**loan requests processing:**

在hacker rank上做题 每个request类似[loan\_id, user\_Id, merchant, amount]

题目是关于loan requests processing 大致考的内容是这些：

part 1:

你有一个map，里面存的是一些公司和它的子公司，但公司本身可能是另一个母公司的子公司，比如：

{"company\_a": [company\_b, company\_c], "company\_d": [company\_a], ...}

给你一个loan request，里面有一个公司，你要找到这个公司最顶上的母公司，然后把这个loan的信息存起来

part 2:

给你一些transaction requests，找出这个transaction可以match part 1里的哪个loan

**字典 get pu delete get random:**

设计一个字典，支持:

o(1) get

o(1) put

o(n) delete

o(1) get\_random\_val

例子：

{a: 5, b: 5, c: 6, d: 5}, the get\_rand\_val()function should return:

5 with a probability of 3/4

6 with a probability of 1/4

Follow up 1：

get\_random\_val returns each unique value with equal probability, 然后 put 的时间复杂度可以是o(n)

Follow up 2:

在follow up 1 的基础上支持 o(1) 的put 和delete

Follow up 2 其实就是 leetcode 381的变种

Question:

In this problem, you're asked to implement a data structure that supports the following operations:

put: Add a key-value pair to the dictionary.

delete: Remove a key-value pair from the dictionary.

get: Retrieve the value associated with a given key.

get\_rand\_val: Return a random value from the dictionary, with each value having a probability proportional to its frequency of occurrence.

For example, given a dictionary {a: 5, b: 5, c: 6, d: 5}, the get\_rand\_val()function should return:

5 with a probability of 3/4

6 with a probability of 1/4

Follow up:

Modify your previous implementation of the dictionary to ensure that get\_rand\_val() returns each unique value with equal probability.

In the previous example, it should return 5 or 6 with a possibility of 1/2

第一题是leetcode原题类似

第二问没有leetcode原题，需要用两个HashMap完成

**处理字符**

一小时一道题。地里没见过

写两个function处理字符，一个compress一个decompress。

input类似“aaabbbbbbcccccddd"，output让自己想。

想了"a3b6c4d3"这种。写完debug完就块没时间了。有一些edge case要考虑比如”abc“而不是”a1b1c1"

面试官白人小姐姐很好会给提醒。

follow up是如果是多位数字decompress怎么写，比如“a12b13"

然后问了如果str很长怎么处理，答split然后merge，注意split错开的情况如何merge比如"aaaa", "aabc"

还问了如果str里本来就有数字怎么办，没答出来，欢迎提供一些想法，面试的加油

LC3163，有的

**PStack：**

In typical Object Oriented world,

a stack has two methods that mutate the data of the stack that is being operated on, push() and pop(). We'd like to implement an immutable version of  the class, which we'll call PStack.

# Psuedocde class PStack

PStack() # constructor

int size() # returns number of elements in the stack

int peek() # returns the most recently pushed element

PStack push(int) # returns an instance of PStack with the element added

PStack pop() # returns an instance of Pstack with the top element removed

需要做到 Push 和 Pop 都是O（1）。Follow up 是实现 reverse()

地理好心人帮忙内推的。亚裔小哥, 一上来聊了一会，然后开始做题，codepair上面。一打开就有个结构写着。

class PStack {

int size() {};

PStack push(int x) {}

PStack pop() {}

}

小哥说你来design 一个persistent stack, push,pop 的时候都得返回一个这个数据PStack. 比如: 现在有s1 = 1,2,3,4, 这个时候call push(5), 应该返回一个新的PStack s2 = 1,2,3,4,5, 而

现有的Stack里面的元素不变。同理如果这个时候call pop(), 应该返回一个新的s3 = 1,2,3,4, 之前的不变。 我就迅速写了个Stack的实现方式，弄个temp Stack 来倒腾元素，然后返回新的

Stack,秒了之后。小哥问这个复杂度是多少，我说O(n),因为你得每次返回一个新的Stack,然后还得装那么多元素啊。然后他说能不能优化到O(1), 想了会，没搞清楚怎么弄。然后他提示说

用shared data structure, 比如用个arrayList, 然后我就写了写，挪一挪指针啥的，返回新的PStack的时候传入指针。然后发现一个bug, 如果你反复push, pop, 还要保留之前的数据的时候，

