University of Scranton
ACM Student Chapter / Computing Sciences Department
13th Annual High School Programming Contest (2003)

Problem 1: Min/Max Search

Develop a program that, given a sequence S of integers as input, produces as output two sequences of positive integers, the first of which indicates all those positions in S at which S's minimum value occurs and the second of which indicates all those positions at which S's maximum value occurs. Positions are numbered starting at one (1).

Input

The first line of input contains a positive integer m indicating how many sequences are to be processed. The next $2m$ lines contain descriptions of the m sequences, two lines per sequence. The first line of each description contains a positive integer n ($n \leq 40$) indicating the length of (i.e., number of values in) the sequence, and the second line contains the values in the sequence. Each such value is separated from the next by at least one space.

Output

For each sequence given as input, there should be four lines of output. The first line echos the given sequence. The second line indicates the positions at which the minimum value occurs. The third line indicates the positions at which the maximum value occurs. The fourth line is blank.

Sample input

```
3
7
3 6 -1 4 6 5 3
4
0 0 0 0
14
-4 45 2 0 3 5 11 -7 854 25 3 -7 4 -3
```

Corresponding output

```
3 6 -1 4 6 5 3
3
2 5

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

-4 45 2 0 3 5 11 -7 854 25 3 -7 4 -3
8 12
9
```

Problem 2: Follow the Yellow Brick Road

Consider the following, which is intended to depict a string M =`OELRO!LWLH D` with characters occupying positions one through twelve and a sequence S of integers whose elements occupy positions zero through twelve. M is a scrambled message and S, if interpreted properly, provides a "roadmap" for unscrambling M.

```
 0  1  2  3  4  5  6  7  8  9  10  11  12
-----------------------------------------
    O  E  L  R  O  !  L  W  L   H       D
10 10  5 -2  5 -1  0 -4 -3  3  -8  -3  -6
```

The zero-th element of S, 10, tells us that we should begin at position ten. As `H` occupies position ten of M, it is the first character in the unscrambled message. As -8 occupies position ten of S, we move eight places to the left (i.e., to position two) in order to find the next character, which is `E`. We find that 5 occupies position two of S; hence we move five positions to the right (i.e., to position seven) to find the next character, which is `L`. We continue in this fashion until reaching a position of S occupied by 0; the character in the corresponding position of M is the last character in the unscrambled message. For this example, the unscrambled message you obtain is `HELLO WORLD!`.

Develop a program that, given as input some scrambled messages together with integer sequences to be used as roadmaps in unscrambling them, outputs the unscrambled messages.

Input

The first line of input contains a positive integer n indicating how many scrambled messages are to be unscrambled. The following $3n$ lines contain descriptions of the scrambled messages and the roadmaps for unscrambling them. Each such description occupies three lines, the first containing a positive integer m ($m \leq 40$) indicating the length of the message, the second containing the message itself (a string of length m), and the third containing a sequence of $m + 1$ integers to be used for unscrambling.

You may assume that the input data is such that the unscrambling process, as described above, will never lead you to a position that is "out of bounds" (i.e., less than one or greater than m). You may also assume that the input data will not cause this process to enter an infinite loop.

Output

For each scrambled message, simply display the string obtained by unscrambling it. Note that it is possible for the unscrambled message to be shorter than the scrambled one. (See sample below.)

Sample input

```
3
12
OELRO!LWLH D
10 10 5 -2 5 -1 0 -4 -3 3 -8 -3 -6
5
TODGB
4 2 0 -1 -2 0
28
Y COFLWR L LDIBHLEOKR OAEOWT
5 24 13 17 20 18 13 15 6 12 16 -10 -6 0 -11 -7 2 -7 -7 -12 -11 -17 6 -11 -11 -8 1 -25 -12
```

Corresponding output

```
HELLO WORLD!
GO
FOLLOW THE YELLOW BRICK ROAD
```

Problem 3: L-I-N-G-O

A current TV game show asks contestants to guess at hidden five-letter words. Contestants are told the first letter of the word and are allowed to make up to five guesses at the word. Feedback is provided in response to each guess, as contestants are told which letters in the guessed word also occur in the hidden word and, for each such letter, whether it occurs in the same position in the two words.

Develop a program that, given as input a number of "games", each described by a hidden five-letter word followed by some guesses (all of which are five letters in length), produces as output the appropriate feedback for each guess, plus the "initial feedback" that is given before the first guess is made. The feedback for a guess is the guess itself (in upper case), except that any letter occurring in the guess that fails to occur in the hidden word is replaced by a period (`.`) and any letter occurring in the guess that occurs in the hidden word, *but not in the same position*, is replaced by the lower case version of that letter.

