Topic 20: Huffman Coding

The author should gaze at Noah, and ... learn, as they did in the Ark, to crowd a great deal of matter into a very small compass.

Sydney Smith, Edinburgh Review

Encoding

- UT CS
- 85 84 32 67 83
- 01010101 01010100 00100000 01000011 01010011

- what is a file?
- open a bitmap in a text editor
- open a pdf in word

Agenda

- Encoding
- Compression
- Huffman Coding
JPEG File

JPEG VS BITMAP

- JPEG File
- Bitmap image
- JPEG Image

Encoding Schemes

- "It's all 1s and 0s"
- What do the 1s and 0s mean?
- 50 121 109
- ASCII -> 2ym
- Red Green Blue-> dark teal?

Agenda

- Encoding
- Compression
- Huffman Coding
Compression

- Compression: Storing the same information but in a form that takes less memory
- lossless and lossy compression
- Recall:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tower-number1-1024x768.bmp</td>
<td>Bitmap image</td>
<td>2,305 KB</td>
</tr>
<tr>
<td>Tower-number1-1024x768.jpg</td>
<td>JPEG Image</td>
<td>283 KB</td>
</tr>
</tbody>
</table>

Lossy Artifacts

Why Bother?

- Is compression really necessary?

2 Terabytes
500 HD, 2 hour movies or 500,000 songs
Price? About $100.00

Home Internet Access
- 40 Mbps roughly $40 per month.
- 12 months * 3 years * $40 = $1,440
- 40,000,000 bits / second = 5.0 * 10^6 bytes / sec

CPU Capability
- $1,500 for a laptop or desktop
- Intel i7 processor
- Assume it lasts 3 years.
- Memory bandwidth
  25.6 GB / sec
  = 2.6 * 10^10 bytes / sec
  on the order of 5.0 * 10^10 instructions / second
Mobile Devices?

**Cellular Network**
- Your mileage may vary …
- Mega bits per second
- AT&T
  - 17 download, 7 upload
- T-Mobile & Verizon
  - 12 download, 7 upload
- 17,000,000 bits per second = 2.125 x 10^6 bytes per second

**iPhone CPU**
- Apple A6 System on a Chip
- Covy about IPS
- 2 cores
- Rough estimates: 1 x 10^10 instructions per second

http://tinyurl.com/q6o7wan

Little Pipes and Big Pumps

![CPU](image)

Data In From Network

Agenda

- Encoding
- Compression
- Huffman Coding

Compression - Why Bother?

- Apostolos "Toli" Lerios
- Facebook Engineer
- Heads image storage group
- jpeg images already compressed
- look for ways to compress even more
- 1% less space = millions of dollars in savings
Purpose of Huffman Coding

- Proposed by Dr. David A. Huffman
  - A Method for the Construction of Minimum Redundancy Codes
  - Written in 1952
- Applicable to many forms of data transmission
  - Our example: text files
  - still used in fax machines, mp3 encoding, others

The Basic Algorithm

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency!
- Yet in ASCII all characters are allocated the same amount of space
  - 1 char = 1 byte, be it e or X

The Basic Algorithm

1. Scan text to be compressed and tally occurrence of all characters.
2. Sort or prioritize characters based on number of occurrences in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all code words.
5. Scan text again and create new file using the Huffman codes
Building a Tree
Scan the original text

• Consider the following short text

Eerie eyes seen near lake.

• Count up the occurrences of all characters in the text

Eerie eyes seen near lake.

Building a Tree
Scan the original text

• What characters are present?

E r i space
y s n a r l k.

Building a Tree
Scan the original text

Eerie eyes seen near lake.

• What is the frequency of each character in the text?

<table>
<thead>
<tr>
<th>Char</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>8</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>space</td>
<td>4</td>
</tr>
<tr>
<td>y</td>
<td>5</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
</tr>
</tbody>
</table>

Building a Tree
Prioritize characters

• Create binary tree nodes with character and frequency of each character

• Place nodes in a priority queue
  – The lower the occurrence, the higher the priority in the queue
Building a Tree

- The queue after inserting all nodes

```
E i k l y . a n r s s p e
1 1 1 1 1 2 2 2 2 4 8
```

- Null Pointers are not shown

Building a Tree

- While priority queue contains two or more nodes
  - Create new node
  - Dequeue node and make it left subtree
  - Dequeue next node and make it right subtree
  - Frequency of new node equals sum of frequency of left and right children
  - Enqueue new node back into queue

```
E i k l y . a n r s s p e
1 1 1 1 1 2 2 2 2 4 8
```
Building a Tree

