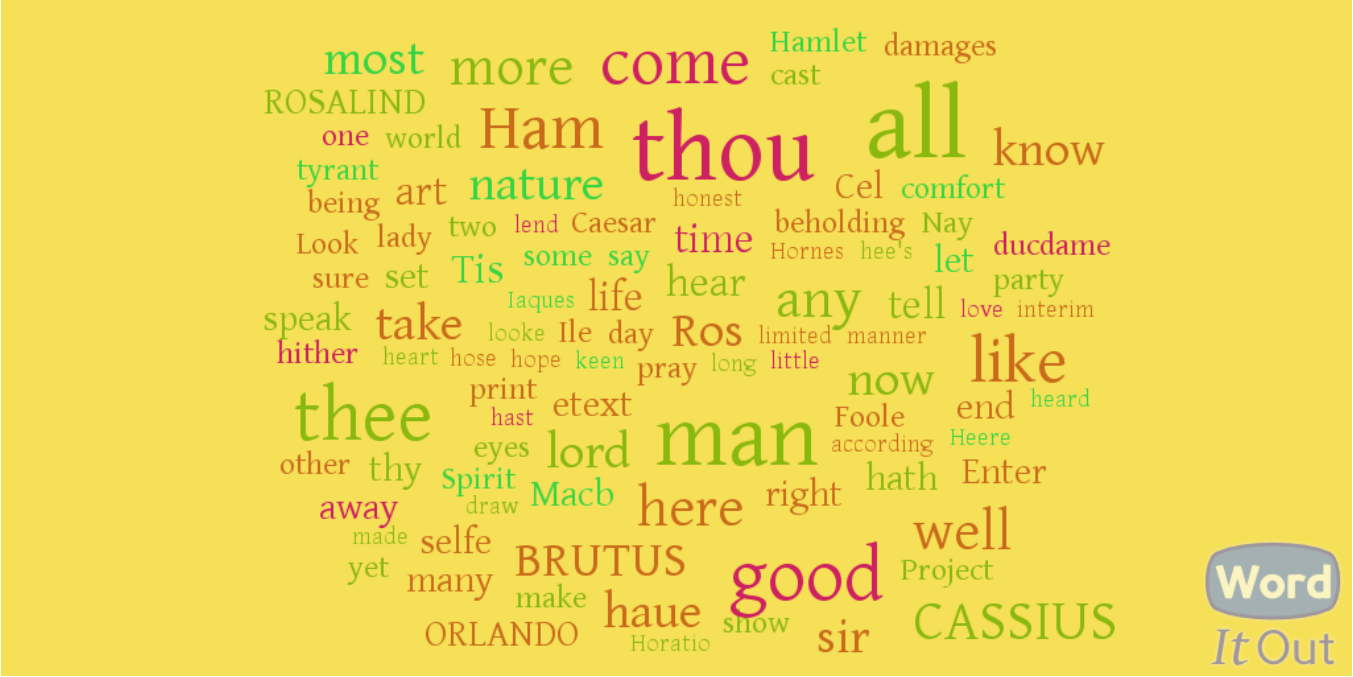
Shakespeare Cuts It Short

In this challenge you will create an efficient compression scheme to compress the works of Shakespeare.

An Introduction to our Compression Scheme

Given a large volume of text from the works of Shakespeare, only containing ASCII characters, your task, is to come up with an encoding scheme, which can compress the text efficiently. Let us take a small example.



Here is a small block of text from Wikipedia:

