Hallgatók tehetséggondozása és felkészítése egyetemi szintű, szakmai tanulmányi versenyekre az algoritmuselmélet és programozástechnika témakörében a 2011/2012. őszi félévben

„Tehetségtámogató programok az 5 kiemelt kutatási területen”

Szakmai dokumentáció, Engedy Balázs, 2011/1438.

Összefoglalás

A Számítástudományi és Információelméleti Tanszék immáron tíz éve szervez „ACM szakkör” néven olyan kiscsoportos foglalkozásokat, amelyek célja a legtehetségesebb hallgatóink algoritmuselmélet- és programozástechnika-tudásának, problémamegoldó készségének fejlesztése, illetve a hallgatók felkészítése napjaink legrangosabb programozási versenyeire. A szakkör hetente egy alkalommal, fakultatív jelleggel, órarenden kívüli foglalkozásként kerül megtartásra, nincsen – a szokásos értelemben vett – számonkérés vagy (formális) előkövetelmény. Ennek ellenére az érdeklődő hallgatók még a félév végéhez közeledve is szép számban látogatják a foglalkozásokat, illetve több más egyetemről is érkeznek résztvevők.

Részletes ismertetés

A szakkör célja, felépítése

A szakkör általános célja, hogy fejlessze a hallgatók készségét programozási versenyfeladatok önálló és csapatban történő megoldására. Egy versenyfeladat megoldása – egy általános mérnöki feladathoz hasonlóan – alapvetően három fázisra bontható:

1. a feladat értelmezése, a feladatban vázolt problémára egy absztrakt, matematikai modell felállítása,
2. a matematikai modellben egy hatékony algoritmus kidolgozása a probléma megoldására,
3. az algoritmus implementálása egy adott programnyelven.

Az első fázishoz jó modellezőképesség kell, amihez nélkülözhetetlen a diszkrét matematika egyes területeinek (pl. a gráfelméletnek) a mélyreható ismerete. A második fázishoz tipikusan gráfelméleti, algoritmuselméleti, számelméleti és csoportelméleti ismeretekre és matematikai gondolkodási képességre van szükség, a harmadik fázishoz pedig természetesen arra, hogy a hallgató precízen, gyorsan és hibamentesen tudjon programozni.

A foglalkozások során tipikusan a feladatmegoldás mindhárom fázisára hangsúlyt fektetünk. Ezzel összhangban a szakkör feladatmegoldó szeminárium jelleggel zajlik, azaz elsősorban a gyakorlatra fókuszál, és elméleti ismereteket, versenyekhez szükséges elméleti alaptudást a gyakorlati feladatokhoz kapcsolódóan ad át. Az óra elején három-négy gondosan összeválogatott, különböző nehézségű versenyfeladatot tartalmazó feladatsort osztunk ki a hallgatóknak. Ezután önálló munka következik, ennek során a szakkörvezető egyénileg konzultál a hallgatókkal, szükség esetén pótolja az elméleti hiányosságokat, apróbb ötleteket ad a feladat megoldásához. A foglalkozás végül a feladatok közös megbeszélésével az egyes megoldások összehasonlításával zárul. Amennyiben az idő engedi, egyes feladatokhoz az implementációt egy-egy on-line kiértékelőrendszerbe is feltöltjük ellenőrzésre.

A 2011/2012. őszi félév során összesen 9 alkalommal tartottunk 90-120 perc terjedelemben foglalkozást. A függelékben csatolva megtalálható az ebben a félévben megoldott minden feladat szövege.

Tematika

A szakkör során megbeszélt feladatok többsége napjaink három legrangosabb programozási versenyének archívumából származik, melyek tipikusan évente kerülnek megrendezésre, és több éves, évtizedes múltra tekintenek vissza. Név szerint:

1. „The ACM-ICPC Internat. Collegiate Programming Contest" – az IBM szervezi 1977 óta,
2. „Google Code Jam" – a Google 2008 óta rendezi,
3. „Topcoder Open, Topcoder SRM" – a Topcoder szervezi, előbbit 2003 óta évente, utóbbit pedig kéthetente (egyfordulós versenyek).

A versenyekhez szükséges elméleti alapismeretek – amely témákhoz a versenyfeladatok kapcsolódnak – az alábbiak:

- gráfreprezentációk és alapvető gráfalgoritmusok, többek között:
  - legrövidebbút-keresés, párosítások, folyamok, többszörös, illetve erősen összefüggőség, színezések, BFS, DFS, DAG-ok,
- dinamikus programozás, backtracking módszerek,
- mintaillesztés, szövegkeresés problémák,
- geometriai problémák,
- kombinatorika, csoportelmélet, számelmélettel kapcsolatos algoritmikus problémák.

Ezen kívül fontos még, hogy a versenyzőkben kialakuljon az a készség, hogy felismerje, egy adott feladathoz milyen tanult eszközt, ismert algoritmust használjon fel. Ennek érdekében egy adott foglalkozás keretében általában a legváltozatosabb problémákat állítjuk össze, és a résztvevőknek azt is feladatuk kitalálni, hogy a feladat szövegezésében milyen ismert probléma bújik meg.

Elért eredmények

A szakkörre az utóbbi félévekben járó hallgatók a 2011/2012. őszi félévben az alábbi eredményeket érték el:

- **11. hely – ACM-ICPC CERC nemzetközi programozási verseny**
  Csapattagok: Eisenberger András, Lám István, Szebeni Szilveszter
- **26. hely – ACM-ICPC CERC nemzetközi programozási verseny**
  Csapattagok: Jámbor Attila, Mártonka Zoltán, Peregi Tamás
- **56. hely – ACM-ICPC CERC nemzetközi programozási verseny**
  Csapattagok: Berecz Dénes, Gönczi Tamás, Tringel Balázs

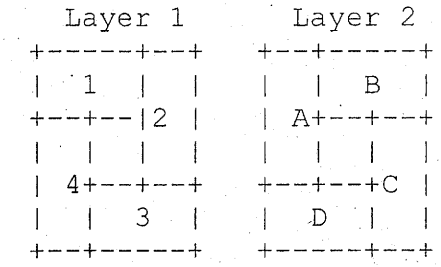
Függelék

A félév során megoldott feladatok

2011. 10. 21.

Problem Statement (Chimney)

A construction worker was assigned the task to build a chimney. The chimney consists of N layers of 4 bricks each. Each layer, when looked at from above, looks like this:



Each layer, except the bottommost one, is placed on top of the previous such that the borders coincide perfectly and each brick is placed on top of exactly two bricks from the layer below. In the example above, brick A is placed on top of bricks 1 and 4, brick B on top of bricks 1 and 2, C on top of 2 and 3 and D on top of 3 and 4.

A brick in a given layer can only be placed after both bricks below have already been placed. Of course, bricks on the bottommost layer have no such restrictions. There are many orders in which bricks can be placed. For instance, if only 1 layer is to be placed, there are 4! = 24 possible orders, because any order of the bricks is valid. However, when more layers are used, not every order of the total number of bricks is valid.

You will be given the total number of layers **n**. Return the number of orders in which bricks can be placed, modulo 1000000007.

Definition

Class:	Chimney
Method:	countWays
Parameters:	long
Returns:	int
Method signature:	int countWays(long n)
(be sure your method is public)	

Constraints

- **n** will be between 1 and 1000000000000000000 (10^18), inclusive.

Examples

0)

1

Returns: 24

There are 4! = 4*3*2*1 = 24 ways of placing 4 bricks without dependencies.

1)

2

Returns: 1088

2)

5

Returns: 110198784

3)

6

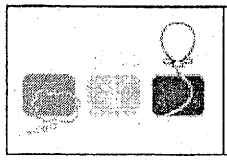
Returns: 138284509

4)

100000

Returns: 19900327

2011.10.21.



4947 - Hill Driving

Europe - Northwestern - 2010/2011

D Hill Driving

You're driving your car in the local hills and returning to your home town. You'd like to get back as quickly as possible; however, you notice that you don't have much fuel left. You know the most efficient route to take. Some parts of this route go downhill, and some go uphill. The different parts have different lengths and slopes. How quickly can you reach home with the little fuel you have left?

We will assume a very simple model for the fuel consumption of your car. Fuel consumption (per unit distance travelled) will increase linearly with your driving speed v. However, there is an offset which depends on the slope s of the hill. For example, when going downhill along a particular road, you might be able to go at 10 km/h without expending any fuel; on the other hand, if you were travelling that same road uphill, you would expend fuel at the same rate as if you were driving 10 km/h faster along a flat road. More specifically, the car's fuel consumption c in liters per kilometer is given by

$$c = \max(0, \alpha v + \beta s), \qquad (1)$$

where α is the standard fuel consumption rate on a flat road, v is your speed in km/h, s is the slope of the road, and β is a positive constant. Acceleration and deceleration do not cost fuel and can be done instantaneously. Note that your car has a maximum (safe) speed which cannot be exceeded.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- One line with four floating point numbers α ($0.1 \le \alpha \le 100$), β ($0.1 \le \beta \le 100$), v_{max} ($10 \le v_{max} \le 200$) and f ($0 \le f \le 50$): the standard (flat road) fuel consumption rate of your car, the slope factor, the maximum speed of your car in km/h, and the amount of fuel you have left in liters, respectively.
- One line with an integer r ($1 \le r \le 10\,000$): the number of road segments.
- r lines with two floating point numbers x_i and y_i ($1 \le x_i \le 1\,000$, $-1\,000 \le y \le 1\,000$) each: the horizontal distance and height change (both in meters) of the i-th road segment. Each road segment has constant slope.

Output

Per test case:

- One line with a floating point number: the fastest time in hours in which you can reach town. It is guaranteed that if it is possible to reach town at all, it will always be possible in less than 24 hours. If it is impossible to reach town, the line must contain "IMPOSSIBLE" instead.

Your output should have a relative or absolute error of at most 10^{-6}.

Sample in- and output

Input	Output
3 10.0 1.0 150 0.0	1.414214 IMPOSSIBLE

1 100.0 -100.0 10.0 100.0 150 1.0 2 100 0 100 100 0.5 0.1 100 10 3 1000 0 100 10 100 -10	0.072120

Northwestern 2010-2011

2011.11.04

5697 Number String

The signature of a permutation is a string that is computed as follows: for each pair of consecutive elements of the permutation, write down the letter 'I' (increasing) if the second element is greater than the first one, otherwise write down the letter 'D' (decreasing). For example, the signature of the permutation {3,1,2,7,4,6,5} is 'DIIDID'.

