COMP251: Data compression

Jérôme Waldispühl
School of Computer Science
McGill University

Based on slides from M. Langer (McGill) and (goodrich & Tamassia, 2009)
Information Theory

• When A communicates a message to B, A sends a bit string that encodes the message.

• The amount of information in the message depends not on the number of bits sent, but rather on the probability of that message being sent.

• How much information does a message transmit?
Data compression

• When A communicates to B, they first agree on a code.
• They choose a code such that are the most likely to be sent are encoded using fewer bits. This yields shorter messages on average.
• The length of the message should be approximately equal to the amount of information communicated (Shannon, 1948).
Suppose you have a sample space $\Sigma$ (often called an alphabet).

Define a code to be a mapping:

$$C : \Sigma \rightarrow \{ \text{bit string} \}$$

For any $x \in \Sigma$, $C(x)$ is the codeword of $x$.

The length of a codeword is the number of bits in that codeword.

**Example:**

$\Sigma = \{ A, C, G, T \}$

$C(A) = 00$, $C(C) = 01$, $C(G) = 10$, $C(T) = 11$
Extension of a code

For an code $C$ on an alphabet $\Sigma$, we have a naturally defined code on sequences of elements from $\Sigma$.

Example:

$C(AGAT) = C(A) \ C(G) \ C(A) \ C(T)$

$= 00 \ 10 \ 00 \ 11$

Note: We concatenate the codewords of the elements (letters) of the sequence.
Fixed length code

All codewords have the same length.

Example:

• $\Sigma = \{ A, C, G, T \}$
  
  $C(A) = 00$, $C(C) = 01$, $C(G) = 10$, $C(T) = 11$

• ASCII (8 bits), Unicode (16 bits)
Variable length code

Codewords can have different lengths.

Example: Morse code

Note: More common letters have shorter codewords
Tree representation

Any code can be represented by a binary tree. Each codeword is a path from the root to a node representing the element (letter) encoded.

**Principle:** 0 for left child, 1 for right child.

**Example:** Morse code (short=0, long=1)

---

C = 1010
Ambiguity

One big problem with Morse code is that messages are ambiguous.

Example:

\[ C(A) = \cdot \quad \text{C}(E) = \cdot \quad \text{C}(T) = \_ \_ \_ \]

\[ \cdot \quad \_ \_ \_ = \text{“A” or “ET” ??} \]

Q: How to distinguish C(A) from C(ET) ?

A: Morse code inserts “space” between codewords... So it’s not really a binary code.
Prefix code

C is a prefix code if no codeword is a prefix of any other codeword.

Q: What does it mean for binary trees?

A: codewords are leaves!

Example:

- $C(A) = 0$
- $C(B) = 100$
- $C(C) = 101$
- $C(D) = 11$
Desambiguation

Prefix codes avoid the ambiguities that we saw with Morse code. How?

• Suppose B is sent a sequence of bits and B wants to decode this sequence.
• B wants to know the sequence of symbols that was encoded.
• If the code is a prefix code, then there is a simple method for decoding:
  o Repeatedly traverse the binary tree from root to leaf.
  o Each time B reaches a leaf, it reads off the symbol at the leaf, then returns to the root.
Desambiguation (Example)

Q: How to decode 0100110 ?

A:

0100110  A
0100110  AB
0100110  ABD
0100110  ABDA

Result: ABDA
Average code length

Given $S$, $c$, $p$, define:

$$\overline{\lambda} \equiv \sum_{s \in S} p(s) \cdot \lambda(s)$$

- Expected value of codework length
- Length of codeword

**Optimal prefix code:** Given $S$ and $p()$, choose a prefix code that *minimizes* the average code length.
Finding an optimal prefix code

First attempt: Recursively partition $S$ into two subsets such that each split divides the probability in half as closely as possible.

**Example:**

$P(A) = 0.32$
$P(B) = 0.25$
$P(C) = 0.2$
$P(D) = 0.18$
$P(E) = 0.05$
Finding an optimal prefix code

Q: How could you improve the code (i.e. reduce the average code length)?

A: Same the codewords of C & D!

Conclusion: This approach does not guarantee an optimal prefix code.
Claim: For any optimal prefix code, you can generate many other optimal prefix codes by swapping left and right children.

Note: This does not change the average code length.
**Optimal prefix code properties**

**Claim:** For an optimal prefix code, internal node of the tree has two children.

**Example:** This tree cannot be optimal we could obtain a code with a lower average code length by contracting the red edge.
Claim: For any optimal prefix code $C$, and any $a, b \in S$, if $p(a) < p(b)$ then $\lambda(a) \geq \lambda(b)$.

Proof: By contradiction.
• Suppose $\lambda(a) < \lambda(b)$. Then swapping $C(a)$ and $C(b)$ reduces $\lambda$.
• Hence $C$ was not optimal.
Claim: For any optimal prefix code $C$, the two least probable elements have the same codeword length.

