The A+B Problem

Calculate the result of A+B, where A and B should be read from stdin. Output should be printed on a single line each.

Input

The input consists of serveral lines, each of which contains two integers A and B, separated by a space.

Output

For each pair of A and B, you should first calculate the sum of A and B and then print the result to the stdout on a single line.

Sample Input

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

Sample Output

```
3
7
11
```

Sample Solution

For the first problem, you may need a sample solution. Read the code below carefully and write a program by yourself. After that submit it to the judge and check the response.

//Program Sample for problem 1000, in C Language

```
#include <stdio.h>
int main()
{
    int a, b;
    while(scanf("%d %d", &a, &b) != EOF){
        printf("%d\n", a + b);
    }
    return 0;
}
```

B • B-Casting

Casting around for problems leads us to combine modular arithmetic with different integer bases, particularly the problem of computing values modulo b-1, where b is the base in which the value is represented. For example,

$7829_{10} \bmod 9 = 8,$
$37777777777777773_{8} \bmod 7 = 6$
$123456_{7} \bmod 6 = 3$

(Note that $37777777777777773_{8} = 1125899906842619_{10}$ and $123456_{7} = 22875_{10}$.)

Your job is to write a program that reads integer values in various bases and computes the remainder after dividing these values by one less than the input base.

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 should be processed identically and independently.

Each data set consists of a single line of input containing three space-separated values. The first is an integer which is the data set number. The second is an integer which is the number, **B** (2 <= **B** <= 10), denoting a numeric base. The third is an unsigned number, **D**, in base **B** representation. For this problem, the number of numeric characters in **D** will be limited to 10,000,000.

Output

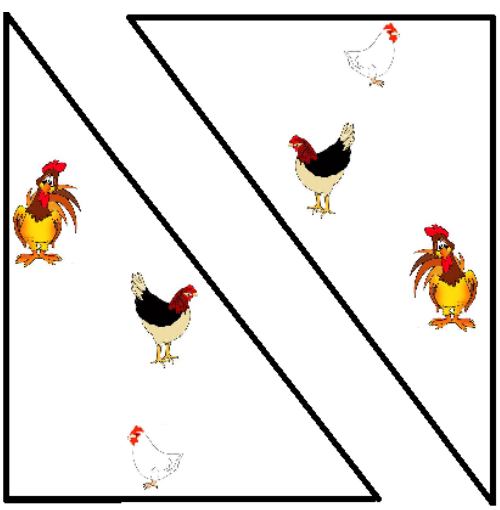
For each data set there is a single line of output. It contains the data set number followed by a single space which is then followed by the remainder resulting from dividing **D** by **(B-1)**.

Sample Input	Sample Output
6 1 10 7829 2 7 123456 3 6 432504023545112 4 8 37777777777777773 5 2 10110100010. . .101010111 6 10 5646046913. . .444716966	1 8 2 3 3 1 4 6 5 0 6 5

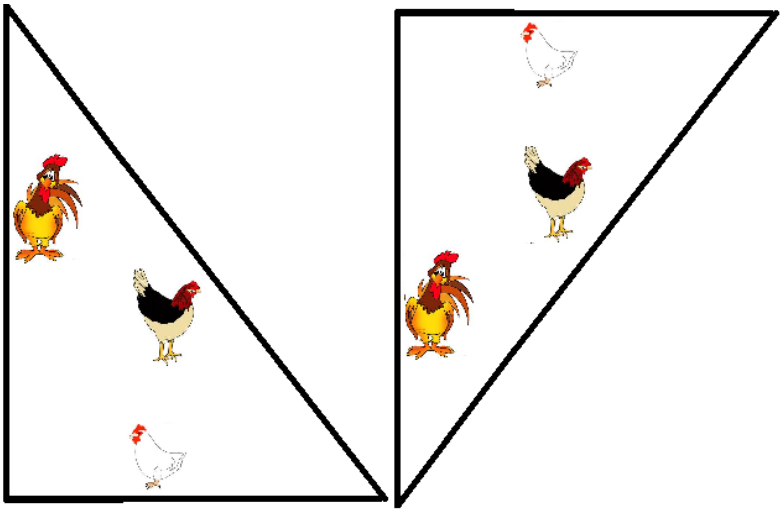
(digits that are omitted here can be seen in the sample data files online)

C • Pen Counts

Chicken farmer Xiaoyan is getting three new chickens, Lucy, Charlie and CC. She wants to build a chicken pen so that each chicken has its own, unobstructed view of the countryside. The pen will have three straight sides; this will give each chicken its own side so it can pace back and forth without interfering with the other chickens. Xiaoyan finds a roll of chicken wire (fencing) in the barn that is exactly ***N*** feet long. She wants to figure out how many different ways she can make a three sided chicken pen such that each side is an integral number of feet, and she uses the entire roll of fence. Different rotations of the same pen are the same, however, reflections of a pen may be different (see below).