指针就不够用了，瞬间想到tree, 这个时候时间已经不多了，他说我们聊天吧。我说我还是写出来吧，他说只有八分钟，你写吧。然后写出来了，自己定义一个class TreeNode{int depth, TreeNode

parent, List<TreeNode> children}, class 里面保留一个curNode, 每次push元素的时候就往curNode加一个child, child的depth是父亲的+1, curNode更新。然后每次pop的时候就从curNode到它的父亲。

**Event List:**

coding: 应该是个新题，一个event list，按要求处理。 part 1 是debug，给了4个case。但是系统有些问题，后两个case文件太大，看不到内容。后来面试官自己找到，paste到题目里面了。bug很好找。注意几个点：

用event\_type做一下过滤 判断一下detail那个map结构的key是否存在。

part 2是给一下处理event的逻辑，让实现。比较坑的是需要自己写parse屏幕输入的逻辑。倒是让google。结果lz惯用的jackson 面试的IDE不支持，面试官帮忙贴了一个gson的code。总算是写出来了，但是import dependency的时间耽误的太久了，只有时间跑过了题目里面给的test case。不知道后面还有没有part 3什么的。

design：设计venmo的data model。最开始有点误会，一位是设计如何保证数据一致性。

其实就是设计数据库的表结构，来支持3个场景：

A给B转钱，用的账户余额，然后B withdraw（注意，这里的withdraw是从B的bank给venmo转钱）

A给B转钱，用的bank。然后Bwithdraw before bank transfer is clear。bank transfer最终成功了。

A给B转钱，用的bank。然后Bwithdraw before bank transfer is clear。bank transfer最终失败了。

**无名题**

不同时间都有贷款进入系统，需要得到最近一小时的贷款总量是多少？

比如 [100, 2:15pm], [150, 3:05pm], [200, 4:05pm]

如果 2：30pm 读取， 得到结果 100

如果 3：05pm 读取， 得到结果 250

如果 3：45pm 读取， 得到结果 150

如果 4：05pm 读取， 得到结果 350

好像不是LC原题，test case都跑过了，感觉没啥问题

Follow up是open question是如果内存非常小，不能存所有的数据都存储，应该怎么读取最近一小时的贷款量？

**Log file：**

同一题两问

part 1.

input 两个 log file 代表两天的log，schema 是

<date, user id, order type, amount>

需要 user id 在两天都出现，然后至少要有两个unique的 order type, eg. phone, web, app

要求 return 满足要求的 user id list

part 2.

读完 log 之后，需要算 新 purchase 里面 user 的 trust score

input(user id, order type, amount)

算分 有两个部分组成：

如果 order type 见过，算50分，没见过，0分

如果 amount 在 user purchase 记录里的 最小 amount 和 最大 amount 之间，算50分。如果小于最小，或者大于最大，difference 每多 10% 那么减10分

