LeetCode 600—799

September 9, 2018

1 600 – 609

1.1 600 – Non-negative Integers without Consecutive Ones

Given a positive integer n, find the number of **non-negative** integers less than or equal to n, whose binary representations do NOT contain **consecutive ones**.

Example 1:

```
Input: 5
Output: 5
Explanation:
Here are the non-negative integers <= 5 with their corresponding binary representations:
0 : 0
1 : 1
2 : 10
3 : 11
4 : 100
5 : 101
Among them, only integer 3 disobeys the rule (two consecutive ones) and the other 5 satisfy the rule.
```

Note: 1 <= n <= 10^9

Note:

如果没有连续1出现，那么这个可能的数字中肯定不会包含0x11...，所以假设对于有n个比特的正整数来说，其范围是从$0 \to \underbrace{11\ldots1}_{n}$。符合题目要求的数字范围为两个部分

1. $10\underbrace{0\ldots0}_{k-2} \to 10\underbrace{11\ldots1}_{k-2}$，即头两位为1和0的数字
2. $0\underbrace{00\ldots0}_{k-1} \to 0\underbrace{11\ldots1}_{k-1}$，即第一位为0的数字

由于剩下的数字都会以11开头，所以都要舍弃。假设$DP(n)$为n个比特的正整数中不连续1的数字个数，根据上述分析，我们有如下递推公式

$$DP(n) = DP(n-2) + DP(n-1)$$

而$DP(1) = 2$, $DP(2) = 3$，典型的Fibnocci公式。题目中是正整数，所以比特位最大为31。

给定任何一个整数，我们先得到他的二进制表达式，然后对其中出现的每一个bit 1，计算出其右边的bit位数。以数字100举例，100的二进制为1100100，

1. 其最高位的1，其右边有6位，那么其所有的不大于100的没有连续bit 1的数字个数就等于$DP(6)$.即从

$$0\underbrace{000000}_{6} \rightarrow 0\underbrace{111111}_{6}$$

这些数字中选择。

2. 接下来第6个bit位也是1，其右边有5位，可选择的数字范围即为

$$10\underbrace{00000}_{5} \rightarrow 10\underbrace{11111}_{5}$$

即$DP(5)$

3. 接下来就没有了，为什么？因为发现已经出现了两个连续的1了。后面可选择的数字肯定都要以11开头的。所以最终结果就是$DP(5) + DP(6)$

所以在实现的时候，如果出现了两个连续1，我们直接返回已经得到的结果，如果一直没有出现，我们需要在得到的结果上加1，因为这个数本身也是符合条件的。

1.2 605 – Can Place Flowers

Suppose you have a long flowerbed in which some of the plots are planted and some are not. However, flowers cannot be planted in adjacent plots - they would compete for water and both would die.

Given a flowerbed (represented as an array containing 0 and 1, where 0 means empty and 1 means not empty), and a number **n**, return if **n** new flowers can be planted in it without violating the no-adjacent-flowers rule.

Example 1:

```
Input: flowerbed = [1,0,0,0,1], n = 1
Output: True
```

Example 2:

```
Input: flowerbed = [1,0,0,0,1], n = 2
Output: False
```

Note:

1. The input array won't violate no-adjacent-flowers rule.
2. The input array size is in the range of [1, 20000].
3. **n** is a non-negative integer which won't exceed the input array size.

Note:

- 基本思想：计算出连续0的子序列，如果这个子序列长度L大于等于3，就可以种花最多的位置个数为(L-2+1)/2，$L-2$是因为两端的0是不能种花的。因为和1相邻。
- 有两个边界情况需要特殊处理：
  1. 数组元素全部为0。在这种情况下，可以种花的最多位置为$(L+1)/2$. L是因为两端都能种花。
  2. 子序列的边界就是原数组的左边界或者是右边界。这种情况下，可以种花的最多位置为$(L-1+1)/2$. $L-1$是因为有一端是不能种花的。

1.3 606 – Construct String from Binary Tree

You need to construct a string consists of parenthesis and integers from a binary tree with the preorder traversing way.

The null node needs to be represented by empty parenthesis pair "()". And you need to omit all the empty parenthesis pairs that don't affect the one-to-one mapping relationship between the string and the original binary tree.

Example 1:

```
Input: Binary tree: [1,2,3,4]
       1
     /   \
    2     3
   /
  4

Output: "1(2(4))(3)"

Explanation: Originallay it needs to be "1(2(4())(3()())",
but you need to omit all the unnecessary empty parenthesis pairs.
And it will be "1(2(4))(3)".
```

Example 2:

```
Input: Binary tree: [1,2,3,null,4]
       1
     /   \
    2     3
     \
      4

Output: "1(2()(4))(3)"

Explanation: Almost the same as the first example,
except we can't omit the first parenthesis pair to break the one-to-one mapping relationship between the input and th
```

Note:

- 如果当前node的left child为null, 而right child不是null，这时，left child所生成的()是不能忽略的。
- 否则的话，在preorder中，先访问当前node，然后加上一个open parenthesis(, 然后在当前node的left继续preorder，结束后，加上close parenthesis)。
- 接着如果当前node的right child不为null，则类似，先加上一个open parenthesis(, 然后在当前node的right继续preorder，结束后，加上close parenthesis)。

1.4 609 – Find Duplicate File in System

Given a list of directory info including directory path, and all the files with contents in this directory, you need to find out all the groups of duplicate files in the file system in terms of their paths.

A group of duplicate files consists of at least **two** files that have exactly the same content.

A single directory info string in the **input** list has the following format:

`"root/d1/d2/.../dm f1.txt(f1_content) f2.txt(f2_content) ... fn.txt(fn_content)"`

It means there are **n** files (`f1.txt` , `f2.txt` ... `fn.txt` with content `f1_content` , `f2_content` ... `fn_content` , respectively) in directory `root/d1/d2/.../dm` . Note that n >= 1 and m >= 0. If m = 0, it means the directory is just the root directory.

