Huffman Codes KT 4.8 DPV 5.2

Frank

September 22, 2021

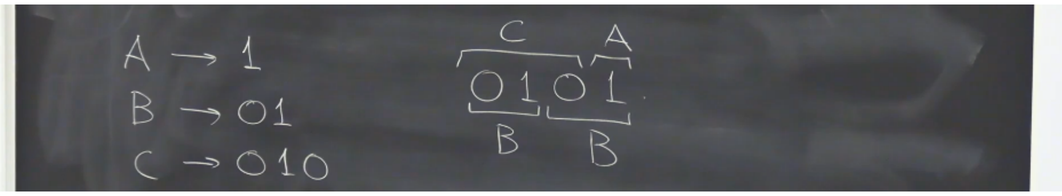
1 Codewords

- Alphabet: Γ finite set of symbols
- Code: maps symbol $a \in \Gamma$ to binary string (*codeword* for a)
- ASCII: fixed-length code - every codeword has exactly the same length

  $\implies$ easy to decode

  However, it makes more sense to use shorter codewords for more frequently used symbols and use the longer codewords for less frequently used symbols
- Variable-length code: codewords may have different lengths

  The potential problem of variable-length code is that given a text of encoding such as below, you cannot decode the text in a unique manner. For example, the below encoding can represent two different words. (CA or BB)



  You cannot recover the above encoding reliably

  The problem is that the encoding for B is a prefix for C (01)

  – In order to avoid this problem, we require our variable-length code be Prefix Code

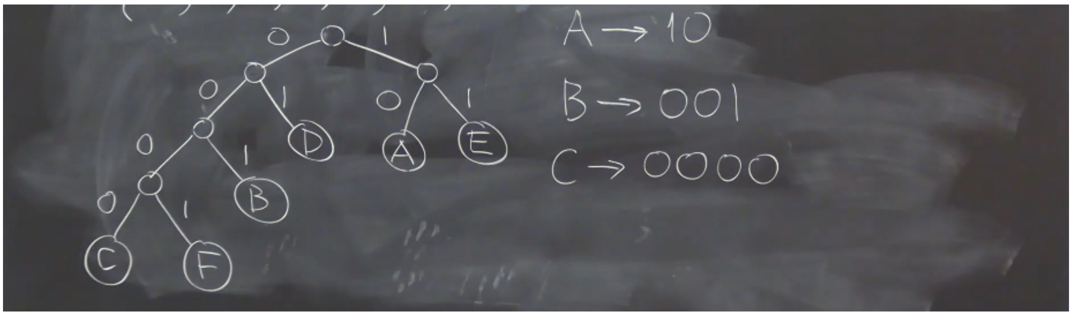
    No codeword is a prefix of another

    $\implies$ unique decoding

2 Prefix Code

Prefix code can be represented as a binary tree

$\Gamma = \{A, B, C, D, E, F\}$, and consider tree



- Symbols are the leaves
- Codeword is the concatenation of labels 0/1 of edges on root-to-leaf path

  The code is prefix because we require the symbols to be leaves, and leaves by default do not have children, so thus it cannot be a prefix to anything. (no path from leaves)

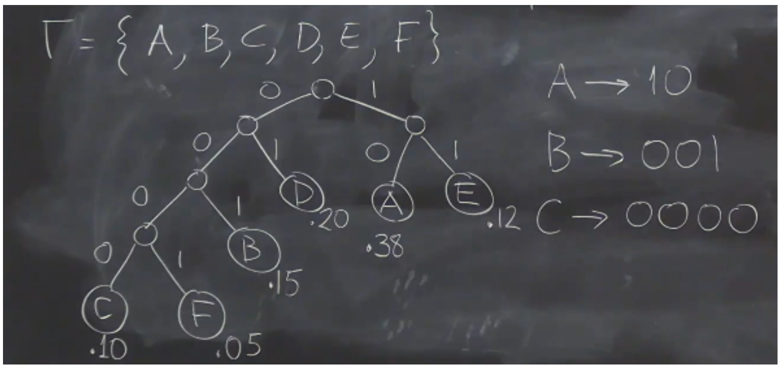
3 Problem

3.1 Input

Alphabet Γ, and $\forall a \in \Gamma$, $f(a)$ = frequency of a

$0 \leq f(a) \leq 1$, $\sum_{a \in \Gamma} f(a) = 1$ i.e. $f(a)$ is a PDF

Adding $f(a)$ the tree will look like the following:



3.2 Output

- Find efficient encoding for the alphabet given frequency.

