Decode password

建议自己练习一下文件读取，面试的时候是要求这个的，我用的 java scanner

(read file) how to optimize the space, only read the content needed

给一个文件 比如 password.txt,

[2, 3]

ABCDEFG

HIGKLMN

OPQRSTU

VWXYZAB

文件内容是是两个数字 (x,y) 和一个 matrix， matrix 左下角坐标(0,0)。

要求： 返回 matrix 中坐标为(x,y)的字符

比如上面返回 C

注意函数接口就是文件路径，所以感觉这道题就是在考文件读取

第二小问：同样文件， 需要解析多个这样的字符，然后根据 index 组成一个 password 返回

文件内容：

1

[2, 3]

ABCDEFG

HIGKLMN

OPQRSTU

VWXYZAB

0

[0, 0]

ABCDEFG

HIGKLMN

OPQRSTU

VWXYZAB

第一个字符为 C, index 为 1， 第二个字符为 V, index 为 0。所以返回“VC”

第三小问：在第二小问基础上，一个文件里有多个这样的 password。但找到第一个 password 结束

比如

1

[2, 3]

ABCDEFG

HIGKLMN

OPQRSTU

VWXYZAB

0

[0, 0]

ABCDEFG

HIGKLMN

OPQRSTU

VWXYZAB

0

[1,1]

ABC

DEF

GHI

1

[2,2]

ABC

DEF

GHI

只需要返回“VC”

1. coding，给定一个文本 path，文本样式：

[2,4]

S3KDA4

4ASDSD

ACEEDS

ASDEED

RTRYYU

读取文本并返回 coordinator 指定位置对的字符。[0,0]对应的是左下角的字符。

2. followup：同样文本路径，但是文件包含多个 block，每个 block 第一行是一个数字，对应的是这个字符在 password 里面的 index，要求返回这个 password。

1

[2,4]

S3KDA4

4ASDSD

ACEEDS

ASDEED

RTRYYU

0

[0,0]

I3KDA4

XTRYYU

给定的例子应该返回 XK

3. followup：和 2 一样文本路径和文本，但是有多个 password，要求返回第一个 password。具体怎么判断当一个 password 结束，是根据当前字符所在的 index 有没有重复。比如：

1

[2,4]

S3KDA4

4ASDSD

ACEEDS

ASDEED

RTRYYU

0

[0,0]

I3KDA4

XTRYYU

1　　#《---index 1 已经出现过，所以从这儿开始是第二个 password，可以返回第一个 password 为：XK

[0,0]

L3BDA4

简单的 parse in 文件合成密码，最后问放在产品端怎么改进。

follow up 是 The chunks look the same as before, but now rather than a single password the input contains a series of passwords, one following the other. You will know one password has ended and the next begun when you see an index repeated. Read only enough of the stream to find the first password and print it out.

读取一个文件，里面给出一个坐标，和一个 matrix，找到 matrix 里面相对应坐标的字符。主要是要熟悉文件读取，和怎么用 regex parse 字符串。虽然可以现场 google，还是要准备一下比较好，这样可以节省不少时间。这题比较简单，不用考虑文件很大的情况，直接把文件 load 进 mem。

加问一：文件含有多个像原题的坐标和 matrix，还有一个 index，中间用空行分隔。考察如何把这样多个字符取出来之后，按照 index 组成一个密码（新的字符串）。关键就是如何分隔好每个 section，之后就可以按原题处理。 最后把字符拼接一下。

加问二：文件含有多个像加问一一样的内容，如何取得第一个密码。其实只要把所有的 index 记录下来，遇到有重复的 index，那肯定是第二个密码开始了，就可以停止了。

这道题有三问 还有一个加问是第三问用最优空间复杂度

很多帖子说会被问 optimize 的方法，所以提前准备好了 read column only + rolling update。

只存每一行的那一个 character，rolling update 是这样的：比如行号是第二行，那就只需要存最新扫描的两行的 character，扫描到第三行的时候把扫描的第一行的 character 扔掉，相当于搞个 fixed sized queue