Same



Different

Input

The first line of input contains a single integer ***P***, (`1 ≤ P ≤ 1000`), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K***, and the length of the roll of fence, ***N***, (`3 <= N <= 10000`).

Output

For each data set there is a single line of output. It contains the data set number, ***K***, followed by a single space which is then followed by an integer which is the total number of different three-sided chicken pen configurations that can be made using the entire roll of fence.

Sample Input	Sample Output
5 1 3 2 11 3 12 4 100 5 9999	1 1 2 5 3 4 4 392 5 4165834

D • Maximum Random Walk

Consider the classic random walk: at each step, you have a 1/2 chance of taking a step to the left and a 1/2 chance of taking a step to the right. Your expected position after a period of time is zero; that is, the average over many such random walks is that you end up where you started. A more interesting question is what is the expected rightmost position you will attain during the walk.

Input

The first line of input contains a single integer ***P***, (**1 ≤ *P* ≤ 15**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input consisting of four space-separated values. The first value is an integer ***K***, which is the data set number. Next is an integer ***n***, which is the number of steps to take (1 ≤ **n** ≤ 1000). The final two are double precision floating-point values ***L*** and ***R*** which are the probabilities of taking a step left or right respectively at each step (0 ≤ ***L*** ≤ 1, 0 ≤ ***R*** ≤ 1, 0 ≤ ***L+R*** ≤ 1). Note: the probably of not taking a step would be **1-*L*-*R***.

Output

For each data set there is a single line of output. It contains the data set number, followed by a single space which is then followed by the expected (average) rightmost position you will obtain during the walk, as a double precision floating point value to four decimal places.

Sample Input	Sample Output
4 1 1 0.5 0.5 2 4 0.5 0.5 3 10 0.5 0.4 4 1000 0.5 0.4	1 0.5000 2 1.1875 3 1.4965 4 3.9995

E • Faulhaber's Triangle

The sum of the m^{th} powers of the first n integers

$$S(n,m) = \text{SUM } (j = 1 \text{ to } n)(j^m)$$

Can be written as a polynomial of degree $m+1$ in n:

$$S(n,m) = \text{SUM } (k = 1 \text{ to } m+1)(F(m,k) * n^k)$$

For example:

$$S(n,1) = (1 + \ldots + n) = (1/2)*n^2 + (1/2)*n$$
$$S(n,2) = (1 + \ldots + n^2) = (1/3)*n^3 + (1/2)*n^2 + (1/6)*n$$
$$S(n,3) = (1 + \ldots + n^3) = (1/4)*n^4 + (1/2)*n^3) + (1/4)*n^2$$
$$S(n,4) = (1 + \ldots + n^4) = (1/5)*n^5 + (1/2)*n^4) + (1/3)*n^3 - (1/30)*n$$

The coefficients $F(m,k)$ of these formulas form *Faulhaber's Triangle*:

1						
1/2	1/2					
1/6	1/2	1/3				
0	1/4	1/2	1/4			
-1/30	0	1/3	1/2	1/5		
0	-1/12	0	5/12	1/2	1/6	
1/42	0	-1/6	0	1/2	1/2	1/7

where rows m start with 0 (at the top) and columns k go from 1 to $m+1$

Each row of *Faulhaber's Triangle* can be computed from the previous row by:

a) The element in row i and column j $(j > 1)$ is $(i/j)*$(***the element above left***); that is:
   $F(i,j) = (i/j)*F(i-1,j-1)$
b) The first element in each row $F(i,1)$ is chosen so the sum of the elements in the row is 1.

Write a program to find entries in *Faulhaber's Triangle* as decimal fractions in lowest terms.

