**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

如果有duplicate key怎么办 following up: duplicated value?

店面 力扣381 follow up 是可以有重复的数字 但是算概率的话每个数字的只算一次。

follow up了 相同值插入但是平均概率输出

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 1/2v

LeetCode 三巴伊的变形， 设计一个data strcuture满足以下要求

    public void put(int key, int val);

    public int get(int key);

    public void remove(int key);

    public int getRandomValue();

getRandomValue 按相同概率返回其中任意一个value

Follow up是允许不同的key有相同的value

比如 （1, 100), (2, 100), (3, 200)

要求设计两个方法

    public int getRandomValue();

    public int getUniqRandomValue();

getRandomValue 需要按frequency来，duplicate越多，概率越大

getUniqRandomValue不看，不同的value概率相同，不论出现多少次duplicate

e.g.

getRandomValue :

100 -> 2/3

200 -> 1/3

getUniqRandomValue :

100 -> 1/2

200 -> ½

**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"]]

老题Shopping Pattern，给一个List of String List，求出每个String的相关Strings是哪些。直接Map + Brute force就好。

Input: [['Casper', 'Purple', 'Wayfair'],['Purple', 'Wayfair', 'Tradesy'],['Wayfair', 'Tradesy', 'Peloton']]

Output:  {

    'Casper': ['Purple', 'Wayfair'],

    'Purple': ['Wayfair'],

    'Wayfair': ['Purple', 'Tradesy'],

    'Tradesy': ['Wayfair'],

    'Peloton': ['Wayfair', 'Tradesy']

}

复制代码

Follow Up：如果这个List of List需要经常添加新的inner List怎么办？

输入 list of strings. 对于每个出现过的字符，找到和它同时在一个string出现过最多的字符。

['abc', 'bcd', 'cde'] =>

{

    a: [b, c], # b appears in 1 string with a, c appears in 1 string with a.

    b: [c], # c appears in 2 strings with b. a and d each appear in only 1 string with b.

    c: [b, d], # b appears in 2 strings with c, d appears in 2 strings with c. But a and e each appear in only 1 string with c.

    d: [c], # c appears in 2 strings with d. But b and e each appear in only 1 string with d.

    e: [c, d], # c appears in 1 string with e, d appears in 1 string with e.

   }

follow up:

1. 如果字符串中有dup怎么办

2. 如果是字符串流怎么办

// Affirm partners with a lot of merchants today and many users will make purchases at more than one merchant.

// We'd like to analyze that cross purchasing behavior to make recommendations to new user about where else they might like to shop.

// Imagine we have a list where each entry is an individual user's history of purchases,

// i.e., the list of merchants that the user has made a purchase at.

// We want to take that list and find, for any merchant with at least one purchase, what other merchant(s) are most frequently seen in users' shopping behavior.

//  e.g. [['Casper', 'Purple', 'Wayfair'],['Purple', 'Wayfair', 'Tradesy'],['Wayfair', 'Tradesy', 'Peloton']]

// [['Casper', 'Purple', 'Wayfair'],['Purple', 'Wayfair', 'Tradesy'],['Wayfair', 'Tradesy', 'Peloton']] =>

