

1. A certain primitive text-formatting system allows one to write superscripts and subscripts in one's text as follows:

In this equation, we may substitute for the derivative to get $x_{n+1} = x_n + (y-x_n^2)/2x_n$, which solves the problem. For the cube root, the corresponding equation is $x_{n+1} = x_n + (y-x_n^3)/3x_n^2$.

and have them converted as follows:

In this equation, we may substitute for the derivative to get $x_{n+1} = x_n + (y-x_n^2)/2x_n$, which solves the problem. For the cube root, the corresponding equation is $x_{n+1} = x_n + (y-x_n^3)/3x_n^2$.

Each character of regular text can have a subscript or superscript (or both). The formatter adds an extra line above any line containing superscripts and below any line containing subscripts, and places the text between the curly braces at the appropriate places on these lines. Use separate sub- and superscript lines for each line that needs them (that is, don't put superscripts from one line on the line containing subscripts from the one above it). Input text to the right of the sub- or superscripted component gets moved to so that it begins just to the right of the subscript or superscript on the preceding text, whichever is longer. For example,

$x^3_{k+1} + 1$ becomes $x_{k+1}^3 + 1$

Write a program that performs these transformations. The input consists of a text file with subscript and superscript annotations. You may assume that the text of subscripts and superscripts does not contain subscripts, superscripts, newlines, or curly braces, and that there is at most one subscript and one superscript for each regular character (so ' $x_n\{y}$ ' is illegal). The input will always be well formed: each underscore or caret is followed by a left curly brace, followed by text that is bracketed by a right curly brace.

Aside from moving around the subscripts and superscripts as shown, preserve all blanks and newlines. There will be no tabs in the input.

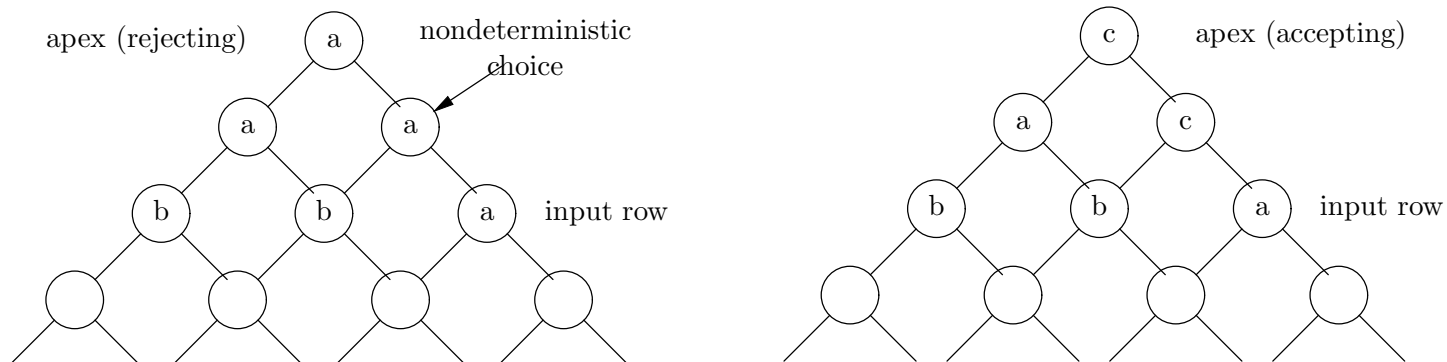
2. [From the 1996 ACM Contest Finals] A nondeterministic trellis automaton (NTA) is a kind of parallel machine composed of identical finite-state processors arranged in an infinite triangular trellis, or series of rows of processors. The top, or apex, of the triangle is a single processor. The next row has two processors and each successive row of an NTA has one more processor than the row above it. Each processor in an NTA is connected to two children in the row below it. Computation in an NTA occurs bottom up; the state of each processor in a row is based on the state of the processor's children and a transition table. The input to an NTA is the initial configuration, or state, of one row of processors. The input is specified by a string that gives the initial state of each processor in a row so that an n -character string specifies the initial configuration for a row of n processors. Computation proceeds up the NTA to the apex by nondeterministically calculating the state of each processor in a row based on the transition table and the state of the processor's children in the row below.

