Al Hamak First Programming Contest

Damascus University

22 / 7 /2014

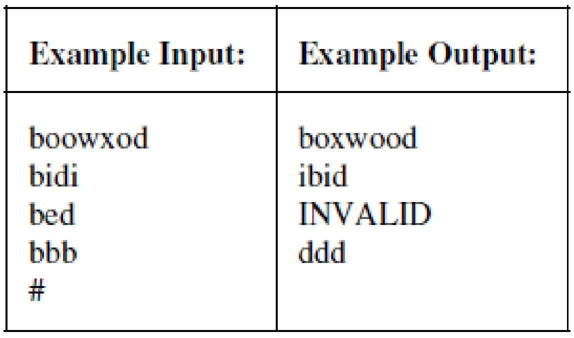
HCPC

A- Mirror, Mirror on the Wall

For most fonts, the lowercase letters `b` and `d` are mirror images of each other, as are the letters `p` and `q`.
Furthermore, letters `i`, `o`, `v`, `w`, and `x` are naturally mirror images of themselves. Although other symmetries exists for certain fonts, we consider only those specifically mentioned thus far for the remainder of this problem. Because of these symmetries, it is possible to encode certain words based upon how those words would appear in the mirror. For example the word `boxwood` would appear as `boowxod`, and the word `ibid` as `bidi`.
Given a particular sequence of letters, you are to determine its mirror image or to note that it is invalid.

Input:
The input contains a series of letter sequences, one per line, followed by a single line with the `#` character. Each letter sequence consists entirely of lowercase letters.

Output:
For each letter sequence in the input, if its mirror image is a legitimate letter sequence based upon the given symmetries, then output that mirror image. If the mirror image does not form a legitimate sequence of characters, then output the word `INVALID`.



Example Input:	Example Output:
boowxod bidi bed bbb #	boxwood ibid INVALID ddd

B- Mad scientist

A mad scientist performed a series of experiments, each having n phases. During each phase, a measurement was taken, resulting in a positive integer of magnitude at most k. The scientist knew that an individual experiment was designed in a way such that its measurements were monotonically increasing, that is, each measurement would be at least as big as all that precede it.

For example, here is a sequence of measurements for one such experiment with $n=13$ and $k=6$:
1, 1, 2, 2, 2, 2, 2, 4, 5, 5, 5, 5, 6

It was also the case that n was to be larger than k, and so there were typically many repeated values in the measurement sequence. Being mad, the scientist chose a somewhat unusual way to record the data.
Rather than record each of n measurements, the scientist recorded a sequence P of k values defined as follows.
For $1 <= j <= k$, $P(j)$ denoted the number of phases having a measurement of j or less. For example, the original measurements from the above experiment were recorded as the P-sequence:
2, 7, 7, 8, 12, 13

as there were two measurements less than or equal to 1, seven measurements less than or equal to 2, seven measurement less than or equal to 3, and so on.
Unfortunately, the scientist eventually went insane, leaving behind a notebook of these P-sequences for a series of experiments.

Your job is to write a program that recovers the original measurements for the experiments.

Input:
The input contains a series of P-sequences, one per line. Each line starts with the integer k, which is the length of the P-sequence. Following that are the k values of the P-sequence. The end of the input will be designated with a line containing the number 0. All of the original experiments were designed with $1 <= k < n <= 26$.

Output:
For each P-sequence, you are to output one line containing the original experiment measurements separated by spaces.

Sample input:
```
6 2 7 7 8 12 13
1 4
3 4 4 5
3 0 4 5
5 2 2 4 7 7
0
```

Sample output:
```
1 1 2 2 2 2 2 4 5 5 5 5 6
1 1 1 1
1 1 1 1 3
2 2 2 2 3
1 1 3 3 4 4 4
```

C- Page count

When you execute a word processor's **print** command, you are normally prompted to specify the pages you want printed. You might, for example, enter:
10-15,25-28,8-4,13-20,9,8-8

The expression you enter is a list of print ranges, separated by commas. Each print range is either a single positive integer, or two positive integers separated by a hyphen. In the latter case we call the first integer **low** and the second one **high**. A print range for which **low > high** is simply ignored. A print range that specifies page numbers exceeding the number of pages is processed so that only the pages available in the document are printed. Pages are numbered starting from 1.
Some of the print ranges may overlap. Pages which are common to two or more print ranges will be printed only once. (In the example given, pages 13, 14 and 15 are common to two print ranges.)

