



SPEED PROGRAMMING COMPETITION

Sample Problems

Problem #1:

Code Numbers

(this problem have a large and a small input data set) (more details provided in rules)

The decimal numeral system is composed of ten digits, which we represent as "0123456789" (the digits in a system are written from lowest to highest). Imagine you have discovered an code numeral system composed of some number of digits, which may or may not be the same as those used in decimal. For example, if the code numeral system were represented as "oF8", then the numbers one through ten would be (F, 8, Fo, FF, F8, 8o, 8F, 88, Foo, FoF). We would like to be able to work with numbers in arbitrary code systems. More generally, we want to be able to convert an arbitrary number that's written in one code system into a second code system.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each case is a line formatted as

code_number source_language target_language

Each language will be represented by a list of its digits, ordered from lowest to highest value. No digit will be repeated in any representation, all digits in the code number will be present in the source language, and the first digit of the code number will not be the lowest valued digit of the source language (in other words, the code numbers have no leading zeroes). Each digit will either be a number 0-9, an uppercase or lowercase letter, or one of the following symbols !"#\$%&'()*+,-./:;<=>?@[\\]^_`{|}~

Output

For each test case, output one line containing "Case #x: " followed by the code number translated from the source language to the target language.

Limits

Small dataset

$1 \leq N \leq 30$.

Large dataset

$30 \leq N \leq 100$.

Sample:

Small dataset input

Input	Output
4	
9 0123456789 oF8	Case #1: Foo
Foo oF8 0123456789	Case #2: 9
13 0123456789abcdef 01	Case #3: 10011
TECH E!TCH? L?EI!	Case #4: ELI!

Large input dataset attached in other file. (another small dataset input is attached)

Problem#2:

Safebreaker

We are observing someone playing the game of Mastermind. The object of this game is to find a secret code by intelligent guess work, assisted by some clues. In this case the secret code is a 4-digit number in the inclusive range from 0000 to 9999, say ``3321''. The player makes a first random guess, say ``1223'' and then, as for each of the future guesses, gets a clue telling him how good his guess is. A clue consists of two numbers: the number of correct digits (in this case 1: the ``2'' at the third position) and the additional number of digits guessed correctly but in the wrong place (in this case 2: the ``1'' and the ``3''). The clue would in this case be: ``1/2''.

Write a program that given a set of guesses and corresponding clues, tries to find the secret code

Input:

The first line of input specifies the number of test cases (N) your program has to process. Each test case consists of a first line containing the number of guesses G ($0 \leq G \leq 10$), and G subsequent lines consisting of exactly 8 characters: a code of four digits, a blank, a digit indicating the number of correct digits, a `/' and a digit indicating the number of correct but misplaced digits.

Output:

For each test case, the output contains a single line saying either:

- * impossible if there is no code consistent with all guesses.
- * the secret code if there is exactly one code consistent with all guesses.
- * indeterminate if there is more than one code which is consistent with all guesses

Sample:

Input:

```
4
6
9793 0/1
2384 0/2
6264 0/1
3383 1/0
2795 0/0
0218 1/0
1
```

1234 4/0

1

1234 2/2

2

6428 3/0

1357 3/0

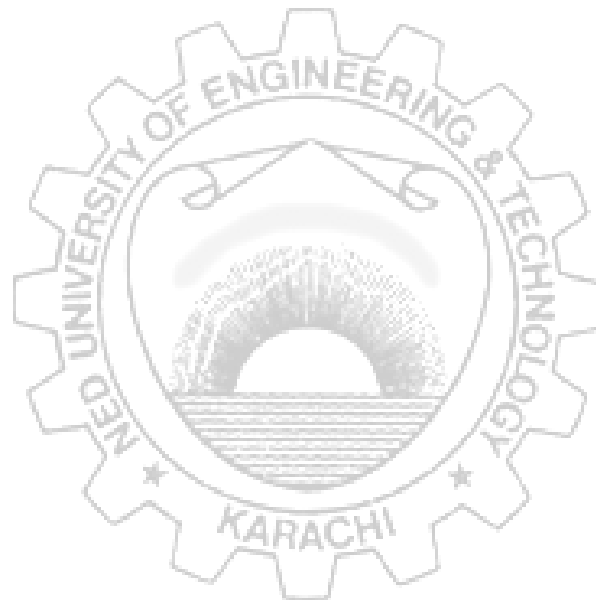
Output:

3411

1234

indeterminate

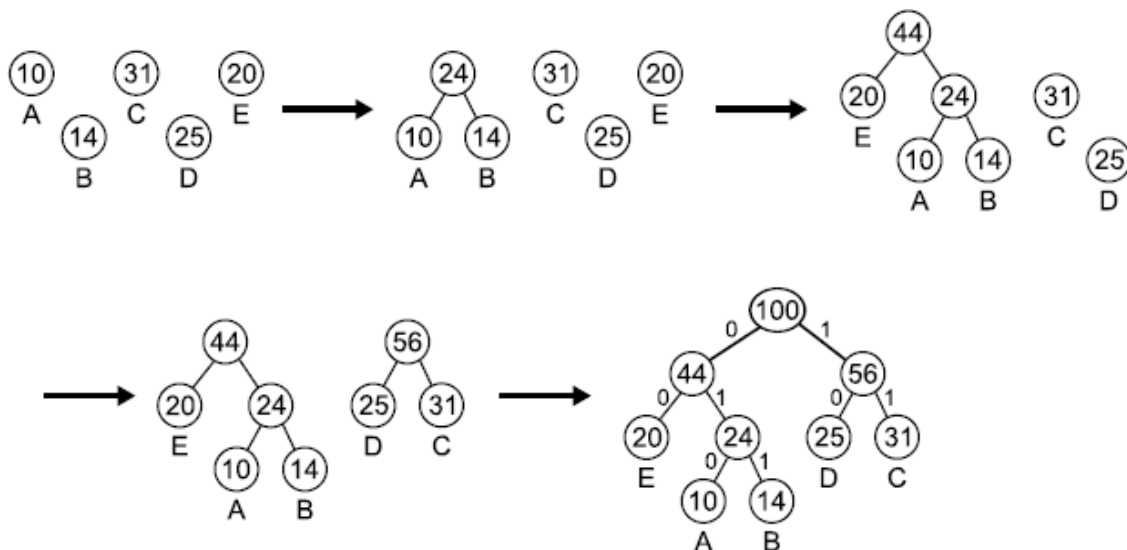
impossible



Problem # 3

Huffman Codes

Dan McAmbi is a member of a crack counter-espionage team and has recently obtained the partial contents of a file containing information vital to his nation's interests. The file had been compressed using Huffman encoding. Unfortunately, the part of the file that Dan has shows only the Huffman codes themselves, not the compressed information. Since Huffman codes are based on the frequencies of the characters in the original message, Dan's boss thinks that some information might be obtained if Dan can reverse the Huffman encoding process and obtain the character frequencies from the Huffman codes. Dan's gut reaction to this is that any given set of codes could be obtained from a wide variety of frequency distributions, but his boss is not impressed with this reasoned analysis. So Dan has come to you to get more definitive proof to take back to his boss. Huffman encoding is an optimal data compression method if you know in advance the relative frequencies of letters in the text to be compressed. The method works by first constructing a Huffman tree as follows. Start with a forest of trees, each tree a single node containing a character from the text and its frequency (the character value is used only in the leaves of the resulting tree). Each step of the construction algorithm takes the two trees with the lowest frequency values (choosing arbitrarily if there are ties), and replaces them with a new tree formed by joining the two trees as the left and right subtrees of a new root node. The frequency value of the new root is the sum of the frequencies of the two subtrees. This procedure repeats until only one tree is left. An example of this is shown below, assuming we have a file with only 5 characters – A, B, C, D and E – with frequencies 10%, 14%, 31%, 25% and 20%, respectively.



After you have constructed a Huffman tree, assign the Huffman codes to the characters as follows. Label each left branch of the tree with a 0 and each right branch with a 1. Reading down from the root to each character gives the Huffman code for that

character. The tree above results in the following Huffman codes: A - 010, B - 011, C - 11, D - 10 and E - 00. For the purpose of this problem, the tree with the lower frequency always becomes the left subtree of the new tree. If both trees have the same frequencies, either of the two trees can be chosen as the left subtree. Note that this means that for some frequency distributions, there are several valid Huffman encodings. The same Huffman encoding can be obtained from several different frequency distributions: change 14% to 13% and 31% to 32%, and you still get the same tree and thus the same codes. Dan wants you to write a program to determine the total number of distinct ways you could get a given Huffman encoding, assuming that all percentages are positive integers. Note that two frequency distributions that differ only in the ordering of their percentages (for example 30% 70% for one distribution and 70% 30% for another) are not distinct.

Input

The input consists of several test cases. Each test case consists of a single line starting with a positive integer n ($2 \leq n \leq 20$), which is the number of different characters in the compressed document, followed by n binary strings giving the Huffman encoding of each character. You may assume that these strings are indeed a Huffman encoding of some frequency distribution (though under our additional assumptions, it may still be the case that the answer is 0 – see the last sample case below). The last test case is followed by a line containing a single zero.

Output

For each test case, print a line containing the test case number (beginning with 1) followed by the number of distinct frequency distributions that could result in the given Huffman codes.

Sample Input	Output for the Sample Input
<pre>5 010 011 11 10 00 8 00 010 011 10 1100 11010 11011 111 8 1 01 001 0001 00001 000001 0000001 0000000 0</pre>	<pre>Case 1: 3035 Case 2: 11914 Case 3: 0</pre>