Your task is as follows: You are given a string describing the signature of many possible permutations, find out how many permutations satisfy this signature.

Note: For any positive integer n, a permutation of n elements is a sequence of length n that contains each of the integers 1 through n exactly once.

Input

Each test case consists of a string of 1 to 1000 characters long, containing only the letters 'I', 'D' or '?', representing a permutation signature.

Each test case occupies exactly one single line, without leading or trailing spaces.

Proceed to the end of file. The '?' in these strings can be either 'I' or 'D'.

Output

For each test case, print the number of permutations satisfying the signature on a single line. In case the result is too large, print the remainder modulo 1000000007.

Hint:
Permutation {1,2,3} has signature 'II'.
Permutations {1,3,2} and {2,3,1} have signature 'ID'.
Permutations {3,1,2} and {2,1,3} have signature 'DI'.
Permutation {3,2,1} has signature 'DD'.
'?D' can be either 'ID' or 'DD'.
'??' gives all possible permutations of length 3.

Sample Input

```
II
ID
DI
DD
?D
??
```

Sample Output

```
1
2
2
1
3
6
```

$T[i, x]$:= az első i számot kirakjuk

tudunk permutálni úgy, hogy az utolsó szám az i szám

úgy, hogy a növ. sor.-nak is eleget tesz

term. sorozatjába az x-edik

kirakjuk olyan i db elem permutációit

hogy az utolsó a term. sor. n. x-edik

és eleget tesz az első $i-1$ db betűnek

$$T[i,x] = \begin{cases} \sum_{j=1}^{x-1} [i-1, j), \text{ha } s[i-1] = \text{'A'} \\ \sum_{j=x+1}^{n} T[i-1, j], \text{ha } s[i-1] = \text{'B'} \end{cases} \quad \text{MOD } 1\,000\,007$$

a betű színe

(-1)

2011.11.04

Miller - Kobin

5701 The Boss on Mars

On Mars, there is a huge company called ACM (A huge Company on Mars), and it's owned by a younger boss.

Due to no moons around Mars, the employees can only get the salaries per-year. There are n employees in ACM, and it's time for them to get salaries from their boss. All employees are numbered from 1 to n. With the unknown reasons, if the employee's work number is k, he can get k^4 Mars dollars this year. So the employees working for the ACM are very rich.

Because the number of employees is so large that the boss of ACM must distribute too much money, he wants to fire the people whose work number is co-prime with n next year. Now the boss wants to know how much he will save after the dismissal.

Input

The first line contains an integer T indicating the number of test cases. ($1 \leq T \leq 1000$) Each test case, there is only one integer n, indicating the number of employees in ACM. ($1 \leq n \leq 10^8$)

Output

For each test case, output an integer indicating the money the boss can save. Because the answer is so large, please module the answer with 1,000,000,007.

Hint:
Case1: sum=1+3*3*3*3=82
Case2: sum=1+2*2*2*2+3*3*3*3+4*4*4*4=354

Sample Input

```
2
4
5
```

Sample Output

```
82
354
```

2011. 11. 04.

Problem Statement (CorrectMultiplication)

Krolik found a sheet of paper with an equation containing three positive integers: "**a** * **b** = **c**". This equation might be wrong, so Krolik is trying to make it correct by adjusting the numbers **a**, **b**, and **c**.

You are given three ints **a**, **b**, and **c**. Return a long containing the minimum value of |A - **a**| + |B - **b**| + |C - **c**|, where A, B, and C are *positive* integers satisfying A * B = C.

Definition

Class:	CorrectMultiplication
Method:	getMinimum
Parameters:	int, int, int
Returns:	long
Method signature:	long getMinimum(int a, int b, int c)
(be sure your method is public)	

Notes

-|x| is the absolute value of x (which is x for non-negative x and (-x) for negative x).

Constraints

-**a**, **b**, and **c** will each be between 1 and 1,000,000,000, inclusive.

Examples

0)

```
19

28

522
```

```
Returns: 2
```

By assigning A = 18, B = 29, and C = 522 the value of |A - **a**| + |B - **b**| + |C - **c**| is minimized.

1)

```
10

30

500
```

```
Returns: 11
```

2)

11111

11111

123454321

Returns: 0

3)
1000

100

10

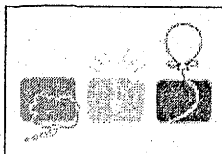
Returns: 1089
Note that A, B, and C must be positive integers.

4)
399

522

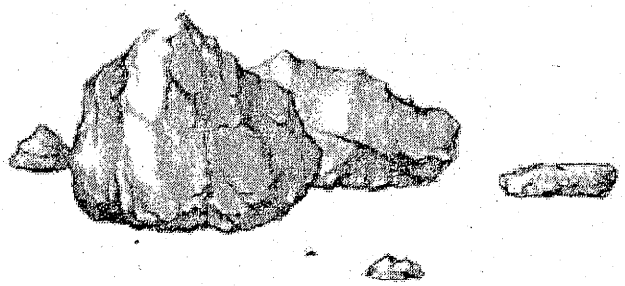
199999

Returns: 24



4290 - Easy Climb

Europe - Northwestern - 2008/2009



Somewhere in the neighborhood we have a very nice mountain that gives a splendid view over the surrounding area. There is one problem though: climbing this mountain is very difficult, because of rather large height differences. To make more people able to climb the mountain and enjoy the view, we would like to make the climb easier.

To do so, we will model the mountain as follows: the mountain consists of n adjacent stacks of stones, and each of the stacks is h_i high. The successive height differences are therefore h_{i+1}-h_i (for $1 \leq i \leq n$-1). We would like all absolute values of these height differences to be smaller than or equal to some number d.

We can do this by increasing or decreasing the height of some of the stacks. The first stack (the starting point) and the last stack (the ending point) should remain at the same height as they are initially. Since adding and removing stones requires a lot of effort, we would like to minimize the total number of added stones plus the total number of removed stones. What is this minimum number?

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with two integers n ($2 \leq n \leq 100$) and d ($0 \leq d \leq 10^9$): the number of stacks of stones and the maximum allowed height difference.
- One line with n integers h_i ($0 \leq h_i \leq 10^9$): the heights of the stacks.

Output

Per testcase:

- One line with the minimum number of stones that have to be added or removed or ``impossible'' if it is impossible to achieve the goal.

Sample Input

```
3
10 2
4 5 10 6 6 9 4 7 9 8
3 1
```

```
6 4 0
4 2
3 0 6 3
```

Sample Output

```
6
impossible
4
```

The 2008 ACM Northwestern European Programming Contest

Northwestern 2008-2009

4794 - Sharing Chocolate

World Finals - Harbin - 2009/2010

Chocolate in its many forms is enjoyed by millions of people around the world every day. It is a truly universal candy available in virtually every country around the world.

You find that the only thing better than eating chocolate is to share it with friends. Unfortunately your friends are very picky and have different appetites: some would like more and others less of the chocolate that you offer them. You have found it increasingly difficult to determine whether their demands can be met. It is time to writte a program that solves the problem once and for all!

Your chocolate comes as a rectangular bar. The bar consists of same-sized rectangular pieces. To share the chocolate you may break one bar into two pieces along a division between rows or columns of the bar. You or the may then repeatedly break the resulting pieces in the same manner. Each of your friends insists on a getting a single rectangular portion of the chocolate that has a specified number of pieces. You are a little bit insistent as well: you will break up your bar only if all of it can be distributed to your friends, with none left over.

For exampla, Figure 9 shows one way that a chocolate bar consisting of 3 x 4 pieces can be split into 4 parts that contain 6, 3, 2, and 1 pieces respectively, by breanking it 3 times (This corresponds to the first sample input.)

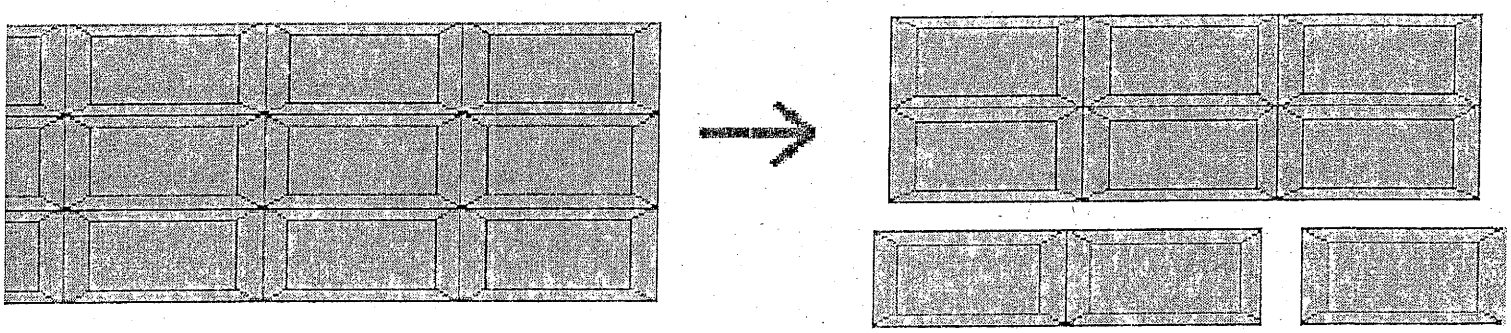


Figure 9

Input

The input consists of multiple test cases each describing a chocolate bar to share. Each description starts with a line containing a single integer n ($1 \leq n \leq 15$), the number of parts in which the bar is supposed to be split.

This is followed by a line containing two integers x and y ($1 \leq x, y \leq 100$), the dimensions of the chocolate bar. The next line contains n positive integers, giving the number of pieces that are supposed to be in each of the n parts.

The input is terminated by a line containing the integer zero.

Output

For each test case, first display its case number. Then display whether it is possible to break the chocolate in the desired way: display ``Yes" if it is possible, and ``No" otherwise. Follow the format of the sample output.

Sample Input

```
4
3 4
6 3 2 1
2
2 3
1 5
0
```

Sample Output

```
Case 1: Yes
Case 2: No
```

Harbin 2009-2010

Problem Statement (PerfectPowers)

A number is called a perfect power if it can be written in the form m^k, where m and k are positive integers, and k > 1.