The **output** is a list of group of duplicate file paths. For each group, it contains all the file paths of the files that have the same content. A file path is a string that has the following format:

`"directory_path/file_name.txt"`

Example 1:

```
Input:
["root/a 1.txt(abcd) 2.txt(efgh)", "root/c 3.txt(abcd)", "root/c/d 4.txt(efgh)", "root 4.txt(efgh)"]
Output:
[["root/a/2.txt","root/c/d/4.txt","root/4.txt"],["root/a/1.txt","root/c/3.txt"]]
```

Note:

1. No order is required for the final output.
2. You may assume the directory name, file name and file content only has letters and digits, and the length of file content is in the range of [1,50].
3. The number of files given is in the range of [1,20000].
4. You may assume no files or directories share the same name in the same directory.
5. You may assume each given directory info represents a unique directory. Directory path and file info are separated by a single blank space.

Note:

- 用一个二维vector（vector(vector(string)))）来保存同一个content所对应的path列表
- 建立一个hashmap，key为content，value为所得到的path所在的数组在二维数组中的位置。把相同内容的path方瑞对应的数组中。
- 最后把只有一个元素的vector(string)去除掉。

1. **Imagine you are given a real file system, how will you search files? DFS or BFS?**

   In general, BFS will use more memory then DFS. However BFS can take advantage of the locality of files in inside directories, and therefore will probably be faster

2. **If the file content is very large (GB level), how will you modify your solution?**

   In a real life solution we will not hash the entire file content, since it's not practical. Instead we will first map all the files according to size. Files with different sizes are guaranteed to be different. We will than hash a small part of the files with equal sizes (using MD5 for example). Only if the md5 is the same, we will compare the files byte by byte.

3. **If you can only read the file by 1kb each time, how will you modify your solution?**

   This will not change the solution. We can create the hash from the 1kb chunks, and then read the entire file if a full byte by byte comparison is required.

4. **What is the time complexity of your modified solution? What is the most time-consuming part and memory consuming part of it? How to optimize?**

   Time complexity is $O(n^2 * k)$ since in worse case we might need to compare every file to all others. k is the file size

5. **How to make sure the duplicated files you find are not false positive?**

   We will use several filters to compare: File size, Hash and byte by byte comparisons.

2 610 – 619

2.1 611 – Valid Triangle Number

Given an array consists of non-negative integers, your task is to count the number of triplets chosen from the array that can make triangles if we take them as side lengths of a triangle.

Example 1:

```
Input: [2,2,3,4]
Output: 3
Explanation:
Valid combinations are:
2,3,4 (using the first 2)
2,3,4 (using the second 2)
2,2,3
```

Note:

1. The length of the given array won't exceed 1000.
2. The integers in the given array are in the range of [0, 1000].

Note:

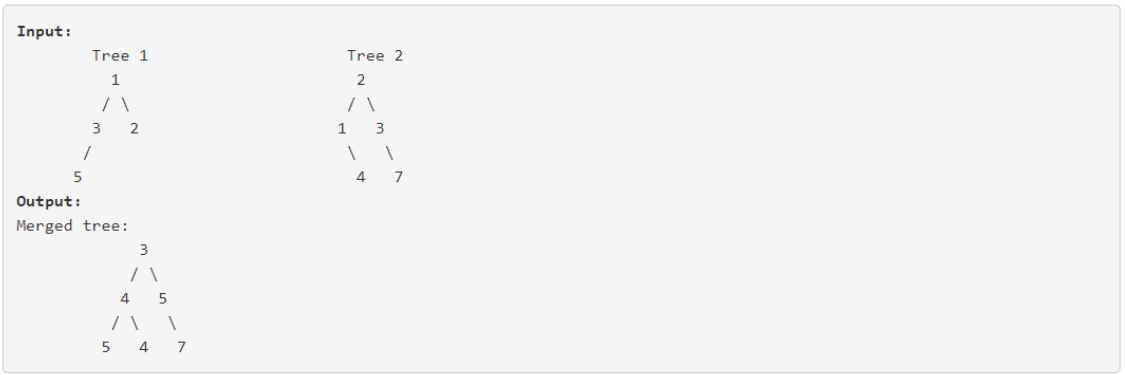
- 首先需要对数组A进行排序。
- 从倒数第二个数即$i = \text{LEN} - 2$ 开始往前遍历，假设$L = 0$, $R = i - 1$。最外侧循环我们把i从$L - 1$一直循环到2. 因为三角形需要三个边，所以至少需要三个数组元素。
- 接着进入到内部循环寻找两个边大于$A[i]$的个数。
- 如果$A[L] + A[R] > A[i]$，那么数组A中，从L到$R - 1$，和$A[R]$相加都会大于$A[i]$, 因为$A[L]$是最小的。所以符合要求的个数为$(R - 1) - L + 1 = R - L$。然后R减1。
- 如果$A[L] + A[R] \leq A[i]$，那么$A[L]$还是有点小，需要增大$A[L]$, 所以L加1。
- 如果$L \geq R$，那么就结束本次寻找，继续外部循环

2.2 617 – Merge Two Binary Trees

Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree.

Example 1:



Note: The merging process must start from the root nodes of both trees.

Note:

Use recursive method. We apply the function at t_1's left with t_2's left, and t_1's right with t_2's right. There are 3 boundaries to be taken care.

- t_1 is empty and t_2 is empty, we just return empty.
- t_1 is not empty but t_2 is empty, return t_1.
- t_1 is empty but t_2 is not, return t_2.
- In other cases, we first replace t_1's value by the sum of values of t_1 and t_2. Then we just let t_1's left equal to the merge of t_1's left and t_2's left, and t_1's right equal to the merge of t_1's right and t_2's right. Return t_1.

3 620 – 629

3.1 621 – Task Scheduler

Given a char array representing tasks CPU need to do. It contains capital letters A to Z where different letters represent different tasks.Tasks could be done without original order. Each task could be done in one interval. For each interval, CPU could finish one task or just be idle.