Input:
The input will contain data for a number of problem instances. For each problem instance there will be two lines of input. The first line will contain a single positive integer: the number of pages in the document. The second line will contain a list of print ranges, as defined by the rules stated above. End of input will be indicated by 0 for the number of pages. The number of pages in any book is at most 1000. The list of print ranges will be not be longer than 1000 characters.

Output:
For each problem instance, the output will be a single number, displayed at the beginning of a new line. It will be the number of pages printed by the **print** command.

Sample input:
```
30
10-15,25-28,8-4,13-20,9,8-8
19
10-15,25-28,8-4,13-20,9,8-8
0
```

Sample Output:
```
17
12
```

D- Matryoshka Dolls

Adam just got a box full of Matryoshka dolls (Russian traditional) from his friend Matryona. When he opened the box, tons of dolls poured out of the box with a memo from her: Hi Adam, I hope you enjoy these dolls. But sorry, I didn't have enough time to sort them out. You'll notice that each doll has a hollow hole at the bottom which allows it to contain a smaller doll inside.
...
Yours,
Matryona
Adam, who already has so many things in his showcase already, decides to assemble the dolls to reduce the number of outermost dolls. The dolls that Matryona sent have the same shape but different sizes. That is, doll i can be represented by a single number h_i denoting its height. Doll i can fit inside another doll j if and only if $h_i < h_j$. Also, the dolls are designed such that each doll may contain at most one doll right inside it. While Adam is stacking his gigantic collection of Matryoshka dolls, can you write a program to compute the minimum number of outermost dolls so that he can figure out how much space he needs in his showcase?

Input:
The input consists of multiple test cases. Each test case begins with a line with an integer N, $1 <= N <= 10^5$, denoting the number of Matryoshka dolls. This is followed by N lines, each with a single integer hi, $1 <= h_i <= 10^9$, denoting the height of the ith doll in cm. Input is followed by a single line with N = 0, which should not be processed. For example:

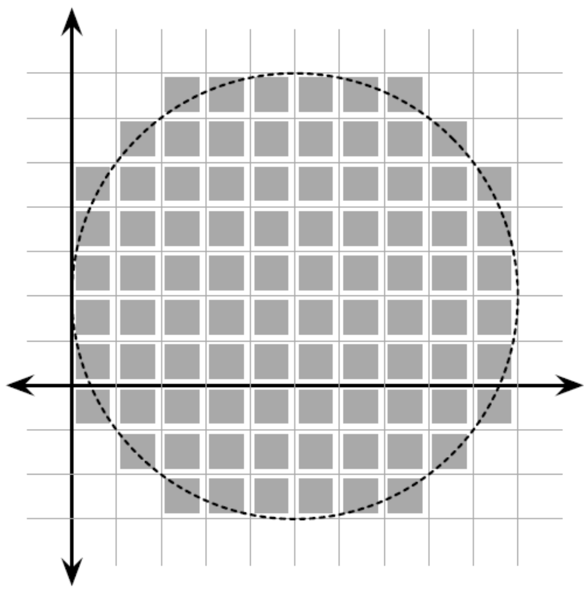
```
4
5
7
7
3
3
10
10
10
3
10
999999999
100000000
0
```

Output:
For each test case, print out a single line that contains an integer representing the minimum number of outermost dolls that can be obtained by optimally stacking the given dolls. For example:

```
2
3
1
```

E- Counting Pixels

Did you know that if you draw a circle that fills the screen on your 1080p high definition display, almost a million pixels are lit? That's a lot of pixels! But do you know exactly how many pixels are lit? Let's find out! Assume that our display is set on a Cartesian grid where every pixel is a perfect unit square. For example, one pixel occupies the area of a square with corners (0,0) and (1,1). A circle can be drawn by specifying its center in grid coordinates and its radius. On our display, a pixel is lit if any part of is covered by the circle being drawn; pixels whose edge or corner are just touched by the circle, however, are not lit.