Given two positive integers **A** and **B**, find the two perfect powers between **A** and **B**, inclusive, that are closest to each other, and return the absolute difference between them. If less than two perfect powers exist in the interval, return -1 instead.

Definition

Class:	PerfectPowers
Method:	nearestCouple
Parameters:	long, long
Returns:	long

Method signature: long nearestCouple(long A, long B)
(be sure your method is public)

Notes

- 1 is a perfect power.

Constraints

- **A** will be between 1 and 10^18, inclusive.-**B** will be between **A**+1 and 10^18, inclusive.

Examples

0)

```
1
4
```

```
Returns: 3
```

1 and 4 are the first pair of perfect powers.

1)

```
8
9
```

```
Returns: 1
```

8 and 9 are the closest pair of perfect powers.

2)

```
10
15
```

```
Returns: -1
```

No pair of perfect powers is present in the interval.

3)

```
1
1000000000000000000
```

```
Returns: 1
```

This is the largest possible range, and 8 and 9 are the closest pair of perfect powers.

4)

```
80000
90000
```

```
Returns: 80
```

Problem Statement (PointErasing)

You are given a int[] **y** of length N. The values in **y** describe N points in the plane: for each x = 0, 1, ..., N - 1 there is a point at coordinates (x, **y**[x]).

Krolik is going to perform the following operation repeatedly:

1. Choose two of the remaining (i.e. not yet erased) points, say A and B, with different y-coordinates.
2. Consider the rectangle with sides parallel to coordinate axes and points A and B in two opposite corners.
3. Erase all points strictly inside the rectangle.

In each iteration, Krolik must choose A and B in such a way that at least one point is erased. If no more operations can be performed, Krolik will count the number of remaining points and write it into a notebook. Return a int[] containing all possible values of the number Krolik can write down, in ascending order.

Notes

- A point is *strictly inside* a rectangle if it is inside the rectangle and does not lie on the border of the rectangle.

Constraints

- **y** will contain between 1 and 50 elements, inclusive.
- Each element of **y** will be between 0 and 1,000,000,000, inclusive.

Examples

0)

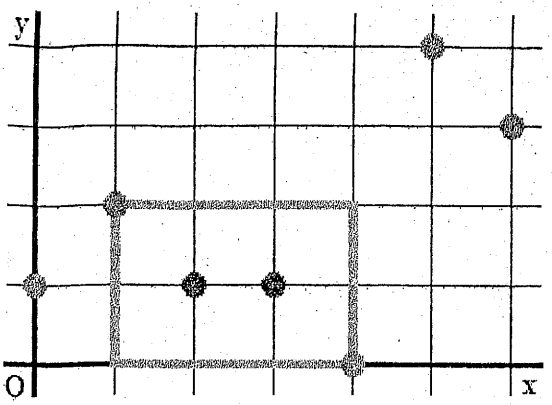
```
{ 1, 2, 1, 1, 0, 4, 3 }
```

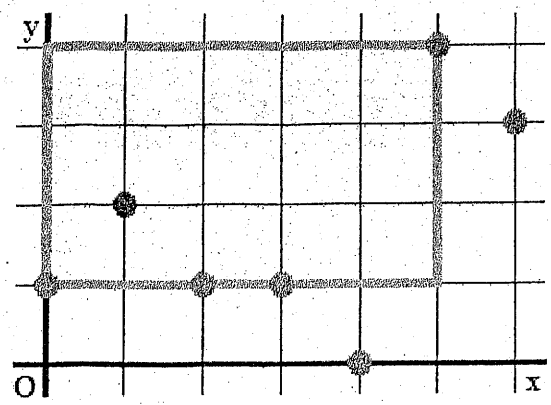
```
Returns: {4, 6 }
```

Krolik will end with 4 points if Krolik takes the following steps:

- Choose (1, 2) and (4, 0) to erase (2, 1) and (3, 1).
- Choose (0, 1) and (5, 4) to erase (1, 2).

If Krolik first chooses (0, 1) and (5, 4) to erase (1, 2), 6 points remain and there are no valid moves left.





1)
{ 0, 0, 4, 4, 8, 8, 4, 4 }

Returns: {6 }

2)
{ 522 }

Returns: {1 }

3)
{ 19, 19, 19, 19, 19, 19 }

Returns: {6 }

4)
{ 0, 1, 2, 3, 4 }

Returns: {2 }

5)
{ 7, 8, 8, 7, 6, 7, 9, 3, 5, 0 }

Returns: {3, 4, 5 }

6)
{ 3, 2, 3, 3, 4, 3, 4, 3, 3, 1, 5, 3 }

Returns: {4, 5, 6, 7, 9 }

7)
{ 5, 5, 4, 4, 5, 8, 5, 5, 5, 5, 5, 9, 2, 0, 9, 4, 5, 5, 3, 4, 4, 4, 4, 7, 4 }

Returns: {6, 7, 8, 10, 11, 12, 13, 15, 16, 17 }

A. Qin Shi Huang's National Road System

Description

During the Warring States Period of ancient China(476 BC to 221 BC), there were seven kingdoms in China ---- they were Qi, Chu, Yan, Han, Zhao, Wei and Qin. Ying Zheng was the king of the kingdom Qin. Through 9 years of wars, he finally conquered all six other kingdoms and became the first emperor of a unified China in 221 BC. That was Qin dynasty --- the first imperial dynasty of China(not to be confused with the Qing Dynasty, the last dynasty of China). So Ying Zheng named himself "Qin Shi Huang" because "Shi Huang" means "the first emperor" in Chinese.



Qin Shi Huang undertook gigantic projects, including the first version of the Great Wall of China, the now famous city-sized mausoleum guarded by a life-sized Terracotta Army, and a massive national road system. There is a story about the road system:

There were n cities in China and Qin Shi Huang wanted them all be connected by n-1 roads, in order that he could go to every city from the capital city Xianyang. Although Qin Shi Huang was a tyrant, he wanted the total length of all roads to be minimum,so that the road system may not cost too many people's life. A daoshi (some kind of monk) named Xu Fu told Qin Shi Huang that he could build a road by magic and that magic road would cost no money and no labor. But Xu Fu could only build ONE magic road for Qin Shi Huang. So Qin Shi Huang had to decide where to build the magic road. Qin Shi Huang wanted the total length of all none magic roads to be as small as possible, but Xu Fu wanted the magic road to benefit as many people as possible ---- So Qin Shi Huang decided that the value of A/B (the ratio of A to B) must be the maximum, which A is the total population of the two cites connected by the magic road, and B is the total length of none magic roads.

Would you help Qin Shi Huang?

A city can be considered as a point, and a road can be considered as a line segment connecting two points.

Input

The first line contains an integer t meaning that there are t test cases(t <= 10).

For each test case:

The first line is an integer n meaning that there are n cities(2 < n <= 1000).

Then n lines follow. Each line contains three integers X, Y and P (0 <= X, Y <= 1000, 0 < P < 100000). (X, Y) is the coordinate of a city and P is the population of that city.

It is guaranteed that each city has a distinct location.

Output

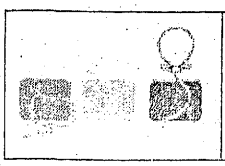
For each test case, print a line indicating the above mentioned maximum ratio A/B. The result should be rounded to 2 digits after decimal point.

Sample Input

```
2
4
1 1 20
1 2 30
200 2 80
200 1 100
3
1 1 20
1 2 30
2 2 40
```

Sample Output

```
65.00
70.00
```



5031 - Graph and Queries

Asia - Tianjin - 2010/2011

You are given an undirected graph with N vertexes and M edges. Every vertex in this graph has an integer value assigned to it at the beginning. You're also given a sequence of operations and you need to process them as requested. Here's a list of the possible operations that you might encounter:

1. Deletes an edge from the graph.

   The format is [D X], where X is an integer from 1 to M, indicating the ID of the edge that you should delete. It is guaranteed that no edge will be deleted more than once.
2. Queries the weight of the vertex with K-th maximum value among all vertexes currently connected with vertex X (including X itself).

   The format is [Q X K], where X is an integer from 1 to N, indicating the id of the vertex, and you may assume that K will always fit into a 32-bit signed integer. In case K is illegal, the value for that query will be considered as undefined, and you should return 0 as the answer to that query.
3. Changes the weight of a vertex.

   The format is [C X V], where X is an integer from 1 to N, and V is an integer within the range [-10^6, 10^6].

The operations end with one single character, E, which indicates that the current case has ended. For simplicity, you only need to output one real number - the average answer of all queries.

Input

There are multiple test cases in the input file. Each case starts with two integers N and M ($1 \leq N \leq 2 * 10^4$, $0 \leq M \leq 6 * 10^4$), the number of vertexes in the graph. The next N lines describes the initial weight of each vertex ($-10^6 \leq [weight][i] \leq 10^6$). The next part of each test case describes the edges in the graph at the beginning. Vertexes are numbered from 1 to N. The last part of each test case describes the operations to be performed on the graph. It is guaranteed that the number of query operations [Q X K] in each case will be in the range [1, $2 * 10^5$], and there will be no more than $2 * 10^5$ operations that change the values of the vertexes [C X V].

There will be a blank line between two successive cases. A case with $N = 0$, $M = 0$ indicates the end of the input file and this case should not be processed by your program.

Output

For each test case, output one real number - the average answer of all queries, in the format as indicated in the sample output. Please note that the result is rounded to six decimal places.

Explanation for samples:

For the first sample:

D 3 - deletes the 3rd edge in the graph (the remaining edges are (1, 2) and (2, 3))

Q 1 2 - finds the vertex with the second largest value among all vertexes connected with 1. The answer is 20.

Q 2 1 - finds the vertex with the largest value among all vertexes connected with 2. The answer is 30.

D 2 - deletes the 2nd edge in the graph (the only edge left after this operation is (1, 2))

Q 3 2 - finds the vertex with the second largest value among all vertexes connected with 3. The answer is 0 (Undefined).

C 1 50 - changes the value of vertex 1 to 50.

Q 1 1 - finds the vertex with the largest value among all vertex connected with 1. The answer is 50.

E - This is the end of the current test case. Four queries have been evaluated, and the answer to this case is (20 + 30 + 0 + 50) / 4 = 25.000.

