Lecture 10

Data Compression

Entropy: Given probability distribution $(p_1, \ldots, p_n)$ of events e_i (symbols or messages), entropy is a measure of quantity of information. The more likely a message is, the less information it contains (need less bits to represent it).

The entropy E_i of event e_i is $-\log p_i$ (need $-\log p_i$ bits to represent e_i), e.g., an event of probability 1/2 needs 1 bit, and an event of probability 1/4 needs 2 bits.

Example

- e_1 with prob $1/2 \to 0$
- e_2 with prob $1/4 \to 10$
- e_3 with prob $1/4 \to 11$

The overall entropy is the average of individual entropies

$$E = \sum_i p_i E_i$$

(need $-\sum_i p_i \log p_i$ bits on average)

Shannon's fundamental result
The entropy is a lower bound of compression.

Prefix property: no code forms a prefix of any other – a code string can be decoded unambiguously.

Huffman Coding

- Given that each symbol in the alphabet must occupy an integral number of bits in the encoding (i.e., instantaneous codes in which the encoding of one event can be decoded before encoding has begun for the next event), Huffman coding achieves "minimum redundancy" (optimal). Hence, it performs optimally if all symbol probabilities are powers of 1/2.
- JPEG, MPEG use run-length coding and Huffman coding.

Arithmetic Coding

- A message is represented by an interval of real numbers in [0,1).
- Arithmetic coding uses $\lfloor -\log_2 p \rfloor + 2$ bits.

Dictionary Techniques

Approaches to text compression can be divided into two classes: statistical (symbolwise) and dictionary (parsing). All previous methods are statistical.

Dictionary coding achieves compression by replacing groups of consecutive characters (phrases) with indexes into some dictionary. The dictionary is a list of phrases.

- Static dictionary encoder: the dictionary is fixed, irrespective of texts.
- Semiadaptive dictionary encoder: generate a dictionary specific to the text being encoded. A drawback is that the dictionary must now be stored and transmitted with the compressed text.
- Adaptive dictionary encoder: Ziv-Lempel

Ziv-Lempel Coding

- Idea: replace a string by a reference (pointer) to an earlier occurrence.
- pointer (m, l): l characters starting at position m.
- A family of algorithms

LZ77

Sliding window of fixed size N: first $N - F$ characters have already been encoded and last F is a lookahead buffer.

1. Initially, $N - F$ are spaces and first F of text are in the buffer.
2. Find a longest match with the buffer in the window. The match may overlap with the buffer. The longest match is coded into (i, j, a), where i is the offset of the match from the buffer, j is the length of the match, and a is the first character after the match in the buffer

3. The window is shifted $j + 1$ positions.

```
N=11, F=4
input:   abcabcbacbab..
output:  (,0,a)(,0,b)(,0,c)(3,3,b)(4,1,c)(3,2,b)..
```

LZ78

- Text seen so far is parsed into phrases, where each phrase is the longest matching phrase seen previously plus one char.
- Each phrase is encoded as an index (or as a pointer (m, l)) to its prefix plus one char. (The longest match cannot overlap because we find the match only with phrases in the dictionary.)
- No restriction on how far back a pointer may reach.

Encoding

1. Find the longest matching phrase.
2. Output (phrase index, next char).
3. Insert new phrase (matching phrase + next char) into dictionary.

```
input:         a    aa   b    ba   baa  baaa bab
parse number: 1    2    3    4    5    6    7
output:      (0,a)(1,a)(0,b)(3,a)(4,a)(5,a)(4,b)
```

Decoding

1. Output a new phrase from (phrase index, next char).
2. Insert the new phrase into dictionary.

- Searching can be implemented efficiently by inserting each phrase into a trie.

- Using suffix trees, the dictionary of phrases is stored in the suffix tree. A phrase is denoted by (i, l), where i is a start position of the phrase and l is the length of the phrase. The longest matching phrase starting from position i is $head_i$. (The longest match can overlap with $head_i$ by the definition of $head_i$ in McCreight's construction.)
- Compression is asymptotically optimal as the size of input increases.

LZW : Unix compress command

- Eliminate the extra char in the output. Output contains pointers only.
- Initialize the list of phrases to include every character in the alphabet. The last character of each new phrase is encoded as the first character of the next phrase.

```
input :  a  a  b  ab  aba  aa
output:  0  0  1  3   5    2

phrase num:  0 1  |  2  3  4  5   6
phrase    :  a b  |  aa ab ba aba abaa
derivation:          0a 0b 1a 3a  5a
```

Encoding

1. Find the longest matching phrase.
2. Output "phrase index".
3. Insert new phrase (matching phrase + next char) into dictionary.

Decoding

1. Output current phrase from "phrase index".
2. Insert a new phrase which is previous phrase + first char of current phrase into dictionary.

Decoding of 5 is tricky because phrase 5 is not available. but we know phrase 5 is abx for some x. If we put abx for the decoding of 5, we can see that phrase 5 is aba. So in this special case the last unknown character of new phrase (phrase 5) is the first character of previous phrase.