However, there is a non-negative cooling interval **n** that means between two **same tasks**, there must be at least n intervals that CPU are doing different tasks or just be idle.

You need to return the **least** number of intervals the CPU will take to finish all the given tasks.

Example 1:

```
Input: tasks = ["A","A","A","B","B","B"], n = 2
Output: 8
Explanation: A -> B -> idle -> A -> B -> idle -> A -> B.
```

Note:

1. The number of tasks is in the range [1, 10000].
2. The integer n is in the range [0, 100].

Note:

有两种方法，一个是基于可产生的slot分析，另一个是用程序的方式，不管哪种方法，其本质都是基于greedy search。第一种方法如下：

- 假设A是所有任务中出现频率最高的，那么我们应当基于A来进行任务安排。例如，假设任务列表为3A，2B和1C，而$n = 2$。那么我们先把安排A，然后在两个A中间插入B和C, 就得到$ABC|AB\star|A$.
- 所以denote出现最高频率的task的总数为K，那么这n个task点之间就会有$K-1$个$slots$，每个slot之间的task和可能的idle的总数为$n-1$.

$$A\underbrace{\ldots}_{n}A\ldots A\underbrace{\ldots}_{n}A\ldots A$$

- 这时候会发现，如果task种类T小于n, idle的个数就为$n-T$.因为我们可以在每一个slot把每个种类的一个task放进去，然后再放idle。所以最终需要的CPU cycyles为

$$(K-1)\times n$$

- 如果task种类T大于n，这时候就不需要idle了，只需要把所有种类的task按照一个个放入slot中，这样由于slot中所有种类的task数量已经超越n了，自然也就符合题目的要求。

- 所以无论是那种情况，我们所能得到的最短cycle数都是

$$(K-1) \times n$$

但是这只是最高频率的task只有一种的情况下。

- 如果具有最高频率的task有多个，我们可以把这些task一个个串起来，比如，$3A$, $3B$, $2C$, $1D$, $n = 3$，我们可以按照如下安排

$$ABCD|ABC\star|AB\star|AB$$

把AB当作一个task

$$(AB)CD|(AB)C\star|(AB)\star|(AB)$$

这与只有A是最高频率的task类似，只不过可用的slot从3变成了2，因为多了一个B.

- 由于其他非最大数量的task有可能在最后一个gap之前就用完了，所以需要通过计算总的slot个数以及可用于task的slot总的个数进行分析，而不是单独对一个gap进行分析。首先要得到在gap中间需要需要放入多少个slots，这个slots可能包含task和idle，取决于task的数量，所以需要计算总数，而不是单独的gap中的task。例如$4A$，$4B$，$2C$，$n = 3$，那么根据分析有

$$ABC\star|ABC\star|AB\star\star|AB$$

在最后一个gap，task C就没有了，已经用完了，需要填入idle。但是总的slots仍然是

$$\texttt{gaps} \times \texttt{n - (number of maximum tasks - 1)} = 3 \times (3 - (2 - 1)) = 6$$

这里需要把AB当作一个整体，因为他们具有相同数量而且都是最大数量的task。

1. $R = \texttt{number of unique tasks with maximum count}$
2. $G = \texttt{total gaps}$
3. $M = \texttt{count of the task which appear most frequently}$
4. $G = M - 1$
5. $S = \texttt{total count of tasks which do not appear most often}$
6. $T = \texttt{total count of tasks}$
7. $S = T - R \times M$
8. $L = \texttt{total slots need to inserted in all gaps}$

9. $L = G \times (n - (R - 1))$
10. $I =$ `total idles needed`
11. $I = \max(0, L - S)$
12. $N =$ `minimum total cycles`
13. $N = T + I$

$$\underbrace{AB}_{R}\overbrace{\underbrace{CDE\ldots}_{S_1}\underbrace{\star\star\ldots}_{I_1}}^{n}AB\ldots AB\overbrace{\underbrace{CDE\ldots}_{S_G}\underbrace{\star\star\ldots}_{I_G}}^{n}AB$$

在这里，$\sum_{i=1}^{G} S_i = S$, $\sum_{i=1}^{G} I_i = I$以及$S + I = L$

3.2 622 – Design Circular Queue

Design your implementation of the circular queue. The circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

One of the benefits of the circular queue is that we can make use of the spaces in front of the queue. In a normal queue, once the queue becomes full, we cannot insert the next element even if there is a space in front of the queue. But using the circular queue, we can use the space to store new values.

Your implementation should support following operations:

- `MyCircularQueue(k)` : Constructor, set the size of the queue to be k.
- `Front` : Get the front item from the queue. If the queue is empty, return -1.
- `Rear` : Get the last item from the queue. If the queue is empty, return -1.
- `enQueue(value)` : Insert an element into the circular queue. Return true if the operation is successful.
- `deQueue()` : Delete an element from the circular queue. Return true if the operation is successful.
- `isEmpty()` : Checks whether the circular queue is empty or not.
- `isFull()` : Checks whether the circular queue is full or not.

Example:

```
MyCircularQueue circularQueue = new MycircularQueue(3); // set the size to be 3
circularQueue.enQueue(1);  // return true
circularQueue.enQueue(2);  // return true
circularQueue.enQueue(3);  // return true
circularQueue.enQueue(4);  // return false, the queue is full
circularQueue.Rear();  // return 3
circularQueue.isFull();  // return true
circularQueue.deQueue();  // return true
circularQueue.enQueue(4);  // return true
circularQueue.Rear();  // return 4
```

Note:

- Requires the following members
  1. `len` — The current number of elements in the queue.
  2. `head` — The head index of the queue. When dequeue one element,
     $$\texttt{head} = (\texttt{head + 1}) \% \texttt{len}$$
  3. `cap` — The maximum capacity of the queue
  4. `tail` — The tail index of the queue. When insert an element
     $$\texttt{tail} = (\texttt{tail + 1}) \% \texttt{len}$$
- When decide if the queue is empty or full, simply check if $\texttt{len} = 0$ or $\texttt{len} = \texttt{cap}$

3.3 623 – Add One Row to Tree

Given the root of a binary tree, then value `v` and depth `d`, you need to add a row of nodes with value `v` at the given depth `d`. The root node is at depth 1.