For the second sample, caution about the vertex with same weight:

Q 1 1 - the answer is 20

Q 1 2 - the answer is 20

Q 1 3 - the answer is 10

Sample Input

```
3 3
10
20
30
1 2
2 3
1 3
D 3
Q 1 2
Q 2 1
D 2
Q 3 2
C 1 50
Q 1 1
E

3 3
10
20
20
1 2
2 3
1 3
Q 1 1
Q 1 2
Q 1 3
E
0 0
```

Sample Output

```
Case 1: 25.000000
Case 2: 16.666667
```

Tianjin 2010-2011

Problem Statement (SRMChallengePhase)

Mr. Dengklek introduced you to an online programming contest called SRM (Special Round Match)!

You are now in the challenge phase of the contest. There are N coders in the contest, conveniently numbered 0 through N-1. In this phase, each coder is given an opportunity to make challenge attempts against another coders. Each challenge attempt can be successful or unsuccessful. In case a challenge attempt against some coder X is successful, it is said that coder X is successfully challenged. A coder may not attempt a challenge against him/herself.

After the challenge phase is over, you remember this information:

- The information about challenge attempts is given in String[] **codersAttempted**. Concatenate all its elements to obtain a single string. This string will contain N characters. If the i-th (0-indexed) character is 'Y', the i-th coder attempted exactly one challenge (which could be successful or unsuccessful). Otherwise, the i-th coder didn't attempt any challenges. No coders attempted two or more challenges.
- The information about successful challenges is given in String[] **codersChallenged**. Concatenate all its elements to obtain a single string. This string will contain N characters. If the i-th (0-indexed) character is 'Y', the i-th coder was successfully challenged exactly once. Otherwise, the i-th coder was never successfully challenged. No coder was successfully challenged two or more times.
- As soon as a coder had been succesfully challenged, the coder became upset and stopped making any challenge attempts.
- Each coder could be unsuccessfully challenged arbitrary many times. Some of the unsuccessful challenges against a coder could be made after the coder had been successfully challenged.

There were no two simultaneous challenge attempts during the challenge phase, so it can be represented as a chronologically ordered sequence of challenge attempts. Unfortunately, you forgot how exactly the challenge phase happened. Return the number of different possible sequences of challenge attempts that are consistent with the information that you remember, modulo 1,000,000,007. The information that you remember may be inaccurate, so if there are no such sequences, return 0. See examples for additional clarification.

Constraints

- **codersAttempted** and **codersChallenged** will each contain between 1 and 50 elements, inclusive.
- Each element of **codersAttempted** and **codersChallenged** will contain between 1 and 50 characters, inclusive.
- The total number of characters in **codersAttempted** and **codersChallenged** will be the same.
- Each character in both **codersAttempted** and **codersChallenged** will be 'Y' or 'N'.

Examples

0)

```
{"YY"}
{"NY"}
```

Returns: 1

The only valid sequence of challenge attempts is:

- Coder 1 unsuccessfully challenged coder 0, then coder 0 sucessfully challenged coder 1.

1)

```
{"YY", "NN"}
{"N", "NYY"}
```

Returns: 4

Here are the four possible sequences:

- Coder 0 successfully challenged coder 2, then coder 1 successfully challenged coder 3.
- Coder 0 successfully challenged coder 3, then coder 1 successfully challenged coder 2.
- Coder 1 successfully challenged coder 2, then coder 0 successfully challenged coder 3.
- Coder 1 successfully challenged coder 3, then coder 0 successfully challenged coder 2.

2)

```
{"YNNN"}
{"NYY", "Y"}
```

Returns: 0

The number of coders that were successfully challenged is greater than the number of attempts. It is impossible.

3)

```
{"N"}
{"N"}
```

Returns: 1

4)

```
{"YYY"}
{"NNY"}
```

Returns: 24

5)

```
{"YY", "N", "YYYY", "NN", "YYYYY"}
{"N", "YYYYY", "N", "Y", "N", "YYYY", "N"}
```

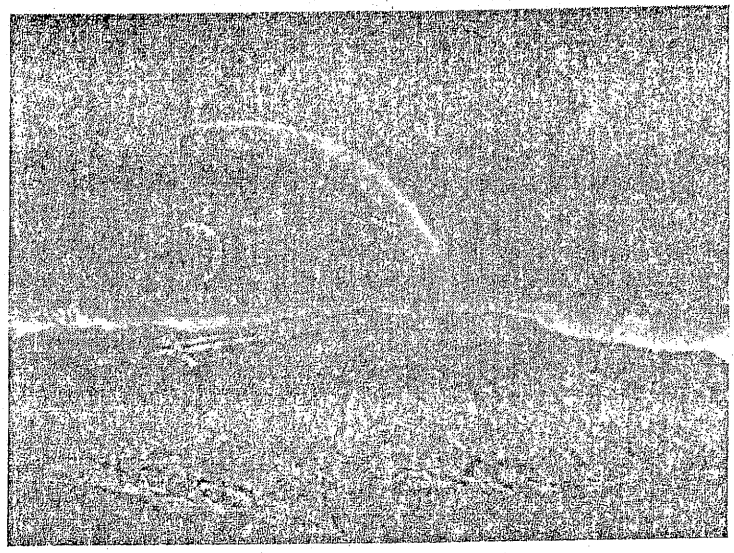
Returns: 807026001

4855 - Hyper Box

Asia - Dhaka - 2010/2011

You live in the universe X where all the physical laws and constants are different from ours. For example all of their objects are N-dimensional. The living beings of the universe X want to build an N-dimensional monument. We can consider this N dimensional monument as an N-dimensional hyper-box, which can be divided into some N dimensional hypercells. The length of each of the sides of a hyper-cell is one. They will use some N-dimensional bricks (or hyper-bricks) to build this monument. But the length of each of the N sides of a brick cannot be anything other than fibonacci numbers. A fibonacci sequence is given below:

1, 2, 3, 5, 8, 13, 21...



As you can see each value starting from 3 is the sum of previous 2 values. So for $N = 3$ they can use bricks of sizes (2,5,3), (5,2,2) etc. but they cannot use bricks of size (1,2,4) because the length 4 is not a fibonacci number. Now given the length of each of the dimension of the monument determine the minimum number of hyper-bricks required to build the monument. No two hyper-bricks should intersect with each other or should not go out of the hyper-box region of the monument. Also none of the hyper-cells of the monument should be empty.

Input

First line of the input file is an integer T ($1 \leq T \leq 100$) which denotes the number of test cases. Each test case starts with a line containing N ($1 \leq N \leq 15$) that denotes the dimension of the monument and the bricks. Next line contains N integers the length in each dimension. Each of these integers will be between 1 and 2000000000 inclusive.

Output

For each test case output contains a line in the format `Case x: M` where x is the case number (starting from 1) and M is the minimum number of hyper-bricks required to build the monument.

Sample Input

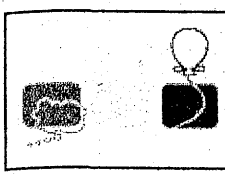
```
2
2
4 4
3
5 7 8
```

Sample Output

```
Case 1: 4
Case 2: 2
```

Dhaka 2010-2011

Problemsetter: Abdullah al Mahmud
Special Thanks: Md. Arifuzzaman Arif



4812 - Hyperactive Girl

Latin America - South America - 2010/2011

Helen is a hyperactive girl. She wants to schedule her activities so that at any moment of the day there is at least one thing she can do. She does not care if her activities overlap in time, as long as every moment of her day has an activity scheduled.

Helen divided the day in a particular way. The day starts at time 0 and finishes at time M. Each moment of the day is represented by a real number between 0 and M, inclusive. Helen made a list of all possible activities, with their start and finish times. Now she must decide which subset of activities to schedule.

If an activity starts at time S and finishes at time F, then we say that it covers all moments between S and F, inclusive. Helen does not want to waste any activities, so she will only choose minimal subsets of activities that cover the day to be scheduled. A subset of activities is a minimal subset that covers the day if and only if:

- every moment of the day is covered by at least one activity of the subset; and
- removing any of the activities from the subset would leave at least one moment of the day uncovered.

Note that some moments of the day may be covered by more than one activity.

Given the list of possible activities for one day, you must help Helen by determining how many distinct minimal subsets cover the day.

Input

Each test case is given using several lines. The first line contains two integers M and N, representing respectively the highest value for a moment in the day ($1 \leq M \leq 10^9$) and the number of possible activities for the day ($1 \leq N \leq 100$). Each of the next N lines describes one possible activity and contains two integers S and F, representing respectively the start and finish times of the activity ($0 \leq S < F \leq M$).

The last test case is followed by a line containing two zeros.

Output

For each test case output a single line with a single integer representing the number of minimal subsets that cover the day. To make your life easier, print the remainder of dividing the solution by 10^8.

Sample Input

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

```
4 6
0 2
1 1
0 1
2 1
0 1
0 0
```

Sample Output

```
4
1
0
```

South America 2010-2011

4809 - Electric Needs

Latin America - South America - 2010/2011

You are going to construct a new factory in your city. Since you have major electric needs, having the factory placed close to a power plant is important. You want to build a prioritized list of possible locations.

The area in which the factory needs to be located can be represented as a rectangular grid of N rows and M columns of cells. Some of those cells contain a power plant. The new factory occupies exactly one cell, and can be placed in any empty cell (i.e., any cell that does not contain a power plant).

Numbering rows from 1 to N and columns from 1 to M, the location of a cell can be described by two integers. Cell (i, j) is the cell in row i and column j. The distance between cell (i_0, j_0) and cell (i_1, j_1) is max $(|\, i_0 - i_1|, |\, j_0 - j_1|)$ where $|\, x|$ represents the absolute value of x. The electric priority of a location is its minimum distance to a power plant.

With this in mind, you will number all possible locations with consecutive integers starting from 1. You will do it in ascending order of electric priority. Among locations with the same electric priority, you will number them in ascending order of their row numbers. Among locations with the same electric priority and row number, you will list them in ascending order of column numbers.

In the following picture you can see a 4 x 7 grid. Black cells are the cells in which there is a power plant. Dark gray cells have an electric priority of 1, light gray cells an electric priority of 2 and white cells an electric priority of 3. The number inside each cell is the number assigned by you to the location.

22	23	12	1	2	3	13
24	14	15	4		5	16
25	17	6	7	8	9	18
26	19	10		11	20	21

You will receive several queries about the prioritized list built. In each query you will be given a number representing a position in the prioritized list and you have to calculate which location was assigned the given position.

