CS 314: Huffman codes

Announcements

- Next week is oral grading - sign up Monday
- HW is out
- Midterm 1 - Friday the 5th (right before break)
  or Friday the 19th (right after break)
Binary encodings

How do computers encode characters?

binary!

ASCII - each character gets 8 bits

01000001 = (41)$_{16}$ - A
(42)$_{16}$ - B
What if sending a bit is expensive?

Won't use same # of bits per letter, since some (e.g., e, a) are more common than others (e.g., 2).

Leads to data compression.
Oldest form (at least common in US):

Morse code

```
e: o, t: −1, a: 0 1
```

(really binary, since uses pause)

Disadvantage: How to translate $0101$?

```
\{aa, et, eta\}
```

ambiguous
Ambiguity comes from one string being a prefix of another.

(So first 0 could be an e, or could start on an a)

Prefix-free codes: No letter's code is a prefix of another.

Advantage: - as we scan, each possibility is unique
- quick linear scan
Example: MISSISSIPPI

\[
\begin{align*}
M & : 111 \quad \rightarrow \quad S & : 0 \\
I & : 10 \quad \rightarrow \quad I & : 1 \\
S & : 0 \quad \rightarrow \quad P & : 110 \\
\end{align*}
\]

Encoding: 11100010001011011010

\[
\frac{\text{MISS}}{21 \text{ bits}}
\]

(ASCII: 88 bits)
Can visualize as a binary tree

M: 111
T: 10
S: 0
P: 110

When reading string, just follow tree and output letter when you reach a leaf.

Ex: 0 1 0 1 1 1 1 0

S I M P
Question: How can we find minimal prefix free codes?

Given n letters plus frequency counts for each letter, $f[1] \ldots f[n]$, find the code minimizing total length:

$$\sum_{i=1}^{n} f[i] \cdot \text{depth}(i) = \text{cost}(T)$$
Huffman codes (1952)

Huffman designed a greedy algorithm:

Merge the two least frequent letters and recurse.

This is optimal!
Pseudo code:

Keep characters in min heap, w/priority = to frequency.

L, R, P keep track of left/right and parent indices.

```
BUILDHUFFMAN(f[1..n]):
    for i ← 1 to n
        L[i] ← 0; R[i] ← 0
        INSERT(i, f[i])
    for i ← n to 2n - 1
        x ← EXTRACTMIN()
        y ← EXTRACTMIN()
        f[i] ← f[x] + f[y]
        L[i] ← x; R[i] ← y
        P[x] ← i; P[y] ← i
        INSERT(i, f[i])
        P[2n - 1] ← 0
```
Example:

This sentence contains three a’s, three c’s, two d’s, twenty-six e’s, five f’s, three g’s, eight h’s, thirteen i’s, two l’s, sixteen n’s, nine o’s, six r’s, twenty-seven s’s, twenty-two t’s, two u’s, five v’s, eight w’s, four x’s, five y’s, and only one z.

Frequency counts:

| A | B | C | D | E | F | G | H | I | L | N | O | R | S | T | U | V | W | X | Y | Z |
| 3 | 3 | 2 | 26| 5 | 3 | 8 | 13| 2 | 16| 9 | 6 | 27| 22| 2 | 5 | 8 | 4 | 5 | 1 |

Two least frequent? combine D + 7, & they become leaves frequency of D = 3
So take this + merge:

```
A  C  E  F  G  H  I  L  N  O  R  S  T  U  V  W  X  Y  Z
3  3  26  5  3  8  13  7  16  9  6  27  22  2  5  8  4  5  3
```

Next?

etc...
This gives us the tree:

Then let U

Note - D + 7 were first 2 merged
Encoding:

1001 0100 1101 00 00 111 011 1001 111 011 110001 111 110001 10001 011 1001 110000 1101

THIS SENTENCE CONTAINS...

Total: 646 bits (versus 1,458 for ASCII)

| freq. | 3 | 3 | 2 | 26| 5 | 3 | 8 | 13| 2 | 16| 9 | 6 | 27| 22| 2 | 5 | 8 | 4 | 5 | 1 |
| depth | 6 | 6 | 7 | 3 | 5 | 6 | 4 | 4 | 7 | 3 | 4 | 4 | 2 | 4 | 7 | 5 | 4 | 6 | 5 | 7 |
| total | 18| 18| 14| 78| 25| 18| 32| 52| 14| 48| 36| 24| 54| 88| 14| 25| 32| 24| 25| 7 |

Claim: This is optimal!

(No other encoding could do better.)
Proof of Correctness:

Lemma: Let $x \neq y$ be the 2 least frequent characters. Then there is an optimal code where $x \neq y$ are siblings and have maximum depth in the tree.

Proof: Let $T$ be an optimal tree, with depth $d$. $T$ must have at least 2 siblings at depth $d$. If $x \neq y$ are there, done. So assume $a \neq b$ are these siblings. (not $x \neq y$)

Swap $a \neq x$. 

Diagram: 

```
  a
 /|
/  \\
\  \\
\  \\
\  \\
\  \\
  b
```

...
Depth of $x$ increases by some $D \geq 0$, and depth of $a$ decreases by $\Delta$.

$$\text{cost}(T') = \text{cost}(T) - f[a] \cdot \Delta + f[x] \cdot \Delta$$

but $f[a] \geq f[x]$.

So $\Delta (f[x] - f[a])$ is positive, and $T'$ must use fewer bits no more.

Similarly, can swap $b + y$. \[\square\]
Thm: Huffman codes are optimal.