10935 Throwing cards away I

Given is an ordered deck of n cards numbered 1 to n with card 1 at the top and card n at the bottom. The following operation is performed as long as there are at least two cards in the deck:

> Throw away the top card and move the card that is now on the top of the deck to the bottom of the deck.

Your task is to find the sequence of discarded cards and the last, remaining card.

Input

Each line of input (except the last) contains a number $n \leq 50$. The last line contains '`0`' and this line should not be processed.

Output

For each number from the input produce two lines of output. The first line presents the sequence of discarded cards, the second line reports the last remaining card. No line will have leading or trailing spaces. See the sample for the expected format.

Sample Input

```
7
19
10
6
0
```

Sample Output

```
Discarded cards: 1, 3, 5, 7, 4, 2
Remaining card: 6
Discarded cards: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 4, 8, 12, 16, 2, 10, 18, 14
Remaining card: 6
Discarded cards: 1, 3, 5, 7, 9, 2, 6, 10, 8
Remaining card: 4
Discarded cards: 1, 3, 5, 2, 6
Remaining card: 4
```

10260 Soundex

Soundex coding groups together words that appear to sound alike based on their spelling. For example, “can” and “khawn”, “con” and “gone” would be equivalent under Soundex coding.

Soundex coding involves translating each word into a series of digits in which each digit represents a letter:

1 represents `B`, `F`, `P`, or `V`
2 represents `C`, `G`, `J`, `K`, `Q`, `S`, `X`, or `Z`
3 represents `D` or `T`
4 represents `L`
5 represents `M` or `N`
6 represents `R`

The letters A, E, I, O, U, H, W, and Y are not represented in Soundex coding, and repeated letters with the same code digit are represented by a single instance of that digit. Words with the same Soundex coding are considered equivalent.

Input

Each line of input contains a single word, all upper case, less than 20 letters long.

Output

For each line of input, produce a line of output giving the Soundex code.

Sample Input

```
KHAWN
PFISTER
BOBBY
```

Sample Output

```
25
1236
11
```

10098 Generating Fast, Sorted Permutation

Generating permutation has always been an important problem in computer science. In this problem you will have to generate the permutation of a given string in ascending order. Remember that your algorithm must be efficient.

Input

The first line of the input contains an integer n, which indicates how many strings to follow. The next n lines contain n strings. Strings will only contain alpha numerals and never contain any space. The maximum length of the string is 10.

Output

For each input string print all the permutations possible in ascending order. Not that the strings should be treated, as case sensitive strings and no permutation should be repeated. A blank line should follow each output set.

Sample Input

```
3
ab
abc
bca
```

Sample Output

```
ab
ba

abc
acb
bac
bca
cab
cba

abc
acb
bac
bca
cab
cba
```

11219 How old are you?

```
...
- Here are the filled form.
- Thank you. Let me check... hum... OK, OK, OK... Wait, how old are you?
- 20. Did I forget to fill it?
- No. It says here that you'll be born next month! The year is wrong...
- Oh... Sorry!
```

The process is going to be automatic and to avoid some human errors there will be a calculated field that informs the age based in the current date and the birth date given. This is your task, calculate the age, or say if there's something wrong.

Input

The first line of input gives the number of cases, T ($1 \leq T \leq 200$). T test cases follow. Each test case starts with a blank line, then you will have 2 lines corresponding to the current date and the birth date, respectively. The dates are in the format $DD/MM/YYYY$, where DD is the day, MM the month and $YYYY$ the year. All dates will be valid.

Output

The output is comprised of one line for each input data set and should be as follow (quotes for clarifying only):

'`Case #`N`:` AGE', where N is the number of the current test case and AGE is one of the 3 following options:

- '`Invalid birth date`', if the *calculated age* is impossible (still going to be born).
- '`Check birth date`', if the *calculated age* is more than 130.
- the *calculated age* (years old only), otherwise.
- If the two dates are the same, the output should be '`0`'.

