Huffman codes P62266_en

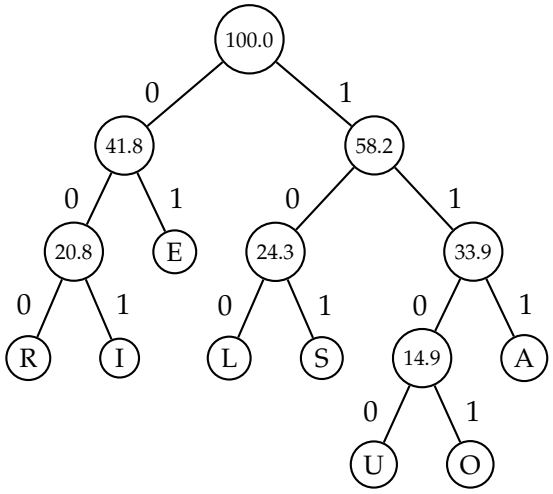
Suppose that n different symbols are used in the writing of a certain language. A simple way to code a text consists in assigning $\lceil \log_2 n \rceil$ bits to each symbol. For instance, consider the five vowels and the three more frequent consonants in Catalan. According to the following table, a "normal" coding of the word LIE would be 011100000:

Letter	"Normal" codification	Relative frequence	Huffman coding
E	000	21	01
A	001	19	111
S	010	12.7	101
L	011	11.6	100
I	100	10.6	001
R	101	10.2	000
O	110	8.6	1101
U	111	6.3	1100

Now, suppose that we know the probabilities (or relative frequences) of every symbol (for instance, according to the table, out of 100 symbols among the eight chosen, 21 are e's, 19 are a's, etcetera). We can obtain a more efficient coding on the average by assigning less bits to frequent letters. According to the table, the Huffman coding of the word LIE would be 10000101, with only eight bits instead of nine.

The construction of a Huffman coding is relatively simple: Repeatedly, choose the two less frequent symbols, arbitrarily assign a bit (0 or 1) to each one to make them different, and consider them both a unique symbol from now on. Stop when only one symbol remains.

For instance, this is the tree corresponding to the Huffman algorithm for the eight characters of the table above. The quantity inside each node is the relative frequence of all the symbols below it.



In this example, the average length of the Huffman coding of a symbol is only

$$0.21 * 2 + 0.19 * 3 + 0.127 * 3 + 0.116 * 3 + 0.106 * 3 + 0.102 * 3 + 0.086 * 4 + 0.063 * 4 \simeq 2.9390,$$

smaller than the average length of a "normal" coding, which obviously is 3. For more biased probability distributions, much more significant savings can be obtained.

Write a program that reads the relative frequences of some letters, and computes the average length of their Huffman coding.

Input

Input consists of the number of symbols $n \geq 2$, followed by the relative frequences of the n symbols. These frequences are all non-negative, and their sum is 100.

Output

Print with four digits after the decimal point the expected number of bits per letter.

Sample input 1

```
8
19 21 10.6 8.6 6.3 12.7 11.6 10.2
```

Sample output 1

```
expected number of bits per letter: 2.9390
```

Sample input 2

```
4
5 2 3 90
```

Sample output 2

```
expected number of bits per letter: 1.1500
```

Problem information

Author : Salvador Roura
Translator : Carlos Molina
Generation : 2018-11-23 11:24:05