Input

Each test case is given using several lines. The first line contains three integers N, M and P, representing the number of rows and columns of the grid ($1 \leq N, M \leq 10^9$) and the number of existing power plants ($1 \leq P \leq 20$). Each of the next P lines contains two integers R and C representing the row and column numbers of the location of a power plant ($1 \leq R \leq N$ and $1 \leq C \leq M$). Within each test case, all power plant locations are different. The next line contains a single integer Q representing the number of queries ($1 \leq Q \leq 50$). Then follows a line with Q integers $p_1, \ldots, p_Q$ representing positions in the prioritized list

$(1 \leq p_i \leq N \times M - P)$.

The last test case is followed by a line containing three zeros.

Output

For each test case output $Q + 1$ lines. Line i of the first Q lines must contain two integers representing the row and column numbers of the location that was assigned number p_i. The last line for each test case must contain a single character `-' (hyphen).

Sample Input

```
4 7 2
2 5
4 4
6
1 6 11 16 21 26
1000000000 1000000000 1
1 1
1
999999999999999999
0 0 0
```

Sample Output

```
1 4
3 3
4 5
2 7
4 7
4 1
-
1000000000 1000000000
-
```

South America 2010-2011

Problem 1: Alien Game

Aliens on the Unknown planet have a tradition of playing a game called Loiten. It is played by two players who alternate turns. There are **N** buckets with apples standing in one line in front of the players. They are numbered from left to right with integers starting from 1.

In one turn a player can select one of the buckets, which is not the first and not the last and has a positive number of apples in it, and move all of that bucket's apples to the bucket adjacent to the left and at the same time move all of them to the bucket adjacent to the right. That's right, the number of apples can be negative as it is a really strange planet. Thus, if there are 3 consecutive buckets with the number of apples being **x**, **y**, **z**, then you can perform the move if **y** is greater than zero. The resulting capacity of the buckets will be as follows: **x+y**, **-y**, **z+y**. The first player who can't make a move loses.

You happen to know one of the aliens from the Unknown planet, named Popo. He is a very good Loiten player, and has reached the Loiten Finals. On the day prior to the game, he found out the number of apples in each of the buckets. Unfortunately, his memory is not that good, and he can't remember the number of apples in the **P**-th bucket. He just remembers that it is a number with absolute value not greater than **F**.

Now, he is asking you to help him to calculate his chances. The players at the Finals are so good that they only make optimal moves to maximize their chance of winning. If neither player can win, the game is considered a draw. You are to find the number of possible apple counts for the bucket with an unknown number of apples where Popo will win. Popo is also sure that he is the one to make the first turn.

Input

The first line of the input file consists of a single number **T**, the number of test cases. Each test case begins with a line containing two integers **N**, the number of buckets and **P**, the number of the bucket with the unknown amount of apples. It is followed by a line containing **N** integers, the numbers of apples in the corresponding buckets. The $\mathbf{P}^{th}$ number on this line is the positive integer **F** and corresponds to the constraint on the number of apples in the **P**-th bucket.

Output

Output **T** lines, with the answer to each test case on a single line, the number of possible values for unknown bucket.

Constraints

$\mathbf{T} = 50$

$1 \leq \mathbf{P} \leq \mathbf{N} \leq 2{,}000.$

$1 \leq \mathbf{F} \leq 1{,}000{,}000{,}000{,}000.$

The number of apples in each bucket at the start of the game has an absolute value not greater than 1,000,000,000,000.

4815 - Kids' Wishes

Latin America - South America - 2010/2011

Kevin is a kid. He has lunch at school together with many more kids. They use to go outdoors and have lunch sitting on the ground. They love to form a big circle in which each kid has exactly two neighbors, one on the left and one on the right. Sometimes the teacher has problems arranging the circle because some kids wish to sit down next to other kids. Each kid may wish to sit down next to at most two other kids, because each kid has just two neighbors in the circle. The teacher wants to know whether it is possible to arrange the circle in such a way that all kids' wishes are satisfied. You clean up the place when the lunch ends. Since you want to finish your work as early as possible, help the teacher in answering that question.

Input

Each test case is given using several lines. The first line contains two integers K and W representing respectively the number of kids ($3 \leq K \leq 10^9$) and the number of wishes ($0 \leq W \leq 10^5$). Kids are identified with numbers between 1 and K. Each of the next W lines describes a different wish using two distinct integers A and B ($1 \leq A, B \leq K$); these values represent that kid A wishes to sit down next to kid B. Each kid has at most two wishes.

The last test case is followed by a line containing two zeros.

Output

For each test case output a single line containing an uppercase `Y' if it is possible to arrange a circle in such a way that all kids' wishes are satisfied, or an uppercase `N' otherwise.

Sample Input

```
4 3
2 3
1 3
2 1
1000000000 0
3 6
3 2
2 1
1 2
1 3
2 3
3 1
0 0
```

Sample Output

```
N
Y
Y
```

Problem 2: Safest Place

While en route to the 295th annual Galactic Dance Party on Risa, you find yourself unceremoniously yanked out of hyperspace and, according to your sensors, surrounded by **N** space bombs. Apparently caught in a trap laid out by some dastardly and unknown enemy, and unable to return to hyperspace, you must find the safest place in the vicinity to weather the detonation of all the space bombs. Your unseen opponent has constructed a cube-shaped space anomaly that you are unable to leave, so your options are limited to points within that cube.

Before the bombs explode (all simultaneously), you have just enough time to travel to any integer point in the cube [0, 0, 0]-[1000, 1000, 1000], both inclusive. You must find the point with the maximum distance to the nearest bomb, which your captain's intuition tells you will be the safest point.

Input

The first line of the input file consists of a single number **T**, the number of test cases. Each test consists of single number **N**, the number of bombs, followed by **3*N** integers describing the positions of the bombs.

Output

Output **T** integers, one per test case each on its own line, representing the square of distance to the nearest bomb from the safest point in the cube.

Constraints

T = 50

1 <= **N** <= 200

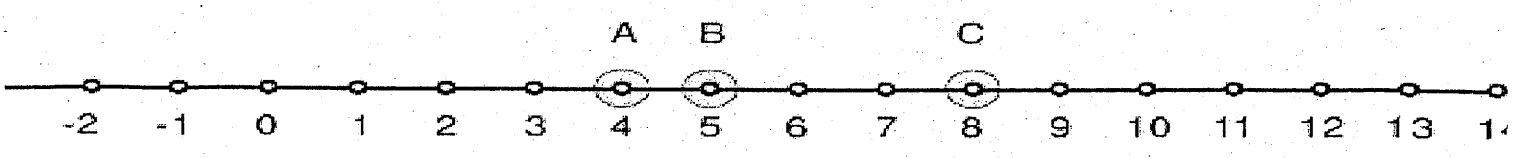
All bombs coordinates will be in [0, 1000], both inclusive.

4813 - Ingenious Metro

Latin America - South America - 2010/2011

The King of Logonia will inaugurate soon a new and revolutionary metro, based on an invention of the Royal Engineers, which allows teletransportation.

The new metro consists of a very long tunnel with a station at each kilometer. There are also T teletransporters, which are located at some of the stations. In each station there is a keyboard with T keys, where each key corresponds to one teletransporter. The figure below illustrates a metro system with three teletransporters, located in stations marked A, B and C.



The metro works as follows. The user goes in a station (the start station) and presses the key corresponding to the teletransporter he wants to use. The user is then teletransported to the station which is at the same distance from the teletransporter as the start station, but on the opposite side relative to the teletransporter. More precisely, if the location of the start station is i and the user presses the key corresponding to the teletransporter located in position j, he will be taken to the station located at position $2 \times j - i$. For example, if the user is in station 6 and wants to go to station -2, he can use the teletransporter C (goes from 6 to 10) and then the teletransporter A (goes from 10 to -2).

The King, however, knows that it is possible that there is no sequence of teletransporters that will take the user from a given station X to a given station Y. To avoid that the users keep trying to go where they cannot go, he wants to make a program available in the Internet to help users. The King wants you to write a program which, given the position of each teletransporter, answers a series of queries. For each query the start and the destination stations are given, and your program must determine if it is possible for the user to go from start to destination.

Input

Each test case is given using several lines. The first line contains two integers T and Q indicating respectively the number of teletransporters ($1 \leq T \leq 10^5$) and the number of queries ($1 \leq Q \leq 10$). The second line contains T different integers t_i indicating the position of the teletransporters ($-10^7 \leq t_i \leq 10^7$). Each of the Q following lines describes a query and contains two distinct integers S and D indicating the position of the start and destination stations ($-10^7 \leq S, D \leq 10^7$).

The last test case is followed by a line containing two zeros.

Output

For each test case output a single line containing the answers to the Q queries, in the same order that the queries were given in the input. For each query you must output an uppercase `Y' if it is possible to reach the destination station from the start station using the metro, or an uppercase `N' otherwise.

Sample Input

```
1 1
-2
-6 2
5 2
10 20 30 40 50
10 15
20 40
5 3
0 5 -3 -8 4
-1 499
4 237
-1 -591
0 0
```

Sample Output

```
Y
N Y
Y N Y
```

South America 2010-2011

4953 - Wormly

Europe - Northwestern - 2010/2011

J Wormly

Jonly is writing his first computer game. For the opening scene he wants to have the main character, Wormly, cross Bridgely, the bridge. Wormly is a worm made of b equal circular bubbles and L legs. At all times each leg has to be under one of the bubbles, and under each bubble there can be at most one leg. Bridgely was supposed to be composed of n planks with the width of each plank equal to the diameter of each of Wormly's bubbles. However, some of the planks are missing.

At every moment, Wormly can do exactly one of the following:

- Move one of its legs forward over any number of (possibly missing) planks. After the move, the leg should be on a plank and underneath one of Wormly's bubbles. A leg isn't allowed to overtake other legs.
- Move all of its bubbles forward one plank while its legs remain on the same planks. After the move each leg must still be under one of Wormly's bubbles.