> William Shakespeare was an English poet and playwright, widely regarded as the greatest writer in the English language and the world's pre-eminent dramatist. He is often called England's national poet and the "Bard of Avon". His extant works, including some collaborations, consist of about 38 plays, 154 sonnets, two long narrative poems, and a few other verses, the authorship of some of which is uncertain. His plays have been translated into every major living language and are performed more often than those of any other playwright.
> Shakespeare was born and brought up in Stratford-upon-Avon. At the age of 18, he married Anne Hathaway, with whom he had three children: Susanna, and twins Hamnet and Judith. Between 1585 and 1592, he began a successful career in London as an actor, writer, and part-owner of a playing company called the Lord Chamberlain's Men, later known as the King's Men. He appears to have retired to Stratford around 1613 at age 49, where he died three years later. Few records of Shakespeare's private life survive, and there has been considerable speculation about such matters as his physical appearance, sexuality, religious beliefs, and whether the works attributed to him were written by others.
> Shakespeare produced most of his known work between 1589 and 1613. His early plays were mainly comedies and histories, genres he raised to the peak of sophistication and artistry by the end of the 16th century. He then wrote mainly tragedies until about 1608, including Hamlet, King Lear, Othello, and Macbeth, considered some of the finest works in the English language. In his last phase, he wrote tragicomedies, also known as romances, and collaborated with other playwrights.
> Many of his plays were published in editions of varying quality and accuracy during his lifetime. In 1623, John Heminges and Henry Condell, two friends and fellow actors of Shakespeare, published the First Folio, a collected edition of his dramatic works that included all but two of the plays now recognised as Shakespeare's. It was prefaced with a poem by Ben Jonson, in which Shakespeare is hailed, presciently, as "not of an age, but for all time."
> Shakespeare was a respected poet and playwright in his own day, but his reputation did not rise to its present heights until the 19th century. The Romantics, in particular, acclaimed Shakespeare's genius, and the Victorians worshipped Shakespeare with a reverence that George Bernard Shaw called "bardolatry".[8] In the 20th century, his work was repeatedly adopted and rediscovered by new movements in scholarship and performance. His plays remain highly popular today and are constantly studied, performed, and reinterpreted in diverse cultural and political contexts throughout the world.

Its usual encoding in the form of bits is given as follows. Every character is converted to the 8-bit representation of its ASCII code. For instance, the ASCII code in decimal for 'a' is 97. In binary, this is 01100001. Simiarly, the encoded representation for 'William' is '01010111011010010110110001101100011010010110000101101101' (a concatenation of the 8-bit representation for the ASCII represeentation of the characters W, i, l, l, i, a, m respectively). Proceeding in

this manner, the encoding for the entire block of text given above will appear as shown below. The total number of bits required, will be the number of characters in the text block, multiplied by 8 (i.e. 8N). Remember that spaces, punctuation, newline: everything counts as a character.
There are 2756 characters in the text block above, and a total of 2756 * 8 = 22048 bits are used while representing it in this way.

```
0101011101101001011011000110110001101001011000010110110100100000010100110110100001100001011010110110010
1011100110111000001100101011000010111001001100101001000000111011101100001011100110010000001100001011011
1000100000010001010110111001100111011011000110100101110011011010000010000001110000011011110110010101110
1000010000001100001011011100110010000100000011100000110110001100001011110010111011101110010011010010110
0111011010000111010000100000011101110110100101100100011001010110110001111001001000000111001001100101011
0010101100111011000010111001001100100011001010110010000100000011000010111001100100000011101000110100001
1001010010000001100111011100100110010101100001011101000110010101110011011101000010000001110111011100100
11010010111010001................................
```

As you might observe above, each character has simply been replaced by its 8-bit ASCII Code in binary representation. **Now, let us define a modified form of encoding, which permits us to encode a given block of text, in fewer bits. i.e, this kind of encoding will help us compress a given block of text.**

This form of encoding can contain two kinds of codes: Codes which are 9-bit in length or 17-bits in length.

```
(1) 9-bit codes should begin with a zero (eg. 011111100). These codes represent the 256 ASCII characters, in 9-bits. For example, the ASCII code for 'a' is 97, so its 9-bit representation is 001100001. The interpretation of these 9-bit codes is already fixed and known, and should not be changed.
(2) 17-bit codes should begin with a one. (eg. 10000000111111110)
(3) Each 17-bit code can either represent one character, or it could represent a string of characters. (For eg. '11100001000000011' could represent 'b', '11100001000000001' could represent 'Shakespeare' and so on).
(4) '1000000001' and '00101111100000001' are examples of invalid codes. The former is a 9-bit code so the first bit on the left should have been zero. The latter is a 17-bit code so the first bit on the left should have been one.
```

Using this form of encoding, we define a set of codes.

