目录

写在前面

我完完整整做出来的题只有 CDEL 四道（C 题还是受到了刘翰之一点启发），另外那三道都是看了攻略才写出来的。这篇题解中也把这三道题也写进去了，是希望能让更多的人明白这些题是怎么搞出来的，评判的时候请只算那四道。

A.Two Friends (Codeforces 8D)

题目

Two Friends

time limit per test: 1 second
memory limit per test: 64 megabytes
input: standard input
output: standard output

Two neighbours, Alan and Bob, live in the city, where there are three buildings only: a cinema, a shop and the house, where they live. The rest is a big asphalt square.

Once they went to the cinema, and the film impressed them so deeply, that when they left the cinema, they did not want to stop discussing it.

Bob wants to get home, but Alan has to go to the shop first, and only then go home. So, they agreed to cover some distance together discussing the film (their common path might pass through the shop, or they might walk circles around the cinema together), and then to part each other's company and go each his

两个朋友

每测试时间限制: 1 秒
每测试内存限制: 64 MB
输入: 标准输入
输出: 标准输出

两个住在同一个城市的邻居，Alan 和 Bob。这个城市中只有三座建筑：电影院、商店和他们住的房屋。其余的部分都是一个大柏油广场。

一天，他们去电影院，看电影感触颇多。当他们离开电影院的时候，他们依旧不愿停止讨论。

Bob 想要回家，但是 Alan 必须得先去商店再回家。所以，他们同意先一起走一段来讨论电影（共同路径可能路过商店，甚至可能绕电影院绕圈）接着离开互相的陪伴走向自己的路。分开后，他们将会思考自己的嗜好，以至于即使他们再度相遇也不能继续之前的讨论。因此，Bob 的路径将是一个从电影院到房屋的连续曲线。而

own way. After they part, they will start thinking about their daily pursuits; and even if they meet again, they won't be able to go on with the discussion. Thus, Bob's path will be a continuous curve, having the cinema and the house as its ends. Alan's path — a continuous curve, going through the shop, and having the cinema and the house as its ends.

Alan 的将会是从电影院出发，路过商店，并在房屋结束的连续曲线。

The film ended late, that's why the whole distance covered by Alan should not differ from the shortest one by more than t_1, and the distance covered by Bob should not differ from the shortest one by more than t_2.

电影结束得很晚。这就导致了 Alan 的路径不能长于 t_1, Bob 的路径不能长于 t_2。

Find the maximum distance that Alan and Bob will cover together, discussing the film.

找出 Alan 和 Bob 能一起讨论电影走过的最大距离。

Input

输入

The first line contains two integers: t_1, t_2 ($0 \le t_1, t_2 \le 100$). The second line contains the cinema's coordinates, the third one — the house's, and the last line — the shop's.

第一行包括两个整数： t_1, t_2 ($0 \le t_1, t_2 \le 100$)。第二行包含电影院的坐标。第三行是房屋的，最后一行是商店的。

All the coordinates are given in meters, are integer, and do not exceed 100 in absolute magnitude. No two given places are in the same building.

所有的坐标都以米为单位，为绝对值不超过 100 的整数。不会有两个地点在同一座楼中。

Output

输出

In the only line output one number — the maximum distance that Alan and Bob will cover together, discussing the film. Output the answer accurate to not less than 4 decimal places.

在唯一的一行中输出一个整数——Alan 和 Bob 能一起讨论电影走过的最大距离。输出精度不少于小数点后 4 位。

Examples

样例

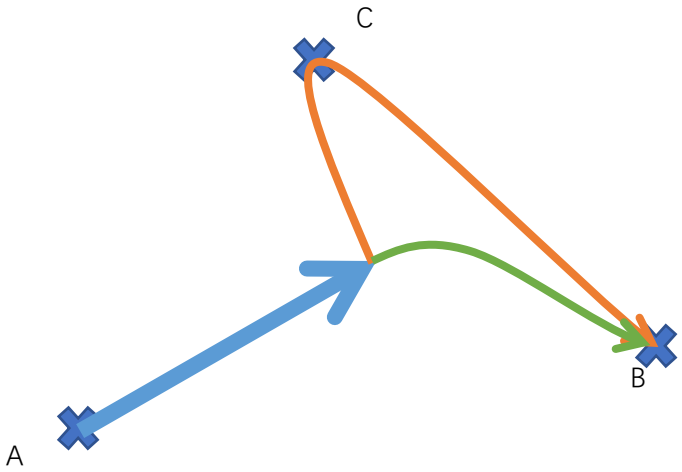
input	输入
0 2 0 0 4 0 -3 0	0 2 0 0 4 0 -3 0
output	**输出**
1.0000000000	1.0000000000
input	**输入**
0 0 0 0 2 0 1 0	0 0 0 0 2 0 1 0
output	**输出**
2.0000000000	2.0000000000

思路

思路来源：AOQNRMGYXLMV

题目简化后大概就是这样一个问题：

有两个人 Alan 和 Bob，他们现在都在 A 点，Bob 想去 B 点，Alan 想先去 C 点再去 B 点。要求 Alan 走过的长度不能超过 t_1，Bob 走过的长度不能超过 t_2，求两人在一起最多走多远。



首先考虑一种特殊情况，如果 Bob 能一直陪着 Alan(即$t_2 \geq \overline{AC} + \overline{CB}$)，那么答案显然为

$\min(t_1, t_2)$。此时他们一定是在 Alan 到达 C 之前分开的。这种特殊情况直接特判掉。而其他情况时，他俩分离的时候一定还没有到达 C 点（不然直接一起回家了更优）。对于一般情况的时候直接想似乎很难，那么这时候就听栗子的，看看如果给一个答案进行判断难度如何。我们发现判断并不难。而且结果是单调的，那么我们便可以使用二分+判断的方法。

