http://www.programming-challenges.com/pg.php?page=studenthome

Python problems

**3n+1 Problem**

Consider the following algorithm to generate a sequence of numbers. Start with an integer *n*. If *n* is even, divide by 2. If *n* is odd, multiply by 3 and add 1. Repeat this process with the new value of *n*, terminating when *n* = 1. For example, the following sequence of numbers will be generated for *n* = 22:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is *conjectured* (but not yet proven) that this algorithm will terminate at *n* = 1 for every integer *n*. Still, the conjecture holds for all integers up to at least 1, 000, 000.

For an input *n*, the *cycle-length* of *n* is the number of numbers generated up to and *including* the 1. In the example above, the cycle length of 22 is 16. Given any two numbers *i* and *j*, you are to determine the maximum cycle length over all numbers between *i* and *j*, *including* both endpoints.

**Input**

The input will consist of a series of pairs of integers *i* and *j*, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

**Output**

For each pair of input integers *i* and *j*, output *i*, *j* in the same order in which they appeared in the input and then the maximum cycle length for integers between and including *i* and *j*. These three numbers should be separated by one space, with all three numbers on one line and with one line of output for each line of input.

**Sample Input**

1 10

100 200

201 210

900 1000

**Sample Output**

1 10 20

100 200 125

201 210 89

900 1000 174

**Minesweeper**

Have you ever played Minesweeper? This cute little game comes with a certain operating system whose name we can't remember. The goal of the game is to find where all the mines are located within a *M* x *N* field.

The game shows a number in a square which tells you how many mines there are adjacent to that square. Each square has at most eight adjacent squares. The 4 x 4field on the left contains two mines, each represented by a ``\*'' character. If we represent the same field by the hint numbers described above, we end up with the field on the right:

|  |
| --- |
| \*...  ....  .\*..  .... |

|  |
| --- |
| \*100  2210  1\*10  1110 |

**Input**

The input will consist of an arbitrary number of fields. The first line of each field contains two integers *n* and *m* ( 0 < *n*, *m*$ \le$100) which stand for the number of lines and columns of the field, respectively. Each of the next *n* lines contains exactly *m* characters, representing the field.

Safe squares are denoted by ``.'' and mine squares by ``\*,'' both without the quotes. The first field line where *n* = *m* = 0 represents the end of input and should not be processed.

**Output**

For each field, print the message Field #*x*: on a line alone, where *x* stands for the number of the field starting from 1. The next *n* lines should contain the field with the ``.'' characters replaced by the number of mines adjacent to that square. There must be an empty line between field outputs.

**Sample Input**

4 4

\*...

....

.\*..

....

3 5

\*\*...

.....

.\*...

0 0

**Sample Output**

Field #1:

\*100

2210

1\*10

1110

Field #2:

\*\*100

33200

1\*100

**Interpreter**

A certain computer has ten registers and 1,000 words of RAM. Each register or RAM location holds a three-digit integer between 0 and 999. Instructions are encoded as three-digit integers and stored in RAM. The encodings are as follows:

|  |  |
| --- | --- |
| 100 | means *halt* |
| 2*dn* | means *set register d to n (between 0 and 9)* |
| 3*dn* | means *add n to register d* |
| 4*dn* | means *multiply register d by n* |
| 5*ds* | means *set register d to the value of register s* |
| 6*ds* | means *add the value of register s to register d* |
| 7*ds* | means *multiply register d by the value of register s* |
| 8*da* | means *set register d to the value in RAM whose address is in register a* |
| 9*sa* | means *set the value in RAM whose address is in register a to the value of register s* |
| 0*ds* | means *goto the location in register d unless register scontains 0* |

All registers initially contain 000. The initial content of the RAM is read from standard input. The first instruction to be executed is at RAM address 0. All results are reduced modulo 1,000.

**Input**

The input begins with a single positive integer on a line by itself indicating the number of cases, each described as below. This is followed by a blank line, and there will be a blank line between each two consecutive inputs.