Your job is to compute the exact number of pixels that are lit when a circle with a given position and radius is drawn.

Input:
The input consists of several test cases, each on a separate line. Each test case consists of three integers, x,y, and r ($1 <= x, y, r <= 10^6$), specifying respectively the center (x, y) and radius of the circle drawn. Input is followed by a single line with x = y = r = 0, which should not be processed. For example:

```
1 1 1
5 2 5
0 0 0
```

Output:
For each test case, output on a single line the number of pixels that are lit when the specified circle is drawn. Assume that the entire circle will fit within the area of the display. For example:

```
4
88
```

F- I love math

Mark loves math so mush especially counting problems that has the famous phrase "In how many ways we can ….", so he can't live a day without solving a counting problem.

Today, while he was searching for a new problem to solve, he found one that looked really hard.

The problem defines a string called "Bit string", a string of only ones and zeroes, and asks: In how many ways we could form a bit string of length n that has no two consecutive zeroes.

For example: the bit strings of length n = 3 are :

000 , 100 , **010** , 001 , **110** , **101** , **011** , **111**
we notice that we have five bit strings of length n = 3 that have no two consecutive zeroes.
Mark needs your help to solve this hard problem!

Input:
The first line has 1 <= T <= 1000, the number of test cases.
The next T lines, each one has 1 <= n <= 1000, the length of the bit string.

Output:
For each test case print the number of bit strings of length n that have no two consecutive zeroes.
The result maybe too large so print the result mod 10^9 + 7.

Sample Input:

```
1
3
```

Sample Output:

```
5
```

G- Overflowing Bookshelf

Agnes C. Mulligan is a fanatical bibliophile – she is constantly buying new books, and trying to find space for those books. In particular, she has a shelf for her “to be read” books, where she puts her newest books. When she decides to read one of these books, she removes it from the shelf, making space for more books.

Sometimes, however, she buys a new book and puts it on the shelf, but because of limited space, this pushes one or more books off the shelf at the other end. She always adds books on the left side of the shelf, making books fall off the right side. Of course, she can remove a book from any location on the shelf when she wants to read one.

Your task will be to write a simulator that will keep track of books added and removed from a shelf. At the end of the simulation, display the books remaining on the shelf, in order from left to right. Books in each simulation will be identified by a unique, positive integer, $0 < I \leq 100$. There are three types of events in the simulation:

Add: A new book is pushed on the left end of the shelf, pushing other books to the right as needed. No book moves to the right unless it is pushed by an adjacent (touching) book on its left. Any book that is not *entirely* on the shelf falls off the right edge. No single book will ever be wider than the given shelf. No book that is currently on the shelf will be added again.

Remove: If the book is on the shelf, then the book is removed from the shelf, leaving a hole. If the book isn't on the shelf, the operation is ignored.

End: End the simulation for this case and print the contents of the shelf.

Input:
The input file will contain data for one or more simulations. The end of the input is signalled by a line containing -1. Each simulation will begin with the integer width of the shelf, s, $5 \leq s \leq 100$, followed by a series of add and remove events. An add event is a single line beginning with an upper case 'A' followed by the book ID, followed by the integer width of the book, w, $0 < w \leq s$. A remove event is a single line beginning with an upper case 'R' followed by the the book ID.
Finally, the end event is a line containing only a single upper case 'E'. Each number in an event is preceded by a single blank.

Output:
For each simulation case, print a single line containing a label (as shown in the output sample), followed by the list of IDs of books remaining on the shelf, in order from left to right.