As an example, suppose that the hidden word starts with `A`. Then the initial feedback would be `A....` . Suppose that our first guess is `APPLE`, which (aside from the first letter) is similar to the hidden word only in that they both contain `L`, but in different positions. The resulting feedback would be `A..l.` . Looking for more help, we next try the word `AMIGO`, whose only similarity to the hidden word (aside from the first letter) is that `I` appears in the same position. The resulting feedback is `A.I..` . Working with the `I`, we next try the word `ASIAN`. The `I` is still correctly placed, but now so too is the (second) `A`. In addition, `S` occurs in both words, but in different positions. Hence, we get as feedback `AsIA.` . From these clues, plus the clue regarding `L` obtained as a result of the first guess, it follows that the hidden word is `ALIAS`, which we make our fourth guess. The resulting feedback is `ALIAS` (indicating an exact match), which ends the game.

Input

The first line of input contains a positive integer n indicating how many games are to be played. Each game is described by a "hidden" five-letter word, followed by at most five guesses, each of which is a five-letter word. Each word appears on a line by itself, left justified and entirely in upper case. Any guess that is correct is the last guess of that game. If no guess is correct, the last guess is the fifth one.

The last guess of one game is separated from the hidden word of the next game by a blank line containing at least one space.

Output

The first line of output for each game contains the "initial feedback", which is the first letter of the hidden word followed by four periods. Each subsequent line of output for that game contains the feedback in response to the corresponding guess (as described above). A single blank line should appear between the outputs for two consecutive games.

Sample input

```
3
ALIAS
APPLE
AMIGO
ASIAN
ALIAS

LAUGH
LAKES
LUGER
LIGHT
LINGO
LAUGH

SWAMP
SOLVE
STUDY
SLIPS
SPATE
SWARM
```

Corresponding output

```
A....
A..l.
A.I..
AsIA.
ALIAS

L....
LA...
Lug..
L.gh.
L..G.
LAUGH

S....
S....
S....
S..ps
SpA..
SWA.m
```

Problem 4: Goldbach's Conjecture

A *prime* number is a positive integer having exactly two divisors among the positive integers (namely, itself and one). The first several primes are 2, 3, 5, 7, 11, and 13. (Note that 1 is *not prime*, as it has only one divisor (namely itself) among the positive integers.)

In a letter to Euler in 1742, Christian Goldbach conjectured that every even integer greater than two is the sum of two prime numbers. Euler agreed that the conjecture was probably true, but, despite the efforts of a number of mathematicians over the past two hundred sixty years, no one has yet been able to prove or disprove it. Recently, with the help of electronic computers, it has been shown that the conjecture holds for numbers up to 4×10^{14}.

Develop a program that, given as input a list of even integers greater than two (but no greater than one thousand), produces as output, for each number in the list, every pair of prime numbers that sum to it.

Input

The first line of input contains a positive integer n indicating how many numbers are to be processed. On each of the next n lines is an even integer greater than two (but no greater than 1000).

Output

For each of the n numbers given as input, that number should be displayed on one line. On the following lines, every distinct pair of primes summing to it should be displayed, one pair per line, with the smaller of the two primes in each pair listed first. The pairs should be listed in increasing order with respect to the smaller prime in each one. Following that should be a blank line.

Sample input

```
4
48
38
12
100
```

Corresponding output

```
48
5 43
7 41
11 37
17 31
19 29

38
7 31
19 19

12
5 7

100
3 97
11 89
17 83
29 71
41 59
```

Problem 5: Decimal to Base Fibonacci Conversion

Let us define the fibonacci sequence $f_0, f_1, f_2, \ldots$ as follows: $f_0 = 1$, $f_1 = 2$, and, for $k > 1$, $f_k = f_{k-2} + f_{k-1}$. That is, the first two elements are 1 and 2, respectively, and each element thereafter is the sum of the two elements preceding it. The sequence begins as follows:

$$1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

In the decimal (i.e., base 10) number system, the numeral 57082 (for example) has value

$$2 \cdot 10^0 \ + \ 8 \cdot 10^1 \ + \ 0 \cdot 10^2 \ + \ 7 \cdot 10^3 \ + \ 5 \cdot 10^4$$

That is, the value of a numeral is obtained by multiplying each of its digits by the appropriate power of 10 (beginning at 10^0 for the rightmost digit) and adding up the results. Writing this more generally, we have that the value of the n-digit numeral $d_{n-1}d_{n-2}\cdots d_1 d_0$ (where each d_i is one of the ten digits 0 through 9) is

$$d_0 \cdot 10^0 \ + \ d_1 \cdot 10^1 \ + \ \cdots \ + \ d_{n-2} \cdot 10^{n-2} \ + \ d_{n-1} \cdot 10^{n-1}$$

