Leetcode原题（绿色高亮为近期高频）

- 10. Regular Expression Matching
- 36. Valid Sudoku
- 37. Sudoku Solver
- 41. First Missing Positive
- 42. Trapping Rain Water
- 43. Multiply Strings
- 44. Wildcard Matching
- 56. Merge Intervals
- 59. Spiral Matrix II
- 72. Edit Distance
- 91. Decode Ways
- 95. Unique Binary Search Trees II
- 96. Unique Binary Search Trees
- 115. Distinct Subsequences
- 128. Longest Consecutive Sequence
- （2）134. Gas Station
- 146. LRU Cache
- 150. Evaluate Reverse Polish Notation
- 152. Maximum Product Subarray
- 153. Find Minimum in Rotated Sorted Array
- （3）162. Find Peak Element
- 165. Compare Version Numbers
- 200. Number of Islands
- （2）205. Isomorphic Strings
- 218. The Skyline Problem
- 222. Count Complete Tree Nodes
- 248. Strobogrammatic Number III
- （3）249. Group Shifted Strings
- （3）253. Meeting Rooms II
- 264. Ugly Number II
- 269. Alien Dictionary
- 270. Closest Binary Search Tree Value
- 274. H-Index
- 286. Walls and Gates
- 289. Game of Life
- 295. Find Median from Data Stream
- 297. Serialize and Deserialize Binary Tree
- 298. Binary Tree Longest Consecutive Sequence
- 305. Number of Islands II

- （2）308. Range Sum Query 2D - Mutable
- 312. Burst Balloons
- 334. Increasing Triplet Subsequence
- 337. House Robber III
- （2）340. Longest Substring with At Most K Distinct Characters
- 346. Moving Average from Data Stream
- （4）359. Logger Rate Limiter
- 362. Design Hit Counter
- 361. Bomb Enemy
- 392. Is Subsequence
- 394. Decode String
- （4）399. Evaluate Division：会写Union-Find解法
- （5）418. Sentence Screen Fitting
- 438. Find All Anagrams in a String
- 465. Optimal Account Balancing
- （2）490. The Maze
- （2）486. Predict the Winner
- 494. Target Sum
- 499. The Maze III
- （2） 496. Next Greater Element I：follow up是如果data是streaming data, 要怎么改代码和设计输出。
- （2）505. The Maze II
- 527. Word Abbreviation
- 529. Minesweeper
- 540. Single Element in a Sorted Array
- 552. Student Attendance Record II
- 560. Subarray Sum Equals K
- 621. Task Scheduler
- 652. Find Duplicate Subtrees
- 659. Split Array into Consecutive Subsequences
- 674. Longest Continuous Increasing Subsequence
- 684. Redundant Connection
- （3）685. Redundant Connection II
- 721. Accounts Merge
- 727. Minimum Window Subsequence
- 731. My Calendar II
- 734. Sentence Similarity
- 737. Sentence Similarity II
- 739. daily temperature
- 750. Number Of Corner Rectangles
- （3）753. Cracking the Safe
- 765. Couples Holding Hands
- 769. Max Chunks To Make Sorted

- 773. Sliding Puzzle
- 787. Cheapest Flights Within K Stops
- 802. Find Eventual Safe States
- 805. Split Array With Same Average
- （4）815. Bus Routes

面经题目

（高频）带时间戳expiration的hashmap以及删除expired的entry

有一堆task，有expiration time，比如1000ms之后expire。然后实现一个generic的hashmap，能够add task，get task（如果task已经expire，就delete）。这两个都实现了，最后要求加一个可以自动clean up没有被get过，但是已经expire的task

应该可以用HashMap + DoublyLinkedList，类似LRU的做法。然后**定期**从头遍历DLL，删除过期的entry from DLL & Map。
如果对于所有task，experation是固定值的话，感觉也可以用circular array做，这样clean up会比较快，空间也比较节省，代码大概这样（get时不更新timestamp的写法）：

```
import java.util.HashMap;
import java.util.Map;

/**
 * get O(1), put O(1), clean O(k), S(k), k is expiration
 */
public class HashMapWithExpirationCirularArray implements IHashMapWithExpiration {
    private int expiration;
    private Map<Integer, Node> map;
    private Node[] arr;

    public HashMapWithExpirationCirularArray(int expiration) {
        this.expiration = expiration;
        map = new HashMap<>();
        arr = new Node[expiration];
    }

    @Override
    public int get(int key) {
        if (map.containsKey(key) && isExpired(map.get(key))) {
            map.remove(key);
        }
        return map.containsKey(key) ? map.get(key).val : -1;
    }

    private boolean isExpired(Node node) {
        return System.currentTimeMillis() - node.timestamp <= expiration;
    }

    @Override
    public void put(int key, int val) {
```

```
        int index = (int) (System.currentTimeMillis() % expiration);
        if (arr[index] != null) {   // clear expired entry
            map.remove(arr[index].key);
        }
        arr[index] = new Node(key, val);
        map.put(key, arr[index]);
    }

    @Override
    public void clean() {
        for (Node node : arr) {
            if (isExpired(node)) {
                map.remove(node.key);
            }
        }
    }

    class Node {
        int key;
        int val;
        long timestamp;

        public Node(int k, int v) {
            key = k;
            val = v;
            timestamp = System.currentTimeMillis();
        }
    }
}
```

Moving puzzle

3x3的grid，放了随机打乱的1-8个数字，和一个empty，最小步数让它还原成1-8
https://www.cs.princeton.edu/courses/archive/fall12/cos226/assignments/8puzzle.html
DFS or BFS, even A*?
就是LC773. Sliding Puzzle

Optimal list of job

list of job, 每个job包含id, required_cpu, priority, 告诉了total available cpu，返回optimal list of job, 讨论了好久好久怎么定义optimal
这个题可能比较open，看你跟面试官讨论如何定义optimal了，我的思路是考虑priority per cpu
Greedy

Find kth frequently number in int[]

HashMap + PriorityQueue

（高频）汇率转换

要写一个online的单位转换器，可以添加随时新的转换关系，e.g. 'USD' -> 'RMB' = 6.8, 'RMB' -> 'JPY' = 10之类的，然后也要支持query，难点在于转换关系可以传递也可以逆转，所以就用了hashmap存adjacent list，然后讨论了应该在每次添加新的关系的时候遍历并且compress path因为config应该没有query的call频繁，小哥表示同意，就开始写两个API，path compression就用DFS做了
类似LC399
聊了时间复杂度，比较了BFS/DFS/UnionFind

Range random query

如果知道有一大堆数据， 数据中只有ABCD, 根据已有数据，得到每个的概率
HashMap

Follow up: 给定每个字符串的出现概率，让实现一个random generator去按概率随机生成字符串，要求时间复杂度越低越好，空间尽量优化，所以不能二分查找。如果是整数就直接用一个array，如果精度很高大概就是用多级不同精度的array进行查询
整数的就是比如A占10%， B占20%， C占30%， D占40%， 然后**随机一个1-100的数，如果是10-30之间就选B**。非整数的情况就要分得更细。（有点consistent hash的感觉！）