判断：设走的公共距离是 x，分离点是 P，那么 P 必须满足：

1. P 必须在以 A 为圆心半径为 x 的圆内，因为他们走的公共距离为 x

$$\overline{PA} \leq x$$

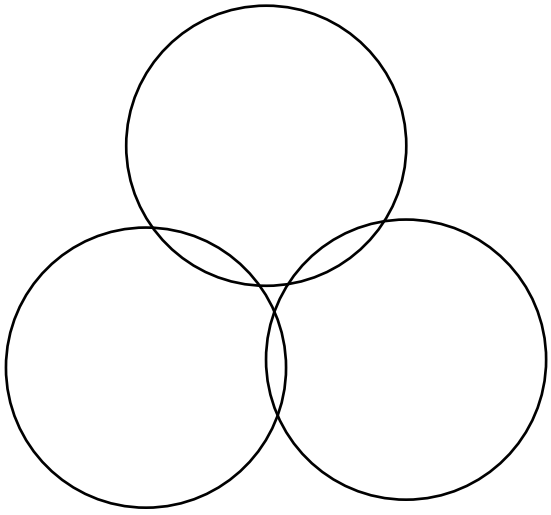
2. P 必须在以 B 为圆心半径为 L2−x 的圆内，为了让 Bob 在分开之后能及时返回 B 点

$$\overline{PB} \leq t_2 - x$$

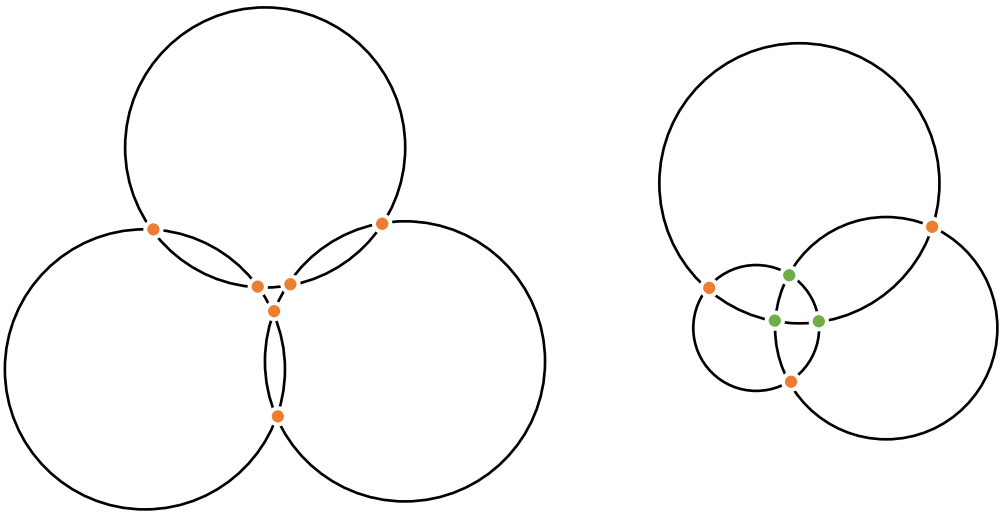
3. P 必选在以 C 为圆心半径为 L1−x−BC 的圆内，因为 Alan 在到达 C 之后还要径直走回 B 点。

$$\overline{PC} \leq t_1 - \overline{BC} - x$$

那么问题就变成了判断三个圆是否有公共部分了。最容易想到的就是三个圆两两相交，但是这并不是充分条件，因为还可能有这样的情况：



两两相交，但是却并没有公共部分。所以还要判断是否有一个交点在三个圆任何一个圆的边界或内部



无效交点以橙色表示,有效交点以绿色表示.

以一个圆为主圆,判断另外两个圆是否和它相交的函数(伪代码，方法来自AOQNRMGYXLMV)

```
bool CheckUnit(point a, double r_a, point b, double r_b, point c, double r_c) {
    if (|c − a| ≤ r_a && |c − b| ≤ r_b) return true;
    b -= a; c -= a;
    point r = (b_x/|b|, b_y/|b|);
    b /= r; c /= r;
    double d = (r_a^2 − r_b^2 + |b|^2)/(2|b|);
    double h = sqrt(max(r_a^2 − d^2, 0));
    if (|h^2 + (d − |b|)^2| == r_b^2) return false;
    if ((d, h) − c ≤ r_c || (d, −h) − c ≤ r_c) return true;
    return false;
}
```

那自然还要有判断三个圆是否有公共部分的伪代码：

```
bool check(point a, double r_a, point b, double r_b, point c, double r_c) {
    if (CheckUnit(a, r_a, c, r_c, b, r_b)) return true;
    if (CheckUnit(b, r_b, a, r_a, c, r_c)) return true;
    if (CheckUnit(c, r_c, b, r_b, a, r_a)) return true;
    return false;
```

```
}
```

在程序里点的坐标怎么保存呀？用 complex 容器呀！运算符和函数都已经给重载好了，直接用就好啦~

C.Black Box

(POJ – 1442, Northeastern Europe 1996)

题目

Black Box

Time Limit :1000MS Memory Limit :10000K

Description

Our Black Box represents a primitive database. It can save an integer array and has a special i variable. At the initial moment Black Box is empty and i equals 0. This Black Box processes a sequence of commands (transactions). There are two types of transactions:

ADD (x): put element x into Black Box;
GET: increase i by 1 and give an i-minimum out of all integers containing in the Black Box. Keep in mind that i-minimum is a number located at i-th place after Black Box elements sorting by non-descending.

