Lossless Compression
Multimedia Systems (Module 2 Lesson 2)

Summary:
- Adaptive Coding
  - Adaptive Huffman Coding
    • Sibling Property
    • Update Algorithm
  - Arithmetic Coding
    • Coding and Decoding
    • Issues: EOF problem, Zero frequency problem

Sources:
- Introduction to Data Compression, by Sayood Khalid
- The Squeeze Page at SFU

Adaptive Coding

Motivations:
- The previous algorithms (both Shannon-Fano and Huffman) require the statistical knowledge which is often not available (e.g., live audio, video).
- Even when it is available, it could be a heavy overhead.
- Higher-order models incur more overhead. For example, a 255 entry probability table would be required for a 0-order model. An order-1 model would require 255 such probability tables. (A order-1 model will consider probabilities of occurrences of 2 symbols)

The solution is to use adaptive algorithms. Adaptive Huffman Coding is one such mechanism that we will study.

The idea of "adaptiveness" is however applicable to other adaptive compression algorithms.

Adaptive Coding

ENCODER
Initialize_model();
do {
    c = getc( input );
    encode( c, output );
    update_model( c );
} while ( c != eof)

DECODER
Initialize_model();
while ( c = decode(input)) != eof)
{    
    putc( c, output)
    update_model( c );
}

- The key is that, both encoder and decoder use exactly the same initialize_model and update_model routines.
The Sibling Property

The node numbers will be assigned in such a way that:

1. A node with a higher weight will have a higher node number.
2. A parent node will always have a higher node number than its children.

In a nutshell, the sibling property requires that the nodes (internal and leaf) are arranged in order of increasing weights.

The update procedure swaps nodes that are in violation of the sibling property.

- The identification of nodes in violation of the sibling property is achieved by using the notion of a block.
- All nodes that have the same weight are said to belong to one block.

Flowchart of the update procedure

- The Huffman tree is initialized with a single node, known as the Not-Yet-Transmitted (NYT) or escape code. This code will be sent every time that a new character, which is not in the tree, is encountered, followed by the ASCII encoding of the character. This allows for the de-compressor to distinguish between a code and a new character. Also, the procedure creates a new node for the character and a new NYT from the old NYT node.
- The root node will have the highest node number because it has the highest weight.

Example

Example Huffman tree after some symbols have been processed in accordance with the sibling property.
Example

Counts:
A:1
B:2
C:2
D:2
E:10

A Huffman tree after first appearance of symbol A

Increment

Counts:
A:1+1
B:2
C:2
D:2
E:10

An increment in the count for A propagated up to the root

Swapping

Counts:
A:2+1
B:2
C:2
D:2
E:10

Another increment in the count for A results in swap

Swap nodes 1 and 5

Counts:
A:3
B:2
C:2
D:2
E:10

Swapping
Another increment in the count for A propagates up

Swap nodes 5 and 6

Further swapping needed to fix the tree

Swap nodes 8 and 9
Arithmetic Coding

Arithmetic coding is based on the concept of interval subdividing.

- In arithmetic coding a source ensemble is represented by an interval between 0 and 1 on the real number line.
- Each symbol of the ensemble narrows this interval.
- As the interval becomes smaller, the number of bits needed to specify it grows.
- Arithmetic coding assumes an explicit probabilistic model of the source.
- It uses the probabilities of the source messages to successively narrow the interval used to represent the ensemble.
  - A high probability message narrows the interval less than a low probability message, so that high probability messages contribute fewer bits to the coded ensemble.

Arithmetic Coding: Description

- In the following discussions, we will use M as the size of the alphabet of the data source,
  - N(x) as symbol x’s probability,
  - Q[x] as symbol x’s cumulative probability (i.e., Q[i]=N[0]+N[1]+…+N[i])
- Assume we know the probabilities of each symbol of the data source,
  - we can allocate to each symbol an interval with width proportional to its probability, and each of the intervals does not overlap with others.
  - This can be done if we use the cumulative probabilities as the two ends of each interval. Therefore, the two ends of each symbol x amount to Q[x-1] and Q[x].
  - Symbol x is said to own the range [Q[x-1], Q[x]].
**Arithmetic Coding: Encoder**

We begin with the interval \([0,1)\) and subdivide the interval iteratively.