//  {

//    'Casper': ['Purple', 'Wayfair'],

//    'Purple': ['Wayfair'],

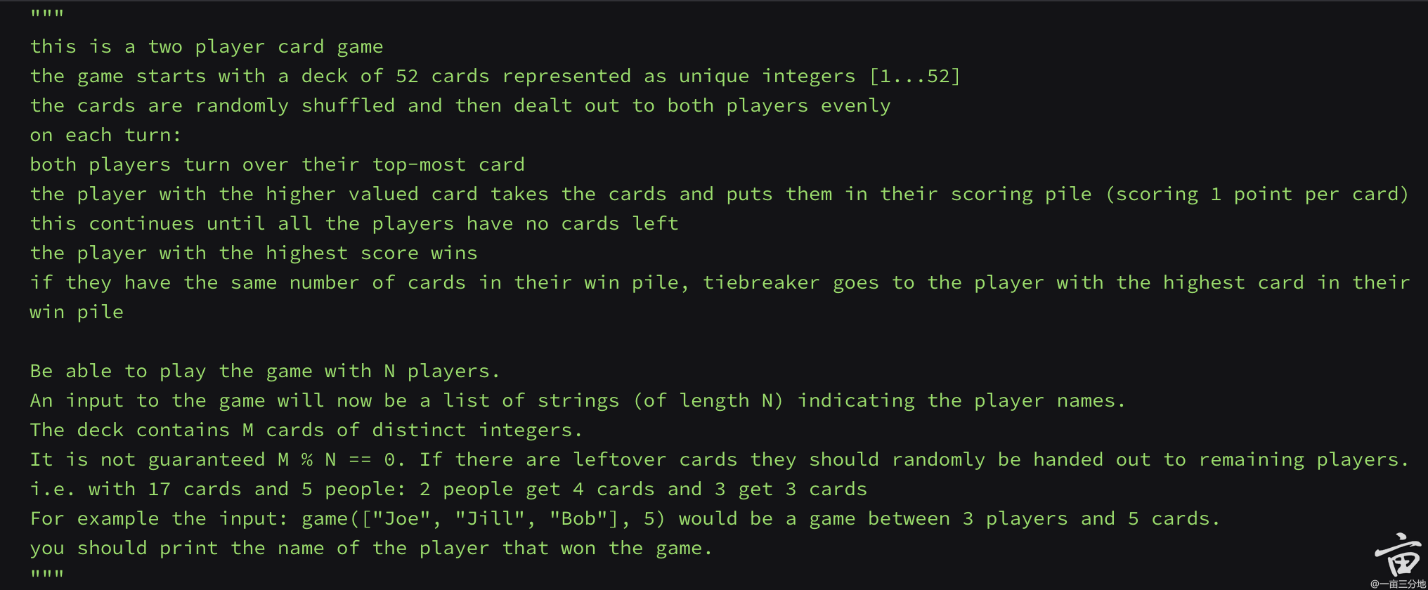
//    'Wayfair': ['Purple', 'Tradesy'],

//    'Tradesy': ['Wayfair'],

//    'Peloton': ['Wayfa‍‍‌‌‌‍‌‌‌‍‌‌‍‍‌‌ir', 'Tradesy']

//  }

**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。

Design a purchase analyzer that builds a two-day per-user history and scores new purchases.

Implement the PurchaseAnalyzer class:

* PurchaseAnalyzer(List<List<String>> day1Logs, List<List<String>> day2Logs) Initializes the analyzer with two days of logs. Each log row is [date, userId, orderType, amount], where amount is a string but parseable to a number. History is built **only** from these two days.
* List<String> crossDayDiverseUsers() Return the **lexicographically sorted** list of user IDs that meets the following reqirements:
  + Appear on both days of logs.
  + Have at least **two distinct** orderType values across the two days combined.
* int trustScore(String userId, String orderType, double amount) Compute and return an integer trust score in [0, 100] for a new purchase, as the sum of two components:
  + **Order-type component (0 to 50):** If the user has ever made a purchase of this order type in the two-day history, award the full 50 points. Otherwise, award 0 points.
  + **Amount component (0 to 50):** Look at the smallest and largest amounts the user has spent across the two days.
    - If the new amount falls within this historical range, award the full 50 points.
    - If it is above or below the range, measure how far it strays from the closest bound in percentage terms. For every full ten percent beyond that bound, deduct 10 points from the 50-point baseline. This component never goes below zero.
    - If the user has no history, award zero for this part.

**Constraints:**

* 1 ≤ day1Logs.length, day2Logs.length ≤ 105105
* 0 < amount
* All input strings are non-empty.

**Example**

**Input:** ["PurchaseAnalyzer", "crossDayDiverseUsers", "trustScore", "trustScore", "trustScore", "trustScore", "trustScore"]

[[[["2025-08-01","u1","web","100"], ["2025-08-01","u1","app","120"], ["2025-08-01","u2","web","50"], ["2025-08-01","u3","phone","200"], ["2025-08-01","u4","web","80"]],[["2025-08-02","u1","web","110"], ["2025-08-02","u2","web","60"], ["2025-08-02","u2","phone","70"], ["2025-08-02","u3","phone","190"], ["2025-08-02","u5","app","40"]]], [], ["u1","app",115.0], ["u2","web",95.0], ["u3","web",210.0], ["u4","web",70.0], ["u2","app",60.0]]