Consider using the Fibonacci numbers as the "weights" of the columns in a numeral, rather than using the powers of ten. That is, suppose that we interpret an n-digit numeral $d_{n-1}d_{n-2}\cdots d_1 d_0$ as having value

$$d_0 \cdot f_0 \ + \ d_1 \cdot f_1 \ + \ \cdots \ + \ d_{n-2} \cdot f_{n-2} \ + \ d_{n-1} \cdot f_{n-1}$$

It turns out that, even if we limit ourselves to using the two digits 0 and 1, every positive integer can be expressed in this way! For example, 111010_{fib} (the subscript indicates that this is a "base-fibonacci" numeral) would be evaluated as follows:

$$\begin{aligned} 111010_{fib} &= 0 \cdot f_0 \ + \ 1 \cdot f_1 \ + \ 0 \cdot f_2 \ + \ 1 \cdot f_3 \ + \ 1 \cdot f_4 \ + \ 1 \cdot f_5 \\ &= 0 \cdot 1 \ + \ 1 \cdot 2 \ + \ 0 \cdot 3 \ + \ 1 \cdot 5 \ + \ 1 \cdot 8 \ + \ 1 \cdot 13 \\ &= 0 + 2 + 0 + 5 + 8 + 13 \\ &= 28 \end{aligned}$$

Note that, unlike the decimal number system, it is possible for two different base-fibonacci numerals, neither having leading zeros, to represent the same number. For example, 10000_{fib}, 1100_{fib}, and 1011_{fib} all represent the number that is written as 8 in decimal. Indeed, within a base-fibonacci numeral, any occurrence of 100 may be replaced by 011, or vice versa, without changing the value associated with that numeral. (This is so because the sum of the weights of the two less significant digits among the three is equal to the weight of the most significant digit among the three.)

We say that a base-fibonacci numeral is in *canonical form* if it has no leading zeros and includes no occurrences of 100 among its digits. Two different canonical-form base-fibonacci numerals necessarily represent two different numbers.

Develop a program that, given a list of positive integers (written in the usual decimal notation) as input, produces as output the same values expressed as canonical-form base-fibonacci numerals.

Input

The first line of input contains a positive integer n, indicating how many numbers are to be processed. Each of the next n lines contains a positive integer no greater than one thousand (1000).

Output

For each of the n integers given as input, display it on one line, display the corresponding canonical-form base-fibonacci number on the next, and make the next line blank.

Sample input

```
5
19
49
248
159
2
```

Corresponding output

```
19
11111

49
1111010

248
10101111010

159
1011011010

2
10
```

Problem 6: Overlapping Taping Schedules

The digital video recorder (DVR) is becoming a popular item for home entertainment systems. One of the benefits of a DVR compared to the older video cassette recorder (VCR) is that the former, unlike the latter, is capable of taping two or more shows simultaneously.

Develop a program that takes as input the taping schedule of a DVR and determines the time periods during which the machine will be required to tape two or more shows simultaneously.

Input

The first line of input contains a positive integer m indicating how many taping schedules are to be analyzed. Subsequent lines of input contain descriptions of m taping schedules.

Each description of a taping schedule begins with a line containing a positive integer n ($n \leq 50$) indicating the number of shows on that schedule. Each of the next n lines contains a day-time interval during which a particular show is to be recorded. A day-time interval consists of a starting day-time and an ending day-time, separated by a space. A day-time consists of a three-letter abbreviation for a day of the week (i.e., one of `SUN`, `MON`, `TUE`, `WED`, `THU`, `FRI`, or `SAT`), followed by a space, followed by the time of day, expressed in "military time", i.e., as a four-digit sequence whose first two digits are in the range 00..23 (indicating the number of hours since midnight) and whose last two digits are in the range 00..59 (indicating the number of minutes since the beginning of the current hour). In order to simplify matters, you may assume that, in any day-time given as input, the last two digits are either 00, 15, 30, or 45.

The day-time intervals given as input should be assumed to fall within the same week but should not be assumed to be in any particular order.

Output

For each schedule given as input, your program should produce as output a *minimum-length* list of day-time intervals covering exactly the time periods during which the DVR is scheduled to record two or more shows simultaneously. These day-time intervals should be listed in chronological order, one per line, beginning with Sunday at midnight (i.e., when Saturday turns into Sunday).

The results for each schedule should be preceded by a message like that shown in the sample output below. The results for each schedule should be followed by a blank line.

Sample input

```
2
8
MON 0200 MON 0500
TUE 1100 TUE 1300
TUE 1200 TUE 1400
WED 0000 WED 0200
TUE 2300 WED 0000
MON 1600 MON 1645
TUE 2330 WED 0030
WED 0100 WED 0130
2
THU 1345 THU 1515
FRI 1500 SAT 0145
```

Corresponding output

```
Schedule 1 overlaps:
TUE 1200 TUE 1300
TUE 2330 WED 0130

Schedule 2 overlaps:
```