12. Tree Decompression

Program Name: TreeDecompression.java   Input File: treedecompression.dat

There are various methods for data compression that trade-off speed of compression and/or decompression with compression size. A relatively simple form of text compression, which we will call Tree Compression, involves using small bit sequences to represent the unique characters in a string and then creating a unique sequence representing each string.

The steps for this tree compression are:

- Count all of the unique characters in the string.
- Sort the characters by the number of times they appear in the sequence from most occurring to least occurring. Characters that occur the same number of times should be sorted by their ASCII numeric value.
- To insure that no two characters begin with the same bit sequence and to use as few bits as possible, assign a binary sequence to each character in your sorted list using the following method:
    - The last character in your list will always be all 1's.
    - If there is only a single character is your list, that character is assigned a 1.
    - Otherwise, if there is more than one character in the list, the first character in the list is assigned the binary value 0.
    - The second character, unless it is the last one, would be assigned 10.
    - The third character, unless it is the last one, would be assigned 110.
    - Continue this process until finally, as stated above, the last character has a 1 at the end instead of 0, i.e., it would be all 1's.

For example, for the string `ADA ATE APPLE` containing 4 `A`'s, 1 `D`, 1 `T`, 2 `E`'s, 2 `P`'s, 1 `L`, and 2 spaces, the encoding for each character is shown in the chart to the right.

Char	encoding
A	0
Space	10
E	110
P	1110
D	11110
L	111110
T	111111

```
        X
       / \
      /   \
     A     X
          / \
         /   \
      space   X
             / \
            /   \
           E     X
                / \
               /   \
              P     X
                   / \
                  /   \
                 D     X
                      / \
                     /   \
                    L     T
```

By reading bits in, one by one, we can tell what character it is since all strings are unique and all sequences are unique from left to right.

One way to figure out what character you are reading is to put the characters into a tree. You start at the top of the tree, and for every bit you move either left for a 0, or right for a 1. When you complete a character, you start again at the top with the next bit. A tree for this example would be as shown in the chart to the left (the X's are just nodes).

Note: some strings require more bits with this compression scheme than if they were in their standard 8-bit ASCII form.

12. Tree Decompression (cont.)

Input

The input file will contain an unknown number of lines, each one containing a string of 1's and 0's which represent the bits of a compressed string. The string will either be in Tree Compression format, or in standard ASCII format, depending on the value of the first byte.

If the first byte is not 0, then the value of that byte is the number of unique characters in the string. Then, there will be a byte for each unique character, in the order the characters appear in the decompression tree. The rest of the string will then be the compressed string representation.

In example 1 below, the first byte is 3 indicating that there are three bytes to follow as they would appear in the decompression tree. Following those three bytes is the compressed string.

```
00000011010000010100011001001100 01110011100
    3       A       F       L    compressed string
```

If the first byte is 0, that means the rest of the string is just a series of bytes that represent ASCII characters in the order they appear in the string. In other words, the string isn't compressed.

In example 2 below, the first byte is zero followed by the corresponding ASCII equivalents.

```
000000000100000100100000010001000100111101000111
    0       A     space     D       O       G
```

Output

For each input line you will print the ASCII version of the string that is represented, one line for each line of input.

Example Input File

```
0000001101000001010001100100110001110011100
000000000100000100100000010001000100111101000111
```

Example Output to Screen

```
ALFALFA
A DOG
```