Each input case consists of up to 1,000 three-digit unsigned integers, representing the contents of consecutive RAM locations starting at 0. Unspecified RAM locations are initialized to 000.

**Output**

The output of each test case is a single integer: the number of instructions executed up to and including the *halt* instruction. You may assume that the program does halt. Separate the output of two consecutive cases by a blank line.

**Sample Input**

1

299

492

495

399

492

495

399

283

279

689

078

100

000

000

000

**Sample Output**

16

**Australian Voting**

Australian ballots require that voters rank all the candidates in order of choice. Initially only the first choices are counted, and if one candidate receives more than 50% of the vote then that candidate is elected. However, if no candidate receives more than 50%, all candidates tied for the lowest number of votes are eliminated. Ballots ranking these candidates first are recounted in favor of their highest-ranked non-eliminated candidate. This process of eliminating the weakest candidates and counting their ballots in favor of the preferred non-eliminated candidate continues until one candidate receives more than 50% of the vote, or until all remaining candidates are tied.

## Input

The input begins with a single positive integer on a line by itself indicating the number of cases following, each as described below. This line is followed by a blank line. There is also a blank line between two consecutive inputs.

The first line of each case is an integer *n*$ \le$20 indicating the number of candidates. The next *n* lines consist of the names of the candidates in order, each up to 80 characters in length and containing any printable characters. Up to 1,000 lines follow, each containing the contents of a ballot. Each ballot contains the numbers from 1 to *n* in some order. The first number indicates the candidate of first choice; the second number indicates candidate of second choice, and so on.

## Output

The output of each test case consists of either a single line containing the name of the winner or several lines containing the names of all candidates who are tied. The output of each two consecutive cases are separated by a blank line.

## Sample Input

1

3

John Doe

Jane Smith

Jane Austen

1 2 3

2 1 3

2 3 1

1 2 3

3 1 2

## Sample Output

John Doe

**HARTALS**

Political parties in Bangladesh show their muscle by calling for regular *hartals* (strikes), which cause considerable economic damage. For our purposes, each party may be characterized by a positive integer *h* called the *hartal parameter* that denotes the average number of days between two successive strikes called by the given party.

Consider three political parties. Assume *h*1 = 3, *h*2 = 4, and *h*3 = 8, where *h*i is the hartal parameter for party *i*. We can simulate the behavior of these three parties for*N* = 14 days. We always start the simulation on a Sunday. There are no hartals on either Fridays or Saturdays.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Days | Su | Mo | Tu | We | Th | Fr | Sa | Su | Mo | Tu | We | Th | Fr | Sa |
| Party 1 |  |  | x |  |  | x |  |  | x |  |  | x |  |  |
| Party 2 |  |  |  | x |  |  |  | x |  |  |  | x |  |  |
| Party 3 |  |  |  |  |  |  |  | x |  |  |  |  |  |  |
| Hartals |  |  | 1 | 2 |  |  |  | 3 | 4 |  |  | 5 |  |  |

There will be exactly five hartals (on days 3, 4, 8, 9, and 12) over the 14 days. There is no hartal on day 6 since it falls on Friday. Hence we lose five working days in two weeks.

Given the hartal parameters for several political parties and the value of *N*, determine the number of working days lost in those *N* days.

**Input**

The first line of the input consists of a single integer *T* giving the number of test cases to follow. The first line of each test case contains an integer *N* ( 7$ \le$*N*$ \le$3, 650), giving the number of days over which the simulation must be run. The next line contains another integer *P* ( 1$ \le$*P*$ \le$100) representing the number of political parties. The *i*th of the next *P* lines contains a positive integer *h*i (which will never be a multiple of 7) giving the *hartal parameter* for party *i* ( 1$ \le$*i*$ \le$*P*).

**Output**

For each test case, output the number of working days lost on a separate line.