**Output:** [null, ["u1", "u2"], 100, 70, 50, 90, 50]

**Explanation:**

* PurchaseAnalyzer pa = new PurchaseAnalyzer(day1Logs, day2Logs);
* pa.crossDayDiverseUsers(); // Return ["u1", "u2"]. Both appear on both days and have ≥ 2 distinct order types across the two days combined.
* pa.trustScore("u1","app",115.0); // Return 100. Order type seen before (+50) and amount within "u1" range [100, 120] (+50).
* pa.trustScore("u2","web",95.0); // Return 70. Type seen (+50); 95 exceeds max = 70 by 25 ⇒ 25 / 70 ≈ 35.7% ⇒ floor to 3 steps ⇒ in-range score 50 - 30 = 20; total 70.
* pa.trustScore("u3","web",210.0); // Return 50. Type unseen (+0); "u3" range is [190,200]; 210 is only 5% over ⇒ floor to 0 steps ⇒ in-range score 50.
* pa.trustScore("u4","web",70.0); // Return 90. Type seen (+50); 70 is 10 below min score (80) ⇒ 10 / 80 = 12.5% ⇒ 1 step ⇒ in-range score 50 - 10 = 40; total 90.
* pa.trustScore("u2","app",60.0); // Return 50. Type unseen (+0); amount within "u2" range [50, 70] ⇒ in-range score +50.

Input

["PurchaseAnalyzer","crossDayDiverseUsers","trustScore","trustScore","trustScore","trustScore","trustScore"]

[[[["2025-08-01","u1","web","100.0"],["2025-08-01","u1","app","120.0"],["2025-08-01","u2","web","50.0"],["2025-08-01","u3","phone","200.0"],["2025-08-01","u4","web","80.0"]],[["2025-08-02","u1","web","110.0"],["2025-08-02","u2","web","60.0"],["2025-08-02","u2","phone","70.0"],["2025-08-02","u3","phone","190.0"],["2025-08-02","u5","app","40.0"]]],[],["u1","app",115],["u2","web",95],["u3","web",210],["u4","web",70],["u2","app",60]]

Output

-

Expected

[null,["u1","u2"],100,70,50,90,50]

Input

["PurchaseAnalyzer","crossDayDiverseUsers","trustScore","trustScore","trustScore","trustScore","trustScore"]

[[[["2025-09-01","u1","web","150.0"],["2025-09-01","u1","app","180.0"],["2025-09-01","u2","web","55.0"],["2025-09-01","u2","phone","70.0"],["2025-09-01","u3","phone","300.0"],["2025-09-01","u4","app","250.0"],["2025-09-01","u5","web","100.0"],["2025-09-01","u6","web","500.0"],["2025-09-01","u6","app","450.0"]],[["2025-09-02","u1","web","160.0"],["2025-09-02","u2","web","65.0"],["2025-09-02","u2","phone","90.0"],["2025-09-02","u3","phone","310.0"],["2025-09-02","u4","app","250.0"],["2025-09-02","u6","app","480.0"]]],[],["u1","app",210],["u2","web",44],["u3","web",210],["u4","app",300],["u6","web",800]]

Output

-

Expected

[null,["u1","u2","u6"],90,80,20,80,50]

Input

["PurchaseAnalyzer","crossDayDiverseUsers","trustScore","trustScore","trustScore","trustScore","trustScore"]

[[[["2025-10-01","u7","web","100.0"],["2025-10-01","u7","app","200.0"],["2025-10-01","u8","web","300.0"],["2025-10-01","u9","app","90.0"],["2025-10-01","u10","phone","100.0"],["2025-10-01","u11","web","50.0"]],[["2025-10-02","u7","phone","180.0"],["2025-10-02","u7","web","220.0"],["2025-10-02","u8","web","330.0"],["2025-10-02","u10","phone","100.0"],["2025-10-02","u11","app","90.0"],["2025-10-02","u11","phone","55.0"]]],[],["u7","app",245],["u7","phone",160],["u10","phone",90],["u11","kiosk",60],["u8","web",297]]

Output

-

Expected

[null,["u11","u7"],90,100,90,50,100]

