Contents

1 Incredibles Poker

The Incredible family is playing poker at home. Violet introduces the family to a game call Razz, which she learned at school while watching her friends (while invisible). Each player receives 7 cards. Instead of trying to get the highest 5-card poker hand, the winner is the player with the *lowest* 5-card hand (using any 5 cards out of the 7 cards the player holds). Straights (5 cards in sequence) and flushes (5 cards of the same suit) do not affect the low hand, only pairs. Aces are treated as low cards.

Since Violet is a very timid poker player, she decides to bet only when she has the best possible Razz hand, which is Ace, 2, 3, 4, and 5 (in any order). Your job is to help Violet determine whether she has this 5-card hand out of her 7 cards.

Input/Output Format:

Cards are treated as numbers, with Aces, Kings, Queens, and Jacks represented as 1, 13, 12, 11, respectively. 7-card hands are read in as 7 numbers on each line. For every hand, your program should output on each line "Bet" if the lowest possible 5-card hand is possible, and "Fold" if it is not.

Example:

Sample Input:

```
1 2 3 4 5 6 7
1 2 3 4 6 7 8
1 2 3 3 4 4 4
11 1 5 4 12 2 3
5 1 3 2 6 7 9
```

Sample Output:

```
Bet
Fold
Fold
Bet
Fold
```

2 Elastigirl's Crime Records

Elastigirl keeps a database of criminal records at home. Recently she decided to write a program for a crime-stopping prioritizing system. It will take as input a set of individual crimes, one per line, each of which consists of a crime ID (a String), state name (a capitalized String), and a dollar value (a floating point value). However, each member of the Parr family has a different idea on the order in which the information should be presented once entered. Mr. Incredible wants to see the list of crime IDs ordered by state name, Elastigirl wants to see the crime IDs ordered by dollar value, and Jack Jack wants to see them listed in the order that they were input. Your task is to read in all of the crime information, and then output three lists of crime IDs; one in each of the specified orders—print Jack Jack's ordering first, then Mr. Incredible's, and finally Elastigirl's.

Example:

Sample Input:

```
32898 NY 35000.45
89328 MD 43500.00
12392 CA 10750.03
```

Sample Output:

```
Jack Jack:      32898  89328  12392
Mr. Incredible: 12392  89328  32898
Elastigirl:     12392  32898  89328
```

3 Syndrome's Run Length Encoder

Run-length encoding is a compression scheme used for images having relatively few colors, such as animation cells and icons. More sophisticated compression algorithms can make even smaller files and work well even when the input is a photograph having many colors.

The Arch-villain Syndrome decides to build a run length encoder, since it will serve two purposes. First, it will allow him to store more films of his former idol, Mr. Incredible. Second, the run length encoder can help protect his files from being viewed by any heroes who may escape from his robots.

Your task is to help Syndrome build a run length encoder for the following byte-oriented run-length encoded format. Runs of the same, repeated byte in the original file are replaced with a count of the number of repeated bytes, stored as a byte, followed by the repeated byte itself. For example, the string "HELLO", would be encoded as follows:

original	H	E	L	L	O			
compressed (encoded)	1	H	1	E	2	L	1	O

Each letter is actually the ascii value of the character. (For example 'h' is 0x68 in base 16 or 104 in base 10). Note that every other byte is a count of how often to repeat the byte that follows.

Note that there is no reason to use a count of zero (0), so we can assume a count of 0 actually represents 256. Also note that if you receive more than 256 letters in a row, you will need to use multiple counts since bytes only store values up to 255. In such cases, all but the final count values should 0. See the example below, which involves encoding a string of 1000 H's.

original	H	H	H	H	H	H	H	...
compressed (encoded)	0	H	0	H	0	H	232	H

Example:

Sample Input:
```
H E L L O
```

Sample Output:
```
1 H 1 E 2 L 1 O
```

Sample Input:
```
H ... (1000 times)
```

Sample Output:
```
0 H 0 H 0 H 232 H
```

4 Mr. Incredible's Run Length Decoder

Mr. Incredible has escaped from Syndrome's robots, and is snooping around the island fortress. He finds several of Syndrome's encoded files, and through much study has determined the files are probably the results of run length encoding.

Your task is to help Mr. Incredible build a run length decoder, using the same format Syndrome used for his run length encoder.

If your decoder works, it should successfully decode any file encoded by the run length encoder and produce the original file.

