How to write an own WinZip

– from basic coding to Huffman codes –

Christiana Mavroyiakoumou & Georg Hahn

Imperial College London

Undergrad Colloquium, 14.03.2014

Overview

What is a file?

- A file is a sequence of 0s and 1s (called bits), e.g. 01010101010101...
- Each block of 8 bits is turned into a symbol, hence there are $2^8 = 256$ symbols, numbered off from 0 to 255
- The alphabet used to convert a number $0 - 255$ into a symbol is called the ASCII code ("American Standard Code for Information Interchange")
- Example: A file with "Hello" is converted into 8 bit ASCII as follows:

$$H = 072 = 01001000$$
$$e = 101 = 01100101$$
$$l = 108 = 01101100$$
$$o = 111 = 01101111$$
$$Hello = 01001000, 01100101, 01101100, 01101100, 01101111$$

Why should I care?

- This talk gives a full introduction to how information is stored digitally
- Main aim: efficient storage, also known as **data compression**
- For this, need efficient codes, introduced on the next slide
- After this talk, you will have all the tools needed to write an own WinZip-like compression program (pretty cool...)

What is a code?

Collection of symbols...

- a (finite or infinite) set of symbols A is called an *alphabet*
- the set of words built from A by concatenation is A^*

Now, can define a code...

- A code is a mapping $C(X) : R \to A^*$, where X is a random variable, A is an alphabet and R is the range of X
- C works for one symbol at a time only!
- Can extend to A^* by $C(x_1 \cdots x_n) := C(x_1) \cdots C(x_n)$

Example:

- $C(A) = 0,\ C(B) = 10,\ C(X) = 11$
- Then $C(BAX) = 10011$

What is a code?

- Suppose each symbol in X occurs with a probability $p(X = x)$
- Example: in a file of length 10 (bytes) with $C(A) = 0$, $C(B) = 10$, $C(X) = 11$, A occurs 2 times, B occurs 5 times, X occurs 3 times
- Then $p(X = A) = 2/10$, etc.
- Expected length of the code C:

$$L(C) = \sum_{x \in R} p(X = x) l(C(x))$$

- Example: in the above, the code has an expected length of

$$L(C) = 0.2(1) + 0.5(2) + 0.3(2) = 1.8$$

- This is the average bitlength of a symbol in the file:

  $\Rightarrow$ it measures how much space the information takes up!

Want to decode as well!

Encoding is not sufficient! Need to be able to uniquely decode:

- C non-singular if it is injective: $x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j)$
- C is uniquely decodable if its extension to A^* is non-singular, that is, if every codeword $C(s) \in A^*$ corresponds to just one string $s = x_1 \cdots x_n$.
- Codeword s is a prefix iff it is the beginning of a longer codeword
- C is a prefix code ("prefix") iff **no** codeword is the prefix of a longer codeword

Two issues:

- Every prefix code is also uniquely decodable!
- Examples!!!

Example: Which code is "good"?

Over the alphabet $A = \{0, 1\}$, consider the following codes for the three symbols $\{x, y, z\} = R$:

Symbol	C_1	C_2	C_3	C_4
x	1	0	00	0
y	1	1	11	10
z	1	01	001	11

- C_1 is clearly singular
- C_2 is not uniquely decodable as 01 could be xy or z
- C_3 is an example of a uniquely decodable code (why? – George!)
- C_4 is prefix and therefore uniquely decodable!

Main message

Observation

The second example shows that prefix codes are a real subset of uniquely decodable codes, i.e. there are more codes which are uniquely decodable than there are prefix codes.

A surprising result will follow in the next section...