Input

The first line of input contains a single integer ***P***, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input consisting of three space separated decimal integers. The first integer is the data set number. The second integer is row number **m**, and the third integer is the index **k** within the row of the entry for which you are to find **F(m, k)**, the *Faulhaber's Triangle* entry **(0 <= m <= 400, 1 <= k <= n+1).**

Output

For each data set there is a single line of output. It contains the data set number, followed by a single space which is then followed by either the value if it is an integer OR by the numerator of the entry, a forward slash and the denominator of the entry.

Sample Input	Sample Output
4 1 4 1 2 4 3 3 86 79 4 400 401	1 -1/30 2 1/3 3 -22388337 4 1/401

F • The King's Ups and Downs

The king has guards of all different heights. Rather than line them up in increasing or decreasing height order, he wants to line them up so each guard is either shorter than the guards next to him or taller than the guards next to him (so the heights go up and down along the line). For example, seven guards of heights 160, 162, 164, 166, 168, 170 and 172 cm. could be arranged as:

160 164 162 168 166 172 170

or perhaps:

172 160 170 162 168 164 166

The king wants to know how many guards he needs so he can have a different up and down order at each changing of the guard for rest of his reign. To be able to do this, he needs to know for a given number of guards, ***n***, how many different up and down orders there are:

For example, if there are four guards: 1, 2, 3, 4 can be arranged as:

1324, 2143, 3142, 2314, 3412, 4231, 4132, 2413, 3241, 1423

For this problem, you will write a program that takes as input a positive integer ***n***, the number of guards and returns the number of up and down orders for ***n*** guards of differing heights.

Input

The first line of input contains a single integer ***P***, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set consists of single line of input containing two integers. The first integer, ***D*** is the data set number. The second integer, ***n*** (1 <= ***n*** <= 20), is the number of guards of differing heights.

Output

For each data set there is one line of output. It contains the data set number (***D***) followed by a single space, followed by the number of up and down orders for the ***n*** guards.

Sample Input	Sample Output
4 1 1 2 3 3 4 4 20	1 1 2 4 3 10 4 740742376475050

G • Mad Veterinarian

Mad Veterinarian puzzles have a mad veterinarian, who has developed several machines that can transform an animal into one or more animals and back again. The puzzle is then to determine if it is possible to change one collection of animals into another by applying the machines in some order (forward or reverse). For example:

> *Machine **A** turns one ant into one beaver.*
> *Machine **B** turns one beaver into one ant, one beaver and one cougar.*
> *Machine **C** turns one cougar into one ant and one beaver.*
>
> *Can we convert a beaver and a cougar into 3 ants?*
>
> Yes. `{b,c}` **C** ➔ `{a,2b}` **A** reversed ➔ `{2a,b}` **A** reversed ➔ `{3a}`
>
> *Can we convert one ant into 2 ants?* `NO`

These puzzles have the properties that:

1. In forward mode, each machine converts one animal of a given species into a finite, non-empty collection of animals from the species in the puzzle.
2. Each machine can operate in reverse.
3. There is one machine for each species in the puzzle and that machine (in forward mode) takes as input one animal of that species.

Write a program to find the **shortest** solution (if any) to *Mad Veterinarian* puzzles. For this problem we will restrict to *Mad Veterinarian* puzzles with exactly three machines, **A**, **B**, **C**.

Input

The first line of input contains a single integer ***P***, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set consists of several lines of input. Each data set should be processed identically and independently.

The first line of each data set consists of two decimal integers separated by a single space. The first integer is the data set number. The second integer is the number, ***N***, of puzzle questions. The next three input lines contain the descriptions of machines **A**, **B** and **C** in that order. Each machine description line consists of three decimal integers separated by spaces giving the number of animals of type **a**, **b** and **c** output for one input animal. The following ***N*** lines give the puzzle questions for the *Mad Veterinarian* puzzle. Each contains seven decimal digits separated by single spaces: the puzzle number, the three starting animal counts for animals **a**, **b** and **c** followed by the three desired ending animal counts for animals **a**, **b** and **c**.