Follow up : 给文章怎么根据每个单词知道它下一个单词概率
Trie，node保存frequency

（高频）多少种走法in matrix

给定一个矩形的长宽，用多少种方法可以从左上角走到右上角 （每一步，只能向正右、右上 或右下走）
Two pass DP即可

follow up：如果给矩形里的三个点，要求解决上述问题的同时，经过这三个点
纵向切割矩形，一个一个地做DP，然后相加

follow up：如何判断这三个点一个是合理的，即存在遍历这三个点的路经
DP时看是否可达就好了呗

follow up：如果给你一个H，要求你的路径必须向下越过H这个界，怎么做
可以再做一次 dp，但是只走 <= H 的路径，再用总数减一下

Follow up：要经过某些特定row怎么走？要先经过一个row再经过另一个row怎么走？
也是矩阵切割的思想，但是要处理先后顺序

Semantic match query

两个query， 每一个都有几个token， 然后给你dictionary， 你需要对这个dictionary先处理， （dfs）之类， 然后问这个query semantic是不是一样， 就是每个单词能不能map到相同的connected graph里。 这个map怎么建立， 要看面试官的需求，是query 被重复call， 还是dictionary经常被更新。
感觉有点类似Accounts merge，HashMap + UnionFind。为每个unique string赋一个id即可，然后用id做Union Find。

设计一个黑白棋游戏的后端API

规则就是如果落子之后可以将同一条线上对方的棋子两头都堵住，那么对方的棋子就会变成你的棋子。比如某一行原来是 ABBB，我走一步变成 ABBBA，那么最终结果就会是AAAAA
需要设计和实现API完成判断落子是否合法，是否有玩家取得胜利，落子之后新的棋盘情况，以及一个提示玩家可选落子位置的API。在墙上写了落子合法性判断和落子后新棋盘更新，之后口述了剩下API的设计和实现方法

找到一条线将坐标系中的点分成两等份

题目是在一个坐标系里给很多离散的点，找到一条线将所有点分成两个数量相等的集合，不用考虑基偶性或者多个点在一条线上的特殊情况。
这题似乎和人工智能的分类器之类的有点关系，可惜楼主基础不牢一开始基本完全懵逼。提出了个O（n^3）的算法，然后只能想到用heuristic来优化。

面试官只说了句可以更快，然后就等我自由发挥，在楼主卡壳五分钟尝试了各种天花乱坠的解法后，问了我一个问题：总共有多少条符合条件的线？
在这个提示下楼主提出了一假设：过任意一点，一定能找到另一个点将坐标系中其余的点分成两个数量相等的组（不知道怎么证也来不及证，但是直觉告诉我是对），基于这个假设之前那个算法的实际复杂度是O（n^2），结合heuristic实际情况会更快

直接比较就好了把，点(x', y') 如果满足 ax' + b < y'，那就在线下面了
比较 tricky 的地方就是和 y 轴平行的线

（高频）长凳坐人问题

有一张长凳一开始分散的坐着一些人，每个新来的都想坐在最宽敞的一段的中间位置，问：如何模拟这一过程
follow up是如果有多个长凳该怎么办？以及如果长凳的数据太大，内存装不下又该怎么办？
follow up: 凳子上一开始没有人，然后一个一个往里面放，每次放的时候O(1)时间求放在哪里距离最大（数学问题）：优先队列，每次弹出最大的间隔，折半加入队列。

这个问题实际就是[0, n)被分成了若干个intervals。可以将intervals加入PriorityQueue，每次新来人的时候，取出最长的interval，split成相等的两个intervals再入队列即可。
对于Follow up，可以维护K个PriorityQueue，每次取出K个peek中最长Interval即可。
数据量太大，那么存文件呗？

Facebook的friend suggestion

Suggested friend should have the maximum common friends as the user while they are not friends already. 举个例子a：bcde (a认识bcde) f:zbcd(f认识zbcd) g:bcx(g认识bcx) 给a找朋友应该输出f.
HashMap

扫雷问题

初始化一个扫雷棋盘，随机布雷，这一问用蓄水池抽样做。
然后是实现一个click功能，点到一个位置，如果是雷游戏结束，不是的话，标出来周围几个雷，如果周围8个位置都没有雷，继续往外扩展，BFS（LC529. Minesweeper）

（高频）扫地机器人

Given a robot cleaner in a room modeled as a grid.
Each cell in the grid can be empty or blocked.
The robot cleaner with 4 given APIs can move forward, turn left or turn right.
When it tries to move into a blocked cell,
its bumper sensor detects the obstacle and it stays on the current cell.

The 4 APIs are:

- Void clean(): clean the current location.
- Void turnleft(k=1): turn left k*90 degrees.
- Void turnright(k=1): turn right k*90 degrees.
- Boolean move(direction=None): move forward for 1 position, return False if that's not possible.

其中关于 `move` 这个 API 看到两个版本：一个是没有 parameter，每次就朝机器人面向的方向前进一步，所以需要自己在递归中维护方向；一个是可以传 direction 进去，让机器人直接朝那个方向走一步。

OOD，要求实现API
或者简单版本，要求用给定的API打扫完所有empty格子。

实现API

Room 负责记录实际的座标和 robot 的所在座标，用来判断 robot 是否撞墙，以及房间是不是已经干净，简单来说这个 API 有上帝视角。
Robot 只记录 robot 面向的方向，以及跟 Room 说我要朝这个方向走，由 Room 返回有没有撞墙。
具体实现其实不难，我只稍微提一下方位，和面经里的分享一样，我用了 0, 1, 2, 3 来代替方位，这样做的好处是要转换方位只需要 `(i + k) % 4` 就行。python 里面能直接 `(i - k) % 4`，也可以直接 `(i - k + 4) % 4` 先换成正数。

```
class Dirs:
    """
    The directions should be in order
    to make turnleft/right in Robot more convient
    if need 8-dirs, the order becomes:
    D, DR, R, UR, U, UL, L, DL
    """
    DOWN = 0
    RIGHT = 1
    UP = 2
    LEFT = 3
    DELTA = (
        ( 1,  0),
```

```
        ( 0,  1),
        (-1,  0),
        ( 0, -1),
    )


class Room:
    EMPTY = 0
    CLEANUP = 1
    OBSTACLE = 2
    ROBOT = 3

    def __init__(self, grid):
        """
        :type grid: list[list[int]]
        """
        self.__room = grid
        self.__cleanups = 0
        self.__robot_at = (0, 0)

        m, n = len(grid), len(grid[0])

        for x in range(m):
            for y in range(n):
                if grid[x][y] == self.CLEANUP:
                    self.__cleanups += 1
                elif grid[x][y] == self.ROBOT:
                    grid[x][y] = self.EMPTY
                    self.__robot_at = (x, y)

    def is_clear(self):
        """
        :rtype: bool
        """
        return self.__cleanups == 0

    def move_robot(self, direction):
        """
        :type direction: int, defined in Dirs
        :rtype: bool
        """
        m, n = len(self.__room), len(self.__room[0])
        x, y = self.__robot_at
        dx, dy = Dirs.DELTA[direction]
        _x, _y = x + dx, y + dy

        if not (0 <= _x < m and 0 <= _y < n):
```

```
            return False

        if self.__room[_x][_y] == self.OBSTACLE:
            return False

        self.__robot_at = (_x, _y)
        return True

    def clean(self, robot):
        """
        :type robot: Robot
        :rtype: void
        """
        if not isinstance(robot, Robot):
            return

        x, y = self.__robot_at

        if self.__room[x][y] == self.CLEANUP:
            self.__room[x][y] = self.EMPTY
            self.__cleanups -= 1

    def _get_robot(self):
        # for testing
        return self.__robot_at

    def _print_room(self):
        # for testing
        print(
            '\n'.join(str(r) for r in self.__room),
            '\nRobot at: ', self.__robot_at,
            '\nCleanups: ', self.__cleanups,
            '\n'
        )


class Robot:
    def __init__(self, room):
        """
        :type room: Room
        """
        self.__room = room
        self.__face = Dirs.DOWN

    def move(self, direction=None):
        """
        :type direction: int, defined in Dirs
```

```
        :rtype: bool
        """
        if direction in range(len(Dirs.DELTA)):
            self.__face = direction

        return self.__room.move_robot(self.__face) is True

    def turnleft(self, k=1):
        """
        :type k: int
        :rtype: void
        """
        n = len(Dirs.DELTA)
        self.__face = (self.__face + k) % n

    def turnrigt(self, k=1):
        """
        :type k: int
        :rtype: void
        """
        # note that, -1 % 4 == 3 in Python, or just (x - k + n) % n
        n = len(Dirs.DELTA)
        self.__face = (self.__face - k) % n

    def clean(self):
        """
        :rtype: void
        """
        self.__room.clean(self)

    def _get_face(self):
        # for testing
        return self.__face
```