return 两部分之和的 trust score

另外问了扩展问题，如果不是 log file，是stream of logs 怎么办。

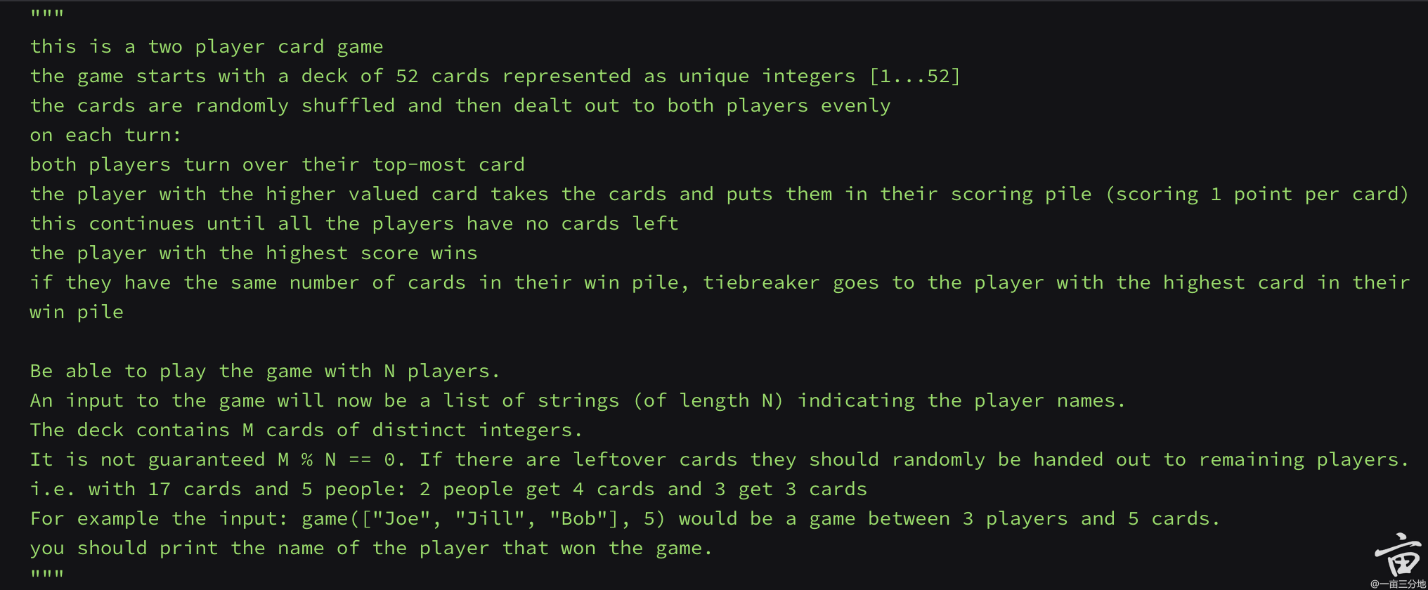
Shopping pattern

Card game

第一个问题说有一堆卡，一共52张，编号1-52，2个玩家，开局洗牌然后一人发一半，然后开始从顶上翻牌比大小，winner takes all，最后赢了多少张牌记多少分，如果平分的话就比谁的最大牌数字大。大概花了20分钟写出来代码并且跑了两个test都过了，然后小哥问了一个拓展问题。现在有m张卡，编号1-m，然后n个玩家，还是轮番发牌，但是m不一定整除n，所以剩下的m%n张牌随机分给这么多人，一人最多一张。比如说，17张牌5个人，先一人3张，余下的2张随机分给5人中的2人。比大小的流程也是一样的，winner takes all，如果分数相同的话就比谁的最大牌数字大

第一部分两人52张牌秒了

第二部分N人M张牌，需要用OOD实现，没写完。问了feedback这是主要的挂点



**End Of Day Balance**

A financial company provides customers a way to buy something now and pay for it later by issuing them loans. However, the correct parties need to be paid the correct amounts - merchants need to be paid the full amount upfront, even though customers are paying that amount over time. This means that money cannot be dispensed directly from the customers to the merchants. Therefore, in order for the logistics of buy now pay later to happen, money needs to be moved between multiple parties - customers, companies, merchants, and third-party entities that the company works with.

Given a list of pending transactions and the current end-of-day bank account balances, determine the updated balance for each party after applying the transactions. Return a list **sorted alphabetically** by party name.

**Constraints:**

* Each transaction involves two parties and a positive integer amount.
* Initial balances and transaction amounts are integers.
* Ignore the number limit of transactions and merchants.

**Example 1:**

**Input:** transactions = [["Alice","Bob","50"],["Bob","Charlie","30"],["Charlie","Alice","20"],["Alice","David","70"]]  
initialBalance = [["Alice","100"],["Bob","50"],["Charlie","75"],["David","25"]]  
**Output:** [0, 70, 85, 95]  
**Explanation:** After all transactions are enacted, the end-of-day balances for "Alice", "Bob", "Charlie," and "David" (sorted alphabetically) are calculated as:

* Alice: 100 (initial) - 50 (to Bob) - 70 (to David) + 20 (from Charlie) = 0
* Bob: 50 (initial) + 50 (from Alice) - 30 (to Charlie) = 70
* Charlie: 75 (initial) + 30 (from Bob) - 20 (to Alice) = 85
* David: 25 (initial) + 70 (from Alice) = 95

**Example 2:**

**Input:** transactions = [["Alice","Bob","300"],["Charlie","David","400"],["Eve","Frank","200"],["George","Hank","100"],["Alice","Charlie","150"],["David","Eve","300"],["Frank","George","250"],["Bob","Hank","100"],["Charlie","Alice","200"],["Eve","David","150"]]  
initialBalance = [["Alice","1000"],["Bob","600"],["Charlie","500"],["David","700"],["Eve","600"],["Frank","400"],["George","200"],["Hank","100"],["Ivy","300"],["Jack","400"]]  
**Output:** [750, 800, 50, 950, 550, 350, 350, 300, 300, 400]