问 production code 要怎么优化，经提醒以后回答把 readblock 的 code 放在 function 里

不需要 stdin 會給 path, 直接讀就可以了

有人用 python

```
file = open("password.txt", "r")
for l in file.readlines():
# parse 每一行
```

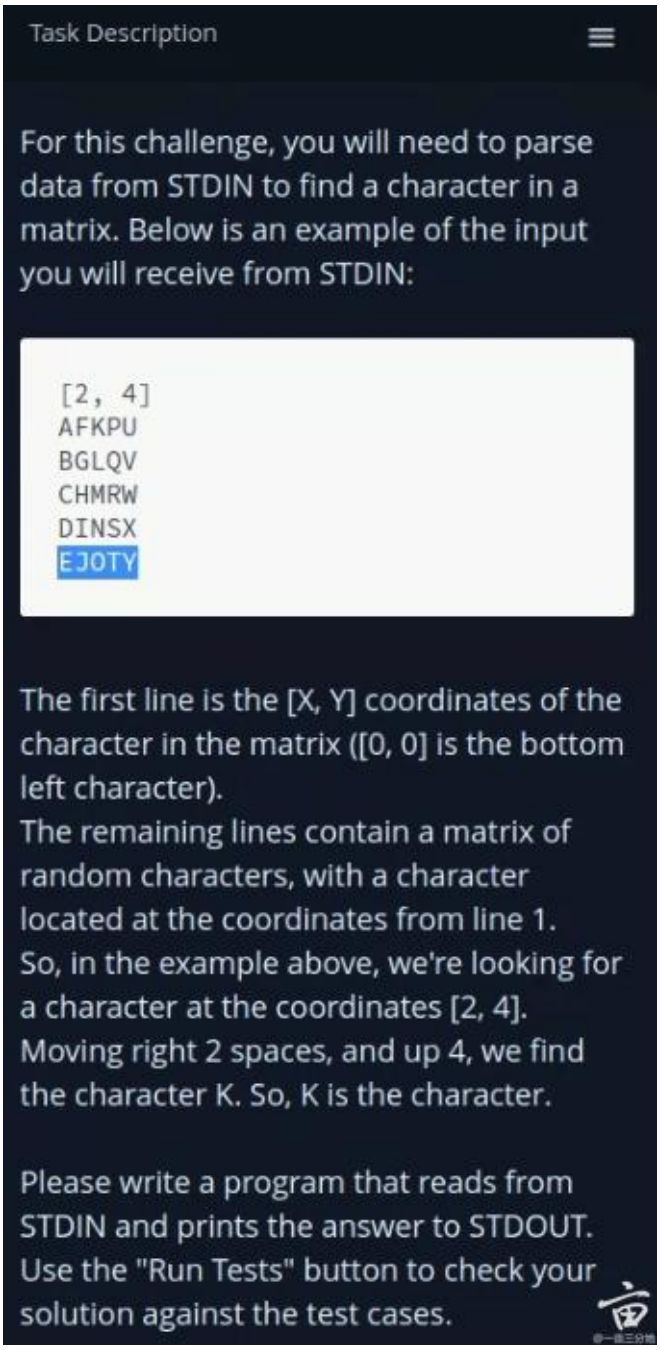
有个人说忘记了怎么处理 Stdin, 但还记得怎么用 Java scanner，就选了用 Java

```
public Character getCode(Scanner scanner) {
   String line;
   int i = 0, x = -1, y = -1;
   List<String> list = new ArrayList<>();
   while (((line = scanner.nextLine()) != null) && !line.isEmpty()) {
      if (i == 0) {
         String[] strs = line.replace("[", "")
            .replace("]", "").replace(" ", "")
```

```
                .split(",");
            y = Integer.parseInt(strs[0]);
            x = Integer.parseInt(strs[1]);
        } else {
            list.add(line);
        }
        i++;
    }
    int m = list.size(), n = list.get(0).length();
    if (x >= m || y >= n) return null;
    return list.get(m - x - 1).charAt(y);
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(new BufferedInputStream(System.in));
    ConstructPassword c = new ConstructPassword();
    System.out.println(c.getCode(scanner));
}




public String genPassword(Scanner scanner) {
    String line;
    char[] chars = new char[100];
    int consecutiveEmptyLines = 0, index = 0, x = 0, y = 0;
    List<String> list = new ArrayList<>();
```

```
    while ((line = scanner.nextLine()) != null && consecutiveEmptyLines <= 1) {
        if (line.matches("[0-9]+")) { // digits
            if (!list.isEmpty() && x < list.size() && y < list.get(0).length()) {
                chars[index] = list.get(list.size() - x - 1).charAt(y);
                list = new ArrayList<>();
            }
            consecutiveEmptyLines = 0;
            index = Integer.parseInt(line);
        } else if (line.indexOf('[') != -1) {
            String[] strs = line.replace("[", "")
                .replace("]", "").replace(" ", "")
                .split(",");
            y = Integer.parseInt(strs[0]);
            x = Integer.parseInt(strs[1]);
        } else if (!line.isEmpty()) {
            list.add(line);
        } else {    // empty
            consecutiveEmptyLines++;
        }
    }
    if (!list.isEmpty()) {
        chars[index] = list.get(list.size() - x - 1).charAt(y);
    }
    return new String(chars);
}
```