The adding rule is: given a positive integer depth `d`, for each NOT null tree nodes `N` in depth `d-1`, create two tree nodes with value `v` as `N's` left subtree root and right subtree root. And `N's` **original left subtree** should be the left subtree of the new left subtree root, its **original right subtree** should be the right subtree of the new right subtree root. If depth `d` is 1 that means there is no depth d-1 at all, then create a tree node with value **v** as the new root of the whole original tree, and the original tree is the new root's left subtree.

Example 1:

```
Input:
A binary tree as following:
       4
     /   \
    2     6
   / \   /
  3   1 5

v = 1

d = 2

Output:
       4
      / \
     1   1
    /     \
   2       6
  / \     /
 3   1   5
```

```
Input:
A binary tree as following:
      4
     /
    2
   / \
  3   1

v = 1

d = 3

Output:
      4
     /
    2
   / \
  1   1
 /     \
3       1
```

Note:

- This is a easy problem. Using recursive method, provide a recursive function with `node`, d, v and `cur_level` as the input parameters
- if `cur_level` $= d - 1$, then for each non null node:
  1. save its left and right subtree first, and then create left and right subtree with v.
  2. assign the saved left subtree to `left child` of the new created left subtree and the saved right substree to `right child` of the new create right subtree.
- otherwise, recursive call the function on current node's left subtree if it is not null with `cur_level` $+ 1$. Same for the current node's right subtree.
- if $d = 1$, create a new root node as require and assign `root` as the `left child` of the new root node.

3.4 624 – Maximum Distance in Arrays

TODO

3.5 625 – Minimum Factorization

TODO

3.6 628 – Maximum Product of Three Numbers

Given an integer array, find three numbers whose product is maximum and output the maximum product.

Example 1:

```
Input: [1,2,3]
Output: 6
```

Example 2:

```
Input: [1,2,3,4]
Output: 24
```

Note:

1. The length of the given array will be in range [3,10^4] and all elements are in the range [-1000, 1000].
2. Multiplication of any three numbers in the input won't exceed the range of 32-bit signed integer.

Note:

- 由于可能有负数，所以需要找到三个最大的整数，和两个最小的负数，然后比较两个负数和最大整数的product，与三个整数的product的大小。

3.7 629 – K Inverse Pairs Array

Given two integers `n` and `k`, find how many different arrays consist of numbers from `1` to `n` such that there are exactly `k` inverse pairs.

We define an inverse pair as following: For `i_th` and `j_th` element in the array, if `i` < `j` and `a[i]` > `a[j]` then it's an inverse pair; Otherwise, it's not.

Since the answer may be very large, the answer should be modulo $10^9 + 7$.

Example 1:

```
Input: n = 3, k = 0
Output: 1
Explanation:
Only the array [1,2,3] which consists of numbers from 1 to 3 has exactly 0 inverse pair.
```

Example 2:

```
Input: n = 3, k = 1
Output: 2
Explanation:
The array [1,3,2] and [2,1,3] have exactly 1 inverse pair.
```

Note:

1. The integer `n` is in the range [1, 1000] and `k` is in the range [0, 1000].

Note:

- 采用DP方式，找到recursive equation $\texttt{dp}(n, k)$，即从$1 \to n$，有k个`inverse pairs`的数列个数。
- 为了找到recursive equation，分析从$n-1$变换到n时，`how many more inverse pairs are added`.
  1. 如果新增的n在位置n,那么由于$1 \to n$中没有数能比n大，所以，在这个位置放n，不会产生新的`inverse pair`. 所以这时候的有k个inverse pairs的array个数仍然是$\texttt{dp}(n-1, k)$
  2. 如果n放在位置$n-1$，由于位置n必然被$1 \to n-1$中的任何一个数所占据，所以把n放在位置$n-1$上的数组中的任何一个都会由于n而产生一个`inverse pair`，那么另外$k-1$个`inverse pair`就要靠安排剩下的$1 \to n-1$产生了。所以这时候有k个`inverse pair`的array个数则是$\texttt{dp}(n-1, k-1)$;
  3. ...
  4. 如果n放在位置1，由于位置$2 \to n$必然被$1 \to n-1$的所有数所占据，所以把n放在位置1上的数组中的任何一个都会由于n而产生$n-1$个`inverse pair`，那么剩下的$k-$

$(n-1)$个`inverse pair`就要靠安排剩下的$1 \to n-1$产生了。所以这时候有k个`inverse pair`的array个数则是$\texttt{dp}(n-1, k-(n-1))$。（注意：这里没有考虑k和n的大小关系，k可能小于或者远小于n-1，所以有的k实际上不会recursive到这里的，在实现的时候需要考虑）。

5. 最后总结得到的recursive equation：

$$\texttt{dp}(n,k) = \texttt{dp}(n-1,k) + \texttt{dp}(n-1,k-1) + \ldots + \texttt{dp}(n-1,k-(n-1))$$

- 如果直接用上述recursive equation, 仍然效率不是很高。通过观察equation中$dp(n,k-1)$，由于

$$\texttt{dp}(n,k-1) = \texttt{dp}(n-1,k-1) + \texttt{dp}(n-1,k-2) + \ldots + \texttt{dp}(n-1,k-1-(n-2)) + \texttt{dp}(n-1,k-1-(n-1))$$

于是结合两个equation，可以得到如下equation

$$\begin{aligned}
\texttt{dp}(n,k) &= \texttt{dp}(n-1,k) + \underbrace{\texttt{dp}(n-1,k-1) + \ldots + \texttt{dp}(n-1,k-(n-1))}_{\texttt{could be part of dp}(n,k-1)} \\
&= \texttt{dp}(n-1,k) + \underbrace{\texttt{dp}(n-1,k-1) + \ldots + \texttt{dp}(n-1,k-(n-1)) + \texttt{dp}(n-1,k-n)}_{\texttt{dp}(n,k-1)} \\
&\qquad - \texttt{dp}(n-1,k-n) \\
&= \texttt{dp}(n-1,k) + \texttt{dp}(n,k-1) - \texttt{dp}(n-1,k-n)
\end{aligned} \tag{1}$$