Transitions are computed nondeterministically; an input is accepted if *some* computation puts the apex processor into an accepting state. An input is rejected if no computation puts the apex processor into an accepting state. For example, the table below shows transitions for a 3-state NTA. States are labeled by characters 'a', 'b', and 'c'. The only accepting state is 'c'.

State Transition Table

Left child state	Right child state		
	a	b	c
a	a	a	c
b	a,c	a	b
c	c	b	a

The diagram below shows two computations for the input "bba". The computation on the left rejects the input, since the state of the apex is 'a'. The computation on the right accepts the input, since the state of the apex is 'c'. Therefore, because some computation results in an accepting state for the apex, the string "bba" is accepted by the NTA. The input "bbb" would be rejected by this NTA, since the only computation results in the state 'a' for the apex.



The states (and inputs) of an NTA are consecutive lowercase letters. Thus the states for a 5-state NTA are 'a', 'b', 'c', 'd', and 'e'. Accepting states are grouped at the end of the letters so that if a 5-state NTA has two accepting states, the accepting states are 'd' and 'e'.

The input for your program is a sequence of NTA descriptions and strings (which are the inputs to the NTAs) in free format. An NTA description is given by the number of states, n , followed by the number of accepting states. An $n \times n$ transition table follows in row-major order. Each NTA description is followed by its input strings. A string of '#' terminates the input strings. An NTA description in which the number of states is zero terminates the input for your program. NTAs will have at most 15 states, input strings will be at most 15 characters.

For each NTA description, print the number of the NTA (NTA #1, NTA #2, etc.). Then for each input string, print the word "accepted" or "rejected" followed by an echo of the input string.

Example:

Input	Output
3 1	NTA # 1
a a c	accept bba
ca a b	reject aaaaa
c b a	reject abab
bba aaaaa abab babbba a	accept babbba
baaab abbbaba baba bcbab #	reject a
3 2	accept baaab
ab a c a ab b c b ab	accept abbbaba
abc cbc #	accept baba
0 0	reject bcbab
	NTA # 2
	reject abc
	accept cbc

3. [From the 1990 Internet Contest] *Arbitrage* is the trading of currency, stocks, or similar items to take advantages of small differences in their relative prices from one market to another in order to make a profit. For example, if \$1.00 in U.S. currency buys £0.7 in British currency, £1 buys 9.5 French francs, and 1 franc buys \$0.16, then an arbitrage trader can start with \$1.00 and earn $\$1 \times 0.7 \times 9.5 \times 0.16 = \1.064 , a 6.4% profit.

Your problem is to write a program that determines whether there is a sequence of such exchanges that yields a profit. To result in successful arbitrage, a sequence of exchanges must begin and end with the same currency, but one may start from any currency.

The input will consist of one or more conversion tables in free format. Each table is preceded by an integer n ($2 \leq n \leq 20$) giving the dimensions of the the table (the number of currencies). The table then follows in row major order but with all diagonal elements of the table missing (they are assumed to have value 1.0). That is, there will be $n(n - 1)$ numbers in all, plus the value n at the beginning. Row k of the table represents the conversion rates between country k and the $n - 1$ other countries, that is, the amount of country j 's currency ($j \neq k$) that may be purchased for 1 unit of country k 's currency.

For each table, your program must determine whether a sequence of exchanges exists that results in a profit of $> 1.0\%$. If such a sequence exists, print it in the format shown in the example below. That is, print the sequence of integers representing the country numbers of the currencies exchanged (the first country, corresponding to the first row of the table, is 1). The first integer in the sequence is the country from which the profiting sequence starts. The last integer in the sequence always equals the first. If there is more than one such sequence, print a sequence of minimal length, i.e., one that uses the fewest exchanges of currencies to yield a profit. If no profitable sequence of n or fewer transactions exists, print the line “no arbitrage sequence exists” as shown in the example.

Because the IRS notices lengthy transaction sequences, all profiting sequences must consist of n or fewer transactions (the sequence “1 2 1” represents two conversions).

Example:

Input	Output
3	1 2 1
1.2 .89	1 4 3 1
.88 5.1	no arbitrage sequence exists
1.1 0.15	
4	
3.1 0.0023 0.35	
0.21 0.00353 8.13	
200 180.559 10.339	
2.11 0.089 0.06111	
2	
2.0	
0.45	