稳定匹配Lab问题分析

赵耀

```
Initially all m ∈ M and w ∈ W are free
While there is a man m who is free and hasn't proposed to
every woman w for which (m, w) ∉ F
    Choose such a man m
    Let w be the highest-ranked woman in m's preference list
        to which m has not yet proposed
    If w is free then
        (m, w) become engaged
    Else w is currently engaged to m'
        If w prefers m' to m then
            m remains free
        Else w prefers m to m'
            (m, w) become engaged
            m' becomes free
        Endif
    Endif
Endwhile
Return the set S of engaged pairs
```

Lab中的典型问题

- 输入和输出用什么数据结构存储？
- 如何高效的找到未配对的大佬？
- 如何高效查询某大佬在某萌新偏好表中的排名？
- 如何记录大佬及萌新的配对状态？
- 如何测试代码？

输入和输出用什么数据结构存储？

- 分析输入输出格式
- 大佬偏好表以及萌新偏好表分别以大佬名字出现的序号和萌新名字出现的序号来逐行输入萌新列表以及大佬列表
- 大佬名字 → 大佬出现序号（Map）
- 萌新名字 → 萌新出现序号（Map）
- 大佬序号 →大佬名字（Array）
- 萌新序号 →萌新名字（Array）
- 偏好表显然应该为二维数组，由于可以从Map中方便得到序号，所以偏好表设计为int[][]都是可以的
- 输出为一列萌新的名字，表示序号为i的大佬与出现在i位置的萌新配对。显然输出用String数组就ok。

如何高效的找到未配对的大佬？

- 最容易想到的： 用一个数组存大佬的配对状态，比如大佬i已经配对，那么isMatched[i]=1不就可以了？如果被挖墙角，那就isMatched[i]=0呗。那怎么找未配对的大佬？一个for循坏不就解决了。

  然而高效么？

- 显然可以O(1)的解决这个环节。推荐用Queue来完成这个工作。
- 初始大佬全都add到一个Queue里。
- 每次while取队首元素pop出来，尝试配对。如果挖墙角成功，则被挖的大佬重新回队列

如何帮大佬找出未曾配对的排名最高的萌新？

- 很多同学没有考虑到这一点，每次都从该大佬的偏好表从头开始遍历
- 要知道，大佬也是会被挖墙角的
- 大佬一旦被挖墙角，应该从未曾配对的排名最高的萌新及往后查找

有的同学可能会说：“可是臣妾不知道哇，谁还记得从哪里被挖的(⊙o⊙)啊！”

- 可以利用很简单的数据结构帮助记录，比如每个大佬都配备一个iterator，更直接的用一个数组，值为大佬i当前未曾求配对的排名最高的萌新在其偏好表中的index

如何高效的查询某大佬在某萌新的排名？

- 在算法过程中，如果大佬A尝试配对的萌新已经有大佬B在带了，那么该大佬就开始尝试挖墙角
- 如果在萌新心目中大佬A的排名比B高，那么大佬A挖墙脚成功；否则大佬A只能尝试配对下一个萌新了
- 简单的处理，在萌新偏好表中根据大佬的序号查询大佬在偏好表中的index，2次for循环得到2个大佬的排名

在这个过程中，如果高效的查询萌新心目中大佬的排名呢？

1、方案一：每个萌新除了偏好表，还维护一张reverse的表，index为大佬序号，值为大佬排名，仔细检查下算法，其实整个算法并不需要从大佬排名→大佬序号，所以一开始输入时只需保存reverse的信息即可。

2、方案二：每个萌新的偏好表为map，大佬序号→大佬排名

上述信息均可在处理输入时构造。

如何记录大佬及萌新的配对状态？

- 输出的时候只需要大佬对应萌新的配对状态就可以输出答案
- 那么萌新对应大佬的配对状态是否需要？答案是要的，这样当大佬尝试跟萌新配对时，可以用O（1）的时间查询萌新的配对状态。

稳定匹配需要的变量列表

- 综上，可以总结稳定匹配需要的变量列表：
- 大佬名字 → 大佬出现序号（Map）
- 萌新名字 → 萌新出现序号（Map）
- 大佬序号 →大佬名字（Array）
- 萌新序号 →萌新名字（Array）
- 大佬偏好表（int[][]）一维为大佬序号-> 该大佬的萌新偏好，二维为萌新排名->萌新序号
- ~~萌新偏好表（int[][]）~~
- 萌新偏好reverse表（int[][]）一维为萌新序号-> 该萌新的大佬偏好，二维为大佬序号->排名
- 未匹配的大佬（Queue）
- 萌新与大佬的匹配关系（Array）
- 大佬与萌新的匹配关系（Array）
- 在算法的执行过程中，更新每个变量的值时考虑要细致

编码过程中容易犯的错误

- 用数组或queue表示未配对的大佬，在匹配成功后忘记删除或者在被挖墙角的时候忘记回收
- 每一次都从偏好表排名最高的萌新开始尝试配对，忘记记录之前已经匹配过的位置
- 使用List\Map\Queue等，不要忘记<类型>
- 对象拷贝的问题，深拷贝和浅拷贝。比如直接用“＝”，那么就会发生不同的变量作用到同一块内存的现象，最后发现3个不同的变量取到的值相同了。有的同学比较谨慎，用到了clone的方法，但也要注意有局限性，如果其对象包含了list等引用对象，那么还是会出问题，一定要注意重载clone的方法。
- 数组越界。比如有的同学boolean[] marked = new boolean[100];写了一个固定大小，那么当输入超过100时，就有可能数组越界导致程序崩溃。
- GS算法类与输入耦合很大，建议将GS算法类与输入分离，减少耦合。