- 在实现的时候，`dp`的`size`为$(n+1)\times(k+1)$，$k=0$时，都为1，然后初始以$n=3$开始，因为$\texttt{dp}(2,1)=1$。$\texttt{dp}(n-1,k-n)$只有当$k \geq n$的时候才会被减掉。

4 630 — 639

4.1 630 – Course Schedule III

There are `n` different online courses numbered from `1` to `n`. Each course has some duration(course length) `t` and closed on `d_th` day. A course should be taken **continuously** for `t` days and must be finished before or on the `d_th` day. You will start at the `1_st` day.

Given `n` online courses represented by pairs `(t,d)`, your task is to find the maximal number of courses that can be taken.

Example:

```
Input: [[100, 200], [200, 1300], [1000, 1250], [2000, 3200]]
Output: 3
Explanation:
There're totally 4 courses, but you can take 3 courses at most:
First, take the 1st course, it costs 100 days so you will finish it on the 100th day, and ready to take the next cour
Second, take the 3rd course, it costs 1000 days so you will finish it on the 1100th day, and ready to take the next c
Third, take the 2nd course, it costs 200 days so you will finish it on the 1300th day.
The 4th course cannot be taken now, since you will finish it on the 3300th day, which exceeds the closed date.
```

Note:

1. The integer 1 <= d, t, n <= 10,000.
2. You can't take two courses simultaneously.

- 基于Greedy Method, 首先按照close day有低到高排序。
- Next，用一个max priority queue里面的node是各个course的duration。然后set 时间总和T为0.
- 从第一个course开始，把当前course的duration放入queue中。如果该course的duration加上到目前为止的T小于course的close time，则继续到下一个course。
- 如果超过了close time，这时候我们需要从queue中remove一个course，那么哪个course是最佳候选呢，很显然，duration越大的越应该移除（Greedy）。而queue的top刚好就是这个需要移除的durtaion。所以把这个duration从T中移除。
- 最后，queue中的node个数就是需要的答案了。

4.2 631 – Design Excel Sum Formula

TODO

4.3 632 – Smallest Range

You have `k` lists of sorted integers in ascending order. Find the **smallest** range that includes at least one number from each of the `k` lists.

We define the range [a,b] is smaller than range [c,d] if `b-a < d-c` or `a < c` if `b-a == d-c`.

Example 1:

```
Input:[[4,10,15,24,26], [0,9,12,20], [5,18,22,30]]
Output: [20,24]
Explanation:
List 1: [4, 10, 15, 24,26], 24 is in range [20,24].
List 2: [0, 9, 12, 20], 20 is in range [20,24].
List 3: [5, 18, 22, 30], 22 is in range [20,24].
```

Note:

1. The given list may contain duplicates, so ascending order means >= here.
2. 1 <= `k` <= 3500
3. -10^5 <= `value of elements` <= 10^5.

Note:

- Basic idea: 用一个min priority_queue，其内部节点结构为`pair<size_t, size_t>`，其中`pair.first`代表当前数字所在数组在输入的二维数组中的`index`，`pair.second`则代表当前输在在其所在数组中的`index`。通过使用这种方法，把**queue**中的node和输入的二维数组联系起来，而不再是单纯的数字了。另外一种方法使用**iterator**，这里采用第一种方法。
- 首先把各个数组的第一个数字放入**queue**中，这样在**queue**中我们都有一个来自各数组的数字。同时需要获得这些数字的最大值H和最小值L，以此作为最终`range`的初始上限和下限
- 接着开始用**queue**是否为**empty**作为循环退出的条件。在循环中，从**queue**中弹出并得到**top**元素，该元素按照前面的设计，是一个**pair**，我们将其**second**加1，也就是当前这个数字M所在数组的下一个数字在当前这个数字所在数组的`index`。如果这个$index$已经和当前这个数字所在数组的长度相等了，表示已经扫描完了这个数组了，这个时候也需要退出循环。因为题目要求是`range`中包含每个数组中的至少一个数字。如果不退出，可能以后更新的`range`就不会包含当前这个数字所在数组的数字了（因为已经扫描完了）。

- 如果还在当前数字M所在数组的范围内，我们将这个数字N也就是当前数字M所在数组中的下一个数字的坐标放入**queue**中。这是因为我们从原来的**queue**中**remove**掉当前的这个数字，也就是说再放入这个数字N之前**queue**中没有了当前数字所在数组的数字了。这时候，需要更新当前`range`的下限为**queue**的top元素。因为这时候top元素可能由于新加入的最新数字而发生了改变。这时候有两种情况

  1. 加入的数字N是最小的，那么当前`range`需要更新为N。
  2. 如果不是，那么当前`range`需要更新为去除top元素之后的最小元素。

  无论发生哪种情况，当前的`range`都需要做更新。同时把R更新为R和N的最大值。这样更新后的`range`同样包含了每个数组中的至少一个数字。只不过是把M替换为了N.

- 之所以上限R需要比较，而下限L不需要比较，是因为下限L已经通过priority_queue的操作已经是最小的了。

- 最后比较R与L是否是当前最小的*range*。接着继续循环。

4.4 634 – Find the Derangement of An Array

TODO

4.5 635 – Design Log Storage System

TODO

4.6 636 – Exclusive Time of Functions

Given the running logs of **n** functions that are executed in a nonpreemptive single threaded CPU, find the exclusive time of these functions.

Each function has a unique id, start from **0** to **n-1**. A function may be called recursively or by another function.

A log is a string has this format : `function_id:start_or_end:timestamp`. For example, `"0:start:0"` means function 0 starts from the very beginning of time 0. `"0:end:0"` means function 0 ends to the very end of time 0.

