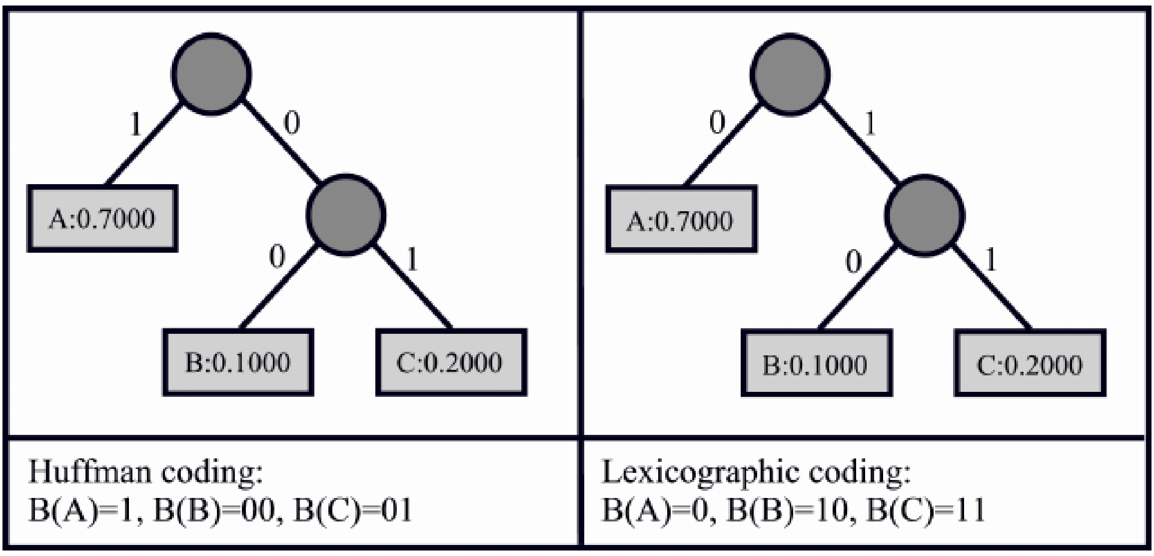
Given an alphabet S, and a probability $Prob(a)$ for each $a \in S$, a binary prefix code represents each a in S as a bit string $B(a)$, such that $B(a_1)$ is not a prefix of $B(a_2)$ for any $a_1 \neq a_2$ in S.

Huffman's algorithm constructs a binary prefix code by pairing the two least probable elements of S, a_0 and a_1. a_0 and a_1 are given codes with a common (as yet to be determined) prefix p and differ only in their last bit: $B(a_0) = p0$ while $B(a_1) = p1$. a_0 and a_1 are removed from S and replaced by a new element b with $Prob(b) = Prob(a_0) + Prob(a_1)$. b is an imaginary element standing for both a_0 and a_1. The Huffman code is computed for this reduced S, and p is set equal to $B(b)$. This reduction of the problem continues until S contains one element a represented by the empty string; that is, when $S = \{a\}$, $B(a) = \epsilon$.

Huffman's code is optimal in that there is no other prefix code with a shorter average length defined as:

$$\sum_{a \in S} Prob(a) \times |B(a)|$$

One problem with Huffman codes is that they dont necessarily preserve any ordering that the elements may have. For example, suppose $S = \{A, B, C\}$ and $Prob(A) = 0.7$, $Prob(B) = 0.1$, $Prob(C) = 0.2$. A Huffman code for S is $B(A) = \texttt{1}$, $B(B) = \texttt{00}$, $B(C) = \texttt{01}$. The lexicographic ordering of these strings is $B(B)$, $B(C)$, $B(A)$ [i.e. `00,01,1`], so the coding does not preserve the original order A, B, C. Therefore, algorithms like binary search might not work as expected on Huffman-coded data.



Given an ordered set S and $Prob$, you are to compute an ordered prefix code — one whose lexicographic order preserves the order of S.

Input

Input consists of several data sets. Each set begins with $0 < n \leq 100$, the number of elements in S. n lines follow; the i-th line gives the probability of a_i, the i-th element of S. Each probability is given as '`0.`*dddd*' (*that is, with exactly four decimal digits*). The probabilities sum to 1.0000 exactly. A line containing '`0`' follows the last data set.

Output

For each data set, compute an optimal ordered binary prefix code for S. The output should consist of one line giving the average code length, followed by n lines, with the i-th line giving the code for the i-th element of S. If you have solved the problem, these n lines will be in lexicographic order. *If there are many optimal solutions, choose any one.*

Output an empty line between cases.

Sample Input

```
3
0.7000
0.1000
0.2000
3
0.7000
0.2000
0.1000
0
```

Sample Output

```
1.3000
0
10
11

1.3000
0
10
11
```