Output

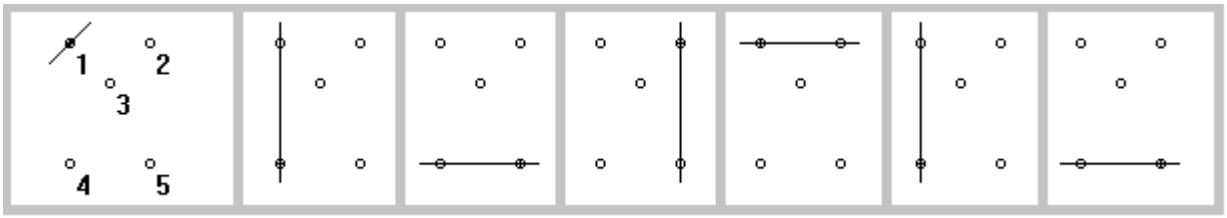
For each input data set there are multiple lines of output. The first line of output for each data set contains the data set number, a space and the number of puzzle questions (***N***). For each puzzle question, there is one line of output which consists of the puzzle question number followed by a space, followed by "**NO SOLUTION**" (without the quotes) if there is no solution **OR** the puzzle question number followed by the shortest number of machine steps used, a space and a sequence of letters [**A B C a b c**] with capital letters indicating applying the machine in the forward direction and lower case letters indicating applying the machine in the reverse direction.

Sample Input	Sample Output
2 1 2 0 1 0 1 1 1 1 1 0 1 0 1 1 3 0 0 2 1 0 0 2 0 0 2 2 0 3 4 0 0 5 0 0 3 1 2 0 0 0 0 5 2 2 0 0 0 0 4	1 2 1 3 Caa 2 NO SOLUTION 2 2 1 NO SOLUTION 2 25 AcBcccBccBcccAccBccBcccBc

H • Windmill Animation

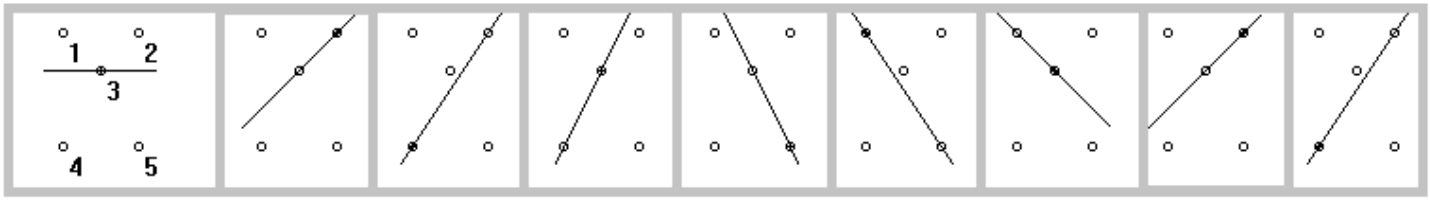
A windmill animation works as follows:

A two-dimensional set of points, no three of which lie on a line is chosen. Then one of the points is chosen (as the first pivot) and a line is drawn through the chosen point at some initial angle. The animation proceeds by rotating the line counter-clockwise about the pivot at a constant rate. When the line hits another of the points, that point becomes the new pivot point. In the two examples below, the points are `(-1, 1)`, `(1,1)`, `(0,0)`, `(-1,-2)` and `(1,-2)`.



Example 1

In Example 1, the start point is point 1 and the line starts rotated 45 degrees from horizontal. When the line rotates to 90 degrees, point 4 is hit and becomes the new pivot. Then point 5 becomes the new pivot, then point 2 then point 1.



Example 2

In Example 2, the initial point is point 3 and the line starts horizontal. At 45 degrees, point 2 becomes the pivot, then at about 56 degrees, point 4 becomes the pivot. At about 63 degrees, point 3 becomes the pivot again, then point 5, point 1 and back to 3 as at the start.

Write a program, which takes as input the points of the set, the initial point and the initial line angle and outputs the sequence of pivot points.

Input

The first line of input contains a single integer P, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of multiple lines of input. The first line of each data set consists of four space-separated decimal integers followed by a single floating-point value. The first integer is the data set number. The second integer is the number of points **`M`** to follow (**`3 <= M <= 20`**). The third integer gives the number, **`S`**, of the pivot points to output (**`3 <= S <= 20`**) and the fourth integer gives the index, **`I`**, of the initial point (**`1 <= I <= M`**). The floating-point value is the angle, **`A`**, in degrees, that the initial line is rotated counter-clockwise from horizontal (**`0 <= A < 180`**).

The remaining **`M`** lines in the data set contain the coordinates of the set of points. Each line consists of an integer, the point's index, **`I`**, and two floating-point values, the **`X`** and **`Y`** coordinates of the point respectively.

