1. [From the 1997 Berkeley contest] The text formatting program TeX breaks paragraphs into lines so as to minimize the total *demerits* of the paragraph. Its definition of “demerit” is a bit complex; here we will deal with a simplified version. In this simple version, we’ll break paragraphs into lines only at word boundaries (blanks). The demerits assigned to a particular choice of where to put line breaks is the sum of the demerits for all the resulting lines.

To compute the demerits, $d$, of one line, we need to know $w$, the number of words on the line; and $c$, the total number of characters on the line (including single blanks between words). Then for that line,

$$
\begin{align*}
  s &= \begin{cases} 
    \min(0, 30 - c), & \text{on the last line,} \\
    30 - c, & \text{otherwise.}
  \end{cases} \\
  b &= \begin{cases} 
    \left( \frac{2s}{\max(1, w-1)} \right)^3, & \text{if } s \geq 0 \\
    \left( \frac{-5s}{\max(1, w-1)} \right)^3, & \text{if } s < 0.
  \end{cases} \\
  d &= (0.1 + b)^2
\end{align*}
$$

assuming that lines are normally 30 characters long. In these formulae, $b$, a floating-point number, is the “badness” of the line, and $s$ is the total stretching of the blanks (or shrinking, if negative) needed to fit all the words in exactly 30 spaces. Again, the total demerits for an entire paragraph is the sum of the $d$ values for the individual lines.

Your program is to find the best selection of line breaks (blanks at which to end one line and begin the next) for a paragraph—one that minimizes the sum of the values of $d$ over all the lines. Each input paragraph will consist of a sequence of words (non-blank characters) separated by single spaces (no line breaks). The output paragraphs are the same, with some of the blanks turned into newlines, and with a blank line between paragraphs. Do not insert extra spaces to justify the lines; leave them ragged, as in the examples below. You may assume that no input paragraph contains more than 500 words. If there is more than one way to break a paragraph optimally, favor one that puts a line break sooner.
Example:  (The backslashes at the ends of lines indicate line continuations, not real line breaks. In the actual input, there would be only three lines of input, with the backslashes and following newlines replaced by single blanks).

Input:

In preparation for the 1997 Pacific Regionals of the 22nd Annual ACM Scholastic Programming Contest, there will be a semi-informal programming contest on Saturday, 25 October 1997, from 1000--1530.

The square of the hypotenuse of a right triangle is equal to the sum of the squares of the two adjacent sides.

You'd not tolerate letting your participle dangle, so please effect the self-same respect for your geometric sides.

Output:

In preparation for the 1997 Pacific Regionals of the 22nd Annual ACM Scholastic Programming Contest, there will be a semi-informal programming contest on Saturday, 25 October 1997, from 1000--1530.

The square of the hypotenuse of a right triangle is equal to the sum of the squares of the two adjacent sides.

You'd not tolerate letting your participle dangle, so please effect the self-same respect for your geometric sides.

Note.  As you can see, some lines can be more than 30 characters long. Also, some lines may not be filled as much as they could be (the second line of output, for example) if that helps later lines.

2.  [From the 1997 Berkeley contest] Sally supervises a group of programmers who are to work on the modules of a large product. She assigns one lead programmer to each module, and asks the lead programmers each to choose a (less-experienced) junior programmer to assist. Sometimes, however, two lead programmers both want the same assistant. At the same time, of course, the junior programmers have their own preferences about which lead programmers they wish to work with.
In an attempt to satisfy all parties at least to some extent, Sally decides on a simple
criterion that any pairing of junior and lead programmers will have to meet. Specifically, if
Jack (lead) and Mary (junior) form a team, there should never be another team—say Toni
(lead) and Jason (junior)—in which Jack would prefer to work with Jason and Jason would
prefer to work with Jack. Sally thinks this problem tricky enough to warrant a program,
which she asks you to write.

The input to this program (in free form) will consist of a positive integer, \( N \), giving the
number of teams, followed by \( 2N \) lists. Each list consists of the name of a programmer (a
string of up to 64 characters without embedded whitespace), followed by the names of \( N \) other
programmers in decreasing order of preference. The first \( N \) lists each begin with the name
of a lead programmer, followed by the names of all \( N \) junior programmers, and the last \( N 
preference lists begin with the names of the junior programmers, followed by the names of all
\( N \) lead programmers. The output of your program should be a possible pairing, in the format
shown in the example below. List the results in the order that the lead programmers were
initially listed. Where there are multiple possible legal pairings, favor the preferences of the
lead programmers—so that they get the assistant they most prefer, subject to the constraints
of the problem.

Example:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 Tim Bryan Mary Binh</td>
<td>Tim Bryan</td>
</tr>
<tr>
<td></td>
<td>Jane Binh</td>
</tr>
<tr>
<td>Jane Mary Binh Bryan</td>
<td>Tommy Mary</td>
</tr>
<tr>
<td>Tommy Mary Binh Bryan</td>
<td></td>
</tr>
<tr>
<td>Mary Tim Tommy Jane</td>
<td></td>
</tr>
<tr>
<td>Binh Tim Jane Tommy</td>
<td></td>
</tr>
<tr>
<td>Bryan Tommy Tim Jane</td>
<td></td>
</tr>
</tbody>
</table>
3. [Augusto Sousa, 2nd round of CPUP’04 (Concurso de Programação da Universidade do Porto 2004)] Suppose you launch a set of cylinders, with different radii, over a hill, in such a way they get stacked like this:

![Diagram of stacked cylinders](image)

Depending to the radius of the cylinders, only subsequences of the cylinders will touch to form a continuous path. In the example represented, the longest such path is composed of cylinders 2, 3 and 6. Cylinder 2 is the *path head* and cylinder 6 is the *path tail*. The *enclosing distance* $d$ is measured from the left support point of the path head until the right point of the path tail.

Given a sequence of cylinders, compute the enclosing distance and the subsequence of cylinders that compose the longest path. You may assume that each cylinder can touch no more than one cylinder at its left and no more than one cylinder at its right. The input will contain several test cases in free format. Each test case begins with a positive integer $N_c \leq 100$ of cylinders. Next come $N_c$ floating-point numbers giving the radii of the cylinders. The first radius is that of cylinder #1, etc.

Output the value of $d$ (to one decimal place) and the sequence of cylinder numbers for each test case, using the format shown in the example below.

**Example.**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Case 1: Distance: 183.1; cylinders 2, 3, 6</td>
</tr>
<tr>
<td>3 25 35 5 4 32 4</td>
<td>Case 2: Distance: 18.3; cylinders 2, 5, 6</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>0.4 3.2 0.4 0.5 3.5 2.5 0.3</td>
<td></td>
</tr>
</tbody>
</table>