CS 240 — Lecture 20

Bartosz Antczak *Instructor: Eric Schost* *March 21, 2017*

20.1 Compression

This section focuses on how we store and transmit data. We'll focus mainly on encoding and decoding text by using algorithms. Other than measuring the efficiency of these algorithms, we can also measure:

- Processing speed
- Reliability
- Security (i.e., how safe is the encryption, or encoded text)
- Size

We define the compression ratio as:

$$\frac{|C| \cdot \log |\Sigma_C|}{|S| \cdot \log |\Sigma_S|}$$

(S represents the source text (i.e., text from the source alphabet Σ_S that is about to be encoded; C represents the encoded text from the coded alphabet Σ_C)

20.1.1 Types of Data Compression

Logical vs. Physical

- **Logical Compression:** uses the meaning of the data and only applies to a certain domain (e.g., sound recordings)
- **Physical Compression:** only knows the physical bits in the data, not the meaning behind them

Lossy vs. Lossless

- **Lossy Compression:** achieves better compression ratios, but the decoding is approximate; the exact source text S is not recoverable
- **Lossless Compression:** always decodes S exactly

We will concentrate on physical, lossless compression algorithms.
We will use ASCII to encode characters.

20.1.2 Decoding

To decode encoded text, we will use a decoding algorithm mapping $\Sigma_C* \rightarrow \Sigma_S*$ (the Kleene start represents every possible character in the respective alphabet).
Our code must be uniquely decodable, and it's helpful if a code is *prefix-free*, which means that for any

character in our encoded alphabet, there exists one and only one character in the decoded alphabet to map to ($\Sigma_{C}* \rightarrow \Sigma_S$). To understand why we need this, consider the following non-uniquely decodable mapping:

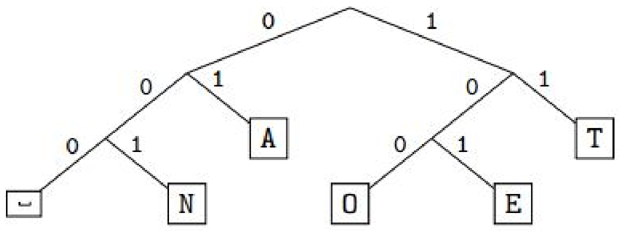
$$A \rightarrow 1$$
$$B \rightarrow 0$$
$$C \rightarrow 01$$

What if we wanted to encode the text 01? Do we map it to C or BA? 01 is on trial right now, and (voice of Roger Waters) *this will not do.*

20.1.3 Huffman Coding

Involves building a binary trie to store the decoding dictionary D. Each character of Σ is a leaf of the trie.
Encoding a text simply involves finding the encoded text representation of each character in Σ_S.
Decoding a text involves reading the encoded text and tracking the characters in the trie. Once we hit a character, print that character, and repeat.

Example 20.1.1. *Huffman Coding trie with decoded alphabet* $\Sigma_S = \{$`A`, `N`, `O`, `E`, `T`, `_`$\}$



Encoding `AN_ANT` returns 01 001 000 01 001 11.
Decoding 11 100 000 101 01 11 returns `TO_EAT`.

20.1.4 Building the Best Trie

For a source text S, how do we determine the best trie which minimizes the length of C?

- Determine the frequency of each character $c \in \Sigma_S$
- Make $|\Sigma_S|$ trie of height 0 holding each character. Assign a weight to each trie, which is the sum of frequencies of all letters in the trie (initially, these are just the character frequencies)
- Merge two tries with the least weights. Their new weight s their sum
- Repeat the previous step until there is only one trie left