Proof: Otherwise we have a contradiction with previous properties.
- Let $a$, $b$ have the smallest probability.
- Suppose $\lambda(a) < \lambda(b)$.
- Then, $b$ has a sibling $d$.
- We can reduce the average length by swapping $a$ with $d$. 
Optimal prefix code properties

**Claim:** There exists an optimal prefix code in which the two least probable elements of D have the same parent in the tree.

**Proof:**
- The 2 least probable elements have the same codeword length.
- So they are at the same level in the tree.
- Swapping codewords of same length does not change the average code length.

This claim suggests an algorithm (Huffman) for finding an optimal prefix code.
Example

\[
P(A) = 0.32 \\
P(B) = 0.25 \\
P(C) = 0.2 \\
P(D) = 0.18 \\
P(E) = 0.05
\]

\[
A \quad B \quad C \quad D \quad E
\]

\[
0.32 \quad 0.25 \quad 0.2 \quad 0.18 \quad 0.05
\]

From previous claim, D and E must have the same codeword length.

\[
P(A) = 0.32 \\
P(B) = 0.25 \\
P(C) = 0.2 \\
P(D) = 0.18 \\
P(E) = 0.05
\]

\[
0.32 \quad 0.25 \quad 0.2 \quad 0.18 \quad 0.05
\]

\[
0.23
\]

\[
P(A) = 0.32 \\
P(B) = 0.25 \\
P(C) = 0.2 \\
P(D) = 0.18 \\
P(E) = 0.05
\]

\[
P(D,E) = 0.23
\]
We have now 4 elements (instead of 5). The next two least probable elements are C and \{D, E\}. Thus, we make them siblings.

\[
\begin{align*}
P(A) &= 0.32 \\
P(B) &= 0.25 \\
P(\{C, D, E\}) &= 0.43
\end{align*}
\]
Example

We have now 4 elements (instead of 5). The next two least probable elements are C and \{D,E\}. Thus, we make them siblings.

\[
\begin{align*}
P(A) &= 0.32 \\
P(B) &= 0.25 \\
P(\{C,D,E\}) &= 0.43
\end{align*}
\]
Example

P(A) = 0.32
P(B) = 0.25
P({C,D,E}) = 0.43

We have now 3 elements. The next two least probable elements are A and B, and we make them siblings.

P({A,B}) = 0.57
P({C,D,E}) = 0.43
Example

Only two elements remain. We make these two elements siblings and we obtain an optimal prefix code!
Huffman’s Algorithm

- Given a string $X$, Huffman’s algorithm constructs a prefix code that minimizes the size of the encoding of $X$.
- It runs in time $O(n + d \log d)$, where $n$ is the size of $X$ and $d$ is the number of distinct characters of $X$.
- A heap-based priority queue is used as an auxiliary structure.
- **Greedy algorithm!**

Algorithm $\text{HuffmanEncoding}(X)$

- **Input** string $X$ of size $n$
- **Output** optimal encoding trie for $X$
- $C \leftarrow \text{distinctCharacters}(X)$
- $\text{computeFrequencies}(C, X)$
- $Q \leftarrow$ new empty heap
- for all $c \in C$
  - $T \leftarrow$ new single-node tree storing $c$
  - $Q.\text{insert}($getFrequency$(c), T)$
- while $Q.\text{size}() > 1$
  - $f_1 \leftarrow Q.\text{minKey}()$
  - $T_1 \leftarrow Q.\text{removeMin}()$
  - $f_2 \leftarrow Q.\text{minKey}()$
  - $T_2 \leftarrow Q.\text{removeMin}()$
  - $T \leftarrow \text{join}(T_1, T_2)$
  - $Q.\text{insert}(f_1 + f_2, T)$
- return $Q.\text{removeMin}()$
$X = \text{abracadabra}$

Frequencies

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

$X = \text{abracadabra}$

Frequencies

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Example

\[ X = \text{abracadabra} \]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Frequencies

\[ X = 01101110100010101101110 \]

Binary tree representation:

- Node 11:
  - Left: a
  - Right: 6

- Node 6:
  - Left: 2
  - Right: 4

- Node 2:
  - Left: c
  - Right: d

- Node 4:
  - Left: b
  - Right: r

- Values:
  - a: 0
  - b: 110
  - c: 100
  - d: 101
  - r: 111
Extended Huffman Tree Example

String: a fast runner need never be afraid of the dark

<table>
<thead>
<tr>
<th>Character</th>
<th>a</th>
<th>b</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>h</th>
<th>i</th>
<th>k</th>
<th>n</th>
<th>o</th>
<th>r</th>
<th>s</th>
<th>t</th>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>