CSE 100: HUFFMAN CODES
Doing data compression the right way...
The Data Compression Problem

Digitize  Store  Communicate
But alas this lovely text must be decomposed to bits.

Good news is: if we decompose well, we can recover it 😊

RIGHT HO, JEEVES

By

P. G. WODEHOUSE

1922

To

RAYMOND NEEDHAM, K.C.
WITH AFFECTION AND ADMIRATION

"Jeeves," I said, "may I speak frankly?"
"Certainly, sir."
"What I have to say may wound you."
"Not at all, sir."
"Well, then—"
No—wait. Hold the line a minute. I've gone off the rails.

I don't know if you have had the same experience, but the snag I always come up against when I'm telling a story is this dashed difficult problem of where to begin it. It's a thing you don't want to go wrong over, because one false step and you're sunk. I mean, if you fool about too long at the start, trying to establish atmosphere, as they call it, and all that sort of rot, you fail to grip and the customers walk out on you.

Get off the mark, on the other hand, like a scalded cat, and your public is at a loss. It simply raises its eyebrows, and can't make out what you're talking about.

And in opening my report of the complex case of Gussie Fink-Nottle, Madeline Bassett, my Cousin Angela, my Aunt Dahlia, my Uncle Thomas, young Tuppy Glossop and the cook, Anatole, with the above spot of dialogue, I see that I have made the second of these two floaters.

I shall have to hark back a bit. And taking it for all in all and weighing this against that, I suppose the affair may be said to have had its inception, if inception is the word I want, with that visit of mine to Cannes. If I hadn't gone to Cannes, I shouldn't have met the Bassett or bought that white mess jacket, and Angela wouldn't have met her shark, and Aunt Dahlia wouldn't have played baccarat.

Yes, most decidedly, Cannes was the point d'appui.

Right ho, then. Let me marshal my facts.

I went to Cannes—leaving Jeeves behind, he having intimated that he did not wish to miss Ascot—round about the beginning of June. With me travelled my Aunt Dahlia and her daughter Angela. Tuppy Glossop, Angela’s betrothed, was to have been of the party, but at the last moment couldn’t get away. Uncle Tom, Aunt Dahlia’s husband, remained at home, because he can’t stick the South of France at any price.
How to represent in bits?
• Encoding with ASCII
• Symbols and dictionaries

How many bits do we need?
Fixed length encoding

• Fixed length: each symbol is represented using a fixed number of bits

• For example if the symbols were ‘s’, ‘p’, ’a’, ‘m’ we might define the following encoding:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code word</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

For a dictionary consisting of M symbols, what is the minimum number of bits needed to encode each symbol (assume fixed length binary codes) ?

A. $2^M$  B. M  C. M/2  D. $\text{ceil}(\log_2 M)$  E. None of these
Binary codes as Binary Trees

Symbols are leaf nodes
Root to leaf node gives the codeword for each symbol

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>
Encoding with Binary Trees

Given the above binary tree, encode the string “papa”
Encoding a symbol- think implementation!

A very bad way is to start at the root and search down the tree until you find the symbol you are trying to encode.
Encoding a symbol

A much better way is to maintain a list of leaves and then to traverse the tree to the root (and then reverse the code)

```
vector<HCNode*> leaves;
...
leaves = vector<HCNode*>(256, (HCNode*)0);
```
Traversing a list, with recursion

class LNode {
    int data;
    LNode* next;
}

void traverse(LNode* n) {
    // 1
    if (n == nullptr) return;
    // 2
    traverse(n->next);
    // 3
}

Where should I put the line to print \texttt{n->data} to print the list in \textit{reverse order}?
\texttt{std::cout << n->data << std::endl;}

A. 1 \quad B. 2 \quad C. 3
A much better way is to maintain a list of leaves and then to traverse the tree to the root. If you use recursion, there’s no need to reverse the list!

Use recursion to easily write a symbol’s code in the correct order!

```cpp
vector<HCNode*> leaves;
...
leaves = vector<HCNode*>(256, (HCNode*)0);
```
Decoding on binary trees

Decode the bitstream 110101001100 using the given binary tree
A. scam
B. mork
C. rock
D. korp
Is there an alternative to fixed length encoding?

- Can we go beyond fixed length encoding?
- What if certain symbols appeared more often than others?

---

Composer: Trad.
Author: Eric Idle

Virgin Records 1989
MONT D1

ALWAYS LOOK ON THE BRIGHT SIDE OF LIFE

Some things in life are bad
They can really make you mad
Other things just make you swear and curse
When you're chewing on life's gristled
Don't grumble, give a whistle
And this'll help things turn out for the best...
And...
(music slides into the song)

... Always look on the bright side of life...
(whistle)
... Always look on the bright side of life...
(whistle)

If life seems jolly rotten
There's something you've forgotten
And that's to laugh and smile and dance and sing
When you're feeling in the dumps
Don't be silly chumps
Just purse your lips and whistle _ that's the thing.
And...