Let us examine a possible sequence of 11 transactions:

Example 1

黑盒

时间限制 :1000MS 内存限制 :10000K

描述

我们的黑盒子代表一个原始数据库。 它可以保存一个整数数组，并具有一个特殊的 i 变量。 在最初的时刻，黑盒子是空的，i 等于 0.这个黑盒子处理一系列命令（操作）。 有两种类型的操作：

ADD（x）：将元素 x 放入黑盒子;
GET：将 i 增加 1，并给出包含在黑盒中的所有整数中的 i 位置元素。 请记住，黑盒子元素排序为升序。

我们来看一下 11 个操作的可能顺序：

例 1

N	Transaction	i	Black Box contents after transaction	Answer
N	操作	i	操作后黑盒内容	答案
	(elements are arranged by non-descending 元素升序排列)			
1	ADD(3)	0	3	
2	GET	1	3	3
3	ADD(1)	1	1, 3	
4	GET	2	1, 3	3
5	ADD(-4)	2	-4, 1, 3	
6	ADD(2)	2	-4, 1, 2, 3	
7	ADD(8)	2	-4, 1, 2, 3, 8	
8	ADD(-1000)	2	-1000, -4, 1, 2, 3, 8	
9	GET	3	-1000, -4, 1, 2, 3, 8	1
10	GET	4	-1000, -4, 1, 2, 3, 8	2
11	ADD(2)	4	-1000, -4, 1, 2, 2, 3, 8	

It is required to work out an efficient algorithm which treats a given sequence of transactions. The maximum number of ADD and GET transactions: 30000 of each type.

需要制定一种有效的算法来处理给定的操作顺序。 ADD 和 GET 操作的最大数量：每种类型为 30000。

Let us describe the sequence of transactions by two integer arrays:

我们用两个整数数组来描述事务的顺序：

1. A(1), A(2), ..., A(M): a sequence of elements which are being included into Black Box. A values are integers not exceeding 2 000 000 000 by their absolute value, M <= 30000. For the Example we have A=(3, 1, -4, 2, 8, -1000, 2).

1.A（1），A（2），...，A（M）：包含在黑盒中的一系列元素。 A 值是绝对值不超过 2 000 000 000 的整数，M <= 30000。对于示例，我们有 A =（3，1，-4，2，8，-1000，2）。

2. u(1), u(2), ..., u(N): a sequence setting

2. u（1），u（2），...，u（N）：设置在

a number of elements which are being included into Black Box at the moment of first, second, ... and N-transaction GET. For the Example we have u=(1, 2, 6, 6).

第一，第二，...和 N 交易时刻被包含在黑盒中的多个元素的序列 得到。 对于示例，我们有 u =（1，2，6，6）。

The Black Box algorithm supposes that natural number sequence u(1), u(2), ..., u(N) is sorted in non-descending order, N <= M and for each p (1 <= p <= N) an inequality p <= u(p) <= M is valid. It follows from the fact that for the p-element of our u sequence we perform a GET transaction giving p-minimum number from our A(1), A(2), ..., A(u(p)) sequence.

黑盒算法假设自然数序列 u（1），u（2），...，u（N）以升序排序，N <= M，对于每个 p（1 <= p < N）不等式 p <= u（p）<= M 是有效的。 从以下事实可以看出，对于我们的 u 序列的 p 元素，我们执行从我们的 A（1），A（2），...，A（u（p））序列得到 p 最小数的 GET 操作。

Input

输入

Input contains (in given order): M, N, A(1), A(2), ..., A(M), u(1), u(2), ..., u(N). All numbers are divided by spaces and (or) carriage return characters.

输入包含（按给定顺序）：M，N，A（1），A（2），...，A（M），u（1），u（2），...，u（N） 所有数字以空格和（或）回车字符分割。

Output

输出

Write to the output Black Box answers sequence for a given sequence of transactions, one number each line.

