HKUST Programming Contest 2004

hosted by
HKUST Programming Team

April 17, 2004

Acknowledgements

The HKUST Programming Team would like to thank the following people for their helps in the HKUST Programming Contest 2004.

Prof. Lionel Ni, department head
Dr. CK Tang, coach of HKUST Programming Team

Judges
Roger Hsiao
Ben Lau

Promotion and graphics design
William Wong

Technical support
Winnie Tse, CSSystem

Yiu-Cho Leung
Martin Ma
HKUST Programming Contest 2004 Coordinators

Contents

Problem A

Ranks

Input: (*stdin*)
Output: (*stdout*)
Time limit: 10 sec(s)

One day, Martin came to Dr. Tang's office.

"Could I know the results of the final exam in advance?" asked Martin.

Dr. Tang knew that Martin is too nervous about his exam results. Although Martin performs quite well in the class, his math and programming background is actually quite limited. So Dr. Tang wants to give him some challenges.

"I'm not going to telling you your exact rank," said Dr. Tang, "but I can tell you who and who have their midterm and final *positions inverted*."

"Sorry," Martin was confused, "but what do you mean by that?"

"That means, say, if students A and B have their midterm and final positions inverted, then A having a higher midterm score than B will imply A has a lower final score than B." Just after Dr. Tang finished his words, he gave Martin a list of such information.

Poor Martin really doesn't know how to work with all these data. So eager to know the final results immediately, he now asks for your help.

Input

In each data block, the first line contains an integer N ($1 < N \leq 1000$) indicating the number of students. It is then followed by N lines, where the ith line contains the name of the student ranked i in the midterm. A name consists of at most 30 alphanumeric characters.

The next line contains an integer K, the number of inverted pairs. Each of the next K lines contains a distinct pair of integers i and j ($1 \leq i, j \leq N$, $i \neq j$), meaning that the students ranked i and j in ther midterm have their positions inverted in the final. A test case with $N = 0$ ends the input, and this test case should not be processed.

Output

For the t-th test case, first print a line "`Case t:`". Then print the names of the students sorted by their final ranks, one name in each line; or print a line

"`Dr. Tang is fooling me!`"

if a valid ordering cannot be deduced from the data.

Examples

Sample Input	Sample Output
5 Joseph Roger Lion Martin Ben 3 1 4 2 4 3 4 5 Joseph Roger Lion Martin Ben 1 1 4 0	Case 1: Martin Joseph Roger Lion Ben Case 2: Dr. Tang is fooling me!

Problem B

Fingerings

Input: (*stdin*)
Output: (*stdout*)
Time limit: 10 sec(s)

The violin is a stringed musical instrument comprising four strings. Pitch is controlled by pressing the string down onto the *fingerboard* (Fig. B.1) with one of the fingers of the left hand.

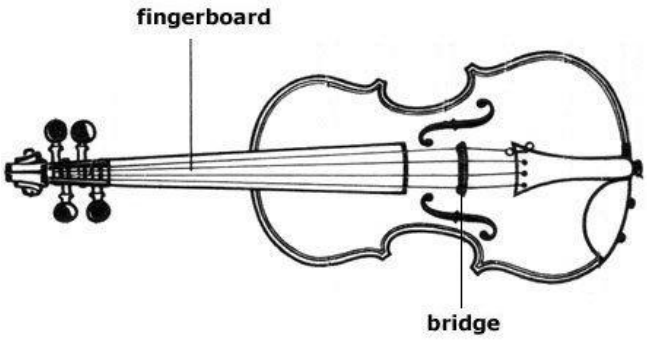


Figure B.1: A violin and its fingerboard

Table B.1 shows all the possible *stops* (places where fingers can press on) on the strings named E,A,D,G and the correponding *pitches* made when they are pressed. O means *open string* – pressed by no fingers.

Stop:	O	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	XIV
E	E_5	F_5	G_5	A_5	B_5	C_6	D_6	E_6	F_6	G_6	A_6	B_6	C_7	D_7	E_7
A	A_4	B_4	C_5	D_5	E_5	F_5	G_5	A_5	B_5	C_6	D_6	E_6	F_6	G_6	A_6
D	D_4	E_4	F_4	G_4	A_4	B_4	C_5	D_5	E_5	F_5	G_5	A_5	B_5	C_6	D_6
G	G_3	A_3	B_3	C_4	D_4	E_4	F_4	G_4	A_4	B_4	C_5	D_5	E_5	F_5	G_5

Bridge

Table B.1: Fingerboard positions

Fingers are conventionally numbered from 1 to 4, the index finger being the first and the little finger the fourth. "No fingers" is conventionally indexed by 0. The

left hand moving along the fingerboard can be thought as a window of width 4 moving along the rows of table B.1. When the left hand is at *position* s, only stops O, s, $s+1$, $s+2$ and $s+3$ are available, which are played by finger 0,1,2,3 and 4 respectively. If a target stop is unreachable from the current hand position, the hand must be *shifted* to another position. Shifting position accurately is a difficult task for violinists.