调用API打扫屋子（DFS + 手动维护方向）

手动维护方向稍微 tricky 一些，可以对照代码仔细思考以下这三句话。
- 进格子：举个实例吧，假设当前位于 O 格子，上下左右分别为 UDLR，那么我要往周围移动的方向要顺着 DFS 的特点，D -> R -> L -> U（只要是十字形的移动就行，使得能够尽可能的直走，以及递归退回来的时候能面向进来时候的反向，比如 R -> U -> D -> L 也行）。
- 换方向：比如以下代码，是对应前一步进格子的 D，也就是往下走的部分 (在 robot_cleaner.py 的 L334-L338)

大白话就是，如果下方 (D) 没去过，而且没墙，就去 (DFS)，**回来之后 (机器人面向上方)** 转右边进去右方 (R)；如果不能去下方，那么 (机器人面向下方) 转左边进去右方。

- 出格子：要让递归返回的时候，Robot 刚好朝向进去格子的反方向(用前述十字形的移动)，如此才能在递归完准备离开当前格子的时候调用 robot.move() 离开。

```
def dfs(self, x, y, to_dir, robot, visited):
    robot.clean()
    visited.add((x, y))

    # down
    d = to_dir
    _x = x + Dirs.DELTA[d][0]
    _y = y + Dirs.DELTA[d][1]

    if (_x, _y) not in visited and robot.move():
        self.dfs(_x, _y, d, robot, visited)
        robot.turnrigt()
    else:
        robot.turnleft()

    # right
    d = (to_dir + 1) % len(Dirs.DELTA)
    _x = x + Dirs.DELTA[d][0]
    _y = y + Dirs.DELTA[d][1]

    if (_x, _y) not in visited and robot.move():
        self.dfs(_x, _y, d, robot, visited)
    else:
        robot.turnleft(2)

    # left
    d = (to_dir + 3) % len(Dirs.DELTA)
    _x = x + Dirs.DELTA[d][0]
    _y = y + Dirs.DELTA[d][1]

    if (_x, _y) not in visited and robot.move():
        self.dfs(_x, _y, d, robot, visited)
        robot.turnleft()
    else:
        robot.turnrigt()

    # up
    d = (to_dir + 2) % len(Dirs.DELTA)
    _x = x + Dirs.DELTA[d][0]
    _y = y + Dirs.DELTA[d][1]

    if (_x, _y) not in visited and robot.move():
        self.dfs(_x, _y, d, robot, visited)
```

```
        robot.turnrigt()

    # move robot when the recursion is back
    robot.move()
```

调用API打扫屋子（DFS + move 可传参方向）

没前面那么复杂，中心思想就两个。1) 如果能走，就直接过去 2) 如果走到一个走过的格子，就退回去，然后转回原来的方向

```
def dfs(self, x, y, from_dir, robot, visited):
    # is there a api to detect the cell need to clean?
    robot.clean()
    visited.add((x, y))

    for to_dir in range(len(Dirs.DELTA)):
        if to_dir == from_dir:
            continue

        # to_dir is index and also the direction defined in Dirs
        dx, dy = Dirs.DELTA[to_dir]
        _x = x + dx
        _y = y + dy

        if (_x, _y) in visited:
            continue

        if robot.move(to_dir):
            self.dfs(_x, _y, (to_dir + 2) % len(Dirs.DELTA), robot, visited)
        else:
            visited.add((_x, _y))

    robot.move(from_dir)
```

然后follow up是如果此机器人有电池电量的限制，如何能尽量减少耗电，其实就是如何能少走几步冤枉路，我说可以把之前撞过的墙都存起来，以后再调用move之前先check一下下一步是不是墙，是就不用往那个方向move了。他说还能不能更优化，比如按房间形状，我没答出来，后来问问题问他怎么弄，他说可以根据墙的一个点试试延伸一下把其他墙都一起存起来更好，不过是bouns，没期待我答出来。

（高频）找最大font fit sentence on screen

已知screen的高和宽，给你最小和最大的fontSize，要求给定一个string，将string用尽可能大的fontSize显示在screen里。字符串如果是句子的话，单词不能被截断，要分配到下一行。
已知两个API getHeight(int fontSize), getWidth(char c, int fontSize)，可以得到每个character在不同fontSize下的高和宽。
和面试官交流后，确认string可以拆分成几行显示在screen中，先提出暴力解法，然后用二分法优化。