输出黑盒子对于给定的操作顺序应答序列，每行一个数字。

Sample Input

样例输入

```
7 4
3 1 -4 2 8 -1000 2
1 2 6 6
```

Sample Output

样例输出

```
3
3
1
2
```

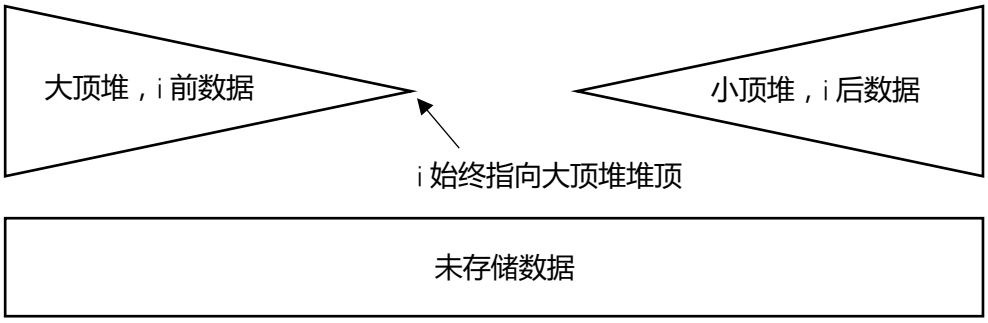
思路

思路来源：刘翰之

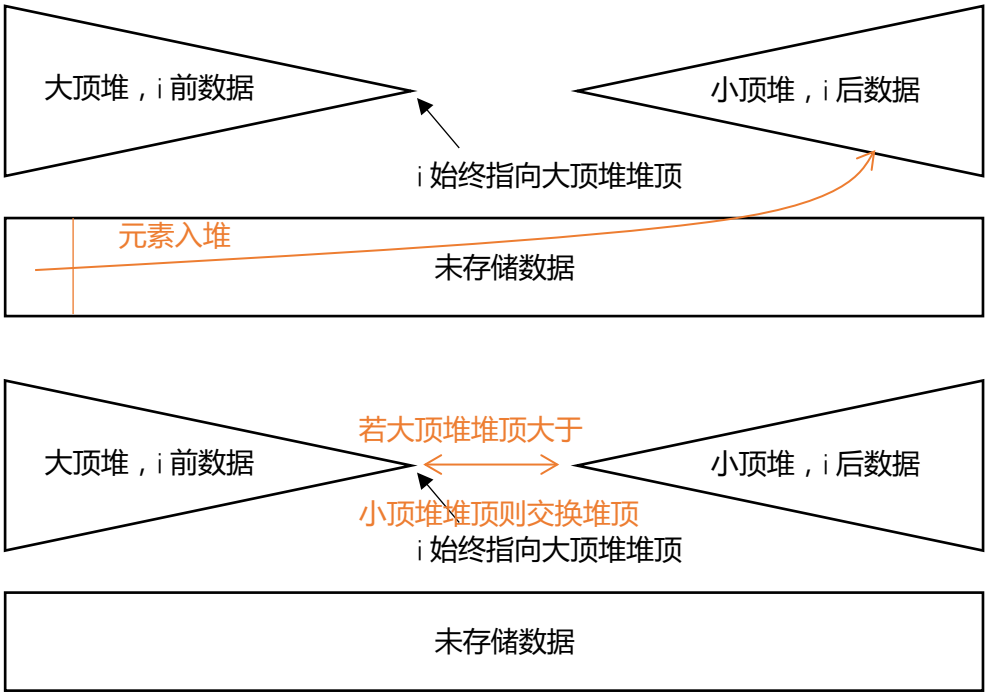
最简单粗暴的方式，直接模拟一遍，不用试，看看数据规模就知道肯定会 Time Limited

Exceeded(怎么会有那么水的题)。这里面有两种操作，存和取，分别有 30 000 次，也就是说存和取都不能超过 O(nlogn)才可以过。但是常用的两种顺序表——连续（存：n^2logn，取 nlogn），链式（存：n^2，取 n^2）都不能满足我们的要求。存取均为 nlogn 的有什么呢？平衡树！可是这我也不会呀。想把 STL 里自带红黑树的几个容器改改好像也想不到怎么改。（会平衡树并且用平衡树 Accepted 掉的大佬请接受我的膜拜）。

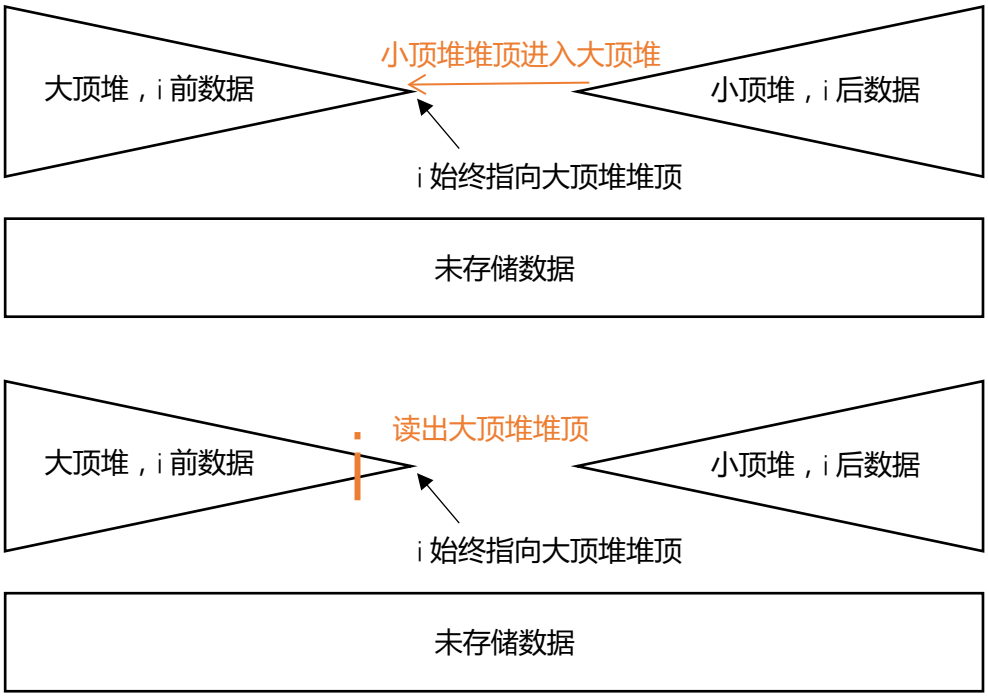
咱们既然不会用那些高级的数据结构就得想点别的办法，受到这东西一直是升序的启发，或许可以用堆来做。当然一个堆肯定是不够的啦，我们需要两个堆。一个大顶堆用于存放 i 和 i 前面的数据，一个小顶堆存放 i 后面的数据，就像这样



那么 ADD 操作就是



GET 操作就是



堆的操作时间复杂度都是 logn，共有 n 次操作，那就是 nlogn，刚好满足题目要求。

你问我堆怎么写？当然是优先队列啦！

首先

```
#include<queue>
#include<functional>
```

然后

```
priority_queue<int, vector<int>, less<int> >BigTopHeap;
priority_queue<int, vector<int>, greater<int> >SmallTopHeap;
```

注意两个连续的尖括号中间加上空格，防止编译器认为这是向右移位而 Compile Error 掉

D.Leading and Trailing (LightOJ - 1282)

题目

Leading and Trailing	领先和落后

You are given two integers: **n** and **k**, your task is to find the most significant three digits, and least significant three digits of n^k.

Input

Input starts with an integer T (≤ 1000), denoting the number of test cases.Each case starts with a line containing two integers: n (2 ≤ n < 231) and k (1 ≤ k ≤ 107).

Output

For each case, print the case number and the three leading digits (most significant) and three trailing digits (least significant). You can assume that the input is given such that n^k contains at least six digits.

给出两个整数 n 和 k，你的任务是找出 n^k 的前三位和后三位。

输入

输入以一个整数 T (≤ 1000)开始，表示数据的组数。每一组数据有同一行的两个整数: n (2 ≤ n < 231) and k (1 ≤ k ≤ 107).