Overview

Leon Kraft (Massachusetts Institute of Technology, MIT)

- Leon Kraft (1949) discovered a powerful property of prefix codes (Kraft in German means power – coincidence?)
- Necessary condition for the existence of prefix codes
- Shows how to construct prefix codes of given lengths

Theorem (Kraft inequality)

Let C be a code over an alphabet A of size $S = |A|$ consisting of n codewords with lengths $l_1, \ldots, l_n$. Then,

$$C \text{ is prefix} \Rightarrow \sum_{i=1}^{n} S^{-l_i} \leq 1.$$

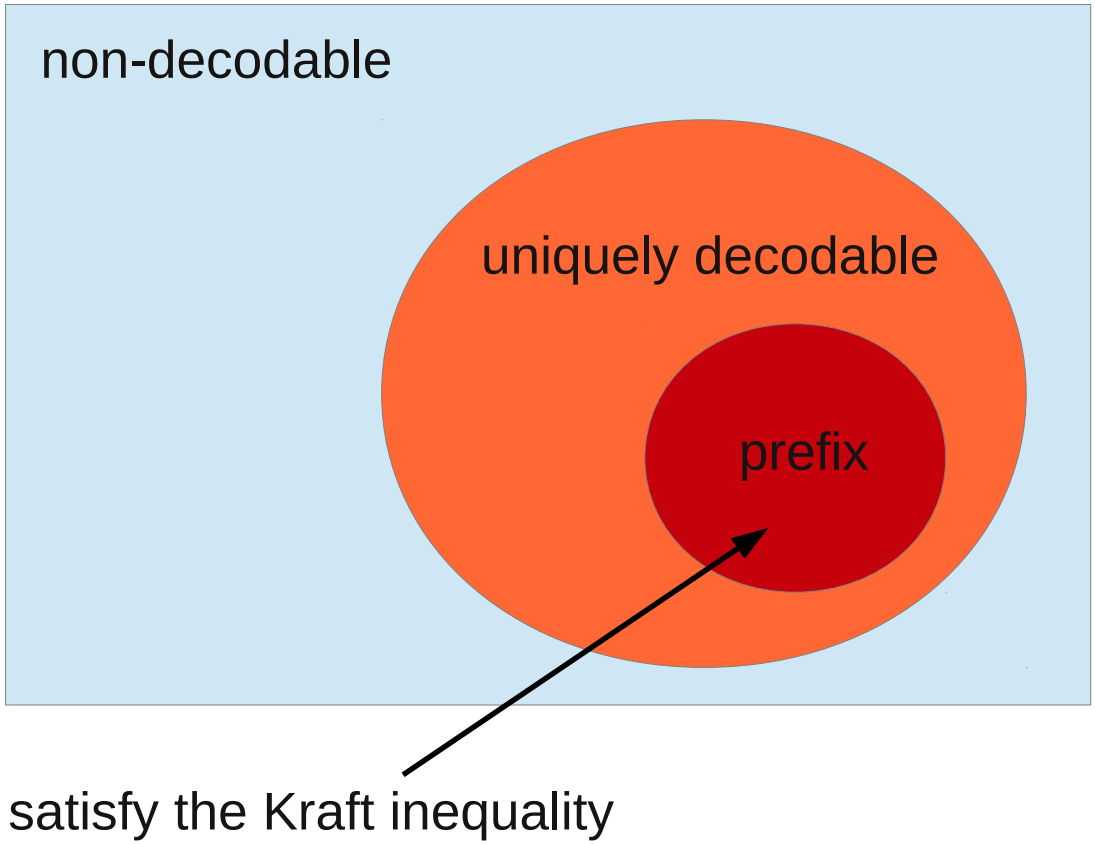
Conversely, for any n numbers $l_1, \ldots, l_n$ satisfying the Kraft inequality, there exists a prefix code $C = \{c_1, \ldots, c_n\}$ having these lengths $l(c_i) = l_i$.

The first statement is "just" interesting. The converse of Kraft is **crucial**!

Proof: George

A picture

All codes – the current situation



This is unexpected!

Surprisingly, exactly the same statement is true not only for prefix codes, but also for the larger set of uniquely decodable codes. This was proved by McMillan in 1956.