可以将屏幕按照font的宽高换算成有多少rows * cols，然后用418. Sentence Screen Fitting的思路判断是否能装下一个字符串或者句子。
上面这个作为binary search的条件，最终锁定满足条件的最大字体。（应该是返回right index）

```
public int getLargestFont(String s, int height, int width
        , int smallestFontSize, int biggestFontSize) {
    int left = smallestFontSize;
    int right = biggestFontSize;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (!canFit(s, height, width, mid)) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return right;
}

/**
 * Decide if can fit a whole string, a char cannot be split into two lines.
 */
private boolean canFit(String s, int height, int width, int font) {
    int rows = height / getHeight(font);
    int len = s.length();
    int i = 0;

    for (int row = 0; row < rows && i < len; ++row) {
        int w = 0;
        // fit as much chars as possible in current row
        while (i < len && w + getWidth(s.charAt(i), font) <= width) {
            w += getWidth(s.charAt(i), font);
            ++i;
```

```
            }
        }

        return i >= len;
    }

    /**
     * Decide if can fit a sentence, a char or a word cannot be split into two lines.
     */
    private boolean canFitSentence(String s, int height, int width, int font) {
        int rows = height / getHeight(font);
        String[] words = s.split(" ");
        int n = words.length;
        int i = 0;
        int currWidth = 0;

        for (int row = 0; row < rows && i < n; ++row) {
            while (i < n && currWidth + getWidth(words[i], font) <= width) {
                ++i;
                currWidth += getWidth(' ', font);   // append a space between words
            }
        }

        return i >= n;
    }

    private int getWidth(String s, int font) {
        int width = 0;
        for (int i = 0; i < s.length(); ++i) {
            width += getWidth(s.charAt(i), font);
        }
        return width;
    }
```

```
 assume different characters have the same height for the same font
def getLargestFont(s, H, W, smallest, largest):

    small = smallest
    large = largest
    while small <= large:
        mid = small + (large - small) // 2
        if canFit(s, mid, H, W):
            small = mid + 1
        else:
            large = mid - 1
```

```
    if large < smallest:
        return -1    # even the smallest number cannot fit
    else:
        return large

def canFit(s, font, H, W):
    curH = H
    curW = W
    length = len(s)
    i = 0
    while i < length:
        if getHeight(s, font) < curH and getWidth(s, font) < curW:
            curW -= getWidth(c.font)
            i += 1
        elif getHeight(s, font) < curH and getWidth(s, font) > curW:   # need to restart with a new
line
            curH = H - curH
            curW = W
        else:
            return False

    return True
```

0-1矩阵找connected components count

find component number in array， leetcode上的那个isolate island那题
然后follow up加一个方法可以把其中的0变成1，然后实时计算component数量
其实就是Number of islands I & II
第一问用DFS/BFS/Union-Find都可以
第二问用Union-Find好做，每加一个1进去，就和四周的1分别union一下，只用union一次就可以

（高频）Maximum sum rectangle in a 2D matrix

https://www.geeksforgeeks.org/dynamic-programming-set-27-max-sum-rectangle-in-a-2d-matrix/
枚举left col和right col，然后以row为单位进行DP。这里对每行计算presum加速。时间复杂度是O(n ^ 3)

用最少的正方形填满矩形

给你一个矩形，用最少的square填充满，我当时想用dp，后来讨论了一下发现不太行，然后他给了一堆hint让我用greedy，然后之后又说greedy也不是最优的，我就懵逼了，之后问问题我问他该咋做，他说这个没有解。。。只能找个optimal solution，类似np
Greedy

（高频）匹配自行车和人

有很多自行车和很多人，如果完美匹配自行车和人，就是匹配最近的自行车和最近的人，至少有一个解，自己设计数据结构。

这道题很开放，需要跟面试官积极交流，厘清条件。

- 比如是人多还是自行车多？如果是人多、自行车少，那么从自行车出发做计算会比较高效。
- 如果出现相等距离怎么办？例如自行车a到两个人x、y的距离相同，match谁呢？

Solution 1: BFS

我用图来存，然后我从自行车做bfs，碰到最近的人就assign。

Solution 2: Greedy, HashMap + MinHeap

我最后写的是一个比较简单的解法，假设函数给定一堆人和车以及他们的坐标，然后求出所有人车距离，把这些数值放在一个minh8eap里面，每次拿最小的距离，同时记录下来这个人和这个车已经match过了，这样依次匹配。**这种解法的缺点是没考虑tie的情况**

之后follow up是如果有的时候很多人到同一辆车距离相同，怎么assign，要求全局最优，

我理解的全剧最优就是让每个人走的路加起来和最小，这样找到一个解使得匹配之后所有人和车的距离之和是最短的。
可以用Greedy，每次都match到最短的pair，但这种未必能确保达到最优解。

或者可以用人为起点做了个bfs，然后得出<Node, distance>的preference list。然后做dfs，找出最短的distance之和。烙印说可以，不过没写代码。

我问他还有什么更好的解法吗，他说有个汉密尔顿什么什么的算法

（高频）找出单词里所有的extension

Extension的定义是，有的人喜欢打字说heeeeloooo，hiiiii，给了个规则说**相同字符连续超过三个就算是extension**，找出一个string中所有的extension的start／end index，自己定义数据结构。

Two-pointer，用个pair来存start index和end index。

follow up是给你一个字典，字典中的单词都是不含extension的。然后给你一个单词，可能有extenstion可能没有，问**去掉了extension的这个单词在不在字典里**。例如，输入是一个有很多extension的单词，比如heeellooo，hiiii，字典是［hello，hi，word］，问你输入的词精简后在不在字典里，比如heeellooo精简后可能是hello，所以在字典里，catttt精简后是cat或者是catt，不在字典里。

对于follow up，**因为exntesion的定义是连续超过三个字母，而又不可能把extension删的一个字母不剩，所以每个extension最后只会留下一个或两个字母**，所以把2^n个可能的情况穷举一遍就行了，或者动态规划也可以，令s为原字符串对所有extension只保留一位后的字符串，f[j][0..1]表示当前处理到s，在trie树上走到j节点，s重复过一次或两次。不过我觉得动态规划应该是想要的解法...感觉好复杂我的妈，要不还是DFS吧？

OOD问题

设计一个ball class，有质量和重量，重量会随环境改变。

是男人就跳问题

给一个二维坐标平面和一个起始点，从这点开始垂直下跳，下面有若干水平挡板，位置长度会在input里给，板的形式是（x, y, distance)，当跳到挡板上时，可以选择走到挡板的左端或者右端，然后继续垂直下跳，直到落地位置，求从开始到落地需要走过的路程最短是多少。

对所有板按照 y 轴从上往下排序，然后给每个板两个编号，id[i][0] 和 id[i][1] 分别表示第 i 个板的左右端点

如果从 id[i][0] 下落能到板 j，那么可以连接两条边 id[i][0] -> id[j][0] 和 id[i][0] -> id[j][1]，权重的话就是下落之后左右走的距离
这样就能构造一个有向图，最后求个最短路

Event fire问题

implement一个event_fire function, 每当有一个event的时候就会调用这个event，input为当前的时间，如果在过去t秒内有超过n个event call了，就return True，否则return False。follow up是优化worst case和space。

假设event的总数为N, 我最开始的思路是用queue, 每当有新的event的时候就放到queue尾，然后从queue的头部开始比对event是不是outdated, 如果是就pop出去，最后判断下queue的长度是不是长于n, 这样的话amortized时间复杂度是O(1)，但是worst case下时间和空间均可能为O(N)。

优化完的思路为每次把新的event放到队尾后，如果queue的长度大于n，就一直pop到n个为止，然后判断queue的头有没有outdated, 这样space不会超过n, worst case时间复杂度也是O(1)， 其实就换了个判断的思路。

Left-diagonally print a list of string

(想象把string排成一个一列), string的长度不一定相等。

Iteration呗？简单

01 matrix toggle to all 1

給定一个全是0和1的matrix, 只给定两种操作，分别是toggle一行和toggle一列，问能不能把matrix全变成1.
我用的是DFS+记录，就是把当前matrix的状态作为一个点，然后做dfs.
面试官给的方法是根据操作可以交换性(先把第一行toggle，再toggle第一列和反过来是一样的),假设是m*n的矩阵，每一行或一列要么toggle要么没有，所有可能的操作方案为2^(m+n),试试就好了。具体实现的话如果m和n不是很大，可以枚举二进制数，来遍历所有操作方案。