输出

对于每种情况，打印案例编号和三个前导数字（最高有效数字）和三位尾数（最低位）。 您可以假设输入被给出，使得 n^k 至少包含六位数字。

Sample Input
样例输入

```
5
123456 1
123456 2
2 31
2 32
29 8751919
```

Output for Sample Input
样例输出

```
Case 1: 123 456
Case 2: 152 936
Case 3: 214 648
Case 4: 429 296
Case 5: 665 669
```

思路

思路来源：原创、Flintx

低三位直接快速幂对 1000 取模就好了。栗子当年手打的快速幂我还留着(代码来自 :栗子)

```
long long fastpowermod(long long a, long long n) {
    long long res = 1;
    while (n > 0) {
        if (n & 1)res = res*a%mod;
        a = a*a%mod;
        n >>= 1;
    }
```

```
    return res;
}
```

高三位怎么办，最开始的思路肯定还是快速幂啦，可是由于乘方后数字极大，就算是 long long int 也不能保存这么大的数字，这可怎么办呢。还有什么类型能保存更大的数字呢？double！double 的上限大概是 10^{308}，我想这应该足够大了吧。可是实践过之后却发现给我输出了一个 inf。说明还是爆掉了。这样看来这个思路大概就是有问题的了。

或许……我们并不需要那么高的精度，我们只要前三位就好了。可是计算中后面的会对前面的有影响呀！影响？影响有多大呢？经过笔算我们发现，低位数字对高位数字的影响主要来自进位，越到后面的影响越小，印象中 double 似乎可以保留 17 位有效数字，那么我们可以试一试这种方法。double 快速幂·改：

```
double fastpower(double a, long long int n) {
    double res = 1;
    while (n > 0) {
        if (n & 1)res = res*a;
        a = a*a;
        n >>= 1;
        while (res >= 1000)res /= 10;
        while (a >= 1000)a /= 10;
    }
    return res;
}
```

这样交了一发之后，竟然直接 Accepted 了！但是这其实是在原来的思路上打了补丁过掉的，而且是有缺陷的，只是数据没有专门卡掉我这种做法（如果一路进位这样做是会 Wrong Answer 的）那么还有没有更好的方法呢？当然是有的，大数一般用到的都是取对数的方法，这题也是一样的

$$f(n) = n^k$$
$$tmp = log_{10} f(n) = k\ log_{10} n$$
$$tmp = tmp - \lfloor tmp \rfloor$$
$$tmp = 10^{tmp}$$

然后把 tmp 放大到所需位数就行啦~

E.Aladdin and the Flying Carpet (LightOJ - 1341)

题目

Aladdin and the Flying Carpet

It's said that Aladdin had to solve seven mysteries before getting the Magical Lamp which summons a powerful Genie. Here we are concerned about the first mystery.

Aladdin was about to enter to a magical cave, led by the evil sorcerer who disguised himself as Aladdin's uncle, found a strange magical flying carpet at the entrance. There were some strange creatures guarding the entrance of the cave. Aladdin could run, but he knew that there was a high chance of getting caught. So, he decided to use the magical flying carpet. The carpet was rectangular shaped, but not square shaped. Aladdin took the carpet and with the help of it he passed the entrance.

Now you are given the area of the carpet and the length of the minimum possible side of the carpet, your task is to find how many types of carpets are possible. For example, the area of the carpet 12, and the minimum possible side of the carpet is 2, then there can be two types of carpets and their sides are: {2, 6} and {3, 4}.

阿拉丁和飞毯

据说，阿拉丁在获得魔法灯之前，必须解决七个奥秘，这是召唤一个强大的精灵。 在这里我们关心第一个谜。

阿拉丁即将进入一个魔法洞穴，由伪装成阿拉丁叔叔的邪恶巫师领导，在入口处发现了一个奇怪的魔法飞地毯。 有一些奇怪的生物守护着洞穴的入口。 阿拉丁可以跑，但他知道很有可能被抓住。 所以，他决定用魔法飞地毯。 地毯是矩形，但不是方形。 阿拉丁拿着地毯，在他的帮助下，他通过了入口。

现在你给了地毯的区域和地毯的最小可能侧面的长度，你的任务是找到可能有多少种类的地毯。 例如，地毯 12 的面积和地毯的最小可能面积为 2，那么可以有两种类型的地毯，它们的侧面是：{2,6}和{3,4}。

Input

Input starts with an integer T (≤ 4000), denoting the number of test cases.

Each case starts with a line containing two integers: a b (1 ≤ b ≤ a ≤ 1012) where a denotes the area of the carpet and b denotes the minimum possible side of the carpet.

Output

For each case, print the case number and the number of possible carpets.

输入

输入以整数 T（≤4000）开始，表示测试用例数。

每个案例以包含两个整数的行开始：a b（1≤b≤a≤1012）其中 a 表示地毯的面积，b 表示地毯的最小可能侧面。

输出

对于每种情况，输出案例编号和可能的地毯数量。

Sample Input
样例输入

```
2
10 2
12 2
```

Output for Sample Input
样例输出

```
Case 1: 1
Case 2: 2
```

思路

思路来源：原创

这题看着真是好啰嗦呀。简单来说就是找一个数比 b 大或相等的因数对的个数。

首先，最笨的方法——爆搜。我竟然天真地交了一发，结果 Time Limited Exceeded 了。

于是就想了，有没有能快速计算一个数因数个数的方法呢？忽然想到小学期课上讲过约数个数定理（其实就数论那节课听得差，各种证明好多都听不懂）。就是分解出来的质因数+1 再相乘就是总的因数个数。于是只要挨个找出那个数的质因数，就知道了它的因数个数，除以 2 后再暴力排除掉比 b 小的那些。于是我又 Time Limited Exceeded 了一发。