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(new BufferedInputStream(System.in));
    ConstructPassword c = new ConstructPassword();
    System.out.println(c.genPassword(scanner));
}
```



Task Description

This challenge builds off of the previous one. Please feel free to copy/paste from the previous challenge.

For this challenge, the goal is to construct a password from a series of chunks. The chunks will now look like:

```
1
[5, 6]
RXIBDIBF
DVMPXTBG
BMWAERXR
UPEIJGMW
YTALDXDH
JNPMOUEJ
XDRHCHWG

0
[0, 1]
HUTQW3
NLEVCU
```

You will notice each chunk looks similar to the previous challenge with one addition — the first line is the (0-based) index of the password character.

In our example
– First chunk: password character index 1, character at [5, 6] is `I`
– Second chunk: password character index 0, character at [0, 1] is `H`

Once you have processed all of the chunks you have the entire password and should print it to STDOUT. In our example the password is `HI`.

Chunks are separated by empty lines.

Expression

用 dfs 来解这道题

第一小问 input:["T2", ["T1 = 1", "T2 = T3", "T3 = T1"]] output: T2 值 公式都是左边一个变量， 右边是变量或者数值

第二小问 input: ["T2", ["T1 = 1", "T2 = 2 + T4", "T3 = T1 - 4", "T4 = T1 + T3]] output:T2 值

公式左边为变量，右边为一个或多个变量/数值，包括加减操作

第三小问： 在第二小问基础上可能存在解不出的情况

这道题我是用 topological sort 的方法，但后来发现 Top-down 递归就够了

然后发现所有的变量都会有一个相应的赋值

比如这种情况的 Testcase 就不存在 ["T2", ["T1=4", "T1 = 2 + T2"]]

1. 给定一个 computeExp 和 expressions。

computeExp

T3

expressions

T1=1

T2=2

T3=T4

T4=T5

T5=T2

应该返回 2

2. followup，expression 可能包含+，-，但最多只有一个+或者-。（不存在比如 T1=T2+T3+1）

3. followup，expression 可能有循环，比如

T1=1

T2=2

T3=T4+T5

T4=T5

T5=T4

这个情况 返回“IMPOSSIBLE”。

给一个公式表 [["t1 = t2 + t3"], ["t2 = t4 + t3"], ["t3 = 5"], ["t4 = 1"]]，求 t1 的值， t1 = 11(t2 = 5 + 1, t3 = 5). 可以用 print debu。 原题比较简单，公式左边都是变量，右边都是数字。

加问一，公式的左边都是变量，右边可能是数字，或者是变量的和或者差，比如 t1=t2+1, t2=1, t3=t1-t2。右边的运算只可能是和或者差。也不需要考虑需要把公式变形的情况。

加问二：可能出现解不出来的情况，如果解不出来就打印 impossible。

Expressions 这道题有人忽略了需要推导多步，比如说 T3=T2，T2=T1，T1=1，一开始我以为过两遍就能找到解，但其实需要 dfs 来找。

只需要知道题目给的 target expression，所以可以从 target expression 开始搜，不需要解出所有。

java split("+")会有解析问题，需要 escape，另外 split 需要加入空格所以是 split(" \\+ ")

java readlines library 提前准备好需要用

第一题： 给一个 target 比如 T4, 还有 T1 = 2, T2 = 3, T2 = T3, T4 = T3, 计算出 T4 = 3, 解法就是 build 一个 graph，然后 traverse graph 一直找到 leaf node;

第二题： 给一个 target 比如 T3, 还有 T1 = 2, T2 = 3, T3 = T4 + T5, T4 = T1 - T2, T5 = T4 + T1, 计算出 T3 = 0, 我的解法是 build mappings from T3 -> T4 + T5, T4 -> T1 - T2, T5 = T4 + T1, 然后用 recursion 计算。

Find Minimal Shoppers