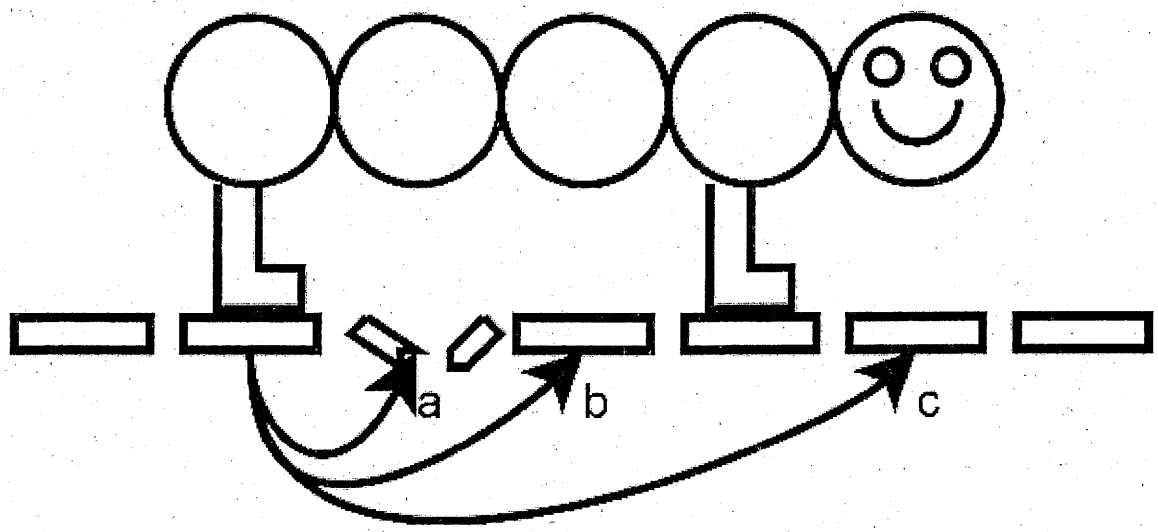


Figure 1: In this example, the only possible move for the last leg is to position b. (The plank at position a is missing, so the leg cannot move there. To get to position c, the last leg would have to overtake the first leg.) Also, in this example, moving all the bubbles forward is not allowed because Wormly's last leg would end up without a bubble over it.

Now Jonly is wondering how long the animation takes until Wormly reaches the end of Bridgely. Initially Wormly's bubbles are directly above the leftmost b planks of the bridge and its legs are on the leftmost L planks. At the end of the animation Wormly's bubbles have to be directly above the rightmost b planks and its legs have to be on the rightmost L planks.

The left- and rightmost L planks of Bridgely are not missing.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- One line with three integers L, b and n ($1 \leq L \leq b \leq n \leq 1\,000\,000$): the number of legs, the number of bubbles and the length of the bridge respectively.
- One line with a string consisting of n characters, either `0' or `1', describing Bridgely. A one indicates a plank and a zero indicates a missing plank.

Output

Per test case:

- One line with an integer: the minimum number of steps it takes Wormly to cross Bridgely. If it is impossible to get across while satisfying the constraints, the line must contain "IMPOSSIBLE" instead.

Sample in- and output

Input	Output
3 1 2 2 11 2 3 5 11011 1 3 5 11011	1 IMPOSSIBLE 5

Northwestern 2010-2011



4379 - iSharp

Europe - Northeastern Europe - 2008/2009

You are developing a new fashionable language that is not quite unlike C, C++, and Java. Since your language should become an object of art and fashion, you call it *i#* (spelled i-sharp). This name combines all the modern naming trends and also hints at how smart you are. Your language caters for a wide auditory of programmers and its type system includes arrays (denoted with "`[]`"), references (denoted with "`&`"), and pointers (denoted with "`*`"). Those type constructors can be freely combined in any order, so that a pointer to an array of references of references of integers (denoted with "`int&&[]*`") is a valid type. Multiple variables in *i#* can be declared on a single line with a very convenient syntax where common type of variables is given first, followed by a list of variables, each optionally followed by additional variable-specific type constructors. For example, the following line:

```
int& a*[]&, b, c*;
```

declares variables *a* , *b* , and *c* with types "`int&&[]*`", "`int&`", and "`int&*`" correspondingly. Note, that type constructors on the right-hand sides of variables in *i#* bind to variable and their order is reversed when they are moved to the left-hand side next to type. Thus "`int*& a`" is equivalent to "`int a&*`". However, you discover that coding style with multiple variable declarations per line is confusing and is outlawed in many corporate coding standards. You decide to get rid of it and refactor all existing *i#* code to a single variable declaration per line and always specify type constructor next to the type it refers to (instead of the right-hand side of variable). Your task it to write such refactoring tool.

Input

The input file contains several test cases, each of them as described below. The input contains a single line with a declaration of multiple variables in *i#* . The line starts with a type name, followed by zero, one, or more type constructors, followed by a space, followed by one or more variable descriptors separated by "`, `" (comma) and space, and terminated by "`;`" (semicolon). Each variable descriptor contains variable name, followed by zero, one, or more type constructors. Type name and variable names are distinct and consist of lowercase and uppercase English letters from "`a`" to "`z`" or "`A`" to "`Z`". The line contains at most 120 characters and does not contain any extra spaces.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

Write to the output a line for each variable declared in the input. For each variable write its declaration on a single line in the same format as in the input, but with all type constructors next to its type. Separate type with all type constructors from a variable name by a single space. Do not write any extra spaces.

Sample Input

```
int& a*[]&, b, c*;
Double[][] Array[];
```

Sample Output

```
int&&[]* a;
int& b;
int&* c;

Double[][][] Array;
```

Northeastern Europe 2008-2009

3392 - Triangular Sums

North America - Greater New York - 2006/2007

The n-th Triangular number, $T(n) = 1 + ... + n$, is the sum of the first n integers. It is the number of points in a triangular array with n points on side. For example $T(4)$:

```
X

X X

X X X

X X X X
```

Write a program to compute the weighted sum of triangular numbers:

$$W(n) = SUM[k = 1..n; k * T(k + 1)]$$

Input

The first line of input contains a single integer N, ($1 \leq N \leq 1000$) which is the number of datasets that follow.

Each dataset consists of a single line of input containing a single integer n, ($1 \leq n \leq 300$), which is the number of points on a side of the triangle.

Output

For each dataset, output on a single line the dataset number, (1 through N), a blank, the value of n for the dataset, a blank, and the weighted sum, $W(n)$, of triangular numbers for n.

Sample Input

```
4
3
4
5
10
```

Sample Output

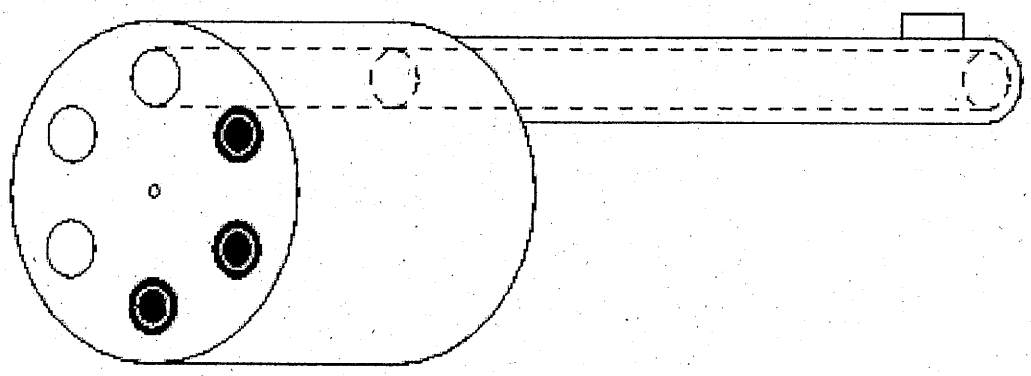
```
1 3 45
2 4 105
3 5 210
4 10 2145
```

Greater New York 2006-2007

4596 - Headshot

Europe - Northeastern Europe - 2009/2010

You have a revolver gun with a cylinder that has n chambers. Chambers are located in a circle on a cylinder. Each chamber can be empty or can contain a round. One chamber is aligned with the gun's barrel. When trigger of the gun is pulled, the gun's cylinder rotates, aligning the next chamber with the barrel, hammer strikes the round, making a shot by firing a bullet through the barrel. If the chamber is empty when the hammer strikes it, then there is no shot but just a ``click".



You have found a use for this gun. You are playing Russian Roulette with your friend. Your friend loads rounds into some chambers, randomly rotates the cylinder, aligning a random chamber with a gun's barrel, puts the gun to his head and pulls the trigger. You hear ``click" and nothing else -- the chamber was empty and the gun did not shoot.

Now it is your turn to put the gun to your head and pull the trigger. You have a choice. You can either pull the trigger right away or you can randomly rotate the gun's cylinder and then pull the trigger. What should you choose to maximize the chances of your survival?

Input

The input file contains several datasets. A dataset contains a single line with a string of n digits ``0" and ``1" ($2 \leq n \leq 100$). This line of digits represents the pattern of rounds that were loaded into the gun's chambers.

``0" represent an empty chamber, ``1" represent a loaded one. In this representation, when cylinder rotates before a shot, the next chamber to the right gets aligned with the barrel for a shot. Since the chambers are actually located on a circle, the first chamber in this string follows the last one. There is at least one ``0" in this string.

Output

For each dataset, write to the output file one of the following words (without quotes):

- ``SHOOT" -- if pulling the trigger right away makes you less likely to be actually shot in the head with the bullet (more likely that the chamber will be empty).
- ``ROTATE" -- if randomly rotating the cylinder before pulling the trigger makes you less likely to be actually shot in the head with the bullet (more likely that the chamber will be empty).

- ``EQUAL" -- if both of the above choices are equal in terms of probability of being shot.

Sample Input

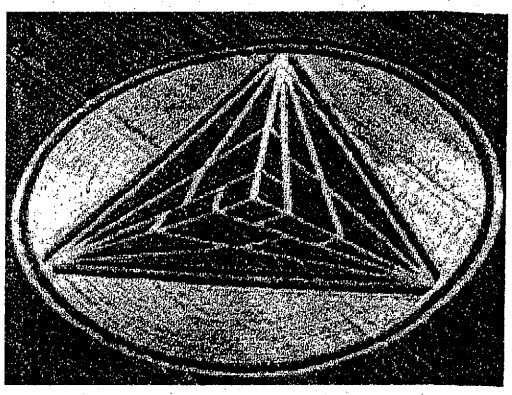
```
0011
0111
000111
```

Sample Output

```
EQUAL
ROTATE
SHOOT
```

Northeastern Europe 2009-2010

A. Equilateral



You are given N rods with sizes S1,S2,S3 ... SN. You can connect two rods with sizes Si and Sj to turn them into one rod with size Si+Sj.