**Coin change leetcode 322变种**

第一问：第一问不要求返回fewest number意思是可以返回任意一种解就行 例如有硬币[1, 2, 5, 10]和总值100, 可以返回{1:100}或者{10:10} 都行

第二问：coin change 变种，给定硬币种类[1,2,5,10]和总值比如100，返回一个map，列出每个硬币个数，比如{1:100}，或者{10:10},可以有很多种解，返回一种就行，但这里的一个假设是input总有1元硬币，也就是说至少有一个解的，我脑子抽了用backtrack，费力不讨好，最优解是直接对每个币值做除法，得到余数，然后用余数在下一个循环里除下一个币值，直到归零，同时记录map就ok了 第二问其实是leetcode

一开始没要求fewest number, 我就写了个dfs，然后问我怎么可以fewest number，口述了一下bfs，没让我写，就说可以了。

**Leetcode 362 Hit counter**

跟刷题网362一样, how many requests in last 3000 seconds

讲了两种, 用queue存全部的然后按timestamp过滤  或者3000数组, 她要求用fix sized数组来做

题目类似于“里扣”里面的“伞刘尔”；时间空间复杂度都要是O（1）

process\_loan(amount)

get\_loan\_volume()；// # the total amount of loans processed in the last 1 hour

写两个API process\_loan 和 getLoan()（名字不记得也不重要） . 第二个API 要求返回1小时内的amount.

LZ最近在刷题期，而且由于这又是第一面脑子一直没转过弯，一直当刷题。。。但其实对面很想听design的东西。

follow up 我答的就是优化用timestmap做key。把popup one hour 的工作丢到process\_loan 这样就可以只maintain 1hour timeframe 的数据。这样就都是constant 了。但是当时脑子没转过弯这段虽然答出来了但是没答好估计挂了。

**End of the day balance**

# Background: Affirm 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, Affirm, merchants, and third party entities that Affirm works with.

# Part 1: Given a list of pending transactions and current bank account balances, return how much money each party would have in their bank account after the transactions are actually enacted.

# return the end of the day balance

def get\_balance\_dict(transactions\_list, beginning\_balance\_dict)->dict[int, int]:

    for transaction in transaction\_list:

        send = transaction[0]

        get = transaction[1]

        cost = transaction[2]

        beginning\_balance\_dict[send] -= cost

        beginning\_balance\_dict[get] += cost

    return beginning\_balance\_dict

# transaction\_list = [[0, 1, 100], [2, 3, 200], [3, 1, 50]]

# start\_dict = { 0:0, 1:0, 2:0, 3:0 }

# Part 2: After some money movements have occurred, the balances between the parties are out of sync,

# meaning some parties still owe others money. Using the least number of transactions,

# how can we settle these balances back out to 0 so that nobody owes anybody money?

# Settled ending balance

# { 0: 0, 1: 0, 2: 0, 3: 0 }

# Unsettled ending balance

# { 0: 100, 1: -50, 2: 0, 3: -20 }

# return transfers and the balance after the settlement

def settle\_balances(beginning\_balance\_dict):

    # Separate into creditors and debtors

    creditors = {k: v for k, v in beginning\_balance\_dict.items() if v > 0}

    debtors = {k: -v for k, v in beginning\_balance\_dict.items() if v < 0}

    transactions = []

    # While there are still debts to settle

    while debtors:

        # Find the largest debtor and creditor

        max\_debtor = max(debtors, key=debtors.get)

        max\_creditor = max(creditors, key=creditors.get)

        # Determine the transaction amount

        amount = min(debtors[max\_debtor], creditors[max\_creditor])

        # Record the transaction

        transactions.append([max\_debtor, max\_creditor, amount])

        # Update balances

        debtors[max\_debtor] -= amount

        creditors[max\_creditor] -= amount

        # If any balance is settled, remove them from the list

        if debtors[max\_debtor] == 0:

            del debtors[max\_debtor]

        if creditors[max\_creditor] == 0:

            del creditors[max\_creditor]

    # All balances are now settled to 0

    settled\_balance = {k: 0 for k in beginning\_balance\_dict.keys()}

    return transactions, settled\_balance

input1 = { 0: -100, 1: 150, 2: -200, 3: 150 }

input2 = { 0: -800, 1: -500, 2: -100, 3: -100, 4: 100, 5: 100, 6: 1300 }

output1 = (

  [

    [3, 2, 150],

    [1, 2, 50],

    [1, 0, 100]

