**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 需要用两个HashMap完成

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

**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。

Implement a persistent stack data structure - PStack, that supports the following operations with each operation executing in *O(1)* time:

* PStack(): Initializes an empty stack.
* int size(): Returns the number of elements in the stack.
* int peek(): Returns the most recently pushed element without removing it.
* PStack push(int val): Returns a new instance of PStack with the element val added on top.
* PStack pop(): Returns a new instance of PStack with the top element removed.

Each push and pop operation should create a **new instance** of the stack without modifying the original, ensuring that all versions of the stack remain accessible.

**Constraints:**

* All operations must run in *O(1)* time.
* The stack should handle a large number of operations efficiently.
* You can assume that pop and peek operations are called on non-empty stacks.

**Example:**

**Input:** ["PStack", "push", "push", "push", "size", "peek", "pop"] [[], [10], [20], [30], [], [], []] **Output:** [[], [10], [20, 10], [30, 20, 10], 3, 30, [20, 10]] **Explanation:**

* PStack stack1 = new PStack(); // Return a new empty stack: []
* PStack stack2 = stack1.push(10) // Return a new stack: [10]
* PStack stack3 = stack2.push(20); // Return a new stack: [20, 10]
* PStack stack4 = stack3.push(30); // Return a new stack: [30, 20, 10]
* stack4.size() // Return 3
* stack4.peek() // Return 30
* PStack stack5 = stack4.pop(); // Return a new stack: [20, 10]

class PStack:

    def \_\_init\_\_(self, head=None, tail=None):

        self.head = head

        self.tail = tail

        self.count = (tail.count if tail else 0) + (1 if head is not None else 0)

    def push(self, val):

        return PStack(val, self)

    def size(self):

        return self.count

    def isEmpty(self):

        return self.head is None

    def pop(self):

        if self.isEmpty():

            raise RuntimeError("Cannot pop from an empty stack")

        return self.tail

    def peek(self):

        return self.head

def visualizeAsList(stack):

    result = []

    current = stack

    while not current.isEmpty():

        result.append(current.head)

        current = current.tail

    return result

def test1():

    print("========= Test1 =========")

    stack1 = PStack()

    print(f"Result: {visualizeAsList(stack1)}, expected: []")

    stack2 = stack1.push(10)

    print(f"Result: {visualizeAsList(stack2)}, expected: [10]")

    stack3 = stack2.push(20)

    print(f"Result: {visualizeAsList(stack3)}, expected: [20, 10]")

    stack4 = stack3.push(30)

    print(f"Result: {visualizeAsList(stack4)}, expected: [30, 20, 10]")

    print(f"Result: {stack4.size()}, expected: 3")

    print(f"Result: {stack4.peek()}, expected: 30")

    # Verify that the original stack is not modified

    print(f"Result: {visualizeAsList(stack1)}, expected: []")

    print(f"Result: {visualizeAsList(stack2)}, expected: [10]")

    stack5 = stack4.pop()

    print(f"Result: {visualizeAsList(stack5)}, expected: [20, 10]")

    # Verify that the original stack is not modified

    print(f"Result: {visualizeAsList(stack4)}, expected: [30, 20, 10]")

def test2():

    print("========= Test2 =========")

    stack = PStack()

    print(f"Result: {visualizeAsList(stack)}, expected: []")

    stack = stack.push(5)

    print(f"Result: {visualizeAsList(stack)}, expected: [5]")

    print(f"Result: {stack.size()}, expected: 1")

def test3():

    print("========= Test3 =========")

    stack = PStack()

    stack = stack.push(1).push(2).push(3)

    print(f"Result: {visualizeAsList(stack)}, expected: [3, 2, 1]")

    stack = stack.pop()

    print(f"Result: {visualizeAsList(stack)}, expected: [2, 1]")

def test4():

    print("========= Test4 =========")

    stack = PStack()

    stack = stack.push(42)

    print(f"Result: {stack.peek()}, expected: 42")

    stack = stack.pop()

    print(f"Result: {visualizeAsList(stack)}, expected: []")

def test5():

    print("========= Test5 =========")

    stack = PStack()

    elements = [10, 20, 30, 40]

    for elem in elements:

        stack = stack.push(elem)

    print(f"Result: {visualizeAsList(stack)}, expected: [40, 30, 20, 10]")

    print(f"Result: {stack.size()}, expected: 4")

if \_\_name\_\_ == "\_\_main\_\_":

    test1()

    test2()

    test3()

    test4()

    test5()

**Shopping pattern**

Assuming you are designing a recommendation system for an online marketplace. The system processes shopping records, where each record lists the stores a customer visited during a single shopping session.