**Sample Input**

2

14

3

3

4

8

100

4

12

15

25

40

**Sample Output**

5

15

**Contest Scoreboard**

Want to compete in the ACM ICPC? Then you had better know how to keep score! Contestants are ranked first by the number of problems solved (the more the better), then by decreasing amounts of penalty time. If two or more contestants are tied in both problems solved and penalty time, they are displayed in order of increasing team numbers.

A problem is considered solved by a contestant if any of the submissions for that problem was judged correct. Penalty time is computed as the number of minutes it took until the first correct submission for a problem was received, plus 20 minutes for each incorrect submission prior to the correct solution. Unsolved problems incur no time penalties.

**Input**

The input begins with a single positive integer on a line by itself indicating the number of cases, each described as below. This line is followed by a blank line. There is also a blank line between two consecutive inputs.

The input consists of a snapshot of the judging queue, containing entries from some or all of contestants 1 through 100 solving problems 1 through 9. Each line of input consists of three numbers and a letter in the format *contestant problem time L*, where *L* can be C, I, R, U, or E. These stand for Correct, Incorrect, clarificationRequest, Unjudged, and Erroneous submission. The last three cases do not affect scoring.

The lines of input appear in the order in which the submissions were received.

**Output**

The output for each test case will consist of a scoreboard, sorted by the criteria described above. Each line of output will contain a contestant number, the number of problems solved by the contestant and the total time penalty accumulated by the contestant. Since not all contestants are actually participating, only display those contestants who have made a submission.

The output of two consecutive cases will be separated by a blank line.

**Sample Input**

1

1 2 10 I

3 1 11 C

1 2 19 R

1 2 21 C

1 1 25 C

**Sample Output**

1 2 66

3 1 11

**File Fragmentation**

Your friend, a biochemistry major, tripped while carrying a tray of computer files through the lab. All of the files fell to the ground and broke. Your friend picked up all the file fragments and called you to ask for help putting them back together again.

Fortunately, all of the files on the tray were identical, all of them broke into exactly two fragments, and all of the file fragments were found. Unfortunately, the files didn't all break in the same place, and the fragments were completely mixed up by their fall to the floor.

The original binary fragments have been translated into strings of ASCII 1's and 0's. Your job is to write a program that determines the bit pattern the files contained.

**Input**

The input begins with a single positive integer on its own line indicating the number of test cases, followed by a blank line. There will also be a blank line between each two consecutive cases.

Each case will consist of a sequence of ``file fragments,'' one per line, terminated by the end-of-file marker or a blank line. Each fragment consists of a string of ASCII1's and 0's.

**Output**

For each test case, the output is a single line of ASCII 1's and 0's giving the bit pattern of the original files. If there are 2*N* fragments in the input, it should be possible to concatenate these fragments together in pairs to make *N* copies of the output string. If there is no unique solution, any of the possible solutions may be output.

Your friend is certain that there were no more than 144 files on the tray, and that the files were all less than 256 bytes in size.

The output from two consecutive test cases will be separated by a blank line.

**Sample Input**

1

011

0111

01110

111

0111

10111

**Sample Output**

01110111

**Doublets (Dictionary exercise)**

**Common Permutation (loops, string matching)**

Given two strings *a* and *b*, print the longest string *x* of letters such that there is a permutation of *x* that is a subsequence of *a* and there is a permutation of *x* that is a subsequence of *b*.

**Input**

The input file contains several cases, each case consisting of two consecutive lines. This means that lines 1 and 2 are a test case, lines 3 and 4 are another test case, and so on. Each line contains one string of lowercase characters, with first line of a pair denoting *a* and the second denoting *b*. Each string consists of at most 1,000 characters.

**Output**

For each set of input, output a line containing *x*. If several *x* satisfy the criteria above, choose the first one in alphabetical order.

**Sample Input**

pretty

women

walking

down

the

street

**Sample Output**

e

nw

et

|  |  |
| --- | --- |
| |  | | --- | | **Where's Waldorf?** | |

**Longest Nap**