  ],

  { 0: 0, 1: 0, 2: 0, 3: 0 } # balance after the settlement

)

output2 = (

  [

    [6, 0, 800],

    [6, 1, 500],

    [4, 2, 100],

    [5, 3, 100],

  ],

  { 0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0 } # balance after the settlement

)

print(settle\_balances(input1))

**Decision Tree:**

A decision tree is a data structure that can be evaluated on a set of signals and return a decision (e.g. Yes or No ("Y" or "N")). Each interior node of the tree is associated with a particular signal and a constant value against which to compare that signal, and each leaf node has a value which will be returned by the tree. To evaluate the tree on a set of signals we traverse the tree, starting at the root and comparing the appropriate signal value to the constant associated with each interior node. If the signal value is less than the constant we proceed down the left subtree and if it is greater than or equal to the constant we proceed down the right subtree. We continue until we reach a leaf at which point we return the value associated with the leaf.

For example, suppose that we have a set of integer-valued signals {X1, X2, X3}. Consider the following decision tree:

           X1 < 3

        ------------

       |            |

    X2 < 1       X1 < 6

-----------    ---------

|           |  |         |

N           Y  N      X3 < 2

                    ----------

                   |          |

                   Y          N

If we evaluate this tree on signals {X1: 2, X2: 1, X3: 11} the result will be Y. Evaluating on signals {X1: 8, X2: 4, X3: 12} we get N. We can use these to implement decisions that need to be made repeatedly on different input values. For instance, a given decision tree might represent a rule to decide whether or not a given transaction looks fraudulent, and the signals could represent different quantities like X1) the age of the account in days, X2) the dollar value of the transaction, and X3) the time in hours since the last transaction attempt.

In real life, we would probably grow a decision tree via some machine learning algorithm. In this exercise, however, we will manually create the tree that we want. We can grow a decision tree by starting with a single-leaf tree and recursively splitting the leaves of the tree. We do this by associating a split condition to a node, creating two new leaves below it, and associating a return value to each of those leaves.

So to grow the tree above we start with a single-leaf tree:

Y

Then add the split condition X1 < 3 and (optionally) assign return values to the new leaves:

     X1 < 3

  ------------

|            |

Y            N

Then add a split condition to the left leaf:

           X1 < 3

        ------------

       |            |

    X2 < 1          N

-----------

|           |

X           X

Assign return values to the new leaves:

           X1 < 3

‍‍‌‌‌‍‌‌‍‌‌‌‍‌‍‌‍‍‍        ------------

       |            |

    X2 < 1          N

-----------

|           |

N           Y

And so on until we are done.

The goal of this question is to implement a decision tree that can be grown incrementally in this fashion and evaluated on a particular set of signals. Concretely you should implement the following pseudocode API in the language of your choice:

```

class DecisionTree:

  method add\_split(leaf, signal\_name, constant):

    Add a split condition to the given leaf node.

    Return the newly created leaves for future calls.

  method set\_leaf\_value(leaf, value):

    Set the return value for a leaf node.

  method evaluate(signals):

    Evaluate the tree on a mapping of signal\_name -> signal\_value.

    Return the value of the leaf node reached by traversing the tree.

```

Afterwards, use your solution to grow the example tree above and write some test cases.

实际操作下来是实现了一个给定输入key =》value json来判断decision结果的功能，这棵树的所有类定义都要自己实现所以还有些boilerplate的活儿。followup是实现一个serialize和deserilize的功能

--------------------------------------

Questions:

1. Implement methods and interface to grow the decision tree incrementally

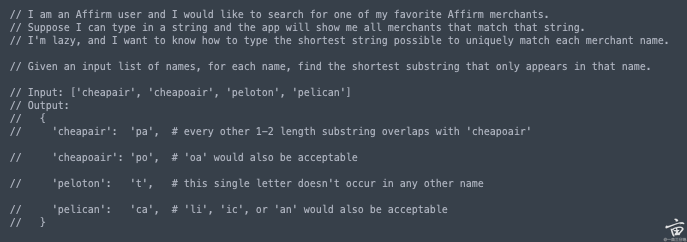
2. Create a decision tree by using the method you implemented in step one