Determine the co-occurrence patterns between stores. For each store, return a list of associated stores that are frequently visited together with it, based on the shopping records.

The results should be:

1. An organized nested list, where each element corresponds to a target store.
2. Sorted by the target store names in lexicographical order.
3. For each target store, the associated stores should be sorted first by the frequency of co-occurrence in non-increasing order. If multiple stores share the same frequency, they should be sorted in lexicographical order.

**Constraints:**

* The number of shopping records and stores is manageable within the scope of this task.
* Shopping records may contain zero stores, indicating the customer didn’t make any purchases.
* Store names consist of uppercase and lowercase English letters and are case-sensitive.

**Example 1:**

**Input:** records = [["Amazon", "Walmart", "Costco"], ["Amazon", "Costco", "BestBuy"], ["Amazon", "BestBuy"], ["HomeDepot", "BestBuy"]] **Output:** [["BestBuy", "Costco", "Walmart"], ["Amazon", "Costco", "HomeDepot"], ["Amazon", "BestBuy", "Walmart"], ["BestBuy"], ["Amazon", "Costco"]] **Explanation:**

* The target stores are: ["Amazon", "BestBuy", "Costco", "HomeDepot", "Walmart"], sorted lexicographically.
* For each target store, the associated stores and their frequencies are calculated:
  + "Amazon": Co-occurs with "Costco" (2 times), "BestBuy" (2 times), and "Walmart" (1 time). Result: ["BestBuy", "Costco", "Walmart"]
  + "BestBuy": Co-occurs with "Amazon" (2 times), "Costco" (1 time), and "HomeDepot" (1 time). Result: ["Amazon", "Costco", "HomeDepot"]
  + "Costco": Co-occurs with "Amazon" (2 times), "Walmart" (1 time), and "BestBuy" (1 time). Result: ["Amazon", "BestBuy", "Walmart"]
  + "HomeDepot": Co-occurs with "BestBuy" (1 time). Result: ["BestBuy"]
  + "Walmart": Co-occurs with "Amazon" (1 time) and "Costco" (1 time). Result: ["Amazon", "Costco"]

**Example 2:**

**Input:** records = [["Amazon"], ["Amazon"], ["Amazon"]] **Output:** [[]] **Explanation:** For "Amazon": There are no other stores purchased alongside it.

**Example 3:**

**Input:** records = [["Amazon", "Walmart", "Costco", "BestBuy"], ["Amazon", "Walmart", "Costco", "BestBuy"], ["Amazon", "Walmart", "Costco", "BestBuy"]] **Output:** [['BestBuy', 'Costco', 'Walmart'], ['Amazon', 'Costco', 'Walmart'], ['Amazon', 'BestBuy', 'Walmart'], ['Amazon', 'BestBuy', 'Costco']]

Input

records =

[["Amazon","Walmart","Costco"],["Amazon","Costco","BestBuy"],["Amazon","BestBuy"],["HomeDepot","BestBuy"]]

Output

-

Expected

[["BestBuy","Costco","Walmart"],["Amazon","Costco","HomeDepot"],["Amazon","BestBuy","Walmart"],["BestBuy"],["Amazon","Costco"]]

Input

records =

[["Amazon"],["Amazon"],["Amazon"]]

Output

-

Expected

[[]]

Input

records =

[]

Output

-

Expected

[]

Input

records =

[["Amazon","Walmart"],["Costco","BestBuy"],["HomeDepot","Target"]]

Output

-

Expected

[["Walmart"],["Costco"],["BestBuy"],["Target"],["HomeDepot"],["Amazon"]]

Input

records =

[["Amazon","Walmart","Costco","BestBuy"],["Amazon","Walmart","Costco","BestBuy"],["Amazon","Walmart","Costco","BestBuy"]]