用n台电脑排序天文数字级数量的数字

divide and concque，分成等长的c个list，quick sort，然后merge sort，问了复杂度，(n/c)log(n/c) + nc，merge sort部分的复杂度有卡壳，不过提示之后还是做出来了。又问有没有快的？语塞，提示space随便用，每个list都有自己的min，开始想到存起来，最后得出用priority queue，merge复杂度降到nlogc。

多线程分布式计算number sum

N个很大的file 里面都是数, 和无限多个machine. 有一个函数 : int computeSum(fileID, machineID) 可以让一个机器计算一个file里面数的和。 最后求所有N个file的总和。
我说可以循环来call 这个函数， 然后累计结果。 但是很慢， 最好能用Thread.
然后大哥说好，然后给了thread的结构。
然后写code, 两个循环，一个建立thread, 一个循环加结果。 大哥说ok。

follow up: machine会有硬件原因卡住无法出结果， 然后computeSum 会return 多一个status, 来告诉你是不是有故障。 问code如何改进来避免卡住超时。
我就比较懵，没有实际thread的工作经验。。。不知道怎么能catch住哪个status, 说了几个想法大哥都摇头。 然后给hint说可以写一个wrapper函数传到thread里，然后让写哪个wrapper函数。
实在时间不够就写了几行， 然后大概意思是如果有问题就用 k*N+i 的 machine再跑一遍，如果还错就k+1, 直到得到status是正常。 没时间写完整代码了，就意思了一下。

follow up2: computeSum 这个函数有p 的概率计算结果是错的， 如果改进让总结果错误率小于p. 本来大哥说只剩一分钟了没时间不问了， 我说你说说我就听听，然后其实也不知道怎么弄，算概率还算错了（很傻）。大哥就上来指点了下就说算了

我的理解是wrapper()函数里面循环地call 那个computeSum(fileID, machineID)，如果状态对就退出循环并且return. 如果状态不对就在machine id 上加N， 继续运行computeSum(), 直到状态对。
类似写一个while(true) 这样的循环吧。
. 1point3acres
然后这个wrapper是在thread的结构器里的。

字符映射问题char mapping

先面了到 简单的题， abc -> edf， a map成e， b map成 d， c map 成f， 多个字母可能map 到同一个字母， 比如 a， e 都可以map 成 f
followup ：在第一步map 完之后， 比如 a map 成 b 之后 ， 原字符串变成了 bbc ，这时候再把b map 成d， 就破坏了之前的map， 问怎么办， 我说中间加一层， 把b map 到d 这个过程改成b map 到4， 4 再 map 到d， 小哥说ok， 那能不能写一个函数， 判断我们在map 的过程当中需要返回是否要加这中间一层的过程

IP address blacklist matching

给一个black list，里面都是ip address，ip address里可能包含wildcard字符 "*", 问如果现在有一个ip address，它是否在black list里。和面试官交流后，确认ip address都是合法的，而wildcard只会出现每个地址的最后，直接用trie做了。

（高频）王位继承

王位继承，长子及其后代有优先级继承权，其次是次子及其后代，以此类推。完成以下几个API：

- void birth(String parent, String name)　　// 父亲名字和孩子名字，生个娃
- void death(String name)　　// 此人要死
- List<String> getOrderOfSuccession()　　// 返回当前的继承顺序，死人不能出现

三个function会被反复调用，实现function细节。

HashMap + DFS

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class SuccessionOrder {
    private Map<String, Node> map;
    private Node root;

    public SuccessionOrder() {
        map = new HashMap<>();
        root = new Node("King");
        root.isDead = true; // current king is set to dead
    }

    public void birth(String parent, String name) {
        Node pNode = map.get(parent);
        Node node = new Node(name);
        pNode.children.add(node);
    }

    public void death(String name) {
        map.get(name).isDead = true;
    }

    public List<String> getOrderOfSuccession() {
        List<String> order = new ArrayList<>();
        dfs(root, order);
        return order;
    }

    private void dfs(Node root, List<String> order) {
        if (root == null) {
```

```
            return;
        }

        if (!root.isDead) {
            order.add(root.name);
        }

        for (Node child : root.children) {
            dfs(child, order);
        }
    }

    class Node {
        String name;
        List<Node> children;
        boolean isDead;

        public Node(String name) {
            this.name = name;
            children = new ArrayList<>();
            isDead = false;
        }
    }
}
```

248. Strobogrammatic Number III变型题

给定一个范围，比如1-100，输出所有翻转180度后还在这个范围内，且不等于它本身的数。

公交车管理问题

公交车站里面有若干公共汽车， 类似这样 terminal:{bus1, bus2, bus3, ...}， bus是一个类， 有int id, String company和一个出发时间 int time. 然后让实现几个函数 ：

- add(bus) 向一个车站里加入一辆车
- getnext() 得到下一辆出发的车
- dispatch() 让下一辆车从车站出发
- removeAll(company) 除掉车站中某一个公司的所有车。

问每个函数的时间复杂度。

Map<String, PriorityQueue> companyToQueue，k个company，每个队列长度为L，则

- add(bus): O(logL)
- getNext(): O(k)
- dispatch(): O(k)
- removeAll(company): O(1)

follow up, 自己实现priority queue 来实现上面的每个问题。

Decide complete binary tree

判断一个binary tree是complete binary tree
In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

isFull(TreeNode) and isComplete(TreeNode)
类似222. Count Complete Tree Nodes
用level order traversal判断？将null也进队列，如果在队列中遇到null了，则读取queue中剩余所有元素，全都为null即为complete tree。

Follow-up: 给一个sorted array， construct complete binary tree.

（高频）Choose a random point uniformly in the list of rectangles

Given a list of rectangles which are **not intersect with others** ,each rectangle has four points, write a method to choose a point uniformly for the list of rectangles

1 算所有的矩形的面积和 S 然后给每一个矩形分配一个对应的区间 比如 3个矩形的面积分别是1，4，9 那可以分别分配[0,1],[1,5],[5,14]
2 随机产生[0，S] 里的数 通过这个数落在哪个区间里来决定哪个矩形被选中.
3 在这个矩形里 根据分别在x 和y 的方向上产生随机坐标 得到的点就是符合要求的点

Follow-up: given a list of rectangles, write a method to choose a random point uniformly in the list of rectangles （**maybe overlapped**）

初步想法是process一遍所有的矩形，处理掉overlap的部分，然后以面积为weight，sample出在哪个矩形取点，然后再sample取哪个点
看了地里面的面经：第一种是找一个最小的大框 把重叠的 矩形都包含住。然后在大框被 随机选点。如果选的点不被包含在任何一个已有的矩阵里面的话 就再次随机选点 知道选中为止。第二种是对重叠的矩形进行分割

（高频）Speed Cluster

input是int[] speeds 长度是n 每个值的范围是1到n代表**没有duplicate**。描述了一个cluster的概念，example: {2, 4} 车速为4的位置但是并不能用4的速度来开，因为前面的车速为2。所以这两个车是一个cluster。再比如{2，4，3}, 3虽然比4小但是不能开到3因为前车是2.所以这三个车也是一个cluster。{2,4,3,1,5} 的话 2,4,3是一个cluster，1,5是一个cluster 要去输出List<Integer> 为各个cluster的长度。

答案：遍历一遍记录最小值么，如果没发现一个新的最小值的话就属于当前cluster，发现一个新的最小值就另起一个cluster。