Output

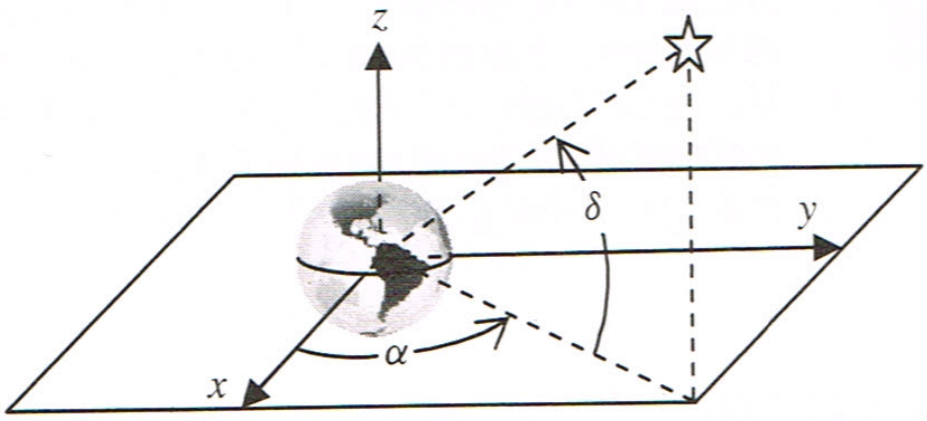
For each data set there is a single line of output. It contains the data set number, followed by **`S`** space separated point indices (excluding the initial point index).

Sample Input	Sample Output
2 1 5 5 1 45 1 -1 1 2 1 1 3 0 0 4 -1 -2 5 1 -2 2 5 7 3 0 1 -1 1 2 1 1 3 0 0 4 -1 -2 5 1 -2	1 4 5 2 1 4 2 2 4 3 5 1 3 2

I • Push-To Telescope

Once a *Push-To Telescope* is setup, it tells you where to point the telescope to see a star in its built-in catalogue by giving an *azimuth* (counter-clockwise angle in the base plane of the telescope) and *elevation* (angle above the base plane of the telescope). The telescope is initialized by pointing it at a known star, finding the star in the catalogue and selecting it. This step is repeated with a second known star which is not too close to the first one.

The direction of stars in the catalogue is given in *geocentric equatorial coordinates*. The origin is the center of the earth. The positive *z-axis* passes through the North Pole and the *xy-plane* contains the equator. The *x-axis* points at the sun at the *spring equinox* (when the sun is in the equatorial plane). The star coordinates are right ascension, α, (angle in *xy-plane* counter-clockwise from the *x-axis* in degrees) and declination, δ, (angle above (positive) or below (negative) the *xy-plane*).



In this coordinate system, the earth rotates at `(2π)(1.0027379093)/86400` radians per second. (Since the earth moves around the sun, it must rotate more than 360 degrees to get the sun over the same point.)

During setup, when you select a star, the system records the time (in seconds since the system was turned on), the azimuth and elevation of the telescope when pointing at the star and the index of the star in the table. After two selections of known stars, the system computes the transformation from geocentric equatorial coordinates to local coordinates. Subsequently, when you select a star to view, the system uses the stars geocentric equatorial coordinates and the current time to compute the azimuth and elevation to point at the star.

Write a program to implement the *Push-To Telescope*.

Since the telescope coordinate system rotates with the earth, it may be useful to use a rotating geocentric equatorial coordinate system for star coordinates. This system aligns with *geocentric equatorial coordinates* when the system turns on and rotates with the earth thereafter. In this system, the declination, δ, is the same but the right ascension angle changes with time:

$$\alpha\_rot = \alpha - t * rotation\_rate$$

where **α_rot** is the right ascension angle in the rotating system (in radians), **α** is the right ascension angle in *geocentric equatorial coordinates* (in radians), **t** is time in seconds since the system was turned on and **rotation_rate** is the earth rotation rate above.

Input

The first line of input contains two decimal integers separated by a single space. The first integer gives the number, **S**, of stars in the catalogue (**0 < S <= 100**) and the second gives the number **P** of data sets **(0 < P < 100**).

The next **S** lines of input are the star table. Each line of the star table consists of a decimal integer and two floating-point values separated by spaces. The integer is the *star index* and the floating-point values are the right ascension (α) and declination (δ) in degrees. There is one star table, and it is used for all data sets. Each data set should be processed identically and independently using the star table.