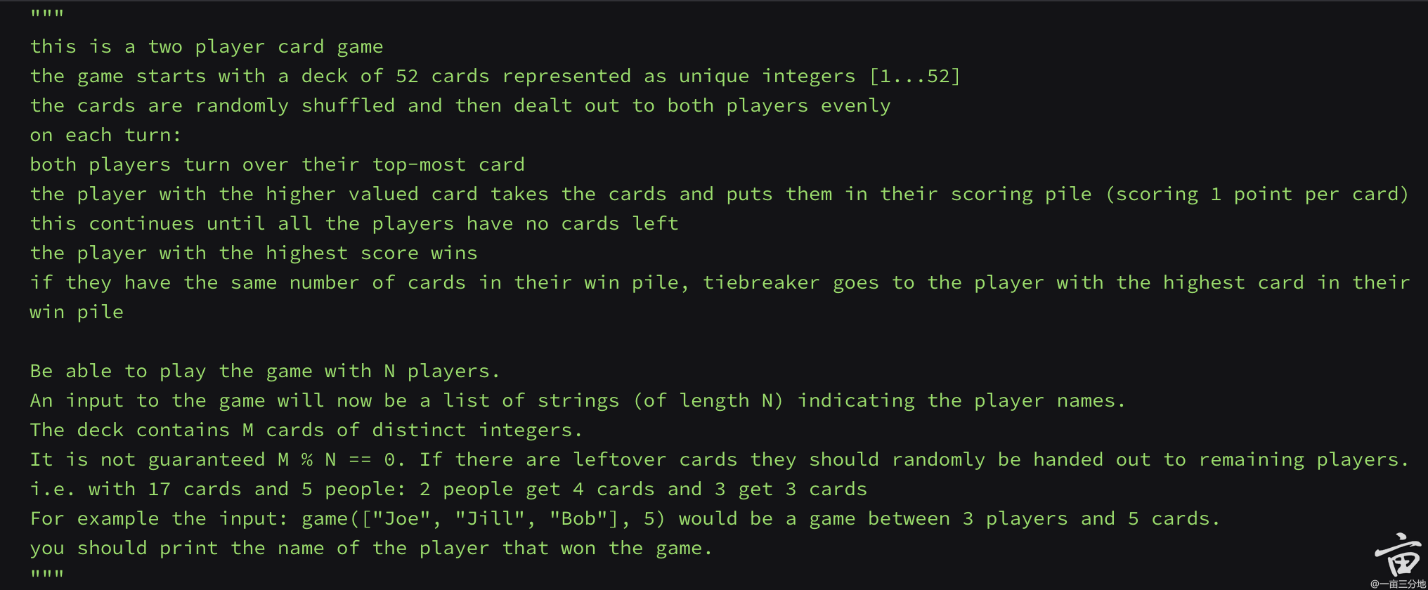
Output

-

Expected

[["BestBuy","Costco","Walmart"],["Amazon","Costco","Walmart"],["Amazon","BestBuy","Walmart"],["Amazon","BestBuy","Costco"]]

**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

Create a simulation of a card game. The game involves a deck of unique cards represented by integers. Assume there are only 2 players "A" and "B", and a deck of 52 unique cards (numbered from 0 to 51). Players "A" and "B" take turns drawing cards **randomly** from the deck. After each draw, the players compare their cards:

* If Player A's card is higher, Player A earns 1 point.
* If Player B's card is higher, Player B earns 1 point.

The game continues until all cards have been drawn and compared. The player with the higher score wins the game. If scores are tied at the end, return "TIE". **Note that your solution should simulate the entire drawing process.**

Implement the CardGames class:

* CardGames(): Initializes the card game.
* String drawAndCompare(): Players take turns drawing and comparing card values until all cards are drawn. Returns the winner or "TIE" if the scores are equal.

**Constraints:**

* Only two players, A and B.
* Only 52 unique cards, from 0 to 51.
* Each card is exactly the same except for the number.
* The total score of two players is always 52 / 2 = 26.

**Example**

**Input:** ["CardGames", "drawAndCompare"] [[], []]

**Output:** [null, "A"]

**Explanation:**

* CardGames game = CardGame();
* game.drawAndCompare(); // Return "A", "B" or "TIE". The drawing process looks like this:
  + 1st round, A and B draw cards and compare, 50 cards left
  + 2nd round, A and B draw cards and compare, 48 cards left
  + 3rd round, A and B draw cards and compare, 46 cards left
  + ... ...
  + 25th round, A and B draw cards and compare, 2 cards left
  + 26th round, A and B draw cards and compare, 0 cards left
  + At the end of the game, compare the scores of Player A and Player B. For example, if Player A has a final score of 17 and Player B has 9, Player A is the winner.

import random

class CardGames:

    def \_\_init\_\_(self):

        self.score1 = 0

        self.score2 = 0

        self.deck = list(range(52))

    def randomTake(self):

        return random.choice(self.deck)

    def remove(self, card):

        self.deck.remove(card)

        return

    def draw(self):

        card1 = self.randomTake()

        self.remove(card1)

        card2 = self.randomTake()

        self.remove(card2)

        if card1 > card2:

            self.score1 += 1

        elif card1 < card2:

            self.score2 += 1

        return

    def drawAndCompare(self):