(This question is a variation of the LeetCode question 1701. Average Waiting Time. If you haven't completed that question yet, it is recommended to solve it first.)

A shopper's logistics company is managing customer orders and needs to maintain efficient service by controlling the average wait time for order fulfillment.

You are given an unsorted array orders, where each orders[i] = [duration, arrivalTime]:

- duration is the time required to complete the ith*i*th customer's order.
- arrivalTime is the time at which the ith*i*th customer places their order.

There are multiple shoppers, and each shopper can only process one order at a time. When a new order arrives at arrivalTime, it is assigned to any available shopper who can begin the order immediately. If all shoppers are busy, the order waits until the next shopper becomes available. All orders must be fulfilled in the order they arrive.

Besides, a customer's waiting time is defined as the time between their arrivalTime and when their order is completed, and the average waiting time is defined as the sum of all customers' waiting times divided by the total number of orders.

Given a floating-point number k, return the minimum number of shoppers required so that the average waiting time for all customers does not exceed k. If it is impossible to achieve this criterion, return -1.

Constraints:

- 1 ≤ orders.length ≤ 104104
- orders[i].length = 2
- 1 ≤ duration, arrivalTime ≤ 104104
- 0 ≤ k ≤ 104104

Example 1:

Input: orders = [[4, 1], [5, 2], [2, 3]], k = 5.0 Output: 2 Explanation:

- With 1 shopper:
    - Order 1: arrives at 1, processing begins at 1, finishes at 5. Wait time = 5 - 1 = 4
    - Order 2: arrives at 2, but the shopper is occupied until 5, so processing starts at 5, finishes at 10. Wait time = 10 - 2 = 8
    - Order 3: arrives at 3, but the shopper is busy until 10, so processing starts at 10, finishes at 12. Wait time = 12 - 3 = 9
    - Average wait time = (4 + 8 + 9) / 3 = 7.0, which exceeds 5.0.
- With 2 shoppers:
    - Order 1: arrives at 1, assigned to shopper 1, so processing starts at 1, ends at 5. Wait time = 5 - 1 = 4
    - Order 2: arrives at 2, and shopper 2 is available, so processing starts at 2, ends at 7. Wait time = 7 - 2 = 5
    - Order 3: arrives at 3, both shoppers are busy (shopper 1 until 5, shopper 2 until 7). The next available shopper is ready at 5, so processing starts at 5, ends at 7. Wait time = 7 - 3 = 4
    - Average wait time = (4 + 5 + 4) / 3 ≈ 4.33, which is less than 5.0.
- Therefore, the minimum number of shoppers required is 2.

Example 2:

Input: orders = [[4, 1], [4, 2], [4, 3], [4, 4]], k = 4.3 Output: 3