3. Analyize the time and space complexity when applying your methods on multiple set of signals.

关于decision tree, 每个结点只有两种情况 比较的是 string 和 integer, 比如 x < 5, 如果 x = 4,去左子树， x = 5,去右子树，给一个所有node label的值，evaluate T or false

本质上是让你create一个二叉树 然后trace 二叉树到哪个叶子结点  需要写测试用例 并且跑几个测试 follow up 是怎么存这个决策树 如果这个决策树很大的话

**3076.Short unique substring:**



给一堆String，求每一个的最短unique substring(给了前提一定存在).

// Input: ["cheapair", "cheapoair", "peloton", "pelican"]

// Output:

// "cheapair": "pa"  // every other 1-2 length substring overlaps with cheapoair

// "cheapoair": "po" // "oa" would also be acceptable

// "pelican": "ca"   // "li", "ic", or "an" would also be acceptable

// "peloton": "t"    // this single letter doesn't occur in any other string

暴力解就可以

followup 1: 优化一下substr(), 就是用字符串长度来做循环

followup 2: 不一定存在unique，输出最短的substring

第一题：unique 最小 Substring, 给一个list ['abc','bde','cbd] 。这个地里可以找到，用string长度来找最小的是最优解

第二题  unique 最小 Sub Sequence, 直接懵逼。

第一题是老题给一堆String，求每一个的最短unique substring(给了前提一定存在). follow up是如果不一定guarantee存在unique，输出最短的match 最少的substring.

第二题是老题给一堆String of characters，求跟某个字母同时出现的最多次数的character."abc", "abe" -> a: b, b: a, c: "a,c", e:"a,b"

shortest unique substring, follow up 1是 不是unique 怎么办， 2是subsequence 怎么办， 就用排列组合想法就可以

Shortest Unique Substring + follow-up 有可能没有unique的，按这个排序 1) appear fewest in all original\_strs, 2) shortest possible

**lc 49 变种**

anagram的各种变式（lc 49，第一个是比较两个string，然后是49原题，然后是优化时间复杂度）

Pop Up node

附一个pop up, 扫一遍 不用加parent

1. 从root到POPUP的path 上所有点 hidden=false

2. 从root到POPUP的path上所有点的sibling hidden=true

3. POPUP 的 silbing hidden=true

4. 其他的点不变

bool dfs(DomNode\* root) {

    if (!root) { return false; }

    if ("POPUP" == root->id) {

        root->hidden = false;

        return true;

    }

    bool has\_popup = false;

    unordered\_map<DomNode\*, bool> node\_has\_popup;

    for (auto child : root->children) {

        node\_has\_popup[child] = dfs(child);

        has\_popup = has\_popup or node\_has\_popup[child];

    }

    if (has\_popup) {

        for (auto child : root->children) {

            if (node\_has\_popup[child]) {

                child->hidden = false;

            }

            else {

                child->hidden = true;

            }

        }

    }

    return has\_popup;

}

void open\_popup(DomNode\* root){

    if (!root) {

        return;

    }

    bool has\_pop = dfs(root);

    if (has\_pop) {

        root->hidden = false;

    }

    return;

}

/\*

// Initial State:

```

    //              ROOT

    //          /     |    \\

    //         /      |     \\

    //       B       C        D

    //    /   |            /  | \\  \\

    //   /    |           /   |  \\  \\

    // F      G      (POPUP)  I  J  (K)

    //              /   |   \\   / \\

    //             /    |    \\  Z  Y

    //            N     O    (P)

    // After openPopup called:

    //              ROOT

    //          /     |    \\

    //         /      |     \\

    //      (B)      (C)      D

    //    /   |            /  | \\   \\

    //   /    |           /   |  \\   \\

    // F      G       POPUP  (I)  (J) (K)

    //              /   |   \\

    //             /    |    \\

    //            N     O     (P)

Requirement:

    Find POPUP, make all the sibling of POPUP to hidden

    Find out POPUP's parent, make all the sibling of parent to hidden

\*/

