Problem H: UTF-8

Unicode is an international standard for encoding characters that was designed to overcome the limitations of older text encoding systems like ASCII. The main deficiency of ASCII and similar systems is that they represent only a very limited number of characters (primarily the characters on an English-language keyboard). In contrast, Unicode currently encodes over $130\,000$ characters (as of 2019), including those from most of the world's languages, and, in theory, could be used to represent over $1\,000\,000$ characters. Perhaps most importantly for modern human communication, Unicode encodes around $3\,000$ *emoji* (again, as of 2019).

Unicode has several encoding "flavours," the most well-known of which are UTF-8, which uses 8, 16, 24, or 32 bits per character; UTF-16, which uses 16 or 32 bits per character; and UTF-32, which uses 32 bits per character. In all these flavours, the encoding of a character is actually the encoding of a non-negative integer corresponding to the character; this integer is called a *code point*.[1]

For this problem, we will focus on UTF-8. A character that is encoded using UTF-8 is stored in 1, 2, 3, or 4 bytes, and we refer to these four options as **Type 1**, **Type 2**, **Type 3**, and **Type 4**, respectively. The following table is useful for illustrating these. In each representation of a byte as 8 bits, the leftmost bit is the most significant.

	byte 1	*byte* 2	*byte* 3	*byte* 4
Type 1	`0xxxxxxx`			
Type 2	`110xxxxx`	`10xxxxxx`		
Type 3	`1110xxxx`	`10xxxxxx`	`10xxxxxx`	
Type 4	`11110xxx`	`10xxxxxx`	`10xxxxxx`	`10xxxxxx`

- **Type 1** – The first (and only) byte begins with `0`. The remaining 7 bits are used to store the code point.
- **Type 2** – The first byte begins with `110`, and second byte begins with `10`. The remaining 5 bits of the first byte and the remaining 6 bits of the second byte (11 bits in total) are used to store the code point.
- **Type 3** – The first byte begins with `1110`, and the second and third bytes begin with `10`. The remaining 4 bits of the first byte and the remaining 6 bits of the second and third bytes are used to store the code point (16 bits in total).
- **Type 4** – The first byte begins with `11110`, and the second, third, and fourth bytes begin with `10`. The remaining 3 bits of the first byte and the remaining 6 bits of the second, third, and fourth bytes are used to store the code point (21 bits in total).

[1] Actually, not every code point corresponds to a character, but that is not important here.

A sequence of bytes adheres to the UTF-8 standard if it consists of one or more character encodings, each of which is of **Type 1**, **Type 2**, **Type 3**, or **Type 4**, in which case we say the byte sequence is *valid* UTF-8. Otherwise, the byte sequence is *invalid* UTF-8. For example, in Sample Input 2, the first (and only) byte begins with `10`, but the first two bits in the first byte of any UTF-8 character encoding can never be `10`. And in Sample Input 3, the first byte appears to begin a character encoding of **Type 4**, but only two bytes follow, instead of three.

Given a sequence of bytes, determine whether or not it is valid UTF-8, and if it is, report the number of character encodings of **Type 1**, **Type 2**, **Type 3**, and **Type 4**.

Input

The first line of input contains a positive integer n ($1 \leq n \leq 1\,200$), the number of bytes in the sequence. This is followed by n lines giving the n bytes in order, one per line. Each byte is represented as a length-8 string of '`0`' and '`1`' characters. The leftmost bit of each byte is the most-significant bit.

Output

If the sequence of bytes is invalid UTF-8, output a single line containing the word "`invalid`". Otherwise output four integers, one per line: the number of character encodings of **Type 1**, **Type 2**, **Type 3**, and **Type 4**, respectively.

Sample Input	*Sample Output*	*Sample Output, with visualized whitespace*
6 11100011 10001111 10101010 00000000 11011011 10001110	1 1 1 0	1\n 1\n 1\n 0\n

Sample Input	*Sample Output*	*Sample Output, with visualized whitespace*
1 10101010	invalid	invalid\n

Sample Input	*Sample Output*	*Sample Output, with visualized whitespace*
3 11110111 10111111 10111111	invalid	invalid\n

Note: ␣ is a space, and \n is a newline character.