Notice that most pitches can be produced in more than one way. For example, A_4 can be produced by pressing stop VIII on the G string, IV on the D string, or O on the A string (i.e. open string). Usually one will choose a way which does not require position shift. But it is not always true. For example, jumping from string G to string E is not prefered because the bow can easily touch the inner strings D,A that cause unwanted sounds. Moreover, sounds made by open strings are too sharp and less desirable. The term *fingerings* refers to the instructions on a *melody* (a sequence of pitches) indicating which string to be selected and what finger to be used to play a pitch.

In this problem you are asked to find, given a melody, the *best* set of fingerings that minimizes the sum of all *badnesses* of all undesirable fingerings appeared (Table B.2):

Undesirable fingering	Badness
For each single note:	
Played at position $pos >$ IV	$(pos - 4) \times 5$
Played by open string	10
Played by finger 4 (little finger)	5
For each consecutive pairs of notes:	
Need a shift from position pos_1 to pos_2	$(pos_1 - pos_2)^2 + 5$
Need a jump between strings D and E / G and A	25
Need a jump between strings G and E	50

Table B.2: Undesirable fingering patterns and their badnesses

Input

Input starts with a single line containing an integer t, the number of melodies. Each of the following t lines contains a melody. A melody begins with an integer $n \leq 500$, the number of pitches, followed by n pitches from the set {G3...E7}. Pitches are separated by spaces.

Output

Print 2 lines for each melody. The first line prints the badness of the best fingerings. The second line prints the best fingerings, where each fingering is of the form $s_i f_i$, where $s_i \in \{E, A, D, G\}$ and $f_i \in \{0, 1, 2, 3, 4\}$. Add one space between two fingerings. If there are more than one set of best fingerings, print the one with smaller *dictionary* order, with $G < D < A < E$ and $0 < 1 < 2 < 3 < 4$.

Examples

Sample Input	**Sample Output**
2 8 G3 A3 B3 C4 D4 E4 F4 G4 9 G3 C4 D4 E4 C4 D4 E4 F4 E4	15 G0 G1 G2 G3 G4 D1 D2 D3 15 G0 G1 G2 G3 G1 G2 G3 G4 G3

Problem C

SimTEX

Input: (*stdin*)
Output: (*stdout*)
Time limit: 10 sec(s)

SimTEX (simplified TEX) is a *macro* based typesetting language: every command expands into a list of other commands or text, and the other commands are then expanded in turn until everything is fully expanded. In this problem you are asked to implement the SimTEX processor that, given lines of commands and text, generate the fully expanded text.

The SimTEX language allows two kinds of characters: (i) *Control* characters: `\`, `{` and `}`. (i) *Text* characters: `A-Z`, `a-z` and space. SimTEX commands start with a `\`. Two build-in commands are: `\def`, which *defines* a command; and `\bye`, which ends the input. Users build their own commands by calling `\def`:

```
\def\cmd{<replacement text>}
```

where `cmd` is the command name containing only `A-Z` and `a-z`, and `<replacement text>` is a string containing commands or text that you want to substitute `\cmd` with. For example, the line

```
\def\martin{MaRTiN tHe CoMPoSeR}
```

will define the command `\martin` so that, when being called, will be expanded to the string "`MaRTiN tHe CoMPoSeR`". Commands other than `\def` and `\bye` can appear in the replacement text also. For example, the lines

```
\def\team{HKUST Programming Team}
\def\coach{\martin is the coach of \team}
\coach
```

will yield the string

"`MaRTiN tHe CoMPoSeR is the coach of HKUST Programming Team`"

No spaces will appear outside `{...}`. Inside `{...}`,

1. Two *tokens* (commands or words) are separated by one or more spaces.
2. Two of more *consecutive* spaces are considered as one space in the output, while leading and trailing spaces are ignored;
3. A `\` will take up consecutive characters of `A-Z` and `a-z` immediately follow it to form the command name;
4. No further `{` and `}` will appear inside `{...}`.

Lastly, defined commands can be redefined. Expansion will choose the most recent definition.

