Data Compression

- We want to represent data in a compact manner using as few bits as possible.
- Applications – data on media (CD, DVD), data over internet, ...
- Encoding (compression) – map input data into compressed format
- Decoding (decompression) – map compressed format back to original data.
- Tradeoffs between time to encode, decode, and the size of the compressed data.

Two kinds of coding

- Lossless – when we encode/decode, we get back original data
- Lossy – when we encode/decode, we get back an approximation of the original data

Huffman Codes

Coding is the problem of representing data in another representation. Typically, we want that representation to be concise. We will encode in binary in this lecture. We call the encoding of a character a codeword .

Different types of codes

- fixed length code. Each codeword uses the same number of bits.
- variable length code. Codewords can use differing numbers of bits.

Example

character	frequency	fixed length code	variable length code
a	.45		
b	.13		
c	.12		
d	.16		
e	.9		
f	.5		

Huffman Codes

Different types of codes

- **fixed length code.** Each codeword uses the same number of bits.
- **variable length code.** Codewords can use differing numbers of bits.

character	frequency	fixed length code	variable length code
a	.45	000	0
b	.13	001	101
c	.12	010	100
d	.16	011	111
e	.09	100	1101
f	.05	101	1100

Evaluation of code: Expected number of bits per codeword.

Fixed length code: 3

Variable length code:

$$.45(1) + .13(3) + .12(3) + .16(3) + .09(4) + .05(4) = 2.24$$

Prefix free codes: No codeword is a prefix of any other codeword.

Decoding: A variable length code must be prefix free.

Trees

Codes can be represented as trees.

- Let T be a tree corresponding to a prefix code,
- Let $f(c)$ denote the frequency of character c.
- Let $d_T(c)$ denote the depth of c's leaf in T ($d_T(c)$ is also the length of the codeword for character c.)

The number of bits required to encode a file is thus

$$B(T) = \sum_{c \in C} f(c) d_T(c) \ ,$$

which we define as the cost of the tree T.

Huffman Coding Algorithm

```
Huffman(C)
1  $n \leftarrow |C|$
2  $Q \leftarrow C$
3  for $i \leftarrow 1$ to $n-1$
4          do allocate a new node $z$
5              $left[z] \leftarrow x \leftarrow$ EXTRACT-MIN$(Q)$
6              $right[z] \leftarrow y \leftarrow$ EXTRACT-MIN$(Q)$
7              $f[z] \leftarrow f[x] + f[y]$
8              INSERT$(Q, z)$
9  return EXTRACT-MIN$(Q)$                    $\triangleright$ Return the root of the tree.
```

Proving Huffman is optimal

Theorem Let C be an alphabet in which each character $c \in C$ has frequency $f[c]$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

Proof

Idea: Take a tree T representing an arbitrary optimal prefix code and modify it to make a tree representing another optimal prefix code such that the characters x and y appear as sibling leaves of maximum depth in the new tree. If we can do this, then their codewords will have the same length and differ only in the last bit.

Proving Huffman is optimal

Theorem Let C be an alphabet in which each character $c \in C$ has frequency $f[c]$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

Details:

- Let a and b be two characters that are sibling leaves of maximum depth in T. (wlog, $f[a] \leq f[b]$ and $f[x] \leq f[y]$.)
- $f[x] \leq f[a]$ and $f[y] \leq f[b]$, since $f[x]$ and $f[y]$ are the two lowest leaf frequencies.
- Exchange the positions in T of a and x to produce a tree T'.
- Exchange the positions in T' of b and y to produce a tree T''.

Proof Continued

- **Let a and b be two characters that are sibling leaves of maximum depth in T. (wlog, $f[a] \leq f[b]$ and $f[x] \leq f[y]$.)**
- **$f[x] \leq f[a]$ and $f[y] \leq f[b]$, since $f[x]$ and $f[y]$ are the two lowest leaf frequencies.**
- **Exchange the positions in T of a and x to produce a tree T'.**
- **Exchange the positions in T' of b and y to produce a tree T''.**

Now look at the difference between $B(T)$ and $B(T')$

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_{T'}(x) - f[a]d_{T'}(a) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_T(a) - f[a]d_T(x) \\ &= (f[a] - f[x])(d_T(a) - d_T(x)) \\ &\geq 0 \ , \end{aligned}$$

Reasons for last inequality:

- **$f[a] - f[x]$ is nonnegative because x is a minimum-frequency leaf,**
- **$d_T(a) - d_T(x)$ is nonnegative because a is a leaf of maximum depth in T.**

Proof Finished

Now look at the difference between $B(T)$ and $B(T')$

$$\begin{aligned} B(T) - B(T') &= \sum_{c\in C} f(c)d_T(c) - \sum_{c\in C} f(c)d_{T'}(c) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_{T'}(x) - f[a]d_{T'}(a) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_T(a) - f[a]d_T(x) \\ &= (f[a] - f[x])(d_T(a) - d_T(x)) \\ &\geq 0\ , \end{aligned}$$

Reasons for last inequality:

- $f[a] - f[x]$ **is nonnegative because x is a minimum-frequency leaf,**
- $d_T(a) - d_T(x)$ **is nonnegative because a is a leaf of maximum depth in T.**

Conclusions:

- $B(T') \leq B(T)$
- **By same argument –** $B(T'') \leq B(T')$
- **Conclusions**
- $B(T'') \leq B(T)$**, which completes the proof.which the lemma follows.**