Exclusive time of a function is defined as the time spent within this function, the time spent by calling other functions should not be considered as this function's exclusive time. You should return the exclusive time of each function sorted by their function id.

Example 1:

```
Input:
n = 2
logs =
["0:start:0",
 "1:start:2",
 "1:end:5",
 "0:end:6"]
Output:[3, 4]
Explanation:
Function 0 starts at time 0, then it executes 2 units of time and reaches the end of time 1.
Now function 0 calls function 1, function 1 starts at time 2, executes 4 units of time and end at time 5.
Function 0 is running again at time 6, and also end at the time 6, thus executes 1 unit of time.
So function 0 totally execute 2 + 1 = 3 units of time, and function 1 totally execute 4 units of time.
```

Note:

1. Input logs will be sorted by timestamp, NOT log id.
2. Your output should be sorted by function id, which means the 0th element of your output corresponds to the exclusive time of function 0.
3. Two functions won't start or end at the same time.
4. Functions could be called recursively, and will always end.
5. 1 <= n <= 100

Note:

- Since each task will reduce the contained tasks' running time, use a **stack** to record the latest `id`.

- At the beginning, parse the **first** log. Push the `id` into the stack. Also, save the time as *prev*.
- Start the loop from the second log.
  - if the log is **start**,
    1. First, add the running time of the `id` which is the top of the stack with the difference of the current time and *prev*.
    2. Next, push current `id` into the stack and update *prev* as the current time.
  - otherwise, this log contains the information of task **ending**.
    1. First, we add the running time of the `id` which is the top of the stack with the difference of the current time and *prev*, also need to add (**1**) since the end of current time needs to be considered.
    2. Next, pop the top `id` from the stack. Also, update *prev* as the current time plus (**1**) since this is the end of the current time.

4.7 637 – Average of Levels in Binary Tree

Given a non-empty binary tree, return the average value of the nodes on each level in the form of an array.

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7
Output: [3, 14.5, 11]
Explanation:
The average value of nodes on level 0 is 3,  on level 1 is 14.5, and on level 2 is 11. Hence return [3, 14.5, 11].
```

Note:

1. The range of node's value is in the range of 32-bit signed integer.

Note:

- An easy problem: use a queue and for each level, get the current size of the queue and pop each node from the queue.
- Need to take consideration of possible overflow. Therefore, each level's sum will be `double` type.

4.8 638 – Shopping Offers

In LeetCode Store, there are some kinds of items to sell. Each item has a price.

However, there are some special offers, and a special offer consists of one or more different kinds of items with a sale price.

You are given the each item's price, a set of special offers, and the number we need to buy for each item. The job is to output the lowest price you have to pay for **exactly** certain items as given, where you could make optimal use of the special offers.

Each special offer is represented in the form of an array, the last number represents the price you need to pay for this special offer, other numbers represents how many specific items you could get if you buy this offer.

You could use any of special offers as many times as you want.

Example 1:

```
Input: [2,5], [[3,0,5],[1,2,10]], [3,2]
Output: 14
Explanation:
There are two kinds of items, A and B. Their prices are $2 and $5 respectively.
In special offer 1, you can pay $5 for 3A and 0B
In special offer 2, you can pay $10 for 1A and 2B.
You need to buy 3A and 2B, so you may pay $10 for 1A and 2B (special offer #2), and $4 for 2A.
```

Example 2:

```
Input: [2,3,4], [[1,1,0,4],[2,2,1,9]], [1,2,1]
Output: 11
Explanation:
The price of A is $2, and $3 for B, $4 for C.
You may pay $4 for 1A and 1B, and $9 for 2A ,2B and 1C.
You need to buy 1A ,2B and 1C, so you may pay $4 for 1A and 1B (special offer #1), and $3 for 1B, $4 for 1C.
You cannot add more items, though only $9 for 2A ,2B and 1C.
```

Note:

1. There are at most 6 kinds of items, 100 special offers.
2. For each item, you need to buy at most 6 of them.
3. You are **not** allowed to buy more items than you want, even if that would lower the overall price.

Note:

- 一个special offer是否被接受，取决于当前所需的quantity是否大于等于对应的**special offer**所提供的quantity。
- 基于recursive原理，不难写出function的原型find_offer(price, specials, current_needs, start, total_cost, minimum_cost)。其中`current_needs`代表recursive到current level时，还有多少需求剩余。`start`代表的是从哪个`index`开始测试**special offer**。`total_cost`代表的是recursive到current level时，已经产生了多少cost，最后的`min_cost`则就是需要返回的最小的cost。
- 在find_offer中，从当前start开始测试每个offer，一直到special offer的最后，首先将当前的`current_needs`保存起来，然后测试当前special offer是否可以接受，如果不可以，需要重

置current_needs为保存的value，继续循环。如果被接受，则开始recursive，这时start为当前的可接受的special offer的**index**，因为题目说同一个offer可以用多次，然后total_cost更新为与当前offer的cost的sum。

- 接着，仍然需要重置current_needs为保存的value，因为需要从下一个offer重新开始测试。
- 最后，循环结束后，由于current_needs可能有剩余，需要在total_cost加上用实际的price得到的**cost**。并比较最小的**cost**。

4.9 639 – Decode Ways II

A message containing letters from `A-Z` is being encoded to numbers using the following mapping way:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Beyond that, now the encoded string can also contain the character '*', which can be treated as one of the numbers from 1 to 9.

Given the encoded message containing digits and the character '*', return the total number of ways to decode it.

Also, since the answer may be very large, you should return the output mod $10^9 + 7$.

Example 1:

```
Input: "*"
Output: 9
Explanation: The encoded message can be decoded to the string: "A", "B", "C", "D", "E", "F", "G", "H", "I".
```

Example 2:

```
Input: "1*"
Output: 9 + 9 = 18
```

Note:

1. The length of the input string will fit in range [1, 10^5].
2. The input string will only contain the character '*' and digits '0' - '9'.

Note:

- 基于**Dynamic Programming**，假设当前字符`index`为i。
- 如果$i = 0$，如果当前字符为$\star$，则有**9 ways**，否则为**1 way**。
- 从第二个字符开始循环，分为三种情形。用n_0代表decode $S[0 \dots i-2]$的ways，n_1代表decode $S[0 \dots i-1]$的ways，n_2代表decode $S[0 \dots i]$的ways

  1. 如果为$\star$，如果只decode这个字符，那么有9 ways，那么这时候产生的$n_2 = 9 \times n_1$。但如果需要把当前这个字符和前一个字符结合起来一起decode，则有以下几种情形。

     (a) 如果前一个字符是2，那么因为最多只能decode 从$21 \to 26$，所以总共有6 ways。所以$n2 = n_2 + n_0 \times 6$。

     (b) 如果前一个字符是1,那么可以decode 从$11 \to 19$，有9 ways，所以$n_2 = n_2 + n_0 \times 9$。

     (c) 如果前一个字符也是$\star$，那么可以decode 从$11 \to 19$ 和$21 \to 26$，总共$6 + 9 = 15$。注意不会产生20，因为$\star$不能decode为zero。所以$n_2 = n_2 + n_0 \times 15$

(d) 如果是剩下的其他字符，都不可能decode，所以为n_2不变。

2. 如果为0，那么因为0本身是不能被decode的，所以必须结合前一个字符一起进行decode。类似的，也有如下情形。

   (a) 前一个字符为2或者为1，只能decode 20或者10，都只有one way，所以$n_2 = n_0 \times 1$。

   (b) 前一个字符为$\star$，那么$\star$能够给出的也只有1和2，所以有2 ways，于是$n_2 = n_0 \times 2$。

   (c) 在其他情形下，都不可能decode，这时候$n_2 = n_0 \times 0 = 0$。

3. 如果为其他字符，那么decode这个字符本身只有one way，所以$n_2 = n_1 \times 1$。然后把当前这个字符和前一个字符结合起来一起decode，则一样会有以下几种情形。

   (a) 前一个字符为1，可以decode，但只有one way，所以$n_2 = n_2 + n_0 \times 1$。

   (b) 前一个字符为2，这时候要看当前字符了，如果当前字符是$1 \to 6$，那么可以decode，也是只有one way。于是$n_2 = n_2 + n_0 \times 1$。

   (c) 如果前一个字符是$\star$，同样也要看当前字符，如果当前字符也是$1 \to 6$，那么$\star$可以是1或者2，总共是2 ways，所以这时候$n_2 = n_2 + n_0 \times 2$。但是，如果当前字符是$7 \to 9$，$\star$只能是1，总共有1 way，这时候$n_2 = n_2 + n_0 \times 1$。

- 继续到下一个字符之前，$n_0 = n_1$，$n_1 = n_2$。因为只需要考虑当前的三个计数值。
- 最后返回n_2
- 实现的时候，最开始的$n_0 = 1$而不是0，另外需要用`long long type`

5 640 — 649

5.1 640 – Solve the Equation

Solve a given equation and return the value of `x` in the form of string "x=#value". The equation contains only '+', '-' operation, the variable `x` and its coefficient.

If there is no solution for the equation, return "No solution".

If there are infinite solutions for the equation, return "Infinite solutions".

If there is exactly one solution for the equation, we ensure that the value of `x` is an integer.

Example 1:

```
Input: "x+5-3+x=6+x-2"
Output: "x=2"
```

Example 2:

```
Input: "x=x"
Output: "Infinite solutions"
```

Example 3:

```
Input: "2x=x"
Output: "x=0"
```

Example 4:

```
Input: "2x+3x-6x=x+2"
Output: "x=-1"
```

Example 5:

```
Input: "x=x+2"
Output: "No solution"
```

Note:

5.2 641 – Design Circular Deque

Design your implementation of the circular double-ended queue (deque).

Your implementation should support following operations:

- `MyCircularDeque(k)` : Constructor, set the size of the deque to be k.
- `insertFront()` : Adds an item at the front of Deque. Return true if the operation is successful.
- `insertLast()` : Adds an item at the rear of Deque. Return true if the operation is successful.
- `deleteFront()` : Deletes an item from the front of Deque. Return true if the operation is successful.
- `deleteLast()` : Deletes an item from the rear of Deque. Return true if the operation is successful.
- `getFront()` : Gets the front item from the Deque. If the deque is empty, return -1.
- `getRear()` : Gets the last item from Deque. If the deque is empty, return -1.
- `isEmpty()` : Checks whether Deque is empty or not.
- `isFull()` : Checks whether Deque is full or not.

Example:

```
MyCircularDeque circularDeque = new MycircularDeque(3); // set the size to be 3
circularDeque.insertLast(1);                    // return true
circularDeque.insertLast(2);                    // return true
circularDeque.insertFront(3);                   // return true
circularDeque.insertFront(4);                   // return false, the queue is full
circularDeque.getRear();                        // return 32
circularDeque.isFull();                         // return true
circularDeque.deleteLast();                     // return true
circularDeque.insertFront(4);                   // return true
circularDeque.getFront();                       // return 4
```

Note:

Algorithm 1 Constructor

1: **procedure** CIRCULARDEQUEUE(k)
2: $cap := k$
3: $size := 0$
4: $v[k]$ ▷ typically a vector with size equal to k
5: $front := 0$ ▷ `index` to the front of the deque
6: $rear := k - 1$ ▷ `index` to the rear of the deque
7: **end procedure**

Algorithm 2 Insert Element into the front

```
procedure InsertFront(value)
    if 8 then
        return false                                  ▷ cannot insert when full
    end if
    size := size + 1                                  ▷ number of elements increment
    v[front] ← value                                  ▷ put value in front index.
    front := (front + 1) mod cap                      ▷ increment front index
                                                      ▷ need to count for the circular property of the queue
    return true
end procedure
```

Algorithm 3 Insert Element into the rear