Input

The input file contains at most 100 SimTEX command lines. Each line has at most 80 characters. You may assume that all the lines are syntactically correct. The input ends with a single line containing the `\bye` command.

Output

For each command except `\def` and `\bye`, print the fully expanded text, or print the line "`<ERROR>`" if (i) an undefined command is met during the expansion, or (ii) the command expands to an infinitely long string. See the examples.

Examples

Sample Input

```
\def\cs{Computer Science}
\def\csprof{ professor \prof from department of \cs }
\csprof
\def\prof{ Chi Keung  Tang}
\csprof
\def\profwho{He is \csprof }
\def\prof{CK Tang}
\profwho
\def\prof{\csprof}
\profwho
\bye
```

Sample Output

```
<Error>
professor Chi Keung Tang from department of Computer Science
He is professor CK Tang from department of Computer Science
<Error>
```

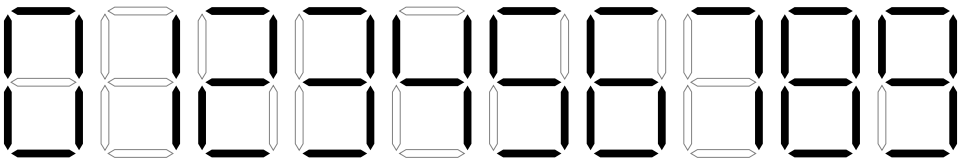
Hong Kong University of Science and Technology Programming Contest 2004

Problem D

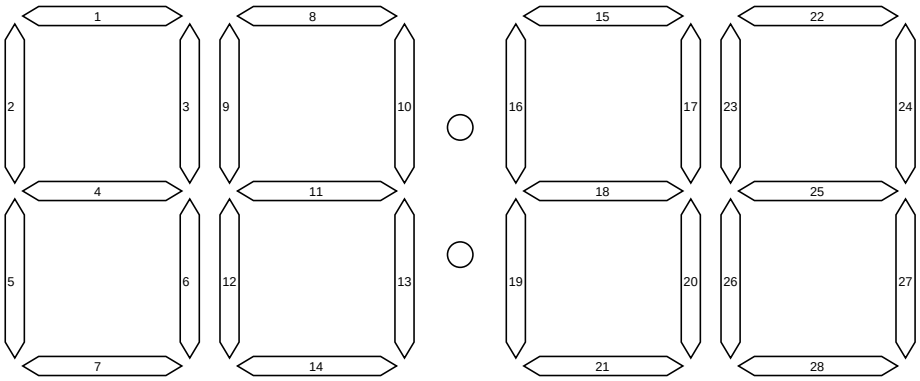
Digital Clock

Input: (*stdin*)
Output: (*stdout*)
Time limit: 10 sec(s)

We are all familiar with digital clock. Digital clock presents digits by different on/off states of a group of electronic diodes.



All the diodes are now assigned with unique ids from 1 to 28 as shown in the following figure.



Suppose now I give you a malfunctioned clock where some diodes are burnt. By looking at several snapshots of the clock, it is possible to figure out which diodes are failed for sure. If a diode is failed, it will always be in OFF state.

Figure D.1 shows three snapshots of a digital clock where diode 10, 18, 23 are failed. Figure D.2 shows one snapshot of a digital clock where diode 5, 7, 21 are

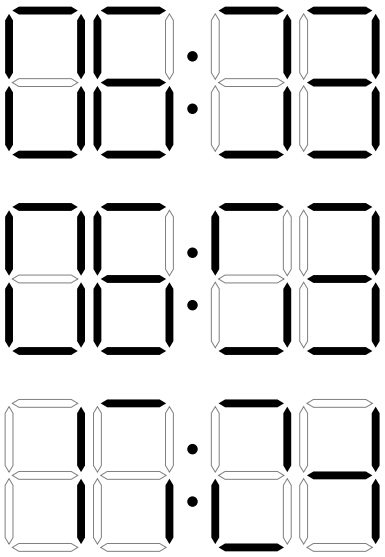


Figure D.1: Diode 10, 18, 23 are failed.

failed. Notice that there may be some other failed diodes as well, but we only know that diode 5, 7, 21 are failed for sure in this case.

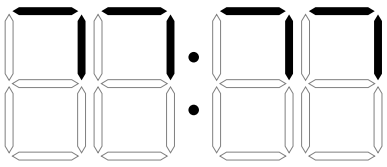


Figure D.2: Diode 5, 7, 21 are failed

Input

The input file contains an integer n in the first line. n input test cases are then followed. For each test case, the first line is an integer m ($1 \leq m \leq 10$), the number of snapshots. It is then followed by m lines, each describing a snapshot that consists of 28 digits of either 0 or 1. The i-th digit in the j-th line indicates whether the i-th diode of the j-th snapshot is ON (1) or OFF (0). All time displays adopt 24-hour system (00:00-23:59).