Example:

Sample Input:
1 H 1 E 2 L 1 O

Sample Output:
H E L L O

Sample Input:
0 H 0 H 0 H 232 H

Sample Output:
H ... (1000 times)

5 Violet's Polynomial Printer

The speedy youth Dash is not so fast when it comes to his algebra homework. His sister Violet helps him by writing a program to print polynomials. This program is given a sequence of integers $(a_n, \ldots, a_0)$, which represents polynomial having these coefficients:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0.$$

For example, the sequence $(3, 4, 2, 5)$ represents the polynomial

$$3x^3 + 4x^2 + 2x + 5.$$

The program prints the polynomial in a nice looking format. Allowing the carat character ('^', which is located above the '6' key on most keyboards) to represent exponentiation, the above polynomial can be expressed as follows:

```
3x^3 + 4x^2 + 2x + 5
```

This sounds easy, but there are dangers! For example, for the sequence $(4, -1, 0)$ do you really want to print the following?

```
4x^2 + -1x + 0
```

No. Instead, you want to print:

```
4x^2 - x
```

There are other dangers too but we won't tell you what they are. Note that format is important, and should match the above form exactly (including where spaces appear). More examples are given below.

Samples

The input will consist of a sequence of lines, each line contains the coefficients for one polynomial. The coefficients are integers and are separated by spaces. Note that leading 0's are possible (as seen in the last example below).

Input:	Polynomial:
`3 4 2 5`	`3x^3 + 4x^2 + 2x + 5`
`4 -1 0`	`4x^2 - x`
`-1 1 -1`	`-x^2 + x - 1`
`-4 -2 003 -5`	`-4x^3 - 2x^2 + 3x - 5`
`-8`	`-8`
`0`	`0`
`0 -3 0`	`-3x`

Input/Output Format

We have provided a main program for you that reads in the coefficients and stores them in an array, and prints the final output. All you need to do is to provide the body for a function `formatPoly` that is given these coefficients and computes and returns the formatted polynomial as a string. The function takes the following argument:

`int[] a:` an array containing the coefficients, where $a[i]$ contains the coefficient for the x^i term. (Note: This is the *reverse* of the input order!)

If you want to do your own input and output, here is the format. Each line of the input contains a sequence of integer coefficients, separated by spaces. The first coefficient belongs to the highest order coefficient and the last belongs to the lowest (0th order) coefficient. For each input line the output consists of the original input line and the final polynomial.

Example:

An example is shown below. Note that for the first sample input the array a will have the values $[5, 2, 4, 3]$.

Sample Input:

```
3 4 2 5
4 -1 0
```

Sample Output:

```
Input: 3 4 2 5
Polynomial: 3x^3 + 4x^2 + 2x + 5
Input: 4 -1 0
Polynomial: 4x^2 - x
```

6 Dash's Palindromic Milestones

One of Mr. Incredible's greatest opponents, *Cixelsyd*, was dyslexic. When reading a word or a number (especially an isolated one), he would get confused and would sometimes read from right to left, instead of left to right. When he was building the Grand National Highways to unite his vast empire, he decided to number the milestones in such a manner that he would never get confused while traveling. His ingenious solution was to leave all milestones blank, except those carrying numbers that read the same both forwards and backwards. So, the first 10 milestones would be 0,1,2,. . . ,9 (where 0 is the starting point) followed by 11,22,33,. . . ,99, followed by 101,111,121, etc. Milestones for distances that are not of this form are left blank.

Dash is running along one such highway starting from the capital, and to keep himself alert, decides to play the following game. When Dash is at distance k miles from the capital, he picks a number n and tries to guess what will be the number on the n-th non-blank milestone from this point (not including the current milestone, if there is any). Design a program to do this.

Input/Output Format:

The input will contain several test cases. The first line contains a positive integer N giving the number of test cases. For every test case, there are 2 lines. The first line contains the positive integer k (the miles covered so far) and the second line contains n (which milestones value is to be calculated).

Your output should be one line per test case, giving the reading on the n-th non-blank milestone after the k-th mile.

