Fuzzy Matching

When comparing two strings, it is straightforward to check if the strings are equal to each other, but the output is one of only two possibilities: they are equal or they are not. In text processing applications, it is often useful to test how similar they are by computing a *fuzzy match* score for two strings, with possible values anywhere between 0 and 1 (often expressed as a percentage).

One system for doing so represents each string as a set (using a technique described below) and then uses set operations to compare the two. The *union* of two sets is a set containing every element that appears in at least one of the two input sets. The *intersection* of two sets is a set containing every element that appears in both of the two input sets. The *resemblance* of two sets is the size of their intersection divided by the size of their union, i.e., for two sets α and β the resemblance is:

$$\frac{|\alpha \cap \beta|}{|\alpha \cup \beta|}$$

For our purposes, we will consider *weighted multisets*. A multiset can contain multiple instances of the same element (unlike a normal set). If multiset α contains 3 instances of one element, and multiset β contains 5 instances of the same element, the union of the two multisets would contain 5 instances of that element, while the intersection would contain 3. In a weighted multiset each element has a specified numerical weight. The size of the multiset is the sum of the weights of its elements. Note that an unweighted multiset is equivalent to a weighted multiset where each element has a weight of one.

For this fuzzy matching scheme, you will represent each string being compared as a weighted multiset, and you will use the set resemblance to compute a fuzzy match score for two strings. In the output file, you will express this as a whole number in the range 0 to 100, using the resemblance score multiplied by 100 and rounded to the nearest whole number.

To represent a string as a set, use q-grams. For a given value of q, divide each string into every possible q-length substring. For example, the string *aabbccdd* yields the 1-grams in the multiset $\{a, a, b, b, c, c, d, d\}$, the 2-grams in the multiset $\{aa, ab, bb, bc, cc, cd, dd\}$, the 4-grams in the multiset $\{aabb, abbc, bbcc, bccd, ccdd\}$, etc. To compute a fuzzy match score, represent a string as the multiset that combines all q-grams for all values of q that are powers of 2, i.e., $\{q : q = 2^n \text{ for some } n \in \mathcal{N}\}$. Furthermore, assign $1/q$ as the weight of each q-gram element. In other words, you must add q-grams that are twice as long and weighted half as much until q is greater than the length of the input string.

As an example, the string *dixie* is represented as the following multiset:

$$\{d, i, x, i, e, di, ix, xi, ie, dixi, ixie\}$$

Each string in the multiset of length one has weight 1, each string of length two has weight 0.5, and each string of length four has weight 0.25. The total weighted size of this set is 7.5.

Input and output

The first line of `input.txt` file consists of a positive integer n indicating how many matches should be performed. The remainder of the file consists of n pairs of lines. For each match, read two lines (strip the newline from the end of each line—it must *not* be included in the score computation), compute the fuzzy match score as described above, and write the score as a whole number between 0 and 100 to the output file on a line by itself (with a newline at the end of each line).

Example:

Note: ↩ represents the newline character at the end of each line.

input.txt:

```
5↩
aaaaaaaaaa↩
aaaaaaaaaa↩
aaaaabbbbb↩
bbbbbaaaaa↩
ababababab↩
aaaaaaaaaa↩
aaaaabbbbb↩
aaaaaaaaaa↩
before locking the door on your way out make sure you have your keys with you↩
make sure you have your keys with you before locking the door on your way out↩
```

output.txt:

```
100↩
82↩
18↩
29↩
93↩
```