Output

For each test case, print exactly one line of output. If there is one or more diodes known to be failed, print their ids in ascending order. Otherwise, print the line "No diode is found to be failed.". If the test case is showing some impossible snapshot, print out "Impossible.".

Examples

Sample Input

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

Sample Output

```
10 18 23
5 7 21
No diode is found to be failed.
Impossible.
```

Problem E

Cough syrup

Input: (*stdin*)
Output: (*stdout*)
Time limit: 10 sec(s)

Robitmartin™ is a brand of cough-curing medicine. Under this brand, there are many kinds of cough syrups, each one having different focus and strength of curing power. However, they actually contain the same kinds of active ingredients, just the amounts are different. The two main ingredients of its cough syrups are:

1. **Dextromethorphan hydrobromide (DMH)**, which selectively depresses the cough center in the medulla, and
2. **Guaifenesin (Guai)**, which increases the output of fluid of the respiratory tract by reducing the viscosity and surface tension of respiratory secretions, thereby facilitating their expectoration.

For example, Table E.1 shows several kinds of Robitmartin™ cough syrups and the amounts of ingredients they contain.

Name	DMH(mg/5mL)	Guai(mg/5mL)
Daytime	10	100
Nighttime	7	0
Maximum Strength	15	100
Pediatric (*for children*)	7	50

Table E.1: Different kinds of cough syrups and their ingredients amounts

Given several bottles of Robitmartin™ cough syrups, how many of them are needed to mix one bottle of target dosage?

Input

Each test case starts with two integers $n < 2000$, the number of available dosages, and $t < 100$, the number of target dosages. They are followed by $n + t$ lines, each containing a pair of integers d and g ($d < 100$, $g < 500$), indicating a dosage of dextromethorphan hydrobromide and guaifenesin, in unit of mg/5mL. First n of them are the available ones, and the rest of them are the target ones. All bottles have volume of 120mL, and should be completely filled with syrups. A test case with $n = t = 0$ ends the input and it should not be processed.

Output

For each target dosage, print the minimum number of bottles that it can be mixed from the available syrups, or print "`NA`" if it can not be mixed from the available ones.

Examples

Sample Input	Sample Output
2 2 10 100 7 0 10 100 10 0 0 0	1 NA

Problem F

Key Agreement Protocol

Input: (*stdin*)
Output: (*stdout*)
Time limit: 10 sec(s)

The *DH key agreement protocol* (also called exponential key agreement) was developed by Diffie and Hellman in 1976 and published in the ground-breaking paper "New Directions in Cryptography." The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.

The protocol has two system parameters p and g. They are both public and may be used by all the users in a system. Parameter p is a prime number and parameter g (usually called a *generator*) is an integer less than p, with the following property: for every integer n between 1 and $p-1$ inclusive, there is a power x of g such that $n = g^x \mod p$. It is assured that, for any prime p, such a g always exists.

Suppose Alice and Bob want to agree on a shared secret key using the DH key agreement protocol. They proceed as follows: First, Alice generates a random private value a and Bob generates a random private value b. Both a and b are drawn from the set of integers $\{1, ..., p-2\}$. Then they derive their public values using parameters p and g and their private values. Alice's public value is $g^a \mod p$ and Bob's public value is $g^b \mod p$. They then exchange their public values. Finally, Alice computes $k_a = (g^b)^a \mod p$, and Bob computes $k_b = (g^a)^b \mod p$. Since $k_a = k_b = g^{ab} \mod p$, Alice and Bob now have a shared secret key $k = g^{ab} \mod p$.

The security of this protocol depends on the fact that, given numbers p, g and n, it is computationally infeasible to find x such that $n = g^x \mod p$ when the prime p is sufficiently large.

In this problem you are asked to implement this protocol.

Input

The first integer t indicates the number of test cases. Each test case consists of 4 positive integers p, g, a and $(g^b \mod p)$, where $p < 2^{128}$ is a prime number and $g < p$; and $a < p - 1$ is the random number you generated (yes, you are Alice).

Output

For each test case, print two lines:

```
Send < x > to Bob.
Calculate key = < y >.
```

where $< x >$ is your public value and $< y >$ is the key. Your algorithm must be very efficient in order to pass the time limit.

Examples

Sample Input	Sample Output
1 11 2 5 8	Send 10 to Bob. Calculate key = 10.

End of Problem Set