Sample Input

```
4

01/01/2007
10/02/2007

09/06/2007
28/02/1871

12/11/2007
01/01/1984

28/02/2005
29/02/2004
```

Sample Output

```
Case #1: Invalid birth date
Case #2: Check birth date
Case #3: 23
Case #4: 0
```

112 Tree Summing

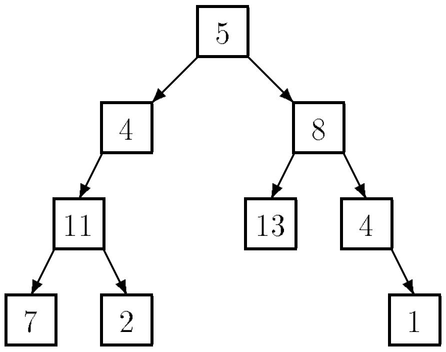
Background

LISP was one of the earliest high-level programming languages and, with FORTRAN, is one of the oldest languages currently being used. Lists, which are the fundamental data structures in LISP, can easily be adapted to represent other important data structures such as trees.

This problem deals with determining whether binary trees represented as LISP S-expressions possess a certain property.

The Problem

Given a binary tree of integers, you are to write a program that determines whether there exists a root-to-leaf path whose nodes sum to a specified integer. For example, in the tree shown below there are exactly four root-to-leaf paths. The sums of the paths are 27, 22, 26, and 18.



Binary trees are represented in the input file as LISP S-expressions having the following form.

empty tree	::=	()
tree	::=	*empty tree* | (integer *tree tree*)

The tree diagrammed above is represented by the expression

(5 (4 (11 (7 () ()) (2 () ())) ()) (8 (13 () ()) (4 () (1 () ()))))

Note that with this formulation all leaves of a tree are of the form

(integer () ())

Since an empty tree has no root-to-leaf paths, any query as to whether a path exists whose sum is a specified integer in an empty tree must be answered negatively.

The Input

The input consists of a sequence of test cases in the form of integer/tree pairs. Each test case consists of an integer followed by one or more spaces followed by a binary tree formatted as an S-expression as described above. All binary tree S-expressions will be valid, but expressions may be spread over several lines and may contain spaces. There will be one or more test cases in an input file, and input is terminated by end-of-file.

The Output

There should be one line of output for each test case (integer/tree pair) in the input file. For each pair I, T (I represents the integer, T represents the tree) the output is the string *yes* if there is a root-to-leaf path in T whose sum is I and *no* if there is no path in T whose sum is I.

Sample Input

```
22 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
20 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
10 (3
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )
5 ()
```

Sample Output

```
yes
no
yes
no
```

10008 What's Cryptanalysis?

Cryptanalysis is the process of breaking someone else's cryptographic writing. This sometimes involves some kind of statistical analysis of a passage of (encrypted) text. Your task is to write a program which performs a simple analysis of a given text.

Input

The first line of input contains a single positive decimal integer n. This is the number of lines which follow in the input. The next n lines will contain zero or more characters (possibly including whitespace). This is the text which must be analyzed.

Output

Each line of output contains a single uppercase letter, followed by a single space, then followed by a positive decimal integer. The integer indicates how many times the corresponding letter appears in the input text. Upper and lower case letters in the input are to be considered the same. No other characters must be counted. The output must be sorted in descending count order; that is, the most frequent letter is on the first output line, and the last line of output indicates the least frequent letter. If two letters have the same frequency, then the letter which comes first in the alphabet must appear first in the output. If a letter does not appear in the text, then that letter must not appear in the output.

Sample Input

```
3
This is a test.
Count me 1 2 3 4 5.
Wow!!!!  Is this question easy?
```

Sample Output

```
S 7
T 6
I 5
E 4
O 3
A 2
H 2
N 2
U 2
W 2
C 1
M 1
Q 1
Y 1
```

UVa Online Judge

10142 Australian Voting

Australian ballots require that the voter rank the candidates in order of choice. Initially only the first choices are counted and if one candidate receives more than 50% of the vote, that candidate is elected. 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 favour of their highest ranked candidate who has not been eliminated. This process continues [that is, the lowest candidate is eliminated and each ballot is counted in favour of its ranked non-eliminated candidate] until one candidate receives more than 50% of the vote or until all candidates are tied.

Input

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

The first line of input is an integer $n \leq 20$ indicating the number of candidates. The next n lines consist of the names of the candidates in order. Names may be up to 80 characters in length and may contain any printable characters. Up to 1000 lines follow; each contains the contents of a ballot. That is, each 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

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The Output consists of either a single line containing the name of the winner or several lines containing the names of the candidates who tied.

