British Informatics Olympiad Final

11–13 April, 2003

Sponsored by Lionhead Studios

Compression

THIS QUESTION IS QUITE LONG — DO NOT WORRY IF YOU DO NOT GET TO THE END

Storing large amounts of text, and transmitting it over computer networks, can be expensive and time-consuming. It is often desirable to apply a *compression algorithm* to some text before it is stored or transmitted. This is a procedure that takes some input text and produces some output text, from which the original can be reconstructed exactly. There are other techniques where the original is only approximately reconstructed; these do not concern us here. The aim is for the output text to be smaller than the input text. In this question, we shall look at two algorithms used for compression.

1. The LZ78 algorithm

The LZ78 algorithm, discovered by Lempel and Ziv in 1978, is the foundation on which a large number of compression algorithms are built, including the LZW algorithm which is used for compressing GIF and ZIP files. As LZ78 works through the input text it keeps a *dictionary* containing some of the patterns it has already seen. When one of these patterns is repeated, the algorithm produces a short code to signify the repetition rather than repeating the pattern. This usually takes up less space. (Initially the dictionary contains a single pattern — the empty string.)

Question 1.1

Consider the string `RobertTheRobotAteTheBananas`. What patterns, with two or more letters, are repeated?

In addition to the dictionary, LZ78 keeps track of a string, B, which contains the characters that it has read, but not yet compressed; initially B is also the empty string. Characters are then read from the input stream in order.

Suppose c has just been read. If the string made from adding c to the end of B is in the dictionary, B becomes this new string and we move on to the next character. If this combined string is not in the dictionary we output the pair (c, i), where B matches the i^{th} entry in the dictionary, then add combined string to dictionary, and finally reset B to the empty string. Since B only grows when the combined string is in the dictionary, we can always find a suitable i.

Finally, after we have processed all the characters in the input, we need to indicate the final value of B. This is done by outputting i, where B matches the i^{th} entry in the dictionary.

For example, the following table shows the LZ78 algorithm being applied to the string `amoamasamat`:

		‘’	
a		‘’, ‘a’	(a,1)
m		‘’, ‘a’, ‘m’	(m,1)
o		‘’, ‘a’, ‘m’, ‘o’	(o,1)
a	a	‘’, ‘a’, ‘m’, ‘o’	
m		‘’, ‘a’, ‘m’, ‘o’, ‘am’	(m,2)
a	a	‘’, ‘a’, ‘m’, ‘o’, ‘am’	
s		‘’, ‘a’, ‘m’, ‘o’, ‘am’, ‘as’	(s,2)
a	a	‘’, ‘a’, ‘m’, ‘o’, ‘am’, ‘as’	
m	am	‘’, ‘a’, ‘m’, ‘o’, ‘am’, ‘as’	
a		‘’, ‘a’, ‘m’, ‘o’, ‘am’, ‘as’, ‘ama’	(a,5)
t		‘’, ‘a’, ‘m’, ‘o’, ‘am’, ‘as’, ‘ama’, ‘t’	(t,1)
		‘’, ‘a’, ‘m’, ‘o’, ‘am’, ‘as’, ‘ama’, ‘t’	1

The first column gives the character *c* read from the input stream. The second and third columns give the contents of the compression buffer and the dictionary, *after the processing of the character c is complete*. The final column gives any output given by the algorithm in response to that input. As you can see, the first row does not have anything in the 'character read' column; it shows the status of the compression buffer and the dictionary at the very beginning of the algorithm.

Question 1.2
Compress the string `luwlueisluei` using LZ78.

Question 1.3
I compress a 15 character string, which uses only the digits 0 to 9, using LZ78. What is the maximum length of the output string? What is the minimum? How about for a 100 character input? [We define the length to be the number of pairs to occur in the output.]

Question 1.4
In the above algorithm, the tests to see if a string is in the dictionary could, if we implement them by comparing against each string in the dictionary in turn, be quite time consuming. Outline, briefly, a way of implementing the algorithm which avoids this.

Question 1.5
Of course, compression is no good unless decompression is possible. Describe an algorithm which inputs a LZ78 compressed message, and outputs the original message.

2. The Burrows-Wheeler transform

The *Burrows-Wheeler transform* is a relatively recent development in compression. It inputs a string, and outputs a string of the same length, together with a number. Thus the transform represents an expansion rather than a compression! The point is that the output string often contains long blocks of a repeated character with occasional occurrences of other characters. For example, here is an excerpt from the transform of the text of this question:

```
diibbbtubupreeeeeeeeeevvthellmL   vhtvtthhehhher glll mpicmmmmm
```

Thus one good strategy in compressing a document is to apply the transform to it first, and then to apply a compression algorithm to the output. (The Burrows-Wheeler transform is reversible so we will be able to recover the original text.) The results will often be much better than if the compression algorithm were applied directly to the original document. We shall illustrate the technique by constructing the Burrows-Wheeler transform of the string `BIOFINAL`.

Consider an array, the i^{th} row of which consists of the input string 'rotated round' $i-1$ times; then sort the rows array into alphabetical order, viewing each row as a single word. For example, on the string `BIOFINAL`:

unsorted	**sorted**
`BIOFINAL`	`ALBIOFIN`
`IOFINALB`	`BIOFINAL`
`OFINALBI`	`FINALBIO`
`FINALBIO`	`INALBIOF`
`INALBIOF`	`IOFINALB`
`NALBIOFI`	`LBIOFINA`
`ALBIOFIN`	`NALBIOFI`
`LBIOFINA`	`OFINALBI`

Observe that each column contains the same letters as the original message, but reordered, and that the first column of the array just consists of the input string sorted into alphabetical order.

Question 2.1
Actually storing the whole of the above array would require a very large amount of memory. (For an n letter string, the array contains n^2 letters.) Explain, briefly, how this can be avoided.

Having done this, the *Burrows-Wheeler transform* of the input data consists of two things; the last column of the sorted two-dimensional array and the *starting index* (the number of the row in the sorted array which contains the original string). In our case, the original string `BIOFINAL` occurs in row 2, so 2 is the starting index; and the final transform is `(NLOFBAII,2)`.

Question 2.2
What is the Burrows-Wheeler transform of the string `IOICOMPETITION`? (You should show your working, including the array in both sorted and unsorted forms).

It might seem surprising at this point, but it is possible to recover the original string from its transform. To see how this can be done, we shall reconstruct the message which gives rise to the transformed string `(SMELLSPI, 5)`. We know that the last column of the sorted array must be `SMELLSPI`. Thus the original message consists of the letters `SMELLSPI` in some order, and so the first column of the array, which consists of these letters in alphabetical order, is `EILLMPSS`. Thus the array looks like:

```
E??????S
I??????M
L??????E
L??????L
M??????L
P??????S
S??????P
S??????I
```

Question 2.3
There is now enough information available to fill in the second column of the array. What should it be? (You should provide a brief justification of your answer.)

Question 2.4
What was the original message that gave the transform `(SMELLSPI, 5)`?

Question 2.5
Describe a general algorithm to restore a message from its Burrows-Wheeler transform. You should try to make it as efficient as possible.

Finally, we now turn to the question of why the results of the Burrows-Wheeler transform are so easily compressible.

Question 2.6
Suppose you are given a string of 9 characters that only uses the characters `a` and `b`, and has no more than 2 consecutive `a`'s or `b`'s. What is the largest number of consecutive `a`'s that can appear after transformation? What is the smallest?