Lempel-Ziv Codes

Last Time: Compression techniques including Huffman codes.
Today : Lempel-Ziv codes.
Reference: Class notes available in the web page.

Lempel-Ziv Codes

Note : There are several variations of Lempel-Ziv Codes. We will look at LZ78.

Ex: zip and unzip and Unix compress and uncompress use Lempel-Ziv codes.

Let us look at an example for an alphabet having only two letters:

`aaababbbaaabaaaaaaabaabb`

Rule : Separate this stream of characters into pieces of text so that the shortest piece of data is a string of characters we have not seen yet.

`a|aa|b|ab|bb|aaa|ba|aaaa|aab|aabb`

1. We see `"a"`.
2. `"a"` has been seen, we now see `"aa"`.
3. We see `"b"`.
4. `"a"` has been seen, we now see `"ab"`.
5. `"b"` has been seen, we now see `"bb"`.
6. `"aa"` has been seen, we now see `"aaa"`.
7. `"b"` has been seen, we now see `"ba"`.
8. `"aaa"` has been seen, we now see `"aaaa"`.
9. `"aa"` has been seen, we now see `"aab"`.

10. `"aab"` has been seen, we now see `"aabb"`.

Notice that this is a dynamic method.

The pieces are indexed from 1 to n.

In the example :

```
Index : 0 1 2  3 4  5  6   7  8    9   10
        0|a|aa|b|ab|bb|aaa|ba|aaaa|aab|aabb
```

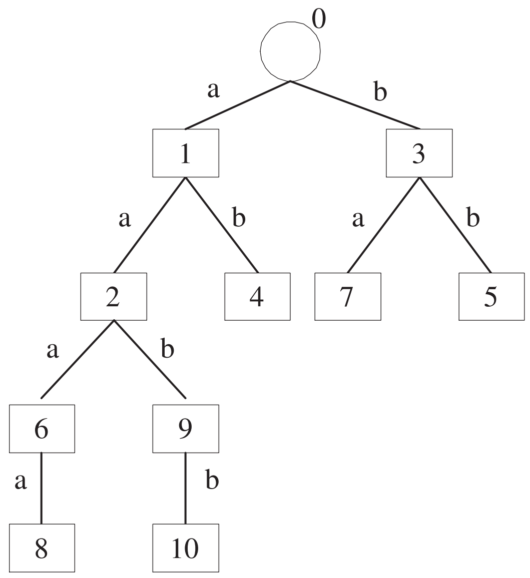
0 = Null string

Encoding :

```
Index : 1  2  3  4  5  6  7  8  9  10
        0a|1a|0b|1b|3b|2a|3a|6a|2b|9b
```

Since each piece is the concatenation of a piece already seen with a new character, the message can be encoded by a previous index plus a new character.

Indeed a digital tree can be built when encoding or decoding :



When a node is inserted the code for the current piece becomes the parent node combined with the new character.

Note that this tree is not binary in general. Here, it is binary because the alphabet has only 2 letters.

Bit Representation of Coded Information

How many bits are necessary to represent each integer with index n ? The integer is at most $n-1$, so the answer is: at most the number of bits to represent the number $n-1$.

```
1   2   3   4   5   6   7   8   9   10
0a|1a|0b|1b|3b|2a|3a|6a|2b|9b
```

Index 1: no bit (always start with zero)
Index 2: at most 1, since previous index can be only 0 or 1.
Index 3: at most 2, since previous index is between 0-2.
Index 4: at most 2, since previous index is between 0-3.
Index 5-8: at most 3, since previous index is between 0-7
Index 9-16: at most 4, since previous index is between 0-15

Each letter is represented by 8 bits. Each index is represented using the largest number of bits possibly required for that position. For the previous example, this representation would be as follows:

```
<a>1<a>00<b>01<b>011<b>010<a>011<a>110<a>0010<b>1001<b>
```

Note that <a> and <b> above should be replaced by the ASCII code for a and b, which uses 8 bits. We didn't replace them for clarity and conciseness.

Total number of bits in the encoded example :
$10 \times 8 + 0 + 1 + 2 \times 2 + 4 \times 3 + 2 \times 4 = 105$ bits

The original message was represented using $24 \times 8 = 192$ bits.

Decompressing

```
1  2  3  4  5  6  7  8  9  10
0a|1a|0b|1b|3b|2a|3a|6a|2b|9b
```

	previous pointer	added character
0	-	-
1	0	a
2	1	a
3	0	b
4	1	b
5	3	b
6	2	a
7	3	a
8	6	a
9	2	b
10	9	b

As the table is constructed line by line, we are able to decode the message by following the pointers to previous indexes which are given by the table. Try it, and you will get:

```
a aa b ab bb aaa aaa ba aaaa aab aabb
```

Irreversible Compression

All previous techniques : we preserve all information in the original data.

Irreversible compression is used when some information can be sacrificed.

Example :
Shrinking an image from 400-by-400 pixels to 100-by-100 pixels. 1 pixel in the new image for each 16 pixels in the original message.

LIt is less common than reversible compression.

Final Notes :

In UNIX:

- **pack** and **unpack** use Huffman codes byte-by-byte. 25-40% for text files, much less for binary files (more uniform distribution)

- **compress** and **uncompress** use Lempel-Ziv.