The interpretation of the 9-bit codes is fixed as per the ASCII chart, and should not be changed.

```
10000000000000001  Shakespeare
10000000000000000  the
10000111111111100  English
10000000000000011  and
10000000000000111  The
10000000000000100  his
10000000000001100  His
10000000000011100  plays
10000000000111100  works
10000000001111100  were
10000000011111100  some
10000000111111100  poet
10000001111111100  other
10000011111111100  known
10001111111111100  Shakespeare's
10011111111111100  published
10111111111111100  language
11111111111111100  including
11011111111111100  which
11001111111111100  until
11000111111111100  three
11000011111111100  often
11000001111111100  mainly
000100001        !
000100010        "
000100011        #
000100100        $
000100101        %
000100110        &
000100111        '
000101000        (
000101001        )
...
001100001        a
001100010        b
```

```
001100011       c
001100100       d
....
```

In deference to space, we won't display all the 9-bit codes, but we have 256 such codes, and each one of them is simply an extra zero prefixed to the 8-bit ASCII representation of the character it corresponds to. For example, 'a' is 97 in ASCII, which is 01100001 in binary; we prefix a zero to this to obtain our 9-bit code for 'a' => '001100001'. We have a total of 256 codes which are 9-bit long and 23 codes which are 17-bit long.

Using this encoding, the block of text from Shakespeare's biography, now encodes as:

```
00101011100110100100110110000110110000110100100110000100110110100010000010000000000000001000100000001111
01110011000010011100110001000000011000010011011100001000001000011111111110000010000010000000111111100000
01000001000000000000001100010000000111000000110110000110000100111100100111011100111001000110100100110011
11001101000001110100000101100000100000001110111001101001001100100000110010100110110000011110010001000000
11100100011001010011001110011000010011100100011001000011001010011001000001000000011000010011100110001000
00010000000000000000000100000001100111001110011000110010100110000100111010000110010100111001100111010000
01000000011101110011100100011010010011101000011001010011100100001000000011010010011011100001000001000000
00000000000000010000100001111111111000001000001011111111111110000010000010000000000000011000100000100000
00000000000000010000000111011100110111100111001000110110000110010000010011100111001100010000000111000000
01110010001100101000101101001100101001101101001101001001101110001100101001101110001110100000100000001100
01000011100100011000010011011010011000010011101000011010010011100110011101000001011100001000000010010000
00110010100010000000110100100111001100010000011000011111111100000100000001100011001100001001101100001100
11000011001010011001000001000000010001010011011100011001110011011000011000010011011100011001000001001111
00111001100010000000110111000110000100111010000110100100110111100110111000 ..........
```

'Shakespeare' has been replaced by '10000000000000001', 'English' has been replaced by '10000111111111100' and so on - as per our encoding scheme.

We now find, that the block of text, can be represented in 21898 bits, using this particular manner of encoding; as opposed to the 22048 bits required by the previous representation. (The size of the encoding-decoding dictionary we develop is not included here.)

Given a large input text file, with one of Shakespeare's works, or a portion of it, your task is to develop an encoding scheme, on the lines of the one shown above, which represents the text, in as few bits as possible. i.e., your primary challenge is to develop the codes in such a way, that you can represent the text given to you, such that it can be encoded into much fewer bits than its original representation.

Input Format

A large input text file, with one of Shakespeare's works, or a portion of it.

Output Format

The first line should contain an integer N. This will indicate the number of different 17-bit codes which you have chosen to use.
This is followed by N lines of the encoding dictionary. Each line contains a character or a string, followed by a tab character, and a 17 bit code which will encode that character or string.
The last line of your output should contain just one line which is the encoded version of the input text, using the encoding scheme determined by you. This should contain only zeroes and ones.
So, totally, there should be N + 2 lines in your file.

Please note that the "expected output" which you see on hitting the "Compile and Test" button is what your bit string should decompress into on using your encoding dictionary. This of course, is same as the input file (i.e. one of the works of Shakespeare). What your program should output, is the output format we have specified. Our grader will handle the de-compression/de-coding part.*

Sample Input

[Text block of Shakespeare's biography from Wikipedia, already displayed above]

Constraints and Restrictions

All test cases have been selected from the works of Shakespeare available on Project Gutenberg. Only the sample text fragment is from Wikipedia.
Only ASCII Characters are present in the file. No text file will contain more than 300k characters.
Please avoid generating 17-bit encodings for strings which contain non-print/feed/control characters (such as newline, tab, form feed etc.). This could possibly lead to inaccuracies in grading.
You should use the same encoding dictionary across all the test cases. Machine Learning libraries are **not** available for this challenge.