Example:

Sample Input 1:
2
13
2
99
3

Sample Output 1:
33
121

Sample Input 2:
1
100000
100000

Sample Output 2:
910999019

7 Two *Colors* Diverged In A Wood...

Our heroes, Mr. Incredible and the gang, have just arrived at the island of *Coloropia*, so called because of its distinctive and bright colored roads. All the major roads on the island are colored with various colors, some red, some blue and some using other colors. Our heroes have to travel along these roads to the capital of Coloropia to confront their arch-nemesis, the King of Coloropia, which they naturally wish to do as soon as possible. To make it hard for the King to follow their movements, Mr. Incredible wants to use different colored roads when coming into any city, and when leaving that city. So if they enter a city on a road colored *red*, they would like to leave the city on a road with any other color than *red*. Unfortunately, Mr. Incredible's job has been made incredibly harder because of the superstitious nature of his companions. For example, Elastigirl will not travel on a red colored road if she has already traveled on a green colored road and *vice versa*. She believes this brings her bad luck. The other companions have similar superstitions, and this has made Mr. Incredible incredibly frustrated. He is considering giving up this life of crime-fighting and taking up his old job as an insurance claims specialist again. For sake of humanity, you must help Mr. Incredible find the shortest such path to the Capital (visiting the fewest number of cities along the way) that satisfies all the constraints outlined above.

Input/Output Format:

The first line of the input contains the number of cities N, and the number of roads between the cities. Cities are numbered from 0 to $N-1$. The next set of lines describes the roads using three numbers each. The first two numbers indicate the two cities that the road connects, and the third number is the color of the road. After that, the next line contains one number that denotes the total number of color constraints, and the next set of lines describe the constraints, one by one. Each of the constraints is given as two numbers, denoting the colors that cannot both be used in a path. Roads are bidirectional, so a road between cities 1 and 2 can be used to go from 1 to 2 or from 2 to 1. There will be at most 10 different colors (from 0 to 9).

Example:

For instance, assume you are given the following:

```
 4 4     - 4 cities, and 4 roads between them.
 0 1 0   - there is a road between cities 0 and 1, and it is colored with color 0.
 1 2 1   - there is a road between cities 1 and 2, and it is colored with color 1.
 2 3 2   - there is a road between cities 2 and 3, and it is colored with color 2.
 1 3 3   - there is a road between cities 1 and 3, and it is colored with color 3.
 1       - 1 constraint
 0 3     - Colors 0 and 3 can't both be used in a path.
```

You are required to find the shortest alternating-colors path satisfying the constraints, between cities 0 and $N-1$ (where N is the total number of cities); in the above case, you need to find the shortest such path between cities 0 and 3. If there is no path between the two cities, the output should simply contain a line:

```
    No Valid Path Found
```

If there is a path between the two cities satisfying the constraints (as in the example above), the output should look something like this:

```
    Start at City 0
```

```
Go to City 1 On Road Colored 0
Go to City 2 On Road Colored 1
Go to City 3 On Road Colored 2
End at City 3
```

There exists a shorter path between 0 and 3, but it does not satisfy the constraints as it uses roads colored 0 and 3 both, which is not allowed.

Note that the shortest path satisfying the constraints might visit a given city multiple times (See Sample 1 below).

Sample Input 1:

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

Sample Output 1:

```
Start at City 0
Go to City 2 on Road Colored 0
Go to City 1 on Road Colored 1
Go to City 3 on Road Colored 0
Go to City 2 on Road Colored 1
Go to City 4 on Road Colored 0
End at City 4
```

Sample Input 2:

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

Sample Output 2:

```
No Valid Path Found
```

Sample Input 3:

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

Sample Output 3:

```
No Valid Path Found
```

8 Edna Goes to Vegas

Super-hero fashion guru Edna Mode decides to take a vacation in Las Vegas. There she plays a new game of chance that involves oddly shaped dice. Each die has two or more sides, and the numbers on the sides run from $1, 2, \ldots$ up to the number of sides. However, the dice are not balanced and different numbers have different probabilities of occurring. (E.g., it may much more likely to throw a 2 than a 5.) Through a secret analysis process, Edna has figured out the probabilities for each dice individually ("Sorry dahling, I'm not at liberty to discuss my secrets"), but she needs to compute the probability of a certain total number when all the dice are thrown. Given an integer value m, Edna wants to compute the probability of obtaining a total value of m when all the dice are rolled at once.

Formally, the input is:

- A positive integer k. (This is the number of dice.)
- For each i, $0 \leq i \leq k-1$, a tuple (vector) of values $(p_{i,1}, \ldots, p_{i,n_i})$, where n_i is the number of sides on the ith die. The value $p_{i,j}$ denotes the probability of throwing j on die i. These probabilities satisfy the following for each i:
  - For all j, $0 \leq p_{i,j} \leq 1$, and
  - $p_{i,1} + p_{i,2} + \cdots + p_{i,n_i} = 1$.
- A positive m, the desired value to be rolled.