- For each symbol entered, the current interval is divided according to the probabilities of the alphabet.
- The interval corresponding to the symbol is picked as the interval to be further proceeded with.
- The procedure continues until all symbols in the message have been processed.
- Since each symbol’s interval does not overlap with others, for each possible message there is a unique interval assigned.
- We can represent the message with the interval’s two ends \([L,H)\). In fact, taking any single value in the interval as the encoded code is enough, and usually the left end \(L\) is selected.

**Arithmetic Coding Algorithm**

\[
L = 0.0; H = 1.0;
\]

\[
\text{While ( (x = getc(input)) \neq EOF ) }
\]

\[
\{ \\
\quad R = (H-L);
\quad H = L + R \times Q[x];
\quad L = L + R \times Q[x-1];
\}
\]

\[
\text{Output(L);} \\
\]

\(R\) is the interval range, and \(H\) and \(L\) are two ends of the current code interval. \(x\) is the new symbol to be encoded. \(H\) and \(L\) are initialized to 0 and 1 respectively.

**Arithmetic Coding: Encoder example**

<table>
<thead>
<tr>
<th>Symbol, (x)</th>
<th>Probability, (N[x])</th>
<th>(Q[x-1])</th>
<th>(Q[x])</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.4</td>
<td>0.0</td>
<td>0.4</td>
</tr>
<tr>
<td>B</td>
<td>0.3</td>
<td>0.4</td>
<td>0.7</td>
</tr>
<tr>
<td>C</td>
<td>0.2</td>
<td>0.7</td>
<td>0.9</td>
</tr>
<tr>
<td>D</td>
<td>0.1</td>
<td>0.9</td>
<td>1.0</td>
</tr>
</tbody>
</table>

String: BCAB
Code sent: 0.6196
Decoding Algorithm

- When decoding the code $v$ is placed on the current code interval to find the symbol $x$ so that $Q[x-1] \leq code < Q[x]$. The procedure iterates until all symbols are decoded.

```c
v = input_code();
for (;;) {
  x = find_symbol_straddling_this_range(v);
 putc(x);
  R = Q[x] - Q[x-1];
  v = (v - Q[x-1]) / R;
}
```

<table>
<thead>
<tr>
<th>$v$</th>
<th>Output Char</th>
<th>$Q[x-1]$</th>
<th>$Q[x]$</th>
<th>$R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.65</td>
<td>B</td>
<td>0.4</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>0.70</td>
<td>C</td>
<td>0.4</td>
<td>0.9</td>
<td>0.2</td>
</tr>
<tr>
<td>0.60</td>
<td>A</td>
<td>0.0</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>0.40</td>
<td>B</td>
<td>0.4</td>
<td>0.7</td>
<td>0.3</td>
</tr>
<tr>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Arithmetic Coding: Issues

- **The zero-frequency problem**: Each symbol’s predicted probability must not be zero or the interval will become zero and interval renormalization would fail. This is called the zero-frequency problem. Models that adapt online may encounter such problem when decaying.

- **The EOF problem**:
  - Assume we pick the lower end of the interval as the encoded code. Two messages may yield the same code if one message is identical to the other except for a sequence of finite number of the first symbol (first in table, not in the sequence) as a suffix.
    - For e.g., both BCAB, BCABA, BCABAA, BCABAAA will have the same lower interval but different upper intervals. (Try it)
  - The simplest solution is to let the decoder know the length of the encoded message. The decoder will know if the message size is fixed or can be transmitted at first. However, this is not plausible if the data size is not known beforehand, such as live broadcasting data or it’s too costly to do so, such as tapes whose size is unknown at the beginning.
  - There is another solution if we introduce a special EOF symbol to the alphabet. The symbol takes a small interval and is used only at the end of the message. When the decoder detects the EOF symbol it knows the end of the message is reached.