  Specifically, find binary tree T representing optimal prefix code for Γ under freq. f

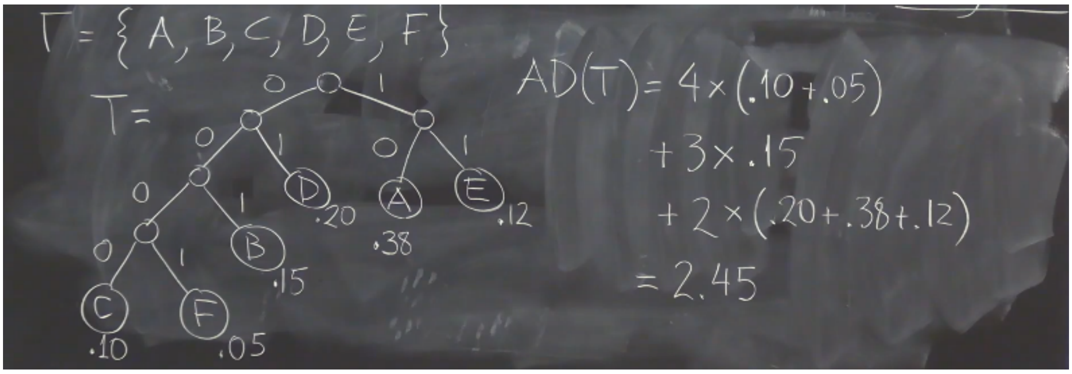
- Define average depth $AD(T) = \sum_{a \in \Gamma} f(a) \cdot \text{depth}_T(a)$, where depth means the number of edges on path from root to leaf of a, or the length of the codeword for a
- We want to find T with minimum $AD(T)$

  Then we want to match high $f(a)$ with low $\text{depth}_T(a)$ to minimize the average depth

  Note we have a probability distribution function $f(a)$ and we are trying to minimize the mean of this probability function.

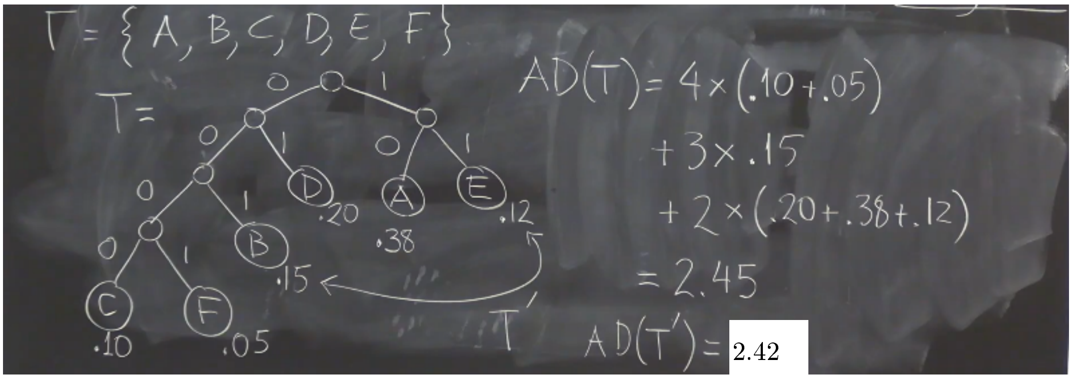
3.3 Average Depth

Given the tree, here are the calculations:



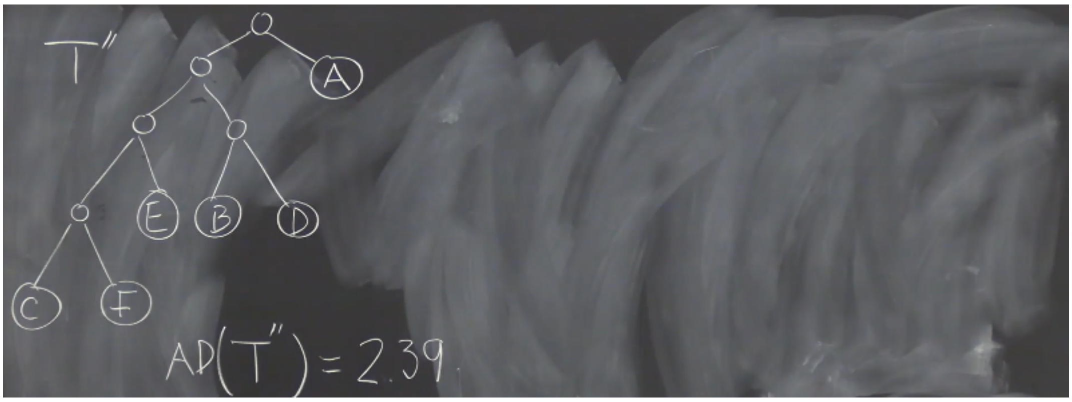
Thus, we need on average 2.45 bits (depth of 2.45) out of how many ever bits we used to encode the alphabet (4 in this case)

- Note that $f(B) > f(E)$ but $\text{depth}(B) < \text{depth}(E)$, so we need to correct this.



This is however, still not the optimal

- This following tree is an optimal tree:



3.4 Easy Observations

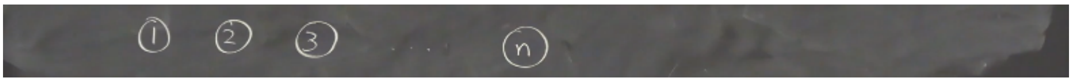
1. An optimal tree must be full, since if we have a node with only one child, then that node is essentially useless (doesn't help to encode a symbol since it is not a leaf node) and only serves as a transition. So we can replace that node with its child. Repeat this process until we have a full tree.

2. A symbol with greater frequency cannot be at a greater depth than a symbol with lesser frequency. Or formally $f(x) > f(y) \implies \text{depth}_T(x) < \text{depth}_T(y)$. Since if not the case, switch the two nodes to get a better tree.

3. 1+2 $\implies$ if x, y are symbols with minimum frequency, then $\exists$ an optimal tree where x and y are siblings at maximum depth. Or for multiple symbols with the same minimum frequency, we can force either two of them to be siblings at maximum depth.

4 Algorithm

Suppose we have n symbols, which will eventually be leaves of some tree

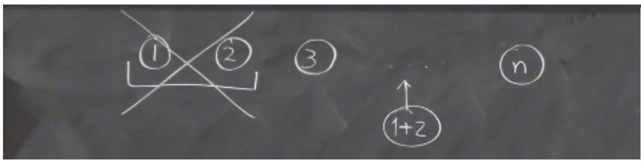


Now order these nodes in non-decreasing frequency and rename them so they retain the same ordering as above

$$f(1) \leq f(2) \leq \ldots \leq f(n)$$

- We know that 1 and 2 can be siblings at maximum depth of some optimal tree by observation 3. Combine 1 and 2 to form a new node at their parent node called $1+2$, then $f(1+2) = f(1) + f(2)$

- Now we have a new node that we have to place somewhere in our list of sorted nodes according to its frequency (to maintain the property of non-decreasing frequency)



Now we solve the same problem, except we have 1 less (throw away 2 insert 1) node in our list of sorted nodes. So recursively do the same thing to obtain a tree in the end, and this process is the **Huffman's Algorithm**

