Problem Statement

Given a finite alphabet S, a binary code over this alphabet S is a function that maps each element of S to some (possibly empty) string over the alphabet {0,1}.

An example of such a code for S={a,b,c,d} is the function f defined by f(a)=1, f(b)=1010, f(c)=01, f(d)=10101.

Any binary code can be naturally extended to encode strings over the alphabet S simply by concatenating the codes of the string's letters, in order. For example, using the code mentioned above we can encode cac as f(cac)=01101.

A code is called ambiguous if there are two different strings over S that have the same encoding. Obviously, in practice we want to avoid using an ambiguous code.

A code is called really ambiguous if there are three different strings over S that have the same encoding. For example, the code from the above example is really ambiguous: the strings ba, acc, and d are all encoded to 10101.

You will be given a code containing the strings over {0,1} used to encode letters of some alphabet S. Your method should check whether this code is really ambiguous. If it is really ambiguous, find a shortest string over {0,1} that is an encoding of (at least) three different strings over S, and return its length. If the given code is not really ambiguous, return -1.

Definition

Class:

BinaryCodes

Method:

ambiguous

Parameters:

vector <string>

Returns:

int

Method signature:

int ambiguous(vector <string> code)

(be sure your method is public)

Limits

Time limit (s):

840.000

Memory limit (MB):

64

Notes

- Your method does not need to know the actual elements of S, and the size of S is obviously equal to the number of elements in code.

Constraints

- code will contain between 2 and 30 elements, inclusive.

- Each element of code will contain between 0 and 50 characters, inclusive.

- Each element of code will only contain the characters '0' (zero) and '1' (one).

Examples

0)

{"1","1010","01","10101"}

Returns: 5

This is the example from the problem statement, and the string 10101 is the shortest string that can be decoded in three different ways.

1)

{"0","1"}

Returns: -1

This code is obviously not ambiguous.

2)

{"0","11","11","11"}

Returns: 2

This is clearly a really ambiguous code, as there are three different one letter strings over S that are encoded to 11.

3)

{"0000","001","01001","01010","01011"}

Returns: -1

This code is a prefix code, i.e., no code word is a prefix of another code word. If a code has this property, it is guaranteed that it is not ambiguous, but the other direction is not true.

4)

{"1","10","00"}

Returns: -1

This is not a prefix code, but it can easily be shown that this code is not ambiguous.

5)

{"","01101001001","111101011"}

Returns: 0

Having an empty code word is a great way how to design a really ambiguous code.

6)

{"00011011","000110","11","0001","1011","00","011011"}

Returns: 8

The shortest proof that this code is really ambiguous is 00011011. Note that this string can in fact be decoded in four different ways.