Sample Input

```
1

3
John Doe
Jane Smith
Sirhan Sirhan
1 2 3
2 1 3
2 3 1
1 2 3
3 1 2
```

Sample Output

```
John Doe
```

574 Sum It Up

Given a specified total t and a list of n integers, find all distinct sums using numbers from the list that add up to t. For example, if $t = 4$, $n = 6$, and the list is [4, 3, 2, 2, 1, 1], then there are four different sums that equal 4: 4, 3+1, 2+2, and 2+1+1. (A number can be used within a sum as many times as it appears in the list, and a single number counts as a sum.) Your job is to solve this problem in general.

Input

The input file will contain one or more test cases, one per line. Each test case contains t, the total, followed by n, the number of integers in the list, followed by n integers $x_1, \ldots, x_n$. If $n = 0$ it signals the end of the input; otherwise, t will be a positive integer less than 1000, n will be an integer between 1 and 12 (inclusive), and $x_1, \ldots, x_n$ will be positive integers less than 100. All numbers will be separated by exactly one space. The numbers in each list appear in nonincreasing order, and there may be repetitions.

Output

For each test case, first output a line containing '`Sums of `', the total, and a colon. Then output each sum, one per line; if there are no sums, output the line '`NONE`'. The numbers within each sum must appear in nonincreasing order. A number may be repeated in the sum as many times as it was repeated in the original list. The sums themselves must be sorted in decreasing order based on the numbers appearing in the sum. In other words, the sums must be sorted by their first number; sums with the same first number must be sorted by their second number; sums with the same first two numbers must be sorted by their third number; and so on. Within each test case, all sums must be distinct; the same sum cannot appear twice.

Sample Input

```
4 6 4 3 2 2 1 1
5 3 2 1 1
400 12 50 50 50 50 50 50 25 25 25 25 25 25
0 0
```

Sample Output

```
Sums of 4:
4
3+1
2+2
2+1+1
Sums of 5:
NONE
Sums of 400:
50+50+50+50+50+50+25+25+25+25
50+50+50+50+50+25+25+25+25+25+25
```

10783 Odd Sum

Given a range $[a, b]$, you are to find the summation of all the odd integers in this range. For example, the summation of all the odd integers in the range $[3, 9]$ is $3 + 5 + 7 + 9 = 24$.

Input

There can be at multiple test cases. The first line of input gives you the number of test cases, T ($1 \leq T \leq 100$). Then T test cases follow. Each test case consists of 2 integers a and b ($0 \leq a \leq b \leq 100$) in two separate lines.

Output

For each test case you are to print one line of output – the serial number of the test case followed by the summation of the odd integers in the range $[a, b]$.

Sample Input

```
2
1
5
3
5
```

Sample Output

```
Case 1: 9
Case 2: 8
```

UVa Online Judge

11995 I Can Guess the Data Structure!

There is a bag-like data structure, supporting two operations:

1 x	Throw an element x into the bag.
2	Take out an element from the bag.

Given a sequence of operations with return values, you're going to guess the data structure. It is a stack (Last-In, First-Out), a queue (First-In, First-Out), a priority-queue (Always take out larger elements first) or something else that you can hardly imagine!

Input

There are several test cases. Each test case begins with a line containing a single integer n ($1 \leq n \leq 1000$). Each of the next n lines is either a type-1 command, or an integer 2 followed by an integer x. That means after executing a type-2 command, we get an element x **without error**. The value of x is always a positive integer not larger than 100. The input is terminated by end-of-file (EOF).

Output

For each test case, output one of the following:

`stack`	It's definitely a stack.
`queue`	It's definitely a queue.
`priority queue`	It's definitely a priority queue.
`impossible`	It can't be a stack, a queue or a priority queue.
`not sure`	It can be more than one of the three data structures mentioned above.

Sample Input

```
6
1 1
1 2
1 3
2 1
2 2
2 3
6
1 1
1 2
1 3
2 3
2 2
2 1
2
1 1
2 2
4
```

```
1 2
1 1
2 1
2 2
7
1 2
1 5
1 1
1 3
2 5
1 4
2 4
```

Sample Output

```
queue
not sure
impossible
stack
priority queue
```

231 Testing the CATCHER

A military contractor for the Department of Defense has just completed a series of preliminary tests for a new defensive missile called the CATCHER which is capable of intercepting multiple incoming offensive missiles. The CATCHER is supposed to be a remarkable defensive missile. It can move forward, laterally, and downward at very fast speeds, and it can intercept an offensive missile without being damaged. But it does have one major flaw. Although it can be fired to reach any initial elevation, it has no power to move higher than the last missile that it has intercepted.

