# Golomb Code

CS6025 Data Encoding

Yizong Cheng

1-29-15

# Entropy and Code Length

- We have shown that the average prefix-free code length in the Huffman tree is bound by the entropy.
- $H = -\Sigma_t\ p_t \log_2 p_t$
- And the ideal code length for a symbol t with probability $p_t$ is $-\log_2 p_t$.
- For unary code, the number n is coded with n bits.
- Unary code is optimal if $n = -\log_2 p_n$. Or $p_n = 2^{-n}$, or $p_n = p_{n-1}/2$.
- For gamma code, the number n is coded with $2\log_2 n$ bits.
- Gamma code is optimal if $2\log_2 n = -\log_2 p_n$. Or $p_n = 1/n^2$.
  - More precisely, n is coded with ceiling($2\log_2 n - 1$) bits.

# Golomb Code with Parameter b

- To code a number n, find the quotient q = n / b and remainder r = n % b.
- Unary code for q with q + 1 bits, followed by $\log_2 b$ bits for the binary code for r.
- The code length for n is $(n/b)+\log_2 b$.
- Make this $-\log_2 p_n = (n/b)+\log_2 b$ and solve for $p_n$.
- $p_n = (1/b)(2^{-1/b})^n = (2^{-1/b}/b)(2^{-1/b})^{n-1}$.
- Geometric distribution with $p_n = (2^{-1/b})p_{n-1}$. (exponential decay)
  - Unary code has b = 1 or $p_n=(1/2)p_{n-1}$.
  - $b = 1/\log_2(p_{n-1}/p_n)$
- $\Sigma_n(2^{-1/b})^{n-1} = 1 - 2^{-1/b}$ .

# Golomb Code G(k)

- G(k) stands for Golumb code for parameter $b = 2^k$.
- No integer division and modulus operations are needed.
- Using bitwise operations only.
- The quotient q = n / b = n >> k.
  - Output q + 1 bits as the unary code for q.
- The binary representation of the remainder is a sequence of k bits and they can be obtained with
- for (int mask = 1 <<= k – 1; mask > 0; mask >>= 1)
- outputBit( n & mask > 0 );

# Example: G(2) for 19

- k = 2, b = $2^k$ = 4, remainderMask = $2^k$ – 1 = 3
- 19 is 0x13 or $10011_2$, 19 / 4 is 4 and 19 % 4 = 3.
- Quotient q = $10011_2$ >> k = $100_2$ = 4
- Remainder r = $10011_2$ & $11_2$ = $11_2$ = 3
- G(2) for 19 is 00001|11 ( unary for q and binary for r )

# Golomb Code LG(k, glimit) with range (256)

- LG(k, glimit) with L = $\log_2$ (range).
- Compute the quotient q = n / $2^k$ (q = n >> k).
- If q < glimit - L – 1, LG(k, glimit) is G(k).
- If the q >= glimit – L – 1, unary code for glimit – L – 1, followed by binary code for n with length = L.
- We will try LG(2, 32) with the range 256.
- L = 8, glimit – L – 1 = 32 – 8 – 1 = 23.
- The smallest n with q >= 23 = $10111_2$ is 23 x 4 = 92 = $1011100_2$.
- From 92 on the codeword length is 24 + 8 = 32 bits.

| Prediction error | Mapped value | Code |
|---|---|---|
| 0 | 0 | 1 00 |
| −1 | 1 | 1 01 |
| 1 | 2 | 1 10 |
| −2 | 3 | 1 11 |
| 2 | 4 | 01 00 |
| −3 | 5 | 01 01 |
| 3 | 6 | 01 10 |
| −4 | 7 | 01 11 |
| 4 | 8 | 001 00 |
| −5 | 9 | 001 01 |
| 5 | 10 | 001 10 |
| −6 | 11 | 001 11 |
| 6 | 12 | 0001 00 |
| −7 | 13 | 0001 01 |
| 7 | 14 | 0001 10 |
| −8 | 15 | 0001 11 |
| 8 | 16 | 00001 00 |
| −9 | 17 | 00001 01 |
| 9 | 18 | 00001 10 |
| −10 | 19 | 00001 11 |
| 10 | 20 | 000001 00 |
| −11 | 21 | 000001 01 |
| 11 | 22 | 000001 10 |
| −12 | 23 | 000001 11 |
| 12 | 24 | 0000001 00 |
| ... | | |
| 50 | 100 | 000000000000 |
| | | 000000000001 |
| | | 01100011 |

5: Prediction Errors, Their Mappings, and $LG(2, 32)$ Codes.