You need to create the largest equilateral triangle possible out of the rods. So connect the rods in a way that you have 3 rods with exactly the same size.

You do not need to utilise all the rods.

Input

Each input will contain several test cases.

The first line of each test case will contain one number N indicating the amount of rods. The next line will contain N numbers (S1 ... SN) separated by spaces. The last line of the input will contain a 0.

Looking into the inputs you can verify that 1 <= Si <= 100 and 4 <= N <= 100.

Output

For each test case print one line with the size of the side of the biggest equilateral triangle that can be built out of the given rods. If no equilateral triangle can be built then print 0.

Example input

```
5
1 1 1 1 1
4
1 2 3 4
8
2 4 4 9 3 7 6 2
0
```

Example output

```
1
0
11
```

4590 - Business Center

Europe - Northeastern Europe - 2009/2010

International Cyber Police Corporation (ICPC) had built a new mega-tall business center to host its headquarters and to lease some space for extra profit. It has so many floors, that it is impractical to have a separate button in each of its m elevator cars for each individual floor. Instead, each elevator car has just two buttons. One button in i-th elevator car makes it move up u_i floors, the other makes it move down d_i floors. The business center is so high, that we can ignore its height for this problem (you will never reach the top floor), but you cannot go below the ground floor. All floors are numbered by integer numbers starting from zero, zero being the ground floor.

You start on the ground floor of the business center. You have to choose one elevator car out of m to ride on. You cannot switch elevators cars after that. What is the lowest floor above the ground floor you can get to after you press elevator car buttons exactly n times?

Input

Input contains several datasets, separated by a blank line. The first line of each dataset contains two integer numbers n and m ($1 \le n \le 1000000$, $1 \le m \le 2000$) -- the number of button presses and the number of elevator cars to choose from. The following m lines describe elevator cars. Each line contains two integer numbers u_i and d_i ($1 \le u_i, d_i \le 1000$).

Output

For each dataset, write to the output file a single positive integer number -- the number of the lowest floor above ground floor that can be reached by one of m elevators after pressing its buttons exactly n times.

Sample Input

```
10 3
15 12
15 4
7 12
```

Sample Output

```
13
```

Northeastern Europe 2009-2010

4864 - Bit Counting

North America - Southeast - 2010/2011

Start with an integer, N_0, which is greater than 0. Let N_1 be the number of ones in the binary representation of N_0. So, if $N_0 = 27$, $N_1 = 4$. For all $i > 0$, let N_i be the number of ones in the binary representation of N_{i-1}. This sequence will always converge to one. For any starting number, N_0, let K be the minimum value of $i \geq 0$ for which $N_i = 1$. For example, if $N_0 = 31$, then $N_1 = 5$, $N_2 = 2$, $N_3 = 1$, so $K = 3$.

Given a range of consecutive numbers, and a value X, how many numbers in the range have a K value equal to X?

Input

There will be several test cases in the input. Each test case will consist of three integers on a single line:

LO HI X

Where *LO* and *HI* ($1 \leq LO \leq HI \leq 10^{18}$) are the lower and upper limits of a range of integers, and X ($0 \leq X \leq 10$) is the target value for K. The input will end with a line with three 0s.

Output

For each test case, output a single integer, representing the number of integers in the range from *LO* to *HI* (inclusive) which have a K value equal to X in the input. Print each integer on its own line with no spaces. Do not print any blank lines between answers.

Sample Input

```
31 31 3
31 31 1
27 31 1
27 31 2
1023 1025 1
1023 1025 2
0 0 0
```

Sample Output

```
1
0
0
3
1
1
```

Southeast 2010-2011

E • The Next Permutation

For this problem, you will write a program that takes a (possibly long) string of decimal digits, and outputs the permutation of those decimal digits that has the next *larger* value (as a decimal number) than the input number. For example:

```
123 -> 132
279134399742 -> 279134423799
```

It is possible that no permutation of the input digits has a larger value. For example, **987**.

Input

The first line of input contains a single integer ***P***, ($1 \le P \le 1000$), which is the number of data sets that follow. Each data set is a single line that contains the data set number, followed by a space, followed by up to 80 decimal digits which is the input value.

Output

For each data set there is one line of output. If there is no larger permutation of the input digits, the output should be the data set number followed by a single space, followed by the string **BIGGEST**. If there is a solution, the output should be the data set number, a single space and the next larger permutation of the input digits.

Sample Input	Sample Output
3 1 123 2 279134399742 3 987	1 132 2 279134423799 3 BIGGEST

J Wormly

Jonly is writing his first computer game. For the opening scene he wants to have the main character, Wormly, cross Bridgely, the bridge. Wormly is a worm made of b equal circular bubbles and L legs. At all times each leg has to be under one of the bubbles, and under each bubble there can be at most one leg. Bridgely was supposed to be composed of n planks with the width of each plank equal to the diameter of each of Wormly's bubbles. However, some of the planks are missing.

At every moment, Wormly can do exactly one of the following:

- Move one of its legs forward over any number of (possibly missing) planks. After the move, the leg should be on a plank and underneath one of Wormly's bubbles. A leg isn't allowed to overtake other legs.
- Move all of its bubbles forward one plank while its legs remain on the same planks. After the move each leg must still be under one of Wormly's bubbles.

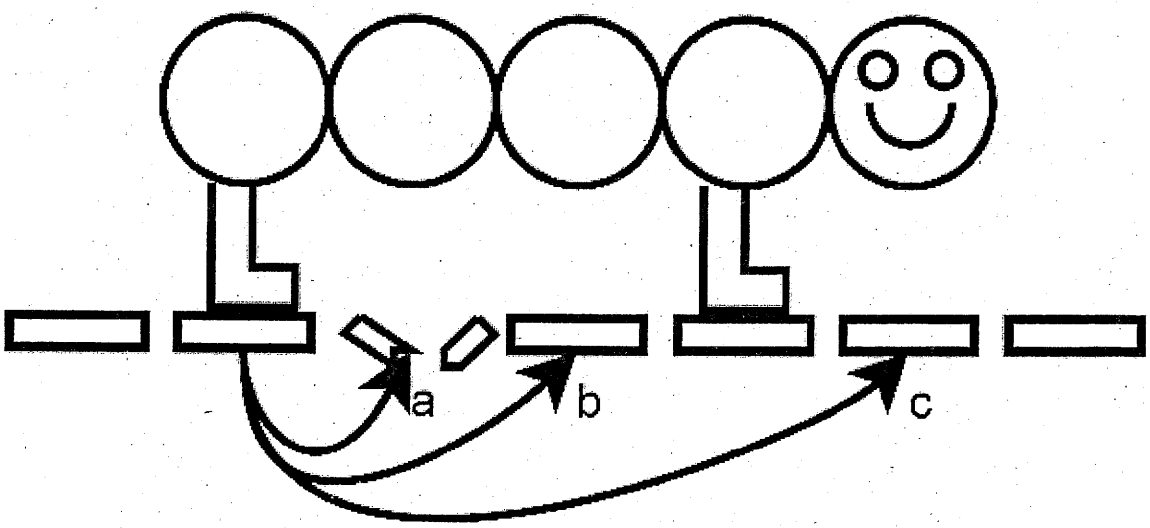


Figure 1: In this example, the only possible move for the last leg is to position b. (The plank at position a is missing, so the leg cannot move there. To get to position c, the last leg would have to overtake the first leg.) Also, in this example, moving all the bubbles forward is not allowed because Wormly's last leg would end up without a bubble over it.

Now Jonly is wondering how long the animation takes until Wormly reaches the end of Bridgely. Initially Wormly's bubbles are directly above the leftmost b planks of the bridge and its legs are on the leftmost L planks. At the end of the animation Wormly's bubbles have to be directly above the rightmost b planks and its legs have to be on the rightmost L planks.

The left- and rightmost L planks of Bridgely are not missing.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- One line with three integers L, b and n ($1 \leq L \leq b \leq n \leq 1\,000\,000$): the number of legs, the number of bubbles and the length of the bridge respectively.
- One line with a string consisting of n characters, either `0' or `1', describing Bridgely. A one indicates a plank and a zero indicates a missing plank.

Output

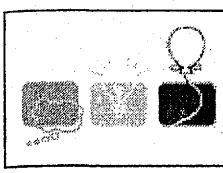
Per test case:

- One line with an integer: the minimum number of steps it takes Wormly to cross Bridgely. If it is impossible to get across while satisfying the constraints, the line must contain "IMPOSSIBLE" instead.

Sample in- and output

Input	Output
3 1 2 2 11 2 3 5 11011 1 3 5 11011	1 IMPOSSIBLE 5

Northwestern 2010-2011



4868 - Palindrometer

North America - Southeast - 2010/2011

While driving the other day, John looked down at his odometer, and it read 100000. John was pretty excited about that. But, just one mile further, the odometer read 100001, and John was REALLY excited! You see, John loves palindromes - things that read the same way forwards and backwards. So, given any odometer reading, what is the least number of miles John must drive before the odometer reading is a palindrome? For John, every odometer digit counts. If the odometer reading was 000121, he wouldn't consider that a palindrome.

Input

There will be several test cases in the input. Each test case will consist of an odometer reading on its own line. Each odometer reading will be from 2 to 9 digits long. The odometer in question has the number of digits given in the input so, if the input is 00456, the odometer has 5 digits. There will be no spaces in the input, and no blank lines between input sets. The input will end with a line with a single `0`.

Output

For each test case, output the minimum number of miles John must drive before the odometer reading is a palindrome. This may be `0` if the number is already a palindrome. Output each integer on its own line, with no extra spaces and no blank lines between outputs.

Sample Input

```
100000
100001
000121
00456
0
```

Sample Output

```
1
0
979
44
```

Southeast 2010-2011

Problem Statement (PermutationSignature)

The *signature* of a permutation is a string that is computed as follows: for each pair of consecutive elements of the permutation, write down the letter 'I' (increasing) if the second element is greater than the first one, otherwise write down the letter 'D' (decreasing).

For example, the signature of the permutation {3,1,2,7,4,6,5} is "DIIDID".