The tests which the contractor completed were computer simulations of battlefield and hostile attack conditions. Since they were only preliminary, the simulations tested only the CATCHER's vertical movement capability. In each simulation, the CATCHER was fired at a sequence of offensive missiles which were incoming at fixed time intervals. The only information available to the CATCHER for each incoming missile was its height at the point it could be intercepted and where it appeared in the sequence of missiles. Each incoming missile for a test run is represented in the sequence only once.

The result of each test is reported as the sequence of incoming missiles and the total number of those missiles that are intercepted by the CATCHER in that test.

The General Accounting Office wants to be sure that the simulation test results submitted by the military contractor are attainable, given the constraints of the CATCHER. You must write a program that takes input data representing the pattern of incoming missiles for several different tests and outputs the maximum numbers of missiles that the CATCHER can intercept for those tests. For any incoming missile in a test, the CATCHER is able to intercept it if and only if it satisfies one of these two conditions:

1. The incoming missile is the first missile to be intercepted in this test.

-or-

2. The missile was fired after the last missile that was intercepted and it is not higher than the last missile which was intercepted.

Input

The input data for any test consists of a sequence of one or more non-negative integers, all of which are less than or equal to 32,767, representing the heights of the incoming missiles (the test pattern). The last number in each sequence is `-1`, which signifies the end of data for that particular test and is not considered to represent a missile height. The end of data for the entire input is the number `-1` as the first value in a test; it is not considered to be a separate test.

Output

Output for each test consists of a test number (`Test #1`, `Test #2`, etc.) and the maximum number of incoming missiles that the CATCHER could possibly intercept for the test. That maximum number appears after an identifying message. There must be at least one blank line between output for successive data sets.

Note: The number of missiles for any given test is not limited. If your solution is based on an inefficient algorithm, it **may not** execute in the allotted time.

Sample Input

```
389
207
155
300
299
170
158
65
-1
23
34
21
-1
-1
```

Sample Output

```
Test #1:
  maximum possible interceptions: 6

Test #2:
  maximum possible interceptions: 2
```

661 Blowing Fuses

Maybe you are familiar with the following situation. You have plugged in a lot of electrical devices, such as toasters, refrigerators, microwave ovens, computers, stereos, etc, and have them all running. But at the moment when you turn on the TV, the fuse blows, since the power drawn from all the machines is greater than the capacity of the fuse. Of course this is a great safety feature, avoiding that houses burn down too often due to fires ignited by overheating wires. But it is also annoying to walk down to the basement (or some other inconvenient place) to replace to fuse or switch it back on.

What one would like to have is a program that checks *before* turning on an electrical device whether the combined power drawn by all running devices exceeds the fuses capacity (and it blows), or whether it is safe to turn it on.

Input

The input consists of several test cases. Each test case describes a set of electrical devices and gives a sequence of turn on/off operations for these devices.

The first line of each test case contains three integers n, m and c, where n is the number of devices ($n \leq 20$), m the number of operations performed on these devices and c is the capacity of the fuse (in Amperes). The following n lines contain one positive integer c_i each, the consumption (in Amperes) of the i-th device.

This is followed by m lines also containing one integer each, between 1 and n inclusive. They describe a sequence of turn on/turn off operations performed on the devices. For every number, the state of that particular devices is toggled, i.e. if it is currently running, it is turned off, and if it is currently turned off, it will by switched on. At the beginning all devices are turned off.

The input will be terminated by a test case starting with $n = m = c = 0$. This test case should not be processed.

Output

For each test case, first output the number of the test case. Then output whether the fuse was blown during the operation sequence. The fuse will be blown if the sum of the power consumptions c_i of turned on devices at some point exceeds the capacity of the fuse c.

If the fuse is not blown, output the maximal power consumption by turned on devices that occurred during the sequence.

Output a blank line after each test case.

Sample Input

```
2 2 10
5
7
1
2
3 6 10
2
5
7
2
1
2
3
1
3
0 0 0
```

Sample Output

```
Sequence 1
Fuse was blown.

Sequence 2
Fuse was not blown.
Maximal power consumption was 9 amperes.
```