```
Int min = Max.Integer, clusters = 0
For (int speed : speeds) {
        If (speed > min) extend current cluster;
        else  ++clusters, min = speed
}
```

Follow-up：黑姐姐说你刚才的那个function已经存在了 List<Integer> clustersLens(int[] speeds). 现在我要加一个车速n+1到speeds里面，我能插入的位置是0-n这n+1个index。每插入不同的位置新的List<Integer> clusterLens 会变成什么样。

简单的想了一下发现，新加的车速就是最大的车速，加在任何位置就只会影响那个位置的cluster的len，从len变成len+1。corner case就是加在第一个位置，会出来一个1为长度的cluster。

以刚才的例子speeds : {2,4,3,1,5} 为例，cluster长度为{3,2} 有6个位置插入，那么输出为6种情况新的cluster length，那么就是一个list<list<Integer>>。

```
{ {1,3,2} insert pos = 0
  {4,2} * 3 insert pos : 1~3
  {3,3} * 2}
```

Double map with eplision

设计一个map, map有一个eplison 比如 0.1
key是 double
put (2.0, "hello)

get的时候在 如果查询的key在 (original key - eplison, original key + eplison)的范围内，则返回original key的value. From 1point 3acres bbs
比如这个栗子

get(2.05) -> hello
get(1.98) -> hello

不考虑overlap collision的情况，面试官提示用bucket做

答案：
如果epsilon 是0.1，代表最小精度是0.1, 1/0.1 = 10, 10就是用来计算bucket的multipler
.1point3acres网

如果put(1.2, "hello") ，那就把 （1.2， "hello")这个pair放到 1.2 x 10 = 12 的bucket里面

如果要查找 get(1.22) , 那么就去bucket 1.22 * 10 = 12的bucket 和他的两边邻居的bucket里面找（ 11， 13，这个case直接在12就找到了，那么返回hello）
如果查 get(1.12)， 那么就去 bucket 1.12 * 10 = 11的bucket 和他两遍邻居（10, 12）找

因为epsilon是0.1. 如果我们把每个bucket的范围限制在0.1 (方便计算，乘以multipler映射成整数），那么我们知道符合结果的key如果存在，那么一定在他整数映射的bucket或者他两遍的邻居里面。

Implement一个Set class

要求4个O（1）时间的API： add, contains, remove, getRandom. 挺简单的OOD，因为getRandom也要常熟时间，所以每次删除就把最后一个元素和被删除的swap就可以.
HashSet + array + random swap

（高频）计算Log中TopK说话最多的用户

有一个hangout的log文件，每一行以preambles <username> "sentences"的形式排列。让找出topK个说话最多的用户。因为之前看了一些system design再加上第二轮的影响，把这道题想复杂了，说了很多种可以边处理边排序的数据结构。把小哥说的一愣愣的，最后才发现原来就是用heap把nlogn的复杂度降低到nlogK而已。。

followup：如何parse文件的每条记录；聊天记录有什么特殊情况会不能parse的；如果某句话非常长有好几M怎么办。如果file很大怎么办。

用preorder和postorder建树

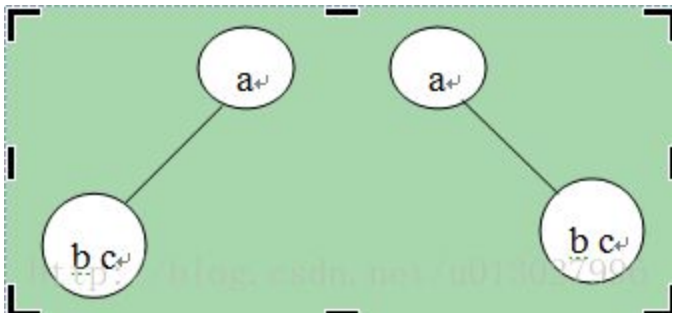
这道题出来时我说好像没有inorder是做不出来的，然后举了个例子，面试管说那就假设总是有left subtree的，然后楼主用recursion写了。完了分析时间复杂度，问怎么优化（hashmap 存一边postorder里的val 对应的Index）

由先序和后序遍历求解符合条件的n叉树个数的方法及程序实现

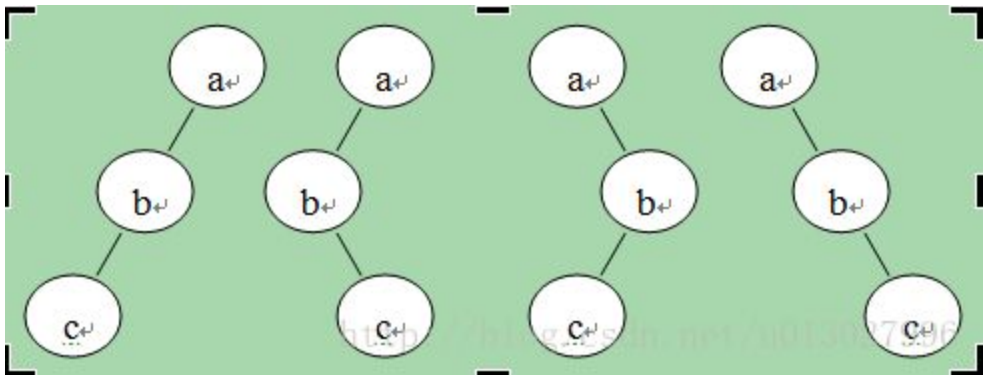
我们都了解二叉树的先序遍历、中序遍历和后序遍历，当知道先序遍历和中序遍历的结果时，可以唯一的确定二叉树;同样的，当知道后序遍历和中序的结果时，也可以唯一的确定二叉树。但是如果只知道先序遍历和后序遍历的结果时，二叉树就不是唯一的了，但是我们可以计算满足条件的不同二叉树的个数。同样，我们可以将问题推广到N叉树。下面我们以例题进行分析。

例一：已知二叉树的先序遍历为：abc，后序遍历为：cba，求满足条件的二叉树的个数。

分析：首先容易得出二叉树的根结点一定是a，再由剩下的先序遍历结点序列bc（第一个结点为b）和后序遍历结点序列cb（最后一个结点为b），可知结点bc共同组成根结点a的一个子树，且其中结点b一定是该子树的根结点。这个子树可以是根结点a的左子树，也可以是右子树。如下图所示：



所以，满足条件的二叉树的个数sum至少为2（sum=2）。又因为对于结点bc来说，c不管是其左结点还是右结点，都满足先序和后序遍历的要求。因此满足条件的二叉树的个数sum=sum*2=2*2=4。其形状如下图所示：



返回words里 包含了所有目标字符串里的所有char 的词中最短的那个

input: 一些字符串words， 一个目标字符串， 要求返回words里 包含了所有目标字符串里的所有char 的词中最短的那个。
eg: words = [study, haha, stone, school, star, store] target = "rts", 需要返回 的词是star, 因为star 包含了所有rts, 同时也是最短。

follow-up: 多种方法优化这道题的方法， 楼主先排序 （要求写出查找的平均时间复杂度），

楼主还答了可以把words 存成 s: study, stone, school, star, stor; t : star, stone, store..... 这样的map, **好处是我们查找的时候只需要看target里出现过的char所对应的词，然后找并集，在目标字符串很小的情况下有可能会省很多时间。**

（高频）Find string matching from stream

给一个char stream，有next(), 和hasNext(), 两个API。 给一些字符串作为目标字符串。要求每当char stream里出现目标字符串任何一个词，就打印这个词。
比如目标 'abc, att, bba, bc, abce', 然后我们对char stream call next， 出来的一些char 是 t, a, b, c, e, t.... 我们需要打印 abc, bc, abce

这个题如果**把词典的word逆序后建Trie树**，再用一个vector存最近遇到的max_length_of_word_in_dict个char，每次来一个char就逆序构造string到Trie中查找

反着建的话，从stream尾到stream头O(n)遍历一遍就能找到所有match到的词。从stream头到stream尾正着走的话，要考虑到每一个字符都作为match开头的情况，所以要检查O(n^2)次。

例：
stream里已有aabca，新来一个字符t，变成aabcat。
字典里有"cat","bcat","aabcat"。
反建trie的话得到t->a->c (isWord) ->b (isWord) ->a->a (isWord)。
aabcat从后往前遍历一次在trie里就能找到所有词。

（高频）Triangle array search

定义是先increase后decrease，无duplicate，要求

- 判断是否是triangle sorted：iteration O(n)
- 找min：O(1)
- 找max peak：binary search, O(logn)
- 找target number：按照peak位置分成两半，然后二分，O(log n)

变型题：有一个曲线，曲线的形状是先递减再增加，找曲线的最低点，如果只考虑int怎么做，如果考虑double怎么做
考虑double的话返回一个range，包含target即可。

随机生成一个Candy Crush

保险箱密码生成

有一个密码箱，密码是4位数，可以是0-9任何一位。在输入密码时可以连续输入，密码会自动匹配最后4位。假设密码为3456，我输入123456时就可以打开了，但是我总共输入了6位数。要求设计一个算法，使得输入的位数尽量少
带restart的
为什么的话你可以看一下详解，每次新添一位得不到新解的话，直接线性穷举出一个没出现过的解，作为新的起始四位数字就可以了。
我说用自动机做，面试官说别别别，你用贪心就行了。

高频面经题，不需要最优解，hashmap记录已经过的4位密码。

拿硬币和最大问题

有N个硬币排成一排，每次要你从最左边或者最右侧拿出一个硬币。总共拿N/2次，写一个算法，使能拿到的硬币的和最大。
Backtracking + memo

BST找range和

一个BST里面存着整数，要求写一个算法，计算树里面在(low, high)之间的数字的和 (low < val <= high)。

可以inorder得到sorted arr，然后遍历，O(n)
可以利用BST特性进行二分查找

（高频）数组shuffle问题

数组1是排好序的1-n个数字，数组2是根据数组1 shuffle得出，给定数组3，要求根据一样的shuffle规则变换成数组4，输出数组4。
第一个follow up，如果数组一是无重复数字组成的无序数组，如何做
第二个follow up，数组1有重复。

假如arr1 = {1, 2, 3, 4}, arr2 = {4, 2, 1, 3}
首先需要预处理arr2，得到val2NewIndex map