Sample Output
(The Dictionary should be the same across all the test cases).

```
23
Shakespeare 10000000000000001
the 10000000000000000
English 10000111111111100
and 10000000000000011
The 10000000000000111
his 10000000000000100
His 10000000000001100
plays   10000000000011100
works   10000000000111100
were    10000000001111100
some    10000000011111100
poet    10000000111111100
other   10000001111111100
known   10000011111111100
Shakespeare's   10001111111111100
.
.
.
.
.
.
.
0010101110011010010011011000011011000011010010 01....
```

Explanation
10000000000000001 can be de-coded to Shakespeare. 10000011111111100 can be de-coded to 'known'. '001100001' can be decoded to the character 'a', an so on. On decoding the long string of bits in the last line, we will get back the original block of text from Wikipedia.

Scoring

Even though compression is accomplished only when you have less than 8N bits, we have relaxed the grading criteria to assign a positive score to those with less than 9N bits. The 9N bits is the number of bits which will be used when every character is replaced by its 9 bit code.

If the input text has N characters, and if your encoding scheme successfully represents that text block with B bits, if B > (9N) the score for the test case will be zero, otherwise, the score for a test case will be:

```
M * ( 1 - B/9N)
Where M is the maximum score for the test.
All test cases carry an equal weightage.
```

Our custom checker will cross check that the encoded representation in the output, matches exactly with the original input, on using the encoding scheme generated by your solution. Representations which do not match exactly with the original input file will result in a zero score for the test case. In case your text contains a code which is neither available in the encoding dictionary provided by you, nor is it a 9-bit representation of one of the ASCII characters, you will be assigned a zero score for that test case.

Resources:

Here are some of the works from Project Gutenberg.

The Taming of the Shrew

A Midsummer Night Dream

The Merchant of Venice

You could use these resources for working on a compression model offline and create an encoding dictionary. **You should use the same encoding dictionary across all the test cases.** All test files have been selected from Shakespeare's works on Project Gutenberg.

Moderator Note and Hint

Originally we had declared that the dictionary could be either the same for all tests, or you could create a separate one for each test if you wished. The second option left open a tricky situation where a contestant can wrap up the entire test case in one code and achieve almost 100% compression for the final output string. And the observant ones need full credit for spotting this loophole and using it in their solutions. This does defeat the purpose of compression though – since this actually results in an expansion of the original data.

Our grader is going to be tightened a bit and we are going to permit that the same dictionary is submitted for all test cases. The dictionary size though, won't be directly factored into the score - after all, the same dictionary, could be used to decompress a hundred encoded works of Shakespeare. We also highly recommend that you do take a look at the links to the works of Shakespeare linked in the problem - we have seen this information hasn't been used by most of those who made submissions; based on which you can develop an efficient encoding scheme offline. The most important part of the code you submit to use will be the map or array of encodings you have arrived at (and how you apply it to the given text). This will also mean, that you will be able to use more resource-intensive algorithms which could lead to better compression ratios - which would not have otherwise run within the time/memory constraints of our environment. And a hint: those of you with NLP or Computational Linguistics toolkits installed on your local developer environment should consider using those and take advantage of Language Modelling techniques.

Template Code for a Solution based on Unigram Frequencies

Please note that this is a Ruby template, provided for your convenience. For those who have decided to encode text using a Unigram model (or word frequences) all you need to do is to fill up the map named 'encoding' with your encodings. No knowledge of Ruby is required, all that needs to be done is to fill up the map. Unigrams might not be the best possible compression method, but they will be able to compress the text to some extent.

```
# Fill up this list with words which occur frequently in the works of Shakespeare. The keys should only be strings which represent One word - containing nothing other than letters, numerals and apostrope marks. # Build a Unigram model from the Shakespeare Resources mentioned in the problem statement and enter the most popular words in this list. # --FILL UP THE LIST OF WORDS HERE --- # --The number of elements in this list should not exceed 2^16 --- # Words shoud only contain lower or upper case letters (A-Z or a-z) encodingList = ["the","an","and","Shakespeare"....] # Do NOT edit the code below this line encoding = {} (0..encodingList.length-1).each{|x| encoding[encodingList[x]] = (2**16+x).to_s(2) } def bitrep(x) rep = x.ord.to_s(2) if rep.length
```