难道这样算得还是不够快？难道还有更快的方法？找到它的质因数……质因数……质因数……质数……对呀！质因数一定是质数，我干嘛去判断那些合数。所以应该先找出所有的质数。那么就有找质数大法——欧拉筛！（为什么都是数论课上的）对于任何一个比 1 大的整

数 n，2n、3n、4n.....一定都不是质数，听大佬说筛到 n^2 是最快的而且没有重复（虽然我也不知道为什么）。这样再判断的时候就只判断质数是不是质因数就好了（不是先判断是不是质数，而是直接跳到下一个质数上！不然还是一样的慢）。顺利 Accepted。

F.Parade (Codeforces – 35E)

题目

Parade

time limit per test: 2 seconds
memory limit per test: 64 megabytes
input: input.txt
output: output.txt

No Great Victory anniversary in Berland has ever passed without the war parade. This year is not an exception. That's why the preparations are on in full strength. Tanks are building a line, artillery mounts are ready to fire, soldiers are marching on the main square... And the air forces general Mr. Generalov is in trouble again. This year a lot of sky-scrapers have been built which makes it difficult for the airplanes to fly above the city. It was decided that the planes should fly strictly from south to north. Moreover, there must be no sky scraper on a plane's route, otherwise the anniversary will become a tragedy. The Ministry of Building gave the data on n sky scrapers (the rest of the buildings are rather small and will not be a problem to the planes). When looking at the city from south to north as a geometrical plane, the i-th building is a

阅兵

每测试时间限制: 2 秒
每测试内存限制: 64 MB
输入: input.txt
输出: output.txt

在没有战争游行的情况下，没有伟大的胜利纪念日。今年不是例外。这就是为什么准备工作充分发挥的原因。坦克正在建造一条线，炮兵准备开火，士兵们正在主要广场上行进。空军将军普罗诺夫先生再次遇到麻烦。今年已经建成了大量的刮铲，这使得飞机难以飞越城市。决定飞机应严格从南向北飞行。此外，飞机上的路线上不能有天空刮板，否则周年将成为悲剧。建筑部给出了 n 个天空刮板的数据（其余的建筑物相当小，对飞机来说不是问题）。当从南到北看城市作为几何平面时，第 i 栋建筑是高度嗨的矩形。其最西点是李和 x 的最东边的 x 坐标。该地区的地形平坦，所有的建筑物都站在一个层面上。你作为国防部的头号程序员的任务是使用天空刮刀上的数据找到一个包络折线。折线的属性如下：

rectangle of height hi. Its westernmost point has the x-coordinate of li and the easternmost — of ri. The terrain of the area is plain so that all the buildings stand on one level. Your task as the Ministry of Defence's head programmer is to find an enveloping polyline using the data on the sky-scrapers. The polyline's properties are as follows:

If you look at the city from south to north as a plane, then any part of any building will be inside or on the boarder of the area that the polyline encloses together with the land surface.

如果你把这个城市从南到北看作一架飞机，那么任何建筑物的任何一部分都将在折线与地面一起包围的区域的内部或边界上。

The polyline starts and ends on the land level, i.e. at the height equal to 0.

折线在地面上开始和结束，即高度等于0。

The segments of the polyline are parallel to the coordinate axes, i.e. they can only be vertical or horizontal.

折线的线段平行于坐标轴，即它们只能是垂直的或水平的。

The polyline's vertices should have integer coordinates.

折线的顶点应该有整数坐标。

If you look at the city from south to north the polyline (together with the land surface) must enclose the minimum possible area.

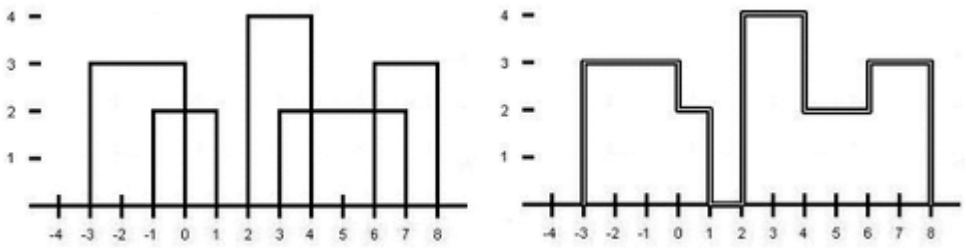
如果您从南到北看城市，折线（与地面一起）必须包围最小可能的面积。

The polyline must have the smallest length among all the polylines, enclosing the minimum possible area with the land.

折线必须在所有折线之间具有最小的长度，包围土地的最小可能面积。

The consecutive segments of the polyline must be perpendicular.

折线的连续片段必须垂直。



Picture to the second sample test (the enveloping polyline is marked on the right).

图片表示样例测试 2（包络折线标记在右侧）。

Input

The first input line contains integer n ($1 \le n \le 100000$). Then follow n lines, each containing three integers h_i, l_i, r_i($1 \le h_i \le 10^9$, $-10^9 \le l_i < r_i \le 10^9$).

输入

第一行包括一个整数 n ($1 \le n \le 100000$). 接下来的 n 行,每一行三个整数 h_i, l_i, r_i($1 \le h_i \le 10^9$, $-10^9 \le l_i < r_i \le 10^9$).

Output

In the first line output integer m — amount of vertices of the enveloping polyline. The next m lines should contain 2 integers each — the position and the height of the polyline's vertex. Output the coordinates of each vertex in the order of traversing the polyline from west to east. Remember that the first and the last vertices of the polyline should have the height of 0.