The **P** problem data sets follow the star table data. Each data set consists of several lines. The first line of each data set consists of two decimal integers separated by a single space. The first integer is the problem instance number **N**. The second integer is the number of stars to find **T**, (**T <= 10**) for the data set.

The next two lines specify the setup data for the data set. Each setup line consists of two integers followed by two floating-point values. The integers are **t**, the number of seconds since the system was started and **I**, the index of the known setup star in the table. The floating-point values are the *azimuth* and *elevation* of the star in degrees (in telescope coordinates).

The remaining **T** lines of input in the data set specify a star the user wants to observe. Each line of input consists of three integers separated by spaces. The first integer is the number of the star to observe (the sub-problem number), the second is the time (in seconds) since the system was started and the last is the index of the star to observe in the star table.

Output

There are several lines of output for each data set. The first line of output contains the data set number, followed by a single space and the number of stars that the user wished to observed (**T**). For each star that was to be observed, there is one line of output. If the computed elevation is less than zero (the star is below the horizon), the line consists of the star (sub-problem) number followed by a space and the string "**NOT VISIBLE**" (without the quotes). Otherwise the line consists of the observed star (sub-problem) number followed by the *azimuth* and *elevation* in degrees to one decimal place.

Sample Input	Sample Output
15 2 1 39.90 -16.70 2 34.65 -52.70 3 76.85 -60.80 4 70.85 19.20 5 95.19 38.80 6 28.12 46.00 7 27.62 -8.20 8 43.17 5.20 9 14.27 -57.20 10 37.62 7.40 11 67.85 -60.40 12 103.47 8.90 13 64.05 -63.10 14 28.10 16.50 15 83.64 -26.40 1 3 20 14 144.70 55.24 73 15 321.36 55.97 1 264 8 2 1121 4 3 2319 9 2 3 39 2 190.46 80.21 73 13 354.21 81.15 1 133 8 2 1846 11 3 2746 5	1 3 1 146.8 73.0 2 70.5 63.7 3 65.8 24.3 2 3 1 133.5 26.7 2 7.7 83.8 3 NOT VISIBLE

X • Mystery

Input

The first line of input contains a single integer ***P***, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set consists of several lines. Each data set should be processed identically and independently.

The first line of each data set contains an integer ***D*** which is the data set number. The second line contains no more than the 93 distinct printable ASCII characters. The third line contains an integer, ***N*** (1 <= ***N*** <= 512), which is the number of integers on the next (fourth) line of the dataset. Each integer on the fourth line is in the range –***X*** to ***X*** where ***X*** is the number of characters on the second line minus **1**.

Output

For each data set there is one correct line of output. It contains the data set number (***D***) followed by a single space, followed by a string of length ***N*** made of the characters on the second line of the input data set.

Sample Input

```
7
1
MAC
3
1 1 1
2
IW2C0NP3OS 1RLDFA
22
0 3 3 -3 7 -8 2 7 -4 3 8 7 4 1 1 -4 5 2 5 -6 -3 -4
3
G.IETSNPRBU
17
2 4 5 -6 -1 -3 -2 -4 -4 1 -1 5 -3 4 1 -2 4
4
PIBN MRDSYEO
16
-4 4 -1 4 5 3 -5 4 -3 -3 -2 -5 -5 -3 1 3
5
D^obV@k"W*B&#]4!NcF$'lj%(d6XG5fi<Hxz7)2Lt~=8aQuvh}r_m+C9eI`-.>EwYyngZRsJKpqO{[\U|MPS,;T?031/A
93
-1 11 44 39 -31 -44 10 5 24 14 1 -33 42 28 -34 7 -37 24 14 3 -7 18 4 19 37 4 20 2 41 -42 18 15 -3 10
7 12 -11 -41 14 8 31 -26 37 -19 -17 -9 -16 15 31 14 29 -22 1 -24 20 -30 6 1 16 -29 31 -30 6 17 -43 -
10 7 7 4 -22 10 -2 15 13 14 2 6 -17 34 -27 28 29 -28 2 33 -13 -15 6 -31 24 41 29 26
6
I
1
0
7
I
13
0 0 0 0 0 0 0 0 0 0 0 0 0
```

Sample Output

```
1 ACM
2 ICPC 2013 WORLD FINALS
3 IN ST. PETERSBURG
4 SPONSORED BY IBM
5 ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz~`!@#$%^&*()_+-={}[]\|'";/.,<>?
6 I
7 IIIIIIIIIIIII
```