33

Building a Tree

34

Building a Tree

35

Building a Tree

36
Building a Tree

```
  a  n  r  s  2  2  2  1  i  1  k  l  1  1  y  1  i  1  sp  4  e  8
```

Building a Tree

```
  r  s  2  2  2  1  i  1  k  l  1  1  y  1  i  1  sp  4  e  8
```

Building a Tree

```
  r  2  2  2  1  i  1  k  l  1  1  y  1  i  1  a  2  n  2
```

Building a Tree

```
  r  s  2  2  2  1  i  1  k  l  1  1  y  1  i  1  sp  4  e  8
```
Building a Tree

What is happening to the characters with a low number of occurrences?
Building a Tree

```
       10
      /  \
     4    6
    /  \
   2    2
  /  \
 e   a
```

Building a Tree

```
      10
     /    \
    4      6
   /       /  \
 2       2    2
 /       /  \
E i   a  n
  /\   /  \
 i k l  2
 /  \
1 1   y
```
Building a Tree

- After enqueueing this node there is only one node left in priority queue.

Encoding the File
Traverse Tree for Codes

- Perform a traversal of the tree to obtain new code words
  - left, append a 0 to code word
  - right append a 1 to code word
  - code word is only completed when a leaf node is reached
Encoding the File

- Rescan text and encode file using new code words
  Eerie eyes seen near lake.

<table>
<thead>
<tr>
<th>Char</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>0000</td>
</tr>
<tr>
<td>i</td>
<td>0001</td>
</tr>
<tr>
<td>k</td>
<td>0010</td>
</tr>
<tr>
<td>ly</td>
<td>0011</td>
</tr>
<tr>
<td>.</td>
<td>0100</td>
</tr>
<tr>
<td>space</td>
<td>0101</td>
</tr>
<tr>
<td>e</td>
<td>0110</td>
</tr>
<tr>
<td>a</td>
<td>1000</td>
</tr>
<tr>
<td>n</td>
<td>1101</td>
</tr>
<tr>
<td>r</td>
<td>1110</td>
</tr>
<tr>
<td>s</td>
<td>1111</td>
</tr>
</tbody>
</table>

00010111000011001110
010010111101111111010
11010111011011001110
011001111000010100101

Encoding the File

- Have we made things any better?
- 82 bits to encode the text
- ASCII would take 8 * 26 = 208 bits
- If modified code used 4 bits per character are needed. Total bits 4 * 26 = 104. Savings not as great.

Decoding the File

- How does receiver know what the codes are?
- Tree constructed for each text file.
  - Considers frequency for each file
  - Big hit on compression, especially for smaller files
- Tree predetermined
  - based on statistical analysis of text files or file types

Decoding the File

- Once receiver has tree it scans incoming bit stream
- 0 ⇒ go left
- 1 ⇒ go right

A. elk nay sir
B. eek a snake
C. eek kin sly
D. eek snarl nil
E. eel a snarl
Assignment Hints

- reading chunks not chars
- header format
- the pseudo eof character
- the GUI

Assignment Example

- "Eerie eyes seen near lake." will result in different codes than those shown in slides due to:
  - adding elements in order to PriorityQueue
  - required pseudo eof character (PEOF)

Assignment Example

<table>
<thead>
<tr>
<th>Char</th>
<th>Freq.</th>
<th>Char</th>
<th>Freq.</th>
<th>Char</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>y</td>
<td>1</td>
<td>k</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>s</td>
<td>2</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>n</td>
<td>2</td>
<td>PEOF</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>a</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>space</td>
<td>4</td>
<td>l</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Assignment Example

```
. E i k l y PEOF a n r s S P e
1 1 1 1 1 1 1 1 2 2 2 2 4 8
```
Codes

value: 32, equivalent char: , frequency: 4, new code 011
value: 46, equivalent char: ., frequency: 1, new code 11110
value: 69, equivalent char: E, frequency: 1, new code 11111
value: 97, equivalent char: a, frequency: 2, new code 0101
value: 101, equivalent char: e, frequency: 8, new code 10
value: 105, equivalent char: i, frequency: 1, new code 0000
value: 107, equivalent char: k, frequency: 1, new code 0001
value: 108, equivalent char: l, frequency: 1, new code 0010
value: 110, equivalent char: n, frequency: 2, new code 1100
value: 114, equivalent char: r, frequency: 2, new code 1101
value: 115, equivalent char: s, frequency: 2, new code 1110
value: 121, equivalent char: y, frequency: 1, new code 0011
value: 256, equivalent char: ?, frequency: 1, new code 0100