```

```java

public static class DomNode {

    String id;

    boolean hidden;

    List<DomNode> children;

    public DomNode(String id, boolean hidden, List<DomNode> children) {

        this.hidden = hidden;

        this.id = id;

        this.children = children;

    }

    public String toString() {

        return hidden ? "(" + this.id + ")" : this.id;

    }

}

public static DomNode openPopup(DomNode root) {

    if (root == null) return root;

    Queue<DomNode> queue = new LinkedList<>();

    queue.offer(root); //Enqueue

    while (!queue.isEmpty()) {

        int size = queue.size();

        List<DomNode> currLevel = new ArrayList<>();

        DomNode parent = null;

        boolean foundInLevel = false;

        for (int i = 0; i < size; i++) {

            boolean found = false;

            DomNode node = queue.poll(); //Dequeue

            currLevel.add(node);

            List<DomNode> children = node.children;

            for (DomNode c : children) {

                queue.offer(c); // Enqueue

                if (c.id.equals("POPUP")) {

                    // set POPUP to visable

                    c.hidden = false;

                    found = true;

                    foundInLevel = true;

                }

            }

            if (found) {

                // set all siblings to hidden

                for (DomNode c : children) {

                    if (c.id.equals("POPUP")) continue;

                    c.hidden = true;

                }

                parent = node;

            }

        }

        // set POPUP's parent's siblings to hidden

        if (foundInLevel) {

            for (DomNode sibling : currLevel) {

                if (sibling.equals(parent)) continue;

                sibling.hidden = true;

            }

            break;

        }

    }

    return root;

}

public static void main(String[] args) {

    // POPUP node

    DomNode N = new DomNode("N", false, new ArrayList<>());

    DomNode O = new DomNode("O", false, new ArrayList<>());

    DomNode P = new DomNode("P", true, new ArrayList<>());

    List<DomNode> popupChildren = new ArrayList<>();

    popupChildren.add(N);

    popupChildren.add(O);

    popupChildren.add(P);

    DomNode POPUP = new DomNode("POPUP", true, popupChildren);

    DomNode I = new DomNode("I", false, new ArrayList<>());

    DomNode J = new DomNode("J", false, new ArrayList<>());

    DomNode K = new DomNode("K", true, new ArrayList<>());

    List<DomNode> dChildren = new ArrayList<>();

    dChildren.add(POPUP);

    dChildren.add(I);

    dChildren.add(J);

    dChildren.add(K);

    DomNode D = new DomNode("D", false, dChildren);

    DomNode C = new DomNode("C", false, new ArrayList<>());

    DomNode F = new DomNode("F", false, new ArrayList<>());

    DomNode G = new DomNode("G", false, new ArrayList<>());

    List<DomNode> bChildren = new ArrayList<>();

    bChildren.add(F);

    bChildren.add(G);

    DomNode B = new DomNode("B", false, bChildren);

    List<DomNode> rootChildren = new ArrayList<>();

    rootChildren.add(B);

    rootChildren.add(C);

    rootChildren.add(D);

    DomNode root = new DomNode("ROOT", false, rootChildren);

    System.out.println("Before: ");

    bfs(root);

    openPopup(root);

    System.out.println("After: ");

    bfs(root);

    // System.out.println(bfs(root));

}

private static void bfs(DomNode root) {

    Queue<DomNode> queue = new LinkedList<>();

    queue.offer(root);

    while (!queue.isEmpty()) {

        int size = queue.size();

        for (int i = 0; i < size; i++) {

            DomNode node = queue.poll();

            System.out.print(node.toString() + ", ");

            for (DomNode c : node.children) {

                queue.offer(c);

            }

        }

        System.out.println("");

    }

}

```

Leetcode 111 102 242 49 127 鬣扣幺幺幺 + 鬣扣幺铃儿 鬣扣洱斯尔 + 鬣扣斯酒 鬣扣幺儿齐

322. Coin Change

380. Insert Delete GetRandom O(1)

465. Optimal Account Balancing

111. Minimum Depth of Binary Tree

102. Binary Tree Level Order Traversal

242. Valid Anagram

49. Group Anagrams

217. Contains Duplicate

146. LRU Cache

713. Subarray Product Less Than K

126. Word Ladder II

362. Design Hit Counter

3076. Shortest Uncommon Substring in an Array