1 → w_1 rejects
4. m_3 proposes to w_2 → w_2 rejects
5. m_3 proposes to w_3 → w_3 accepts

- We can see that for w_1 by lying about its own preferences with switching the position of m_1 and m_3. w_1 ended up matching with m_2 which has a rank of 1 in contrast to m_1 which has a rank of 2. This shows that w_1 achieved a better outcome by lying

Steps for $I = (M, W_{Lie})$

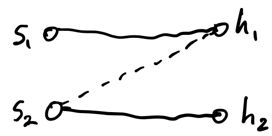
1. m_1 proposes to w_1 → w_1 accepts
2. m_2 proposes to w_2 → w_2 accepts
3. m_3 proposes to w_1 → w_1 rejects m_1
   w_1 accepts m_3
4. m_1 proposes to w_2 → w_2 rejects m_2
   w_2 accepts m_1
5. m_2 proposes to w_1 → w_1 rejects m_3
   w_1 accepts m_2
6. m_3 proposes to w_2 → w_2 rejects m_3
7. m_3 proposes to w_3 → w_3 accepts

Problem 3:

$k_i > h_i$

Type 1:

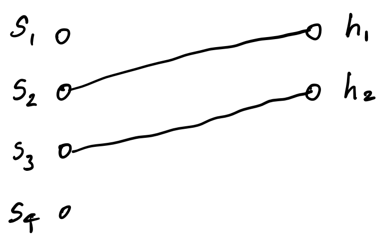
Type 2:



a)

$s_1 : [h_1, h_2]$
$s_2 : [h_2, h_1]$
$s_3 : [h_2, h_1]$
$s_4 : [h_1, h_2]$

$h_1 : [s_3, s_2, s_1, s_4]$
$h_2 : [s_3, s_4, s_2, s_1]$



→ In this case I would modify the algorithm such that it only terminates when all h's are matched and all h's are matched with their highest possible preference

b) → I can adapt the algorithm such that it would firstly look at the number of h's present. Afterwards each h will cut its preference lists, s.t. if there are **m** h's present it will exclude preferences from **m** onwards. Lastly, I will look at all of the h's and only keep s's that remained.

e.g.

~~$s_1 : [h_1, h_2]$~~
$s_2 : [h_2, h_1]$
$s_3 : [h_2, h_1]$
$s_4 : [h_1, h_2]$

$h_1 : [s_3, s_2 \mid \cancel{s_1, s_4}]$
$h_2 : [s_3, s_4 \mid \cancel{s_2, s_1}]$

→

$h_1 : [s_3, s_2]$
$h_2 : [s_3, s_4]$

$\{s_2, s_3, s_4\}$ remains
so we can mark s_1 as unacceptable

Problem 4:

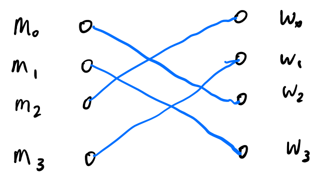
$m_0 : [w_2, w_1, w_3, w_0]$ $\quad w_0 : [m_0, m_2, m_1, m_3]$

$m_1 : [w_0, w_1, w_3, w_2]$ $\quad w_1 : [m_2, m_0, m_3, m_1]$

$m_2 : [w_0, w_1, w_2, w_3]$ $\quad w_2 : [m_3, m_2, m_1, m_0]$

$m_3 : [w_0, w_1, w_2, w_3]$ $\quad w_3 : [m_2, m_3, m_1, m_0]$

Final Matching for given Example in Question:



$\rightarrow$ $[(m_0, w_2), (m_1, w_3), (m_2, w_0), (m_3, w_1)]$

$[(0, 2), (1, 3), (2, 0), (3, 1)]$

```python
def gale_shapley(m_prefs, w_prefs):
    m_pointers = [0 for m in range(len(m_prefs))]
    w_matches = {w : None for w in range(len(w_prefs))}
    all_males = set([i for i in range(0, len(m_prefs))])
    engaged = set()
    matches = []
    while engaged != all_males:
        for male in m_prefs:
            if male in engaged:
                continue
            # Propose to the most preferable female on their list
            highest_w = m_prefs[male][m_pointers[male]]
            m_pointers[male] += 1
            print(f'male {male} proposes to female {highest_w}')
            # Female will have to decide to either accept or reject the current offer made to them
            if not w_matches[highest_w]:
                w_matches[highest_w] = (male, w_prefs[highest_w].index(male))
                engaged.add(male)
                print(f'--> male {male}\'s proposal is accepted')
            else:
                if w_matches[highest_w][1] > w_prefs[highest_w].index(male):
                    engaged.remove(w_matches[highest_w][0])
                    print(f'--> female {highest_w} rejects male {w_matches[highest_w][0]}')
                    w_matches[highest_w] = (male, w_prefs[highest_w].index(male))
                    print(f'--> female {highest_w} accepts male {w_matches[highest_w][0]}\'s proposal')
                    engaged.add(male)
                else:
                    print(f'--> male {male}\'s proposal is rejected')
    for w, m in w_matches.items():
        matches.append((m[0], w))
    return matches
```

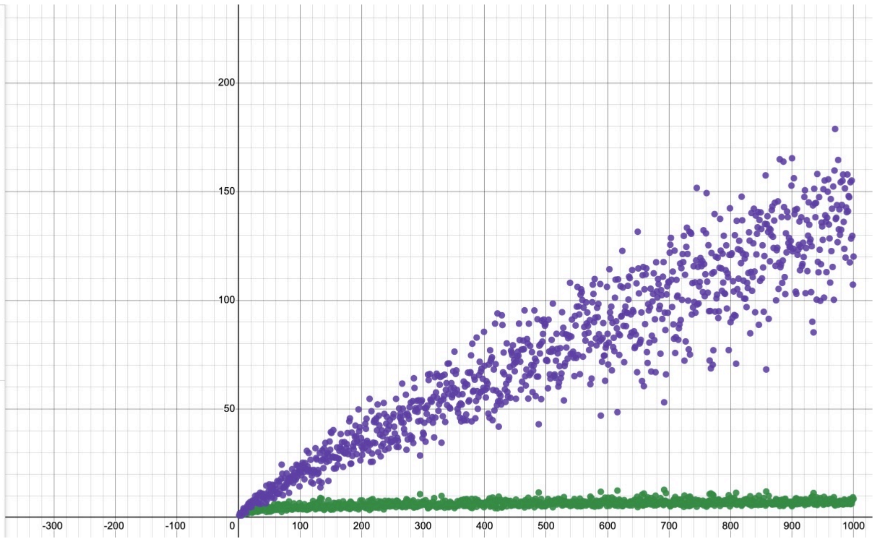
```
male 0 proposes to female 2
--> male 0's proposal is accepted
male 1 proposes to female 0
--> male 1's proposal is accepted
male 2 proposes to female 0
--> female 0 rejects male 1
--> female 0 accepts male 2's proposal
male 3 proposes to female 0
--> male 3's proposal is rejected
male 1 proposes to female 1
--> male 1's proposal is accepted
male 3 proposes to female 1
--> female 1 rejects male 1
--> female 1 accepts male 3's proposal
male 1 proposes to female 3
--> male 1's proposal is accepted
Result Matching: [(2, 0), (3, 1), (0, 2), (1, 3)]
```

$\rightarrow$ My implementation takes in male and female preferences as dictionaries $\{0: [1, 2, 3], \ldots\}$, means m_0 or f_0 prefers f_1 to f_2 to f_3 or m_1 to m_2 to m_3. then outputs all the proposals made by males and if the female accepted or rejected their proposal.

$\rightarrow$ The result matching is outputted at the very end and it reads as $[(m_i, f_i), \ldots]$ where m_i is matched with f_i

Problem 5:

x_1	m_1	f_1
1	1.0	1.0
2	1.0	2.0
3	1.3333333333333333	2.0
4	1.5	2.25
5	2.0	1.0
6	2.1666666666666665	2.33333333333333
7	1.5714285714285714	3.142857142857
989 more rows Show all		
997	6.3901705115346035	155.08726178535
998	7.741482965931864	129.63126252505
999	9.36136136136136	107.20720720720
1000	8.64	120.109



→ We can see that the goodness for males are better as a whole for n up to 1000 because males have a lower goodness meaning that on average that are matched with a partner higher on their preference list. Whereas females has a worse goodness as they are matched with partners that are lower on their preference list on average.