... Always look on the bright side of life...
(whistle)
Come on,
(others start to join in)
... Always look on the bright side of life...
(whistle)

For life is quite absurd
And death's the final word
You must always face the curtain with a bow
Forget about your sin
Give the audience a grin
Enjoy it _ it's your last chance anyhow.

So always look on the bright side of death
Just before you draw your terminal breath
Variable length codes

Text file

```
ssssssssssssssssss
ssppppaampamm
```

Symbol Counts
- s: 18
- p: 6
- a: 3
- m: 3

Symbol Frequency
- s: 0.6
- p: 0.2
- a: 0.1
- m: 0.1

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Code B

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Is code B better than code A?
A. Yes
B. No
C. Depends
Comparing encoding schemes

ssssssssssssssssss
ssppppaampamm

Symbol   Frequency
s        0.6
p        0.2
a        0.1
m        0.1

Symbol Codeword
s        00
p        01
a        10
m        11

Code A

Symbol Codeword
s        0
p        1
a        10
m        11

Code B

Average length (code A) = 2 bits/symbol
Average length (code B) = 0.6 * 1 + 0.2 * 1 + 0.1 * 2 + 0.1 * 2
= 1.2 bits/symbol
Decoding variable length codes

Symbol Frequency
s 0.6
p 0.2
a 0.1
m 0.1

Code A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Code B

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Decode the binary pattern 0110 using Code B?
A. spa  
B. sms  
C. Not enough information to decode
Variable length codes

Variable length codes have to necessarily be prefix codes for correct decoding.

A prefix code is one where no symbol’s codeword is a prefix of another.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

Code B is not a prefix code
Variable length codes

Is code C better than code A and code B? (Why or why not?)
A. Yes
B. No

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code A</th>
<th>Code B</th>
<th>Code C</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
<td>10</td>
<td>110</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
<td>11</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>
### Variable length codes

#### Symbol Frequency

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>

#### Code A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

#### Code B

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

#### Code C

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>10</td>
</tr>
<tr>
<td>a</td>
<td>110</td>
</tr>
<tr>
<td>m</td>
<td>111</td>
</tr>
</tbody>
</table>

Average length (code A) = 2 bits/symbol
Average length (code B) = 0.6 * 1 + 0.2 * 1 + 0.1 * 2 + 0.1 * 2
= 1.2 bits/symbol
Average length (code C) = 0.6 * 1 + 0.2 * 2 + 0.1 * 3 + 0.1 * 3
= 1.6 bits/symbol
What is the advantage of thinking of codes as trees?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>00</td>
</tr>
<tr>
<td>p</td>
<td>01</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

**Code A**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>10</td>
</tr>
<tr>
<td>m</td>
<td>11</td>
</tr>
</tbody>
</table>

**Code B**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
</tr>
<tr>
<td>p</td>
<td>10</td>
</tr>
<tr>
<td>a</td>
<td>110</td>
</tr>
<tr>
<td>m</td>
<td>111</td>
</tr>
</tbody>
</table>

**Code C**
Problem Definition

Given a frequency distribution over $M$ symbols, find the optimal prefix binary code i.e. one that minimizes the average code length

**In short:**

I give you frequencies, you give me the best tree!

---

**Symbol** | **Frequency**
---|---
$s$ | 0.6
$p$ | 0.2
$a$ | 0.1
$m$ | 0.1

Huffman coding is one of the fundamental ideas that people in computer science and data communications are using all the time - Donald Knuth

David Huffman
Problem Definition (reworded for trees)

Input: The frequency \( (p_i) \) of occurrence of each symbol \( (S_i) \)

Output: Binary tree \( T \) that minimizes the following objective function:

\[
L(T) = \sum_{i=1:N} p_i \cdot \text{Depth}(S_i \text{ in } T)
\]

Solution: Huffman Codes
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node

A: 6; B: 4; C: 4; D: 0; E: 0; F: 0; G: 1; H: 2
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them

A: 6; B: 4; C: 4; D: 0; E: 0; F: 0; G: 1; H: 2
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them

A: 6; B: 4; C: 4; D: 0; E: 0; F: 0; G: 1; H: 2
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree
Huffman’s algorithm: Bottom up construction

- Build the tree from the bottom up!
- Start with a forest of trees, all with just one node
- Choose the two smallest trees in the forest and merge them
- Repeat until all nodes are in the tree
PA2: encoding/decoding

ENCODING:
1. Scan text file to compute frequencies
2. Build Huffman Tree
3. Encode: Find code for every symbol (letter)
4. Create new compressed file by saving the entire code at the top of
   the file followed by the code for each symbol (letter) in the file

DECODING:
1. Read the file header (which contains the code) to recreate the tree
2. Decode each letter by reading the file and using the tree

This is the logical flow of what needs to happen, it need not be the
way you develop the code: Brainstorm in class and in section to see
how you can design in a modular way