Example 3:

Input: orders = [[10, 1], [10, 2], [10, 3], [1, 4]], k = 2.0 Output: -1

第一小问是排序 第二问是二分加 最小堆/优先队列

第二问用 priority queue + binary search

这道题的第一问和 leetcode 1701 Avg Wait Time 很像 这里一定要注意如果用 leetcode 1701 练习了的话别说漏嘴，题目本身跟餐饮没有任何关系。其他注意的细节有输入的格式并不会提前按照时间排序，楼主的解法是直接先排序。第一问毕竟原题，很简单

第二问是如果在第一题基础上问如果有个目标时间，几个工人可以让你的平均等待时间达标。最快解法就是用到 bs 和 heap，建议假装思考，别看到题就写，解法很短，楼主半小时就写完了，奖励了自己半小时尬聊。

补问的问题：

- 问某步骤或者某参数的意义是什么，
- 某个步骤效率是什么，
- 总体效率是什么，
- 是否能更快，
- 存储效率是多少，
- 为什么你选择用二分
- 为什么你选择用 heap
- 问时间&空间复杂度和优化
- 并且如果是 production code，如何改进

Data pivot table

第一问输入是 `List[List[str]]` 然后`List[i].length==5`(就是每行)，相当于总共五列，（细节我记不清了 面完没有趁热打铁记下来），有`date`，`region`，`sales`什么的。

然后`List[i]`就是一个“订单"

但是 tricky 的点是每个列指标不固定，就比如这个测试数据`List[i][1]` 是“sales“但是下个测试数据的 List[i][2]是“sales”

总共是三问，

第一问送分的，就是问总共表里 sales 综合

第二问是给一个`column name`，然后根据这个来 filter 一些 sales。比如说我指定“region”，然后根据 region 生成一个 table。比如“Canada”对应的原表中的数据有很多条，输出的时候应该把这些都合总了。

第三问是个排序，给一个 date，需要 filter 在此之后（包括该 date）的订单。然后表里面也会有 `cost`和`sales`，需要计算一个 profit，并且是根据第二问的基础上也会制定一个`column_name`. 这一些条件统计出来一个 profit 最大的项，再根据特定的输出格式输出（就是字符串 `f"The most xxx is xxx"`)

sum of a column

sum of a column based on country

sum of a column based on date

第一问是很简单的根据不同的 country aggregate total sale。 follow up 就是在第一问的基础上加一些附加条件和进行一个简单的计算（比如 profit）做完会问你复杂度和 production 中如何优化

第一问：返回总销售额。

第二问：返回某个 column（例如 state）的 pivot table，

第三问：利润

会让分析时间复杂度。并且如果是 production code，如何改进

需要注意的是 Column 的名字不能 hard code，具体三问都是有关某列-某列，做些比较，或者找出前五名之类，计算很简单，用到 hash。要注意审题交流来确认输出格式，最后转换 hash 到对应的格式。楼主练习的时候并没有转换格式，现场写并不难。

题目做完三问补问一些问题，比如现在的效率是多少，是否有更高的效率，每个步骤效率是多少，每个步骤是否能提前结束，比如找到了提前终止之类。做完会问你复杂度和 production 中如何优化

A retail analytics platform stores order records in a data warehouse, with each record represented as a list of strings. The first row contains column headers, while all subsequent rows are order records with each value as a string. Every record includes at least these columns: order_id, cost, sell_price, product, and date (formatted as "YYYY-MM-DD"). Additional columns (e.g., state) may also be present.

Your task is to simulate the creation of a pivot table that computes the **net profit** (sell_price minus cost) for each unique value in a specified column (pivotColumn), using

only orders **on or after** a given start date (startDate). The pivotColumn can be "state", "product", or any other column.

Implement the DataWarehouse class:

- DataWarehouse(List<List<String>> data) Initializes the data warehouse with the provided dataset. The first row is the header and the rest are data rows.
