Problem Analysis Of Stable Matching

YAO ZHAO

Propose–And–Reject Algorithm

(also known as the deferred acceptance algorithm or the Gale–Shapley algorithm)

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

Common Problems

What data structures are used for input and output ?

How to find the unmatched men efficiently?

How to efficiently query the ranking of a man in a woman's preference list?

Not to test the code sufficiently

What data structures are used for input and output?

- Analysis of Input and Output Formats
- Men's name → Men's Appearance No. （HashMap）
- Women's name → Women's Appearance No. （HashMap）
- Men's Appearance No. → Men's name （Array）
- Women's Appearance No. → Women's name （Array）
- Apparently, the preference list should be a **two-dimensional array**. Since the Appearance No. can be easily obtained from Map, it is possible to design the preference list as int [][]
- The output is a list of Women's names. The i-th Man matches the i-th Woman. Obviously, the output is OK using a String array.

Men's Preference Profile

	0th	1st	2nd	3rd	4th
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Women's Preference Profile

	0th	1st	2nd	3rd	4th
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Men's name → Men's Appearance No. (HashMap)
Victor → 0
Wyatt → 1
Xavier → 2
Yancey → 3
Zeus → 4

Men's Appearance No. → Men's name (Array)
0 → Victor
1 → Wyatt
2 → Xavier
3 → Yancey
4 → Zeus

Women's name → Women's Appearance No. (HashMap)
Amy → 0
Bertha → 1
Clare → 2
Diane → 3
Erika → 4

Women's Appearance No. → Women's name (Array)
0 → Amy
1 → Bertha
2 → Clare
3 → Diane
4 → Erika

Men's preference list(int [][]):

Men's Appearance No.	0th	1st	2nd	3rd	4th
0	1	0	3	4	2
1	3	1	0	2	4
2	1	4	2	3	0
3	0	3	2	1	4
4	1	3	0	4	2

Women's preference list(int [][]):

Women's Appearance No.	0th	1st	2nd	3rd	4th
0	4	0	1	3	2
1	2	1	3	0	4
2	1	2	3	4	0
3	0	4	3	2	1
4	3	1	4	2	0

How to find the unmatched Men efficiently?

- Queue or Stack: O(1)
- Initial, all Men are free and added to a queue
- Each iterator pop a man from the queue, and try to match, If a woman prefers this man over her current provisional partner, the woman will dump her current provisional partner who must go back to the queue.

How to find a woman of the highest rank and not be tried match before for a man?

- Simple solution: find from head to tail every time
  - But if a man was dumped by a woman, he should find lower rank women than the last woman.
  - We can use **an array** to store the current preference index of the woman.

In the following case, Victor is dumped by Bertha, go back to queue. We can record the index of Bertha. When he is popped from queue again, he can propose to Amy(the index of Bertha+1).

Men's Preference Profile

	0th	1st	2nd	3rd	4th
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Women's Preference Profile

	0th	1st	2nd	3rd	4th
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Xavier proposes to Bertha.

Men's Preference Profile

	0th	1st	2nd	3rd	4th
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Women's Preference Profile

	0th	1st	2nd	3rd	4th
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Xavier proposes to Bertha.

- Bertha dumps Victor and accepts Xavier.

How to efficiently query the ranking of a man in a women's preference list?

In the following case, Xavier proposes to Bertha. Bertha is matched. Now Bertha should find the rank of Xavier and her current partner Victor, to decide whether to accept or reject Xavier

Men's Preference Profile

	0th	1st	2nd	3rd	4th
Victor	Bertha	Amy	Diane	Erika	Clare
Wyatt	Diane	Bertha	Amy	Clare	Erika
Xavier	Bertha	Erika	Clare	Diane	Amy
Yancey	Amy	Diane	Clare	Bertha	Erika
Zeus	Bertha	Diane	Amy	Erika	Clare

Women's Preference Profile

	0th	1st	2nd	3rd	4th
Amy	Zeus	Victor	Wyatt	Yancey	Xavier
Bertha	Xavier	Wyatt	Yancey	Victor	Zeus
Clare	Wyatt	Xavier	Yancey	Zeus	Victor
Diane	Victor	Zeus	Yancey	Xavier	Wyatt
Erika	Yancey	Wyatt	Zeus	Xavier	Victor

Xavier proposes to Bertha.

- Simple solution: using a loop to find a man's rank according to the men's Appearance No. in the women's preference list. O(n)
- More efficient solution:

1、Maintain an inverse list of a women's preference list.

Index: men's appearance No. → value: men's rank

We don't need men's rank → men's appearance No.

2、using a HashMap to store men's appearance No. → value: men's rank

Women's preference inverse list?

Women's preference list(int [][]):

Women's Appearance No.	0th	1st	2nd	3rd	4th
0	4	0	1	3	2
1	2	1	3	0	4
2	1	2	3	4	0
3	0	4	3	2	1
4	3	1	4	2	0

Women's preference **inverse list**(int [][]):

Women's Appearance No.	0	1	2	3	4
0	1st	2nd	4th	3rd	0th
1	3rd	1st	0th	2nd	4th
2	4th	0th	1st	2nd	3rd
3	0th	4th	3rd	2nd	1st
4	4th	1st	3rd	0th	2nd

Data Structure List

- men's name → *men's appearance No.* (HashMap)
- women's name → women' *appearance No.* (HashMap)
- men's *appearance No.* → men's name (Array)
- women's *appearance No.* → women's name (Array)
- men's preference list (int[][])
  - the first dimension: men's appearance No.–> men' preference list;
  - the second dimension: the rank of women–> women's appearance No.
- women's preference inverse list (int[][])
  - the first dimension: women's appearance No.–> women's preference list;
  - the second dimension: men's appearance No.–> the rank of men
- Free men (Queue)
- Matching status of woman –> man (Array)
- Matching status of man –>woman (Array)
- When you update above variables , you should be full thought.

Test

- Construct Test Data:
  - Generate random names but do not repetitive:Simple and efficient way： w1,w2, w3 ..or m1,m2, m3 … and so on.
  - Prefer Lists: generate 1 to n for priority. Random swap 2 elements. You can also construct some special cases, for example, all men’s preference lists are the same.
- Check Results:
  - Check the pairs number
  - Check every man has no repetition and exists in men list.
  - Check every man’s partner has no repetition and exists in women list
  - Check every pair whether satisfy stable match condition.(no unstable pair)

Unstable pair condition

- woman x and man y are unstable if:

x prefers y to its assigned man.

y prefers x to its assigned woman.

Pay Attention

- Object copying
  - deep copy
  - shallow copy