Example input:

```
10
R 3
A 6 5
A 42 3
A 3 5
A 16 2
A 15 1
R 16
E
7
A 49 6
A 48 2
R 48
E
5
A 1 1
A 2 1
A 3 1
R 2
A 4 1
A 5 1
R 5
R 4
A 6 1
A 7 4
E
-1
```

Example output:

```
PROBLEM 1: 15 3
PROBLEM 2:
PROBLEM 3: 7 6
```

H- Game Rigging

You've decided to host a StarCraft II tournament to decide who is the very best StarCraft II player at Stanford. A lot of your friends are entering, and you want one of them to win because the prize is two tickets to the next StarCraft II World Finals! Luckily, you get to arrange the games and you have some information about which players will beat which other ones, i.e. match-ups which have a guaranteed outcome. The tournament is played as a series of games where in each game two players (of your choice) who have not yet been eliminated play each other. This continues until only one player remains, and he is the winner. Can you set the tournament up so that one of your friends wins?

Input:
The input consists of multiple test cases. Each test case will start with two integers N (2 <= N <= 100000), the number of players in the tournament, K (1 <= K <= N), the number of your friends in the tournament, and M (0 <= M <= 100000), the number of match-ups for which you know the outcome. The next line will contain K integers between 1 and N, indicating which of the players are your friends (indices corresponding to the order of the input). The following M lines will contain two integers each; a line containing A B indicates that if players A and B play each other, player A will always beat player B. Input is followed by a single line with N = K = 0, which should not be processed.

Output:
For each test case, output one line which contains either "yes" or "no" (without quotes) indicating whether you can guarantee that one of your friends wins the tournament.

Sample Input:
```
4 1 3
1
1 2
1 3
2 4
0 0 0
```

Sample Output:
```
yes
```

I- Flag

According to a new ISO standard, a flag of every country should have, strangely enough, a chequered field $n \times m$, each square should be wholly painted one of 26 colours. The following restrictions are set:

- In each row at most two different colours can be used.
- No two adjacent squares can be painted the same colour.

Pay attention, please, that in one column more than two different colours can be used.

Berland's government took a decision to introduce changes into their country's flag in accordance with the new standard, at the same time they want these changes to be minimal. By the given description of Berland's flag you should find out the minimum amount of squares that need to be painted different colour to make the flag meet the new ISO standard.

Input:
The first line has T the number of test cases.
For each test case The first line contains 2 integers n and m $(1 \leq n, m \leq 500)$ — amount of rows and columns in Berland's flag respectively. Then there follows the flag's description: each of the following n lines contains m characters. Each character is a letter from `a` to `z`, and it stands for the colour of the corresponding square.

Output:
For each test case output the minimum amount of squares that need to be repainted to make the flag meet the new ISO standard.

Sample input:
```
1
3 4
aaaa
bbbb
cccc
```

sample output:
```
6
```

J- Lifting the stone

There are many secret openings in the floor which are covered by a big heavy stone. When the stone is lifted up, a special mechanism detects this and activates poisoned arrows that are shot near the opening. The only possibility is to lift the stone very slowly and carefully. The ACM team must connect a rope to the stone and then lift it using a pulley. Moreover, the stone must be lifted all at once; no side can rise before another. So it is very important to find the centre of gravity and connect the rope exactly to that point. The stone has a polygonal shape and its height is the same throughout the whole polygonal area. Your task is to find the centre of gravity for the given polygon.

Input:
The input consists of *T* test cases (equal to about 500). The number of them (*T*) is given on the first line of the input file. Each test case begins with a line containing a single integer *N* ($3 <= N <= 1000000$) indicating the number of points that form the polygon. This is followed by *N* lines, each containing two integers X_i andY_i ($|X_i|, |Y_i| <= 20000$). These numbers are the coordinates of the *i*-th point. When we connect the points in the given order, we get a polygon. You may assume that the edges never touch each other (except the neighboring ones) and that they never cross. The area of the polygon is never zero, i.e. it cannot collapse into a single line.

Output:
Print exactly one line for each test case. The line should contain exactly two numbers separated by one space. These numbers are the coordinates of the centre of gravity. Round the coordinates to the nearest number with exactly two digits after the decimal point (0.005 rounds up to 0.01). Note that the centre of gravity may be outside the polygon, if its shape is not convex. If there is such a case in the input data, print the centre anyway.

Sample Input:
```
2
4
5 0
0 5
-5 0
0 -5
4
1 1
11 1
11 11
1 11
```

sample output:
```
0.00 0.00
6.00 6.00
```