- String getMostProfitable(String pivotColumn, String startDate) Considers only records where date is **on or after** startDate. Finds the value in the pivot column with the **highest** total net profit.
    - In case of a tie, returns the **lexicographically smallest** value.
    - If no records match, returns "".

Constraints:

- The table data contains at least one row (the header).
- Each row has the same length as the header.
- All values are non-null strings.
- Dates are in "YYYY-MM-DD" format.
- 1 ≤ data.size() ≤ 105105
- 1 ≤ data[0].size() ≤ 20

Example:

Input: ["DataWarehouse", "getMostProfitable", "getMostProfitable", "getMostProfitable"]

[[["order_id", "cost", "sell_price", "product", "date", "state"], ["23", "12", "18", "cheese", "2023-12-04", "CA"], ["24", "5", "12", "melon", "2023-12-04", "OR"], ["25", "25", "31", "cheese", "2023-12-05", "OR"], ["26", "4", "12", "bread", "2023-12-05", "CA"], ["25", "10", "14", "cheese", "2023-12-06", "CA"], ["26", "5", "6", "bread", "2023-12-06", "OR"]], ["state", "2023-12-05"], ["product", "2023-12-05"], ["product", "2025-12-05"], ["color", "2023-12-01"]]

Output: [null, "CA", "cheese", ""]

Explanation:

order_id	cost	sell_price	product	date	state
23	12	18	cheese	2023-12-04	CA
24	5	12	melon	2023-12-04	OR
25	25	31	cheese	2023-12-05	OR
26	4	12	bread	2023-12-05	CA
25	10	14	cheese	2023-12-06	CA
26	5	6	bread	2023-12-06	OR

- DataWarehouse warehouse = new DataWarehouse(data);
- warehouse.getMostProfitable("state", "2023-12-05"); // Returns "CA". For all records on or after "2023-12-05", the net profit for "CA" is (12 - 4) + (14 - 10) = 8 + 4 = 12, for "OR" is (31 - 25) + (6 - 5) = 6 + 1 = 7.
- warehouse.getMostProfitable("product", "2023-12-05"); // Returns "cheese". Net profit for "cheese" is 10, "bread" is 9, and "melon" is 0.
- warehouse.getMostProfitable("product", "2025-12-05"); // Returns "", as no dates qualify.
- warehouse.getMostProfitable("color", "2023-12-01"); // Returns "", as no column qualify.

56. merge interval / 435. Non-overlapping Intervals

wildcard search

leetcode 44 的变种

Coding 1: 实现在字符串中找 target 字符串那道题，第二问是如果有 wildcard (*)怎么办

1.1 实现 string.indexOf, 不能用 indexOf (废话), 还挺简单的，2-3 分钟就写好了

1.2 pattern 里带有 `*`, 这里和 LC 的 wildcard 不太一样，LC 的是完整匹配， 这个是返回 index. 楼主在这里卡壳了，没把逻辑整理好就开始写代码，过不了 case 以后急躁了，越改越错。最后没能完成这题的所有 case. 事后楼主重新尝试了一下，思路理清以后， 十分钟就搞定了。属于不幸卡壳没能自救回来的例子。

很快做完了前两问，于是面试官掏出了之前没见过的第三问和第四问，跟之前两个题感觉关联不大，有可能是题库里面的新题。第三问是判断两个 string 是否只有 at most 1 edit distance，允许的 edit 是 replace a char or add/delete a char。第四问是判断两个 string 是否有 at most K edit distances，K 是 input。楼主做完第三问还有 15 分钟时间，说出第四问的解法（DFS+backtracking）