Your task is to reverse this computation: You are given a String **signature** containing the signature of a permutation. Find and return the lexicographically smallest permutation with the given signature. If no such permutation exists, return an empty int[] instead.

Notes

-For any positive integer N, a permutation of N elements is a sequence of length N that contains each of the integers 1 through N exactly once.-To compare two permutations A and B, find the smallest index i such that A[i] and B[i] differ. If A[i] < B[i], we say that A is lexicographically smaller than B, and vice versa.

Constraints

-**signature** will contain between 1 and 50 characters, inclusive.-Each character in **signature** will be either 'I' or 'D'.

Examples

0)
```
"IIIII"
Returns: {1, 2, 3, 4, 5, 6 }
```

1)
```
"DI"
Returns: {2, 1, 3 }
```
There are two permutations with this signature: {3,1,2} and {2,1,3}. You must return the lexicographically smaller one.

2)
```
"IIIID"
Returns: {1, 2, 3, 4, 6, 5 }
```

3)
```
"DIIDID"
Returns: {2, 1, 3, 5, 4, 7, 6 }
```
This is the signature from the problem statement. Note that the correct answer is not the permutation from the problem statement.

Problem Statement (NetworkSecurity)

TopCoder Security Agency (TSA) has been hired to secure a network system. In this network, there are several client computers and server computers which for the sake of conciseness will be called clients and servers, respectively. There are several one directional data cables in the network: between clients and from clients to servers. A data path in the network is defined as a series of 2 or more computers C1, C2, C3, ..., CN such that for each i between 1 and N-1, there exists a data cable from Ci to C(i+1). To avoid infinite data loops, the network is acyclic, that is, there does not exist a data path that originates and ends at the same computer. A client C is connected to a server S if there exists a data path originating at C and ending at S.

TSA is going to secure the network by installing data gates on some of the cables. The network is said to be secure if for every pair of connected client and server, there exists at least one data path between them which uses at least one cable on which a data gate is installed.

There are N clients numbered 0 through N-1 and M servers numbered 0 through M-1. The data cables are given as String[] **clientCable** and String[] **serverCable**. The j-th character of the i-th element of **clientCable** is 'Y' if there exists a data cable from client i to client j, or 'N' otherwise. The j-th character of the i-th element of **serverCable** is 'Y' if there exists a data cable from client i to server j, or 'N' otherwise.

Return the minimum number of data gates that needs to be installed to make the network secure.

Definition

Class:	NetworkSecurity
Method:	secureNetwork
Parameters:	String[], String[]
Returns:	int
Method signature:	int secureNetwork(String[] clientCable, String[] serverCable)
(be sure your method is public)	

Constraints

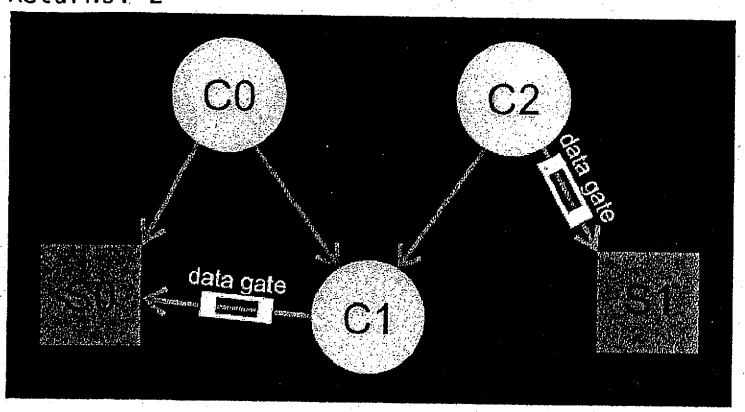
-**clientCable** will contain between 1 and 50 elements, inclusive.-Each element of **clientCable** will contain exactly N characters, where N is the number of elements in **clientCable**.-Each character in **clientCable** will be 'Y' or 'N'.-The i-th character of the i-th element of **clientCable** will be 'N'.-**serverCable** will contain the same number of elements as **clientCable**.-Each element of **serverCable** will contain between 1 and 50 characters, inclusive.-All elements of **serverCable** will contain the same number of characters.-Each character in **serverCable** will be 'Y' or 'N'.-The network will be acyclic as described in the problem statement.

0)

```
{"NYN"
,"NNN"
,"NYN"}

{"YN"
,"YN"
,"NY"}
```

Returns: 2



All three clients are connected to server 0 and only client 2 is connected to server 1. If the data gates are installed on the two cables as shown in the picture above, then each of the following data paths between a client and server will contain at least one cable on which a data gate is installed:

- C0->C1->S0
- C1->S0
- C2->C1->S0
- C2->S1

1)

```
{"NN"
,"NN"}

{"NNN"
,"NNN"}
```

Returns: 0

No client is connected to any server and hence the network is secure.

2)

```
{"NYNN"
,"NNNN"
,"NNNY"
,"NNNN"}

{"YYN"
,"NNN"
,"NNY"
,"NNN"}
```

Returns: 3

4888 - Railroad

North America - Rocky Mountain - 2010/2011

A train yard is a complex series of railroad tracks for storing, sorting, or loading/unloading railroad cars. In this problem, the railroad tracks are much simpler, and we are only interested in combining two trains into one.

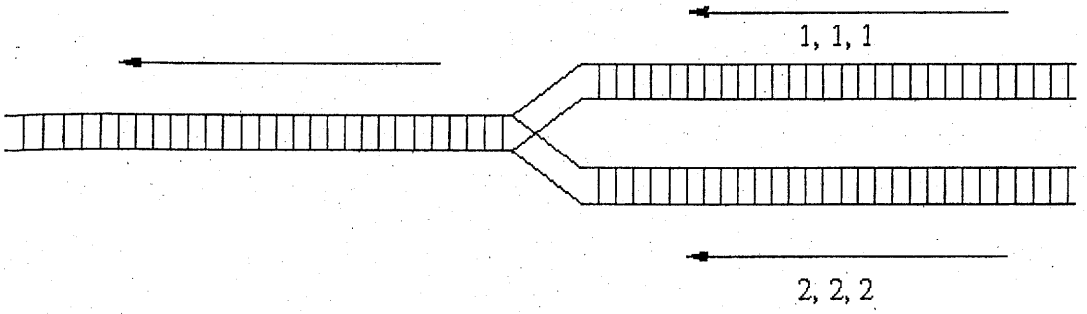


Figure 1: Merging railroad tracks.

The two trains each contain some railroad cars. Each railroad car contains a single type of products identified by a positive integer up to 1,000,000. The two trains come in from the right on separate tracks, as in the diagram above. To combine the two trains, we may choose to take the railroad car at the front of either train and attach it to the back of the train being formed on the left. Of course, if we have already moved all the railroad cars from one train, then all remaining cars from the other train will be moved to the left one at a time. All railroad cars must be moved to the left eventually. Depending on which train on the right is selected at each step, we will obtain different arrangements for the departing train on the left. For example, we may obtain the order 1,1,1,2,2,2 by always choosing the top train until all of its cars have been moved. We may also obtain the order 2,1,2,1,2,1 by alternately choosing railroad cars from the two trains.

To facilitate further processing at the other train yards later on in the trip (and also at the destination), the supervisor at the train yard has been given an ordering of the products desired for the departing train. In this problem, you must decide whether it is possible to obtain the desired ordering, given the orders of the products for the two trains arriving at the train yard.

Input

The input consists of a number of cases. The first line contains two positive integers N_1 N_2 which are the number of railroad cars in each train. There are at least 1 and at most 1000 railroad cars in each train. The second line contains N_1 positive integers (up to 1,000,000) identifying the products on the first train from front of the train to the back of the train. The third line contains N_2 positive integers identifying the products on the second train (same format as above). Finally, the fourth line contains N_1+N_2 positive integers giving the desired order for the departing train (same format as above).

The end of input is indicated by $N_1 = N_2 = 0$.

Output

For each case, print on a line `possible` if it is possible to produce the desired order, or `not possible` if not.

Sample input

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

Sample Output

```
possible
not possible
```

Rocky Mountain 2010-2011

5119 - Zombie Containment

North America - Pacific Northwest - 2010/2011

Associated with the zombie apocalypse is the notion of a critical mass of zombies: once the number of zombies exceeds a certain threshold T, all is lost. Your city is in a precarious state: its current population is $3T$. Even worse, you suspect someone in the city may be infected, although you have no idea who it may be. Once the symptoms manifest, it will be too late for that person, as well as for everyone else in the city that person can reach.



To prevent the possibility of reaching critical mass, you are going to divide the city into 3 regions via a set of walls. The city is triangular, and so you are going to divide the city into 3 smaller triangles by choosing a single splitting point and erecting straight walls between that point and the three vertices of the outer wall. This must be done immediately, i.e., there is no time to relocate people. Can you find a splitting point such that the three resulting regions each have exactly T people?

Input

The input consists of multiple test cases. Each test case is formatted as follows:

- Line 1: An integer n, $3 \leq n \leq 30000$, denoting the number of people. n is always a multiple of 3.
- Lines 2 to 4: Two numbers x_i and y_i denoting the coordinates of the i-th corner of the city, with $-10 \leq x_i, y_i \leq 10$. The three corners are specified in counterclockwise order.
- Lines 5 to $n + 4$: Two numbers x_i and y_i denoting the coordinates of the i-th person, with $-10 \leq x_i, y_i \leq 10$. All people lie in the interior of the city triangle.

Input is followed by a single line with a `0', which should not be processed. It is guaranteed that no two people will be collinear with any of the corners of the city. In particular, it is guaranteed that for any two people I and J and any corner A, the angle IAJ will be at least 10^{-7} rad.

Output

For each test case, print out a single line with two numbers x and y denoting the splitting point of the city. Each person must lie in the interior of one of the three generated triangles. Assume a person is a single point with zero radius, and that a wall is a line with zero thickness. It is guaranteed that there exists a splitting point; in fact, it is guaranteed that there exists a splitting point such that if the point were moved by 10^{-7} in any direction, it would still be a splitting point. Any splitting point that partitions the people into 3 sets of equal size will be judged correct.

Sample Input

```
3
0.0 0.0
10.0 0.0
0.0 10.0
4.0 4.0
1.0 4.0
4.0 1.0
0
```

Sample Output

```
3.0 3.0
```

Pacific Northwest 2010-2011