Class 10: Huffman Coding

February 15th, 2012

Outline
- Prefix Codes and Huffman Coding
- Making the Dictionary
- Encoding
- Decoding

February 15th, 2012

Dr David A. Huffman

David Albert Huffman (1925–1999) was a pioneer in computer science, known for his Huffman coding.

In 1951, David Huffman and his MIT information theory classmates were given the choice of a term paper or a final exam.

Huffman chose the latter and finished with a paper in 1952: “A Method for the Construction of Minimum Redundancy Codes”

February 15th, 2012
**Morse Code**

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
| . | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

**Morse Code - Solution**

In Morse code: Leave a three dot space between the letters of one word.

- J
- E O
- A M
- W T

**Prefix Code**

- A prefix code is a type of variable-length code system which has **prefix property**.

- Prefix property states that there is no valid code word in the system that is a prefix (start) of any other valid code word in the set.
  - The code set {9, 59, 55} has the prefix property.
  - The code set {9, 5, 59, 55} does not.

- With a prefix code, a receiver can identify each word without requiring a special marker between words.
  - Huffman coding system is a prefix code system.
ASCII Code (Character = Byte)

<table>
<thead>
<tr>
<th>Character</th>
<th>Code</th>
<th>Number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>01000001</td>
<td>8</td>
</tr>
<tr>
<td>d</td>
<td>01000000</td>
<td>8</td>
</tr>
<tr>
<td>e</td>
<td>01000001</td>
<td>8</td>
</tr>
<tr>
<td>n</td>
<td>01101110</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>00101110</td>
<td>8</td>
</tr>
<tr>
<td>blank</td>
<td>00000000</td>
<td>8</td>
</tr>
</tbody>
</table>

Consider the text:
“ed needed a deed. anna added a dead end.”

Size: 40 characters
• Needs 40 bytes (or 320 bits) to encode in ASCII.

February 15th, 2012

Squeezing Each Character to 3 Bits

<table>
<thead>
<tr>
<th>Character</th>
<th>Code</th>
<th>Number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>000</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>001</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>010</td>
<td>3</td>
</tr>
<tr>
<td>n</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>blank</td>
<td>101</td>
<td>3</td>
</tr>
</tbody>
</table>

Text: “ed needed a deed. anna added a dead end.”
Size: 40 characters

Encoding: 40x3 = 120 bits
Compression factor: 120 / 320 = 37.5%

February 15th, 2012

Example Huffman Code

<table>
<thead>
<tr>
<th>Character</th>
<th>Code</th>
<th>Number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>00</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>110</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1110</td>
<td>4</td>
</tr>
<tr>
<td>blank</td>
<td>1111</td>
<td>4</td>
</tr>
</tbody>
</table>

Text: “ed needed a deed. anna added a dead end.”
Size: 40 characters

Encoding: 8x2 (blanks) + 11x2 (d's) + 9x2 (e's) + 6x3 (a's) + 4x4 (n's) + 2x4 (dots) = 98 bits

Compression factor: 98/320 = 30.63%

February 15th, 2012

Huffman Code

Huffman Code is one of the statistical coding systems.

In reality, not all the symbols (characters) occur with the same frequency (or the same probability).
• Should we assign the same length of code for all of them (fixed coding like in ASCII)?
• Can we save space by tailoring the codes to the frequency of the symbols?

Code word length will be variable and shorter for more frequently used symbols.

February 15th, 2012
How to Encode and Decode

- Compute the frequency or probability of each symbol in the target data.
- Build the Huffman code tree based on the probabilities.
- Traverse the tree to determine all the code words.
- To encode (compress) scan the data, encode using the code table and attach the code table to the data.
- To decode (decompress) scan the data and decode using the code table.

How To Build the Tree (1)

Text: “ed needed a deed. anna added a dead end.”
- $A_x=\{a,d,e,n,.,blank\}$
- $P_x=\{6,11,9,4,2,8\}$

How To Build the Tree (2)

Each symbol (character) is a single node tree at first.
Sort roots of all trees by frequency:
- Sorted list = $\{\text{dot}(2), n(4), \text{a}(6), \text{blank}(8), \text{e}(9), \text{d}(11)\}$
Join first two trees in a new tree and set the root value to sum of the values.
**How To Build the Tree (3)**

Re-sort roots of all trees:
- Sorted list = \{a(6), T1(6), blank(8), e(9), d(11)\}

Join first two trees in a new tree and set the root value to sum of the values.

*Repeat process until we have a single tree.*

![Tree Diagram](February 15th, 2012)

**How To Build the Tree (4)**

Re-sort roots of all trees:
- Sorted list = \{blank(8), e(9), d(11), T2(12)\}

Join first two trees...

![Tree Diagram](February 15th, 2012)

**How To Build the Tree (5)**

Re-sort roots of all trees:
- Sorted list = \{d(11), T2(12), T3(17)\}

Join first two trees...

![Tree Diagram](February 15th, 2012)

**How To Build the Tree (6)**

Re-sort roots of all trees:
- Sorted list = \{T3(17), T4(23)\}

Join first two trees...

![Tree Diagram](February 15th, 2012)
How To Build the Tree (7)

Encode And Decode

After we found the Huffman Codes tailored to our data...

To Compress:
- Include the code table before the compressed data in the output
- Simply replace every symbol in the uncompressed input data with its corresponding code

To Decompress:
- Read the code table block at the beginning of the compressed data
- Replace every prefix code with its correspondent symbol

Huffman Code - Compression Algorithm

Input: sequence of bytes (original file)
Output: sequence of bits (compressed file)

Read Huffman code table
Repeat for each byte \( b \) in the input file:
- Read \( b \) from the file
- In Huffman code table, look up code sequence corresponding to \( b \)
Output code sequence

February 15th, 2012

Huffman Code – Decompression Algorithm

Input: sequence of bits (compressed file)
Output: sequence of bytes (original file)

Read Huffman code table
While there are more bits left in the input file:
- Read a bit
While the bits read since last output do not form a Huffman code sequence:
- Read another bit
When we have the bits of a Huffman code sequence:
- Output corresponding symbol (byte)

February 15th, 2012
Huffman Code

• Zip compression format uses Huffman Coding.

• Including a code table in case of small files may defeat the purpose by increasing the resulting compressed file.

• To solve that, compression software has some standard known code tables for different types of data (e.g. different languages texts).

• They can include a code in the compressed data which implies that a predefined code table is used.

Summary

• Each Huffman code assigns a sequence of bits to a character:
  • More frequent = shorter sequence

• Each Huffman code sequence corresponds to a Huffman tree.

• Can always tell where each code sequence ends.