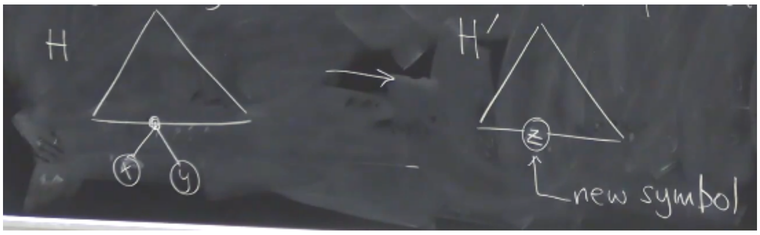
We start with a bunch of nodes (symbols), then we gradually combine nodes to form a bigger and bigger tree (or many trees, but combine these trees later) until we reach a single tree containing all nodes

- Note that this algorithm does not give a unique tree:
  1. Having many symbols with the same minimum frequency allows us to choose any two, resulting in a different tree with the same average depth
  2. You could swap the order of the siblings to get a different tree, but maintain the same average depth of the tree

Nevertheless, it gives an optimal tree (code)

5 Optimality/Correctness

- Let Γ = alphabet, frequencies f, and H be the tree produced by Huffman's algorithm on (Γ, f), $|\Gamma| = n+1$
- x, y = symbols with minimum frequencies that are siblings in H (first two symbols that the algorithm combines), and combine it into a new symbol z, (which is now a tree with 2 children x, y)



Then we end up with new alphabet

$$\Gamma' = (\Gamma - \{x, y\}) \cup \{z\}$$

and new frequency

$$f'(a) = \begin{cases} f(a) & a \neq z \\ f(x) + f(y) & a = z \end{cases}$$

- H' is output of Huffman on (Γ', f'), note $|\Gamma'| = n < |\Gamma|$

  $\overset{IH}{\Longrightarrow}$ H' is optimal for (Γ', f')

  Base case: tree with 2 symbols (trivial)

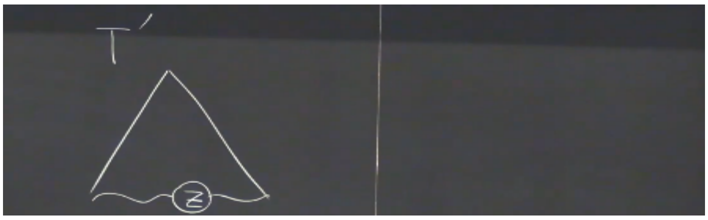
  IH: Suppose Huffman's returns an optimal tree for an alphabet with n symbols

  Now consider an alphabet with $n+1$ symbols, let H be the tree that Huffman's produces for $n+1$ symbols

  We combine x, y into z to produce an alphabet with n symbols, and then we can apply our IH to get that H' is optimal, which in turn shows that H is optimal. See following arguments for clearification.

- Consider optimal tree T for (Γ, f), then by observation 3, we can assume that x, y are siblings.

Now apply the same transformation to T and get T', a tree for (Γ', f'), which may not be optimal



- We claim that $AD(H) = AD(H') + [f(x) + f(y)]$

  Think of the difference between $AD(H)$ and $AD(H')$
  everything cancels out except for $AD(H)$, we have

  $$[f(x) + f(y)] \cdot depth_{x,y}$$

  and for $AD(H')$ we have

  $$f(z) \cdot depth_z = [f(x) + f(y)] \cdot [depth_{x,y} - 1]$$

  Rearrange to get that $AD(H) = AD(H') + [f(x) + f(y)]$

- Then by the same reasoning,

  $$AD(T) = AD(T') + [f(x) + f(y)]$$

  But then $AD(H') \leq AD(T')$ by optimality of H' for (Γ', f'), so

  $$AD(H) \leq AD(T)$$

  which proves that H is an optimal tree.

6 Time Complexity

See website for full proof, basically, consider our array of nodes and use a heap to organize them into a tree.