Chapter 25

Compression, Information and Entropy – Huffman's coding

NEW CS 473: Theory II, Fall 2015
December 1, 2015

25.1 Huffman coding

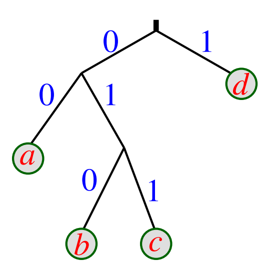
25.2 start

25.2.0.1 Codes...

(A) Σ: alphabet.
(B) ***binary code***: assigns a string of 0s and 1s to each character in the alphabet.
(C) each symbol in input = a codeword over some other alphabet.
(D) Useful for transmitting messages over a wire: only 0/1.
(E) receiver gets a binary stream of bits...
(F) ... decode the message sent.
(G) ***prefix code***: reading a prefix of the input binary string uniquely match it to a code word.
(H) ... continuing to decipher the rest of the stream.
(I) binary/prefix code is ***prefix-free*** if no code is a prefix of any other.
(J) ASCII and Unicode's UTF-8 are both prefix-free binary codes.

25.2.0.2 Codes...

(A) Morse code is binary+prefix code but **not** prefix-free.
(B) ... code for `S` $(\cdots)$ includes the code for `E` $(\cdot)$ as a prefix.
(C) Prefix codes are binary trees...



(D) ...characters in leafs, code word is path from root.

(E) prefix treestree!prefix tree or ***code trees***.
(F) Decoding/encoding is easy.

25.2.0.3 Codes...

(A) Encoding: given frequency table:
$f[1 \ldots n]$.
(B) $f[i]$: frequency of ith character.
(C) code(i): binary string for ith character.
len(s): length (in bits) of binary string s.
(D) Compute tree $\mathcal{T}$ that minimizes

$$\operatorname{cost}(\mathcal{T}) = \sum_{i=1}^{n} f[i] * \operatorname{len}(\operatorname{code}(i)), \tag{25.1}$$

25.2.1 Frequency table for...

25.2.1.1 "A tale of two cities" by Dickens

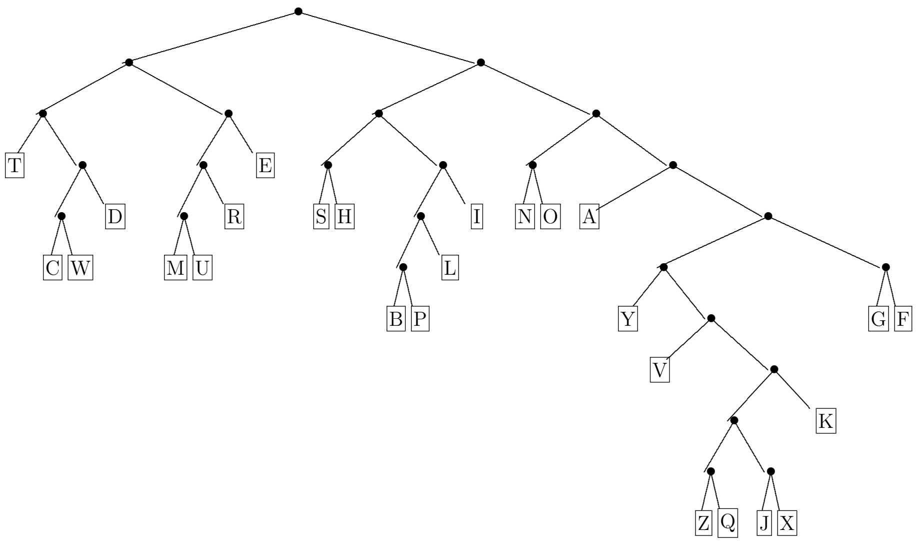
\ n	16,492	'1'	61	'C'	13,896	'Q'	667
' '	130,376	'2'	10	'D'	28,041	'R'	37,187
'!'	955	'3'	12	'E'	74,809	'S'	37,575
'"'	5,681	'4'	10	'F'	13,559	'T'	54,024
'$'	2	'5'	14	'G'	12,530	'U'	16,726
'%'	1	'6'	11	'H'	38,961	'V'	5,199
'''	1,174	'7'	13	'I'	41,005	'W'	14,113
'('	151	'8'	13	'J'	710	'X'	724
')'	151	'9'	14	'K'	4,782	'Y'	12,177
'*'	70	':'	267	'L'	22,030	'Z'	215
','	13,276	';'	1,108	'M'	15,298	'_'	182
'–'	2,430	'?'	913	'N'	42,380	'‘'	93
'.'	6,769	'A'	48,165	'O'	46,499	'@'	2
'0'	20	'B'	8,414	'P'	9,957	'/'	26