然后遍历arr1，新建index2NewIndex，那么index2NewIndex.put(i, val2NewIndex.get(a[i]))
对于arr3，根据index2NewIndex逐个填写元素到arr4中即可。

Follow-up 1
跟上面做法相同

Follow-up 2
将val2NewIndex map的每个值变成List<Integer>就行了

（高频）User CPU Peak问题

给了一堆log，log里有用户id，resource id以resource在某个起始时间和终止时间的使用量，比如用户abc在1到5秒钟使用了CPU的数量是2，用户abc在2到3秒使用的CPU数量是4，也就是一个用户对某个resource的使用在某个时间是可以叠加的，**给定一个resource id，根据用户对这个resource的peak使用量，找到top k的用户**。上面的例子中abc的CPU的peak使用量是2+4=6

follow up：如果数据量很大怎么办。

这题就是找根据resource id，找到每个user的这个resource的使用情况，然后根据start，end time排序，从左到右扫，每个时间段，开始时间标识+1，结束时间标识-1，记录最大值即可。
每个log按照开始结束分成2段，(int time, int cpu, boolean isStart)，然后排序，扫描即可。

Follow up: 数据量很大的问题也没有太好的办法，如果时间范围跨总体很小的话，可以用map-reduce。

数组中找孤儿

是给一堆数组，数组中的每个数都是一对，只有一个是单独，同时相同的数字一定都是挨着的，找到单独的数字
Iteration就行了，i += 2，比较i < length - 1 && a[i] == a[i + 1]

找到乘积最大的三个数字

给一个数组，找到三个数字, 他们相乘的乘积最大
不用sort，扫描一遍就搞定了，因为max product只可能是最大三个数的乘机或者是最小两个数和最大数乘机这两种可能

Insert into intervals

给一堆排好序的non-overlapped interva，要插入一个数字，如果这个数字可以和某个interval的起始时间或者结束时间挨着，就要把这个数字merge到那个interval之中，如果不挨着，就把数字当成interval插入到那堆interval当中，比如给定的interval是[1,2], [4,5], [9,10], 插入7的话，结果是[1,2], [4,5], [7,7], [8, 9]，插入3的话，结果是[1,5], [9,10]
很像56. Merge Intervals

（高频）拿卡片游戏

每张卡片都有一个值，给定一堆卡片从一头拿，每次可以拿一到三张，两人轮流拿，求最高得分。考虑卡片值为负的情况。
dp[i] is the max score of current player choosing from arr[i] to arr[n - 1]
dp[i] = sum[i] - dp[i + k], 1 <= k <= 3, sum[i] is sum of arr[i] to arr[n - 1]
return max(dp[0], sum[0] - dp[0]), because dp[0] is the max score of player1, and sum[0] - dp[0] is the max score of player2.

（高频）maximum vacation days

有几座城市，每个月在每个城市都有不同的假期，然后每个城市有飞往不同城市的航班，求最大能获取的假期和Path.
dp(i)(j) 代表第i个月在第j个城市所能获得的最大假期. DP 方程大概是 dp(i)(j) = Math.max(dp(i-1)(fromCity)+map(i)(j), dp(i)(j)

找超过数组长度1/4的数

输入一个排好序的数组，输出任意一个数量超过数组长度1/4的数。
先O(n)解决
优化一下，到O(logn)
类似skyline简化，给了一堆list (start, end, label), start是inclusive的，end是exclusive的，然后让输出所有连续的不同的region
就是 (2, 8, 'a'), (3, 5, 'c'), (6, 9, 'b')的话， 不考虑没有label的地方， 输出就是 (2, 3, 'a'), (3, 5, 'ac'), （5， 6， ‘a’）, (6, 8, 'ab'), (8,9, 'b') . from: 1point3acres

实现找k-th 最小的数api/数据结构，数据是在不断增加的, k也是在变的。 brute force就是来一个新数，就插入到一个数组中，O(n); 然后提出了用binary search tree + 存左树的结点数量，这样能做到 log（n） 插入和查找。

给了一棵树，里面有些结点需要被删除 （给了一个函数 need_to_delete(node)， 返回boolean表示是否要删除）。要求删除这些需要删除的结点，这样的话，这棵树会变成一个森林，输出这个森林。
基本就是BFS/DFS, 加删除节点

设计股票价格的记录系统

只观察一支股票的价格，实现几个function：

- vold update(double price, int timestamp) 更新/修正/删除股票价格，其中timestamp大部分时间是递增的，但是有时候发现过去的记录有错误，所以会对过去数据修正或invalidate。对过去数据修正是指input argument中的timestamp是一个已经记录在案的timestamp，让price为function新提供的price；invalidat时候function argument中的price为-1，删除timestamp对应的记录
- double max() 返回历史最大值
- double min() 返回历史最低值
- double current() 返回最近一次的记录

针对各种情况进行实现和算法复杂度讨论。（大量更新？大量查询？invalidate很多很少？etc.）

第一种实现：假设update频率比较低
Map<Integer, Double> time2Price;
Double min;
Double max;
Double current;

那么

- update：大多数O(1)，少数O(n)
- Max, min, current：O(1)
- S(n)

第二种实现：update频率较高
Map<Integer, Double> time2Price;
MinHeap<int[]> minHeap;
MaxHeap<int[]> maxHeap;
LinkedList<int[]> list;

那么：

- update：O(log n)
- max(), min(): 最好O(1)，最坏O(k log k)
- list：最好O(1)，最坏O(k)
- S(n)

产品中要monitor service response time，假设已经收集了大量数据，给定percentile的定义：如果说20ms的响应时间是90 percentile，代表收集的数据中90%的response time小于20ms，10%的response time大于20ms。同理，如果是说50 percentile，就是50%小于，50%大于。分析如何处理数据从而得到percentile。（这里假设了已经收集了大量数据，但是我们仍然就大量采集数据的系统环境进行了分析。最终还是回归到已经收集好的情况下）假设数据量大到memory无法存下，怎么处理？不考察系统设计，只根据实际需求在数据和算法层面进行优化。

01矩阵走路问题

0和1的grid，1是墙，0是路，从左上角走到右下角，最少多少步。
Follow-up: 现在说能把grid中的一个1变成0，问新的最小步数是多少步。

第一问BFS

Follow-up
题目是说最多能把一个 1 变成 0，那最优解就分两种情况：
* 原题解，路径上全是 0
* 路径上中间某个点是 1

这里考虑第二种，走的路径肯定是 "左上角 -> 0 -> 0 .. -> 1 -> 0 -> 0 -> 右下角"
然后，"左上角 -> 0 -> 0 .. -> 1" 就是求从左上角做为起点，每个 1 做为终点的 BFS，
类似，"1 -> 0 -> 0 -> 右下角" 就是求从右下角做为起点，每个 1 做为终点的 BFS
最后就是拼凑路径（枚举每个 1），把两段路径加起来求个最小值

设计贪吃蛇游戏

实现一个贪吃蛇游戏，贪吃蛇一开始在［0，0］位置且长度为1。每当贪吃蛇吃到食物，长度和获得的分数都增加一。对于每次move输出move之后的分数。如果贪吃蛇走出边界或和自己的身体相撞则游戏结束，输出－1。下面地图中 S是Snake， F是food

Follow-up: 多人玩怎么实现

给出一个地图长为3，宽为2，food的位置是[1,2],[0,1]

一开始snake出现在[0,0]，food出现在[1,2]

以下用S表示snake，用F表示food

|S| | |
| | |F|

snake.move("R"); -> Returns 0

| |S| |
| | |F|

snake.move("D"); -> Returns 0

| | | |
| |S|F|

snake.move("R"); -> Returns 1 (Snake eats the first food and right after that, the second food appears at (0,1))

| |F| |
| |S|S|

snake.move("U"); -> Returns 1

| |F|S|
| | |S|

snake.move("L"); -> Returns 2 (Snake eats the second food)

| |S|S|
| | |S|

snake.move("U"); -> Returns -1 (Game over because snake collides with border)

我写的代码，稍微有点问题，判断是否撞到自己时，要区分撞到身体还是撞到尾巴，撞到尾巴可能不会造成实际碰撞（因为尾巴会向前移动），但是有一种corner case就是，snake size = 2，然后向后撞向尾巴，会game over。

```
import java.util.*;

public class SnakeGame {
    private boolean isGameOver;
    private int[][] food;  // only one food available at the same time. If current food is ate, next food shows
    private int nextFood;
    private int width;
    private int height;
    private Deque<int[]> snake; // use Deque because we need to access both head and tail
    private Set<Integer> used;  // store snake for O(1) access

    public SnakeGame(int w, int h, int[][] food) {
        width = w;
        height = h;
        this.food = food;
        isGameOver = false;
        snake = new LinkedList<>();
        used = new HashSet<>();
        // snake initial at (0, 0)
        snake.offer(new int[]{0, 0});
        used.add(0);
        // decide if the snake initial position has a food already there, if has, score
        if (food.length > 0 && Arrays.equals(food[nextFood], new int[]{0, 0})) {
            ++nextFood;
        }
    }

    public int move(String direction) {
        if (isGameOver) {
            return -1;
        }

        int[] offset = new int[]{0, 0};
        switch (direction.toUpperCase()) {
            case "U":
```

```
            offset = new int[]{-1, 0};
            break;
        case "D":
            offset = new int[]{1, 0};
            break;
        case "L":
            offset = new int[]{0, -1};
            break;
        case "R":
            offset = new int[]{0, 1};
            break;
        default:
            break;
    }

    int x = snake.peekLast()[0] + offset[0];
    int y = snake.peekLast()[1] + offset[1];

    if (x < 0 || x >= height || y < 0 || y >= width  // invalid
            || used.contains(x * width + y)) {          // newHead hit snake itself
        isGameOver = true;
        return -1;
    }

    // add newHead to snake
    int[] newHead = new int[]{x, y};
    snake.offer(newHead);
    used.add(x * width + y);

    // check if score, snake length increase
    if (nextFood < food.length && Arrays.equals(newHead, food[nextFood])) {
        ++nextFood;
    } else {    // no score, snake remain same length, should remove tail
        int[] tail = snake.pollFirst();
        used.remove(tail[0] * width + tail[1]);
    }

    return nextFood;
  }
}
```

Follow-up: 把所有蛇存在一個list裏面，再用一個2D array去記錄有哪些格子已經被蛇占領了。