COMP250: 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 of the number of bits sent, but rather on the probability of that message being sent.

• How much information does a message transmit?

A

Blah blah blah blah
Blah blah blah blah
Blah blah blah blah
Blah blah blah blah
Blah blah blah blah

B

I know that! Tell me something I do not know already!
Data compression

- When A communicates to B, they first agree on a code.
- They choose a code such that 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:

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

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

Codewords can have different lengths.

Example: Morse code

```
A:B:C:::D:::E::E::E:F:GG:::H:::I:::J:KK::L::LL::M::NN::OO::PP::QQ::RR::ST::
```

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)

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
C = 1010
```
Ambiguity

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

Example:

\[ C(A) = \text{• —} \quad C(E) = \text{•} \quad C(T) = \text{—} \]

• — = “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 in binary tree representations?

A: codewords are leaves!

Example:

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

Q: How to decode 0100110?

A:

0100110  A
0100110  AB
0100110  ABD
0100110  ABDA

Result: ABDA
Huffman’s Algorithm

- Given a string $X$, Huffman’s algorithm construct 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 $c$
  - $Q.\text{insert}(\text{getFrequency}(c), T)$
- 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 $Q.\text{removeMin}()$
Example

Preprocessing: Compute the frequency of characters in string X.

\[
\begin{align*}
P(A) &= 0.32 \\
P(B) &= 0.25 \\
P(C) &= 0.2 \\
P(D) &= 0.18 \\
P(E) &= 0.05
\end{align*}
\]

Merge two element with the lowest probabilities:

\[
\begin{align*}
P(A) &= 0.32 \\
P(B) &= 0.25 \\
P(C) &= 0.2 \\
P(D,E) &= 0.23 \\
P({D,E}) &= 0.23
\end{align*}
\]
We have now 4 elements. The probability of this new element \(\{D,E\}\) is the sum of the probability of the elements merged. We merge the next two least probable elements.

\[
\begin{align*}
P(A) &= 0.32 \\
P(B) &= 0.25 \\
P(C) &= 0.2 \\
P(D) &= 0.18 \\
P(E) &= 0.05 \\
P(\{D,E\}) &= 0.23
\end{align*}
\]
We have now 3 elements. The next two least probable elements are A and B, and we merge them again.
Example

Only two elements remain. We make these two elements siblings and we obtain an **optimal prefix code**!
Example (Encoding)

Original message
T = 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>

Huffman’s algorithm

Optimal prefix code

Encrypted message
T’ = 01101110100010101101110

Codewords

a : 0
b : 110
c : 100
d : 101
r : 111
Example (Decoding)

Encoded message
M = 01101110100010101101110

Optimal prefix code

Decoded message
T = abracadabra
Example (Huffman)

T = abracadabra

Frequencies:

<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>2</td>
<td></td>
</tr>
</tbody>
</table>

![Huffman Tree Diagram]