        # play game until no card

        while self.deck:

            self.draw()

        # decide winner

        if self.score1 > self.score2:

            return "A"

        elif self.score1 < self.score2:

            return "B"

        else:

            return "TIE"

if \_\_name\_\_ == "\_\_main\_\_":

    def test(i):

        game = CardGames()

        res = game.drawAndCompare()

        print("======== Test {0} =======".format(i))

        print("Player A's score: {0}, palyer B's score: {1}".format(game.score1, game.score2))

        print("The result is: " + res)

    for i in range(1,6):

        test(i)

**Follow-up:**

Maintaining the same game rules, extend the game to include N players and M cards. Each player is assigned a unique index ranging from 0 to N-1, and card values range from 0 to M-1.

In each round, all N players take turns drawing cards randomly and comparing their values. The game continues until the remaining cards are fewer than the number of players.

Return a list of the winning players' indices. If multiple players achieve the highest score, they are all considered winners.

Implement the CardGames class:

* CardGames(int players, int deck): Initializes the card game with given players' number and deck size.
* List<Integer> drawAndCompare(): Players take turns drawing and comparing card values until the deck is empty or lacks enough cards for a full round. Returns a list containing the index or indices of the winning player(s).

**Example**

**Input:** ["CardGames", "drawAndCompare"] [[3, 52], null]

**Output:** [null, [2]]

**Explanation:**

* CardGames game = CardGame(3, 52); // Initialize card game with 3 players and 52 cards
* game.drawAndCompare(); // Returns [2]. Additionally, any combination of elements from [0, 1, 2], such as [0, 1] or [0], is also considered valid.
  + 1st round, 3 players draw cards and compare, 49 cards left
  + 2nd round, 3 players draw cards and compare, 46 cards left
  + 3rd round, 3 players draw cards and compare, 43 cards left
  + ... ...
  + 16th round, 3 players draw cards and compare, 4 cards left
  + 17th round, 3 players draw cards and compare, 1 card left
  + Stop the game because 1 card is not enough for a new round. Compare players' scores and return the index/indices of winner(s).

import random

class CardGames:

    def \_\_init\_\_(self, players, deck):

        self.users\_count = players

        self.users\_score = [0] \* self.users\_count

        self.deck = list(range(deck))

    def randomTake(self):

        return random.choice(self.deck)

    def remove(self, card):

        self.deck.remove(card)

        return

    def draw(self):

        m = -1

        idx = -1

        for i in range(self.users\_count):

            card = self.randomTake()

            self.remove(card)

            if card > m:

                m = card

                idx = i

        self.users\_score[idx] += 1

        return

    def drawAndCompare(self):

        # play game until no card

        while len(self.deck) >= self.users\_count:

            self.draw()

        # decide winner

        maxind = []

        maxnum = max(self.users\_score)

        for i, num in enumerate(self.users\_score):

            if num == maxnum:

                maxind.append(i)

        return maxind

if \_\_name\_\_ == "\_\_main\_\_":

    # Test case 1

    print("============== Test 1 ==============")

    players1 = 3

    deck1 = 52

    print("Player number: ", players1)

    print("Deck size:", deck1)

    test1 = CardGames(players1, deck1)

    rslt1 = test1.drawAndCompare()

    print("Winner Index: ", rslt1)

    print("Players' scores: ", test1.users\_score)

    # Test case 2

    print("============== Test 2 ==============")

    players2 = 4

    deck2 = 52

    print("Player number: ", players2)

    print("Deck size: ", deck2)

    test2 = CardGames(players2, deck2)

    rslt2 = test2.drawAndCompare()

    print("Winner Index: ", rslt2)

    print("Players' scores: ", test2.users\_score)

    # Test case 3

    print("============== Test 3 ==============")

    players3 = 5

    deck3 = 68

    print("Player number: ", players3)

    print("Deck size: ", deck3)

    test3 = CardGames(players3, deck3)

    rslt3 = test3.drawAndCompare()

    print("Winner Index: ", rslt3)

    print("Players' scores: ", test3.users\_score)

**处理字符 （**整体题目和Leetcode3163很像**）**

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"

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

两个追问：

1. 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"

2. 还问了如果str里本来就有数字怎么办

**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。