The output is the probability that, if all the dice are rolled, the total number of dots showing will be m. (If m is less than the minimum possible value or greater than the maximum possible value that can be rolled, the probability of course is 0.)

Example:

Here is an example. Suppose that there are three dice, having 3, 4, and 2 sides, respectively, and with the following probabilities:

Die 0: 0.5, 0.2, 0.3.
Die 1: 0.2, 0.1, 0.3, 0.4.
Die 2: 0.1, 0.9.

Also suppose that $m = 5$. There are 5 ways to roll $m = 5$ with the above dice:

$$(1,2,2), \quad (1,3,1), \quad (2,1,2), \quad (2,2,1), \quad (3,1,1).$$

The respective probabilities of these events are:

$$\begin{aligned}
0.5 \cdot 0.1 \cdot 0.9 &= 0.045 \\
0.5 \cdot 0.3 \cdot 0.1 &= 0.015 \\
0.2 \cdot 0.2 \cdot 0.9 &= 0.036 \\
0.2 \cdot 0.1 \cdot 0.1 &= 0.002 \\
0.3 \cdot 0.2 \cdot 0.1 &= 0.006
\end{aligned}$$

Thus, the total probability of rolling 5 is the sum of these individual probabilities, which is 0.104, which is the program's output.

On the other hand, suppose that $m = 2$. It is impossible to roll 2 with the above dice, so the result would be 0.

The final answer must be accurate to decimal 6 digits. Hence, it is strongly recommended that you use double (rather than float) in your computations, to avoid accumulations of round-off errors.

Note that the number of dice may be large, so a brute force solution, based on enumerating all the possibilities will take too long. Also, with 6 digits of precision required, you will not have time to estimate the probability by simply randomly tossing the dice over and over.

Input/Output Format:

We have provided a main program for you that inputs all these quantities stores them in arrays, and formats and prints the final output. All you need to do is to provide the body for a function `findProbability` that computes and returns the final probability (as a double). The arguments to this function are:

`int nDice`: the number of dice. You may assume there are at most 50 dice.

`int[] nSides`: `nSides[i]` is the number of sides on the ith die, for $0 \leq i \leq$ `nDice` $- 1$. You may assume that $2 \leq$ `nSides[i]` ≤ 20.

`double[][] prob`: The value of `prob[i][j]` is the probability of rolling the value of j on the ith die, where j runs from 1 up to `nSides[i]`. (You should not attempt to access `prob[i][j]` for $j >$ `nSides[i]`).

`int rollValue`: The target value to be rolled.

If you want to do your own input and output, here is the format. The first line of the input file is the number of dice. The next lines contain the probabilities, one line for each die. The probabilities are given as floating point values, separated by spaces. The number of entries on a line is the number of sides on the die, with the first being the probability of rolling 1, the second the probability of 2, and so on. The last number is the value m to be rolled. The output consists of the final probability, accurate (and printed) to 6 decimal positions. The input quantities and the output should be printed as shown below.

Sample Input:

```
3
0.5 0.2 0.3
0.2 0.1 0.3 0.4
0.1 0.9
5
```

Sample Output:

```
Dice[0] probabilities:  (1): 0.5  (2): 0.2  (3): 0.3
Dice[1] probabilities:  (1): 0.2  (2): 0.1  (3): 0.3  (4): 0.4
Dice[2] probabilities:  (1): 0.1  (2): 0.9
Roll value = 5
Final probability = 0.104000
```

0 Dash's Poker Game

The Incredible family is playing poker at home. Dash likes to play a game called 7-card Stud. Each player receives 7 cards, one at a time, and tries to get the best 5-card poker hand.

Since Dash is a very fast poker player, he decides to bet whenever he has either an Ace or a pair (two cards with the same value). He will fold his hand otherwise. Your job is to help Dash determine whether he has an Ace or a pair out of his 7 cards.

Input/Output Format:

Cards are treated as numbers, with Aces, Kings, Queens, and Jacks represented as 1, 13, 12, 11, respectively. 7-card hands are read in with 7 cards on each line. For every hand, your program should output on each line "Bet" if there is an Ace or a pair present, and "Fold" if not.

Example:

Sample Input:

```
1 2 3 4 5 6 7
2 3 4 6 7 8 9
9 2 3 4 7 4 11
11 13 5 4 8 2 3
5 1 3 2 6 7 9
```

Sample Output:

```
Bet
Fold
Bet
Fold
Bet
```