Theorem (McMillan)

Any uniquely decodable code C over an alphabet of size S satisfies the Kraft inequality, i.e. $\sum_{i=1}^{n} S^{-l_i} \leq 1$, where l_i are the codeword lengths.
Conversely, for any numbers $l_1, \ldots, l_n$ satisfying $\sum_{i=1}^{n} S^{-l_i} \leq 1$, there exists a **uniquely** *decodable code $C = \{c_1, \ldots, c_n\}$ with lengths $l(c_i) = l_i$.*

Proof: slightly more complicated than the one of the Kraft inequality and omitted

Important: McMillan's code is not prefix, it's just uniquely decodable.

A new picture

Both theorems together establish a surprising result:

$$\text{C prefix} \Rightarrow \text{C uniquely decodable}$$

as well as

C' uniquely decodable with $l_1, \ldots, l_n$
$\Rightarrow \sum_{i=1}^{n} S^{-l_i} \leq 1$ by McMillan
$\Rightarrow \exists$prefix code C'' with same codeword lengths $l_1, \ldots, l_n$ by converse of Kraft
$\Rightarrow L(C') = L(C'')$

Unexpected result

$L(C') = L(C'')$ means that the set of prefix codes is as powerful for encoding as the larger set of uniquely decodable codes.

Overview

Preliminaries

What can be said for the optimal code?

- Should be optimal with respect to minimal expected length: data compression!
- Application: computer, so will consider binary alphabet $A = \{0, 1\}$
- Can visualise the codes in a tree!

Main insight

By McMillan and converse of Kraft can assume that the optimal code is prefix.

The optimal code

Theorem

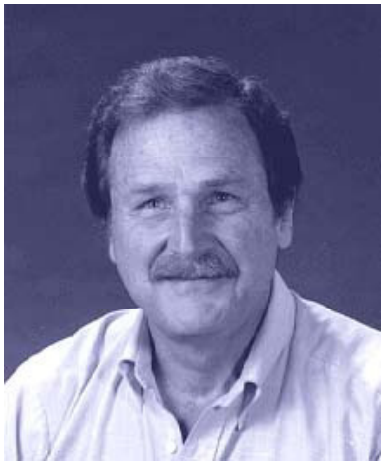
Let C be an optimal code over a binary alphabet to encode symbols $\{b_1, \ldots, b_n\}$, where symbol b_i occurs with probability p_i. Then the lengths l_i of the codewords $C = \{c_1, \ldots, c_n\}$, where $c_i \in \{0,1\}^*$, must satisfy the following properties.

1. *If $p_r > p_s$ for any $r, s \in \{1, \ldots, n\}$, then $l_r \leq l_s$.*
2. *The two longest codewords have equal length.*
3. *There is a pair of two longest codewords which are siblings in a tree representation.*

Proof: George

The Huffman code

- The last theorem gives the theoretical justification for the following optimal code for **single character compression**, called Huffman code.
- Developed by David Huffman (MIT):



The Huffman code: Example on the board + programs!

Construction of the Huffman code

1. The two least likely symbols will have the longest code lengths by property (1) and are siblings by (3), so they're joined into a supernode with added probability. This ensures that they will have equal length at the end as required by (2).
2. This reduces the number of nodes by one, so repeat the first step until only one node remains and the tree (*called Huffman tree*) is constructed.
3. Once the tree is constructed, label the two branches leaving every node with the symbols 0 and 1 and read off the codewords by following the paths from the root to the leaves. *This is the Huffman code.*
4. Replacing every symbol of the input by its Huffman code then yields a message length which is equal or shorter than the original length when compared in a joint alphabet. *No special symbol for separating the codes is needed as the Huffman code is a prefix code.*

Can you do better?

- Answer: Yes, a lot better
- Idea: don't compress single characters, compress substrings
- Example: ABABAB does contain two characters with equal probability, but the structure is clearly visible

Lempel-Ziv coding is optimal (in the limit) and cannot be beaten

Abraham Lempel

Jacob Ziv