25.2.1.2 Computed prefix codes...

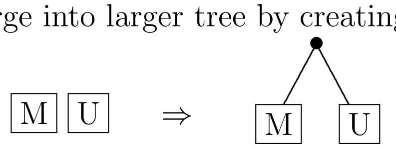
char	frequency	code	char	freq	code
'A'	48165	1110	'N'	42380	1100
'B'	8414	101000	'O'	46499	1101
'C'	13896	00100	'P'	9957	101001
'D'	28041	0011	'Q'	667	1111011001
'E'	74809	011	'R'	37187	0101
'F'	13559	111111	'S'	37575	1000
'G'	12530	111110	'T'	54024	000
'H'	38961	1001	'U'	16726	01001
'I'	41005	1011	'V'	5199	1111010
'J'	710	1111011010	'W'	14113	00101
'K'	4782	11110111	'X'	724	1111011011
'L'	22030	10101	'Y'	12177	111100
'M'	15298	01000	'Z'	215	1111011000

25.2.2 The Huffman tree generating the code

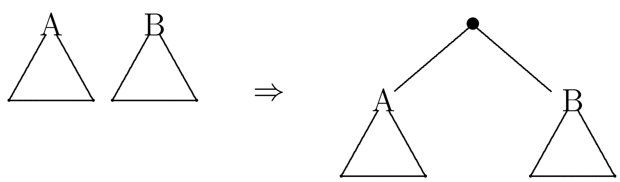
25.2.2.1 Build only on A-Z for clarity.



25.2.2.2 Mergeablity of code trees

(A) two trees for some disjoint parts of the alphabet...
(B) Merge into larger tree by creating a new node and hanging the trees from this common node.
(C) 

(D) ...put together two subtrees.



25.2.3 The algorithm to build Hoffman's code

25.2.3.1 Building optimal prefix code trees

(A) take two least frequent characters in frequency table...
(B) ... merge them into a tree, and put the root of merged tree back into table.
(C) ...instead of the two old trees.
(D) Algorithm stops when there is a single tree.
(E) Intuition: infrequent characters participate in a large number of merges. Long code words.
(F) Algorithm is due to David Huffman (1952).
(G) Resulting code is best one can do.
(H) ***Huffman coding***: building block used by numerous other compression algorithms.

25.2.4 Analysis

25.2.4.1 Lemma: lowest leafs are siblings...

Lemma 25.2.1. *(A) $\mathcal{T}$: optimal code tree (prefix free!).*
(B) Then $\mathcal{T}$ is a full binary tree.

(C) ... every node of $\mathcal{T}$ has either 0 or 2 children.
(D) If height of $\mathcal{T}$ is d, then there are leafs nodes of height d that are sibling.

25.2.4.2 Proof...

(A) If $\exists$ internal node $v \in \mathsf{V}(\mathcal{T})$ with single child...
...remove it.
(B) New code tree is better compressor: $\mathrm{cost}(\mathcal{T}) = \sum_{i=1}^{n} f[i] * \mathrm{len}(\mathrm{code}(i))$.
(C) u: leaf u with maximum depth d in $\mathcal{T}$. Consider parent $v = \overline{\mathrm{p}}(u)$.
(D) $\implies$ v: has two children, both leafs

■

25.2.4.3 Infrequent characters are stuck together...

Lemma 25.2.2. *x, y: two least frequent characters (breaking ties arbitrarily).*
$\exists$ optimal code tree in which x and y are siblings.

25.2.4.4 Proof...

(A) Claim: $\exists$ optimal code s.t. x and y are siblings + deepest.
(B) $\mathcal{T}$: optimal code tree with depth d.
(C) By lemma... $\mathcal{T}$ has two leafs at depth d that are siblings,
(D) If not x and y, but some other characters α and β.
(E) $\mathcal{T}'$: swap x and α.
(F) x depth inc by Δ, and depth of α decreases by Δ.
(G) $\mathrm{cost}(\mathcal{T}') = \mathrm{cost}(\mathcal{T}) - \big(f[\alpha] - f[x]\big)\Delta$.
(H) x: one of the two least frequent characters.
...but α is not.
(I) $\implies$ $f[\alpha] \geq f[x]$.
(J) Swapping x and α does not increase cost.
(K) $\mathcal{T}$: optimal code tree, swapping x and α does not decrease cost.
(L) $\mathcal{T}'$ is also an optimal code tree
(M) Must be that $f[\alpha] = f[x]$.

25.2.4.5 Proof continued...

(A) y: