11. Tree Compression

Program Name: TreeCompression.java **Input File: treecompression.dat**

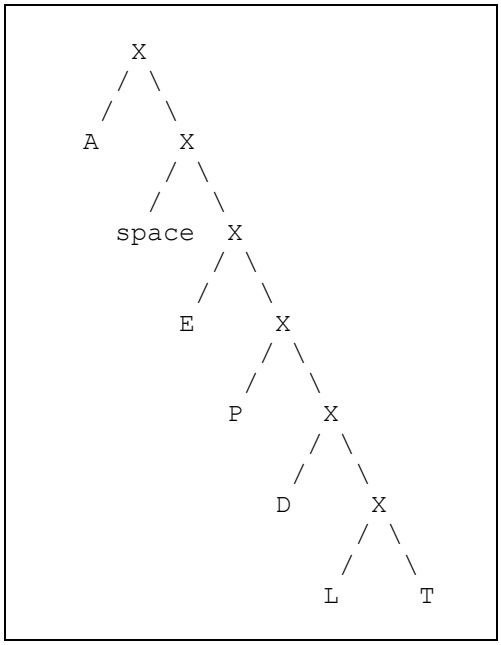
There are various methods for data compression that trade-off speed of compression and/or decompression with compression size. A relatively simple form of text compression, which we will call Tree Compression, involves using small bit sequences to represent the unique characters in a string and then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



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.

11. Tree Compression (cont.)

Input
The input file will contain an unknown number of lines, where each line contains a single ASCII string to compress.

Output
For each line of input, use the format described below to print the parts of the compressed string that would represent the binary form of the most compact storage of the string, either the Tree Compression form, or the standard ASCII string, whichever has the least amount of total bits.

The format for printing a compressed string is as follows:

- The first line of output is the binary representation of the byte (8 bits) that indicate the most compact storage form of the string. For a string in Tree Compression form, the 8 bits represent the number of unique characters in the compressed string. For a string in standard ASCII form, these bits will be all 0's.
- The remaining lines will be in one of these two formats:
  - If the string is in **compressed** form:
    - Print the byte (8 bits) representing the ASCII characters, one per line and in order of most used to least used. Characters that share the same count should be printed in the order of least to greatest by their ASCII order.
    - On the last line, print the string using the encoded values.
  - If the string is in **standard ASCII** form:
    - Print the binary representation of each character of the text string, one byte per line.
- Print a blank line after each set of output.

Note: For the example `ADA ATE APPLE`, the compressed form requires fewer bytes as shown below:
The **compressed** form would be (but your output will be one byte per line for the benefit of the judges):

```
00000111010000010010000001000101010100000100010001001100010101000
```

and the string using the encoded values would be:

```
01111001001111111101001110111011110110
```

for a total of 104 bits.

The **ASCII** form would be (but your output will be one byte per line for the benefit of the judges):

```
0000000001000001010001000100000100100000010000010101010001000101
0010000001000001010100000101000001001100010000101
```

for a total of 112 bits (there is no carriage return, the bits are wrapped to the next line).

Example Input File

```
ADA ATE APPLE
ABCDEFG
ALFALFA
```

Example Output to Screen (cont. on next page)

11. Tree Compression (cont.)

Example Output to Screen (cont. on next page)

```
00000111
01000001
00100000
01000101
01010000
01000100
01001100
01010100
01111001001111111101001110111011111110110

00000000
01000001
01000010
01000011
01000100
01000101
01000110
01000111

00000011
01000001
01000110
01001100
01110011100
```