输出

在第一行输出整数 m - 包络折线的顶点的数量。 接下来的 m 行应该包含 2 个整数 - 折线的顶点的位置和高度。 按照从西向东遍历折线的顺序输出每个顶点的坐标。 请记住，折线的第一个和最后一个顶点的高度应为 0。

Examples

样例

input

```
2
3 0 2
4 1 3
```

输入

```
2
3 0 2
4 1 3
```

output

```
6
0 0
0 3
1 3
1 4
```

输出

```
6
0 0
0 3
1 3
1 4
```

```
3 4
3 0
```

input

```
5
3 -3 0
2 -1 1
4 2 4
2 3 7
3 6 8
```

output

```
14
-3 0
-3 3
0 3
0 2
1 2
1 0
2 0
2 4
4 4
4 2
6 2
6 3
8 3
8 0
```


```
3 4
3 0
```

输入

```
5
3 -3 0
2 -1 1
4 2 4
2 3 7
3 6 8
```

输出

```
14
-3 0
-3 3
0 3
0 2
1 2
1 0
2 0
2 4
4 4
4 2
6 2
6 3
8 3
8 0
```


思路

思路来源：AOQNRMGYXLMV

首先，这道题是文件输入输出（已在题目中用红色标记）!

```
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
```

将每个矩形拆成两个事件：{l,y,+}和{r,y,−}分别表示在扫描先到 l 位置将 y 插入数据结构，和

扫描线到 r 位置将 y 从数据结构种删除。

并且该数据结构支持查询最大值

维护数据结构中的当前最大值，每当最大值发生变化时，便找到轮廓线上的两个点(x,preH)，

(x,curH)。

初始时将 0 加入数据结构表示地平线，就不用特判集合是否非空。

STL 中的 multiset 就可以完成上述操作。

两个变量的时候可以直接用 pair 而不用开结构体

```
multiset<int> contain;
vector<pair<int, int> > tag, out;
```

储存

```
tag.push_back(make_pair(l, y));
tag.push_back(make_pair(r, -y));
```

排序

```
sort(tag.begin(), tag.end());
```

初始化

```
contain.insert(0);
int maxh = 0;
```

一波扫描

```
for (int i = 0, j; i < n * 2; i = j) {
    for (j = i; j < n * 2 && tag[i].first == tag[j].first; j++) {
        if (tag[j].second > 0) contain.insert(tag[j].second);
        else contain.erase(contain.lower_bound(-tag[j].second));
    }
    if ((*contain.rbegin()) != maxh) {
        out.push_back(make_pair(tag[i].first, maxh));
        out.push_back(make_pair(tag[i].first, maxh = (*contain.rbegin())));
    }
}
```

然后再把 out 里的东西打出来

H.Winning an Auction

(HDU – 5951，

2016ACM/ICPC 亚洲区沈阳站-重现赛)

题目

Winning an Auction

Time Limit :2000MS Memory Limit : 131072K

Problem Description

Alice and Bob play an auction game. Alice has A dollars and Bob has B dollars initially. There are N items on sale. In each round, an item will be sold by the following way. Alice writes down an integer a (0 ≤ a ≤ A) and Bob writes down an integer b (0 ≤ b ≤ B), which are the amount of dollars they want to pay for the item. If a > b, then Alice gets the item and pays a dollars to the seller. If a < b, then Bob gets the item and pays b dollars to the seller. If a = b, then for the 1st, 3rd, 5th, 7th ... round, Alice gets the item and pays a dollars; for the 2rd, 4th, 6th, 8th ... round, Bob gets the item and pays b dollars. Since all the items have the same value, the goal of the auction game is to get as many items as possible. Both Alice and Bob know the values of N,A and B. Your task is to calculate how many items they will get if both of them play optimally.

Input

赢得拍卖

时间限制 :2000MS 内存限制 : 131072K

描述

爱丽丝和鲍勃玩一个拍卖游戏。 爱丽丝有一美元，鲍勃最初有 B 美元。 有 N 件出售。 在每一轮中，一个物品将通过以下方式出售。 Alice 写下一个整数 a（0≤a≤A），Bob 写下一个整数 b（0≤b≤B），这是他们要支付的项目金额。 如果 a> b，那么 Alice 将会收到该商品，并向卖家支付一美元。 如果 a<b，那么鲍勃得到的项目，并向卖方支付 B 美元。 如果 a = b，那么对于第 1，第 3，第 5，第 7，...轮，爱丽丝得到项目并支付一美元; 在第二，第四，第六，第八，第八轮，鲍勃得到的项目，并支付 b 美元。 由于所有项目都具有相同的价值，拍卖游戏的目标是尽可能多地获得项目。 爱丽丝和鲍勃都知道 N，A 和 B 的价值。你的任务是计算如果两个玩家都玩得最好，他们将获得多少个项目。

Input

The first line is the number of test cases. Each test case contains 3 integers N,A and B, which are no larger than 255.

Output

For each test case, output the number of items Alice and Bob will get if both of them play optimally.

Sample Input

```
3
1 1 2
2 4 2
3 3 3
```

Sample Output

```
Alice 0 Bob 1
Alice 1 Bob 1
Alice 2 Bob 1
```

第一行是测试用例的数量。 每个测试用例包含 3 个整数 N，A 和 B，不大于 255。

Output

对于每个测试用例，输出 Alice 和 Bob 如果它们都能最佳地游戏，将得到的项目数。

样例输入

```
3
1 1 2
2 4 2
3 3 3
```

样例输出

```
Alice 0 Bob 1
Alice 1 Bob 1
Alice 2 Bob 1
```

思路

思路来源：snowy_smile

这题需要用 short，不然会爆内存（题目中红色标出）

用 f[o][i][j][k]

表示初始 n 的奇偶性为 o，现在考虑到倒数第 i 件物品，Alice 还剩下 j 元钱，Bob 还剩下 k 元钱

对于第 1 ~ i 的所有物品，Alice 在最优策略下所能获得的最大物品数。

那么我们可以从 0 开始枚举物品的实际购买价格。

例如假设现是 Alice 的奇偶性优势点，Alice 可以以 x 元购买这个物品，那么 Alice 的收益会是 f[o][i - 1][j - x][k] + 1

而 Bob 如果以 x + 1 元购买这个物品，使得 Alice 的收益会是 f[o][i - 1][j][k - x - 1]