# LG232.txt

| | | |
|---|---|---|
| 100 | 00000100 | 0000000000100 |
| 101 | 00000101 | 0000000000101 |
| 110 | 00000110 | 0000000000110 |
| 111 | 00000111 | 0000000000111 |
| 0100 | 000000100 | 00000000000100 |
| 0101 | 000000101 | 00000000000101 |
| 0110 | 000000110 | 00000000000110 |
| 0111 | 000000111 | 00000000000111 |
| 00100 | 0000000100 | 000000000000100 |
| 00101 | 0000000101 | 000000000000101 |
| 00110 | 0000000110 | 000000000000110 |
| 00111 | 0000000111 | 000000000000111 |
| 000100 | 00000000100 | 0000000000000100 |
| 000101 | 00000000101 | 0000000000000101 |
| 000110 | 00000000110 | 0000000000000110 |
| 000111 | 00000000111 | 0000000000000111 |
| 0000100 | 000000000100 | 00000000000000100 |
| 0000101 | 000000000101 | 00000000000000101 |
| 0000110 | 000000000110 | 00000000000000110 |
| 0000111 | 000000000111 | 00000000000000111 |

0000000000000100
0000000000000101
0000000000000110
0000000000000111
0000000000000100
0000000000000101
0000000000000110
0000000000000111
0000000000000100
0000000000000101
0000000000000110
0000000000000111
0000000000000100
0000000000000101
0000000000000110
0000000000000111
0000000000000100
0000000000000101
0000000000000110
0000000000000111

0000000000000000000100
0000000000000000000101
0000000000000000000110
0000000000000000000111
0000000000000000000100
0000000000000000000101
0000000000000000000110
0000000000000000000111
0000000000000000000100
0000000000000000000101
0000000000000000000110
0000000000000000000111
00000000000000000000101011100
00000000000000000000101011101
00000000000000000000101011110
00000000000000000000101011111
00000000000000000000101100000
00000000000000000000101100001
00000000000000000000101100010
00000000000000000000101100011

# Homework 6: due 2-4-15

- Complete the deGolomb() function in H6.java that reads a codeword of LG(2,32) and returns the byte (0-255) it encodes.
- LG232.txt contains 256 codewords of LG(2,32) and can be used with H3A on any file and the completed H6.java is the inverse of H3A LG232.txt.
  - java H3A LG232.txt < original > encoded
  - java H6 < encoded > original
- LG(2,32) is best used on images transformed with H4A or H5A. The file test6.lg2 is the result of H3A LG232.txt on the result of H5A on an original image. Use H6 on test6.lg2 an then your H5C or H5B on the result to recover the original image.

```
public class H6{
  static final int GolombK = 2;
  static final int GolombLimit = 23;
  static final int[] powersOf2 = new int[]{
   1, 2, 4, 8, 16, 32, 64, 128 }; // used by deGolomb
  int buf = 0; int position = 0;



  void decode(){
   int symbol = -1;
   while ((symbol = deGolomb()) >= 0)
     System.out.write(symbol);
   System.out.flush();
  }

  public static void main(String[] args){
   H6 h6 = new H6();
   h6.decode();
  }
 }
```

```
int inputBit(){  // 0, 1, or -1 for end of file
  if (position == 0)
    try{
      buf = System.in.read();
      if (buf < 0){ return -1;
}
      position = 0x80;
    }catch(IOException e){
       System.err.println(e);
       return -1;
    }
  int t = ((buf & position) == 0) ? 0 : 1;
  position >>= 1;
  return t;
}

int deGolomb(){ // get the next codeword, return the symbol it encodes
  int value = 0;
  // Your code
  return value;
}
```

# Pseudo Code for deGolomb

- Initialize ints value to 0, q (quotient) to 0, and bit to -1.
- While bit = inputbit() is 0, increment q.
- If bit is -1, return -1 (end of file).
- If q < golombLimit do
  - value is q * $2^{golombK}$, which can be done as q << golombK.
  - Input golombK bits and turn them into the remainder and add to value
- Otherwise read the next 8 bits and they represent the symbol.
- Return value.
- powersOf2 can be used to turn a bitstring into a number.