```
procedure InsertLast(value)
    if 8 then
        return false                                  ▷ cannot insert when full
    end if
    size := (size + 1)                                ▷ number of elements increment
    v[rear] ← value                                   ▷ put value in front index.
    rear := (rear − 1 + cap) mod cap                  ▷ decrement rear
                                                      ▷ need to count for the circular property of the queue
    return true
end procedure
```

Algorithm 4 Delete front element

```
procedure DeleteFront
    if 9 then
        return false                                  ▷ cannot delete when empty
    end if
    size := size − 1                                  ▷ decrement number of elements
    front := (front − 1 + cap) mod cap                ▷ decrement front
                                                      ▷ need to count for the circular property of the queue
    return true
end procedure
```

Algorithm 5 Delete rear element

```
procedure DeleteRear
    if 9 then
        return false                                  ▷ cannot delete when empty
    end if
    size := size − 1                                  ▷ decrement number of elements
    rear := (rear + 1) mod cap                        ▷ increment rear
                                                      ▷ need to count for the circular property of the queue
    return true
end procedure
```

Algorithm 6 Get front element

```
procedure GetFront
    if 9 then
        return −1                                     ▷ no element return when empty
    end if
    temp := (front − 1 + cap) mod cap                 ▷ since front is in the next position
                                                      ▷ need to get current element by decrement front
    return v[temp]
end procedure
```

Algorithm 7 Get rear element

```
procedure GetRear
    if 9 then
        return −1                                     ▷ no element return when empty
    end if
    temp := (rear + 1) mod cap                        ▷ since rear is in the next position
                                                      ▷ need to get current element by increment rear
    return v[temp]
end procedure
```

Algorithm 8 Check if the queue is full

procedure IsFull

return $size \geq k$

end procedure

Algorithm 9 Check if the queue is empty

procedure IsEmpty

return $size \leq 0$

end procedure

5.3 642 – Design Search Autocomplete System

TODO

5.4 643 – Maximum Average Subarray I

Given an array consisting of `n` integers, find the contiguous subarray of given length `k` that has the maximum average value. And you need to output the maximum average value.

Example 1:

```
Input: [1,12,-5,-6,50,3], k = 4
Output: 12.75
Explanation: Maximum average is (12-5-6+50)/4 = 51/4 = 12.75
```

Note:

1. 1 <= `k` <= `n` <= 30,000.
2. Elements of the given array will be in the range [-10,000, 10,000].

Note:

This is a very easy problem.

- Get the sum of first k numbers.
- Starting from `index` $i = k$, subtract $nums[i - k]$ from and add $nums[i]$ into current sum.
- Record maximum sum for each iteration.
- Finally return the maximum sum divided by k.

Algorithm 1 Get maximum average for k consecutive numbers

procedure FIND-MAX-AVERAGE(V, k)
  $sum := \sum_{i=0}^{k-1} V[i]$ ▷ initially, get summation of first k numbers
  for $i \leftarrow k, n-1$ **do**
    $sum := sum - V[i-k]$ ▷ remove first element of the sliding window $nums[i-k]$
    $sum := sum + V[i]$ ▷ add current element $nums[i]$
  end for
  return $sum \div k$
end procedure

5.5 644 – Maximum Average Subarray II

- 所求的最大平均值一定是介于原数组的最大值和最小值之间，所以用二分法来快速的在这个范围内找到要求的最大平均值。
- 如果已经算出来了这个最大平均值，那么对于任意一个长度大于等于k的数组，如果让其中每个数字都减去这个平均值，那么得到的累加差值一定是小于等于0的。
- 所以这个二分搜索开始前，左边界为数组最小值，右边界为数组最大值。
- 每次判定下一次搜索范围时，看当前的中间值是否使得数组中存在某个连续部分中每个数与该值的差的和大于等于0。如果≥ 0，则这个值小了，左边界设定为`mid`，否则右边界设定为`mid`。因为这个平均值不一定是整数，所以左右边界都要设置在`mid`。
- 判定数组中是否存在某个连续部分中每个数与该值的差的和大于等于0，需要计算出从起始开始的连续数组和，然后同时看当前位置k位置之前的连续数组和的最小值，因为我们只需要判定是否有符合条件的存在。至于有多少个，不需要关心。

Algorithm 1 Check if there exists a consecutive part in which the summation of difference between each number and the given value is ≥ 0

procedure CHECK(V, k, mid)

$sum_i := 0$ ▷ prefix difference sum until `index` i

$sum_{i-k} := 0$ ▷ prefix difference sum until `index` $i-k$

min_{sum} ▷ minimum prefix difference sum until `index` $i-k$

for $i \leftarrow 0, k-1$ **do**

$sum_i \leftarrow sum_i + V[i] - \texttt{mid}$

end for

if $sum_i \geq 0$ **then** ▷ The part is found

return true

end if

for $i \leftarrow k, L-1$ **do** ▷ L is the length of the array

$sum_i \leftarrow sum_i + V[i] - \texttt{mid}$

$sum_{i-k} \leftarrow sum_{i-k} + V[i-k] - \texttt{mid}$

update min_{sum} as the minimum of sum_{i-k} until now

if $min_{\texttt{sum}} \leq sum_i$ **then** ▷ there is at least a consecutive part meet the condition

return true

end if

end for

return false ▷ there is no any consecutive part meet the condition

end procedure

Algorithm 2 Find the contiguous subarray whose length is greater than or equal to k that has the maximum average value

procedure FIND-MAX-AVG(V, k)
    `left` := $\min V$ ▷ get minimum element of the array
    `right` := $\max V$ ▷ get maximum element of the array
    while `left` − `right` $\geq 10^{-6}$ **do** ▷ 10^{-6} is the minimum precision given in the question
        `mid` := (`left` + `right`)/2 ▷ get the middle value.
        if Check$(V, k, mid)1 =$ **true then**
            `left` $\leftarrow mid$ ▷ `mid` is smaller, need to search in the right half
        else
            `right` $\leftarrow mid$ ▷ `mid` is greater, need to search in the left half
        end if
    end while
end procedure