如果 Bob 的这个"x + 1 元购买"的决策，相比于 Alice 的"x 元购买"策略，能使得 Alice 的收益

变少，则 Bob 肯定会花这 x + 1 元

同理，如果 Alice 的"x + 1"元购买策略，相比于 Bob 的"x + 1 元购买策略"，能使得 Alice 的收益增加，则 Alice 肯定会花这 x + 1 元

……

两人一直进行到，其中一人无力出更高价或者即使出更高价也不优。

DP 转移过程（代码来自 snowy_smile）：

```
if ((o - p + 1) & 1)//Alice占据优势
{
    int va = f[o][p - 1][a][b] + 1; //初始a买的状态
    int vb;
    for (int u = 1; ; ++u)
    {
        if (b < u || (vb = f[o][p - 1][a][b - u]) >= va) //b多花钱没意义
        {
            f[o][p][a][b] = va;
            break;
        }
        if (a < u || (va = f[o][p - 1][a - u][b] + 1) <= vb)//a多花钱没意义
        {
            f[o][p][a][b] = vb;
            break;
        }
    }
}
else//Bob占据优势
{
    int vb = f[o][p - 1][a][b]; //初始b买的状态
    int va;
    for (int u = 1; ; ++u)
    {
        if (a < u || (va = f[o][p - 1][a - u][b] + 1) <= vb)//a多花钱没意义
        {
            f[o][p][a][b] = vb;
            break;
        }
        if (b < u || (vb = f[o][p - 1][a][b - u]) >= va) //b多花钱没意义
        {
            f[o][p][a][b] = va;
```

```
                break;
            }
        }
    }
```

L.Bi-shoe and Phi-shoe (LightOJ – 1370)

题目

Bi-shoe and Phi-shoe	【不会翻译】
Bamboo Pole-vault is a massively popular sport in Xzhiland. And Master Phi-shoe is a very popular coach for his success. He needs some bamboos for his students, so he asked his assistant Bi-Shoe to go to the market and buy them. Plenty of Bamboos of all possible integer lengths (yes!) are available in the market. According to Xzhila tradition,	竹撑竿跳高是 Xzhiland 非常受欢迎的运动。大师鞋是他成功的非常受欢迎的教练。他需要一些竹子给他的学生，所以他请他的助手 Bi Shoe 去市场买。大量竹子可能的整数长度（是的！）市场上有现货。据 xzhila 传统，
Score of a bamboo = Φ (bamboo's length)	一个竹竿的评分=Φ（竹竿的长度）
(Xzhilans are really fond of number theory). For your information, Φ (n) = numbers less than n which are relatively prime (having no common divisor other than 1) to n. So, score of a bamboo of length 9 is 6 as 1, 2, 4, 5, 7, 8 are relatively prime to 9.	（xzhilans 真的喜欢数理论）。给您的信息，Φ（n）=数小于 n 互质（没有公约数 1 除外），所以，一根长度为 9 的得分为 6，1, 2, 4，5, 7, 8 是互质的 9
The assistant Bi-shoe has to buy one bamboo for each student. As a twist, each pole-vault student of Phi-shoe has a lucky number. Bi-shoe wants to buy bamboos such that each of them gets a bamboo with a score greater than or equal to his/her lucky number. Bi-shoe wants to	Bi-shoe 助理必须为每个学生买一把竹子。作为一个转折点，每一个撑杆跳的学生都有一个幸运数字。鞋要买竹子，使他们每个人得到一个竹子的得分大于或等于他的幸运数字。BI 想尽量减少购买竹子所花的钱。一个单位成本 1 xukha。帮助他。

minimize the total amount of money spent for buying the bamboos. One unit of bamboo costs 1 Xukha. Help him.

Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing an integer n (1 ≤ n ≤ 10000) denoting the number of students of Phi-shoe. The next line contains n space separated integers denoting the lucky numbers for the students. Each lucky number will lie in the range [1, 106].

Output

For each case, print the case number and the minimum possible money spent for buying the bamboos. See the samples for details.

输入

输入以整数 T（≤100）开始，表示测试用例数。

每个案例以包含 Phi-shoe 学生人数的整数 n（1≤n≤10000）开始。 下一行包含 n 个空格分隔的整数，表示学生的幸运数字。 每个幸运号码将在[1,106]的范围内。

输出

对于每种情况，打印案件编号和购买竹子所需的最低可能花费。 详见样例。

Sample Input
样例输入

```
3
5
1 2 3 4 5
6
10 11 12 13 14 15
2
1 1
```

Output for Sample Input
样例输出

```
Case 1: 22 Xukha
Case 2: 88 Xukha
Case 3: 4 Xukha
```

思路

思路来源：原创

这题看似复杂，其实就只是要欧拉函数大于等于这个数的最小的输（好拗口）。

那既然是这样的话最简单最方便的方法就是打个表出来。先打欧拉函数的表（代码来源：

ACM 小学期课）：

```
void getphi()
{
    int i, j;
    phi[1] = 1;
    for (i = 2; i <= N; i++)
    {
        if (!mark[i]) { prime[++tot] = i; phi[i] = i - 1; }
        for (j = 1; j <= tot; j++)
        {
            if (i*prime[j]>N)  break;
            mark[i*prime[j]] = 1;
            if (i%prime[j] == 0)
            {
                phi[i*prime[j]] = phi[i] * prime[j]; break;
            }
            else  phi[i*prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    phi[1] = 0;
}
```

接着再整理那个反向的表

```
void tidy(void) {
    long long curmax = 0;
    for (int i = 2; i <= N; i++) {
        if (phi[i] > curmax) {
            int t = phi[i] + 1;
            curmax = phi[i];
            while (t--) {
                if (f[t] != 0)break;
                f[t] = i;
            }
        }
    }
}
```

读入数据后只要查表 f[x]就行了

累加，输出，Accepted