**Example 3:**

**Input:** transactions = [["Alice","Bob","150"],["Charlie","Alice","200"],["David","Bob","50"],["Eve","Charlie","100"],["Bob","Eve","300"],["David","Alice","50"],["Alice","Eve","100"],["Charlie","Bob","150"]]  
initialBalance = [["Alice","500"],["Bob","300"],["Charlie","200"],["David","150"],["Eve","250"]]  
**Output:** [500, 350, -50, 50, 550]

Input

transactions =

[["Alice","Bob","50"],["Bob","Charlie","30"],["Charlie","Alice","20"],["Alice","David","70"]]

initialBalance =

[["Alice","100"],["Bob","50"],["Charlie","75"],["David","25"]]

Output

-

Expected

[0,70,85,95]

Input

transactions =

[["Alice","Bob","300"],["Charlie","David","400"],["Eve","Frank","200"],["George","Hank","100"],["Alice","Charlie","150"],["David","Eve","300"],["Frank","George","250"],["Bob","Hank","100"],["Charlie","Alice","200"],["Eve","David","150"]]

initialBalance =

[["Alice","1000"],["Bob","600"],["Charlie","500"],["David","700"],["Eve","600"],["Frank","400"],["George","200"],["Hank","100"],["Ivy","300"],["Jack","400"]]

Output

-

Expected

[750,800,50,950,550,350,350,300,300,400]

Input

transactions =

[["Alice","Bob","150"],["Charlie","Alice","200"],["David","Bob","50"],["Eve","Charlie","100"],["Bob","Eve","300"],["David","Alice","50"],["Alice","Eve","100"],["Charlie","Bob","150"]]

initialBalance =

[["Alice","500"],["Bob","300"],["Charlie","200"],["David","150"],["Eve","250"]]

Output

-

Expected

[500,350,-50,50,550]

Input

transactions =

[]

initialBalance =

[["Alice","100"],["Bob","200"],["Charlie","300"]]

Output

-

Expected

[100,200,300]

Follow-up:

After processing a series of financial transactions, the net balances among various parties may become unbalanced, resulting in some parties owing money to others. Determine the **minimal** number of transactions required to settle all debts so that every party's balance returns to zero.

You can assume that it is always possible to settle all debts, i.e., the sum of all balances is zero.

**Example 1:**

**Input:** balanceToSet = [["Alice","-100"],["Bob","70"],["Charlie","65"],["David","-35"]] **Output:** 3 **Explanation:** There are two ways to settle all debts, and both require 3 transactions:

1. Bob pays Alice $70; Charlie pays David $35; Charlie pays Alice $30.
2. Charlie pays Alice $65; Bob pays Alice $35; Bob pays David $35.

**Example 2:**

**Input:** balanceToSet = [["Alice", "-100"],["Bob", "200"],["Charlie", "-50"],["David", "150"],["Eve", "-150"],["Frank", "100"],["George", "50"],["Hank", "-100"],["Ivy", "0"],["Jack", "-100"]] **Output:** 5

**Example 3:**

**Input:** balanceToSet = [["Alice","-250"],["Bob","50"],["Charlie","-50"],["David","100"],["Eve","150"]] **Output:** 3

Input

balanceToSet =

[["Alice","-100"],["Bob","70"],["Charlie","65"],["David","-35"]]

Output

-

Expected

3

Input

balanceToSet =

[["Alice","-100"],["Bob","200"],["Charlie","-50"],["David","150"],["Eve","-150"],["Frank","100"],["George","50"],["Hank","-100"],["Ivy","0"],["Jack","-100"]]

Output

-

Expected

5

Input

balanceToSet =

[["Alice","-250"],["Bob","50"],["Charlie","-50"],["David","100"],["Eve","150"]]

Output

-

Expected

3

Input

balanceToSet =

[["Charlie","-50"],["David","150"],["Eve","-150"],["Frank","100"],["George","50"],["Hank","-100"],["Ivy","0"]]

Output

-

Expected

3

Input

balanceToSet =

[["Alice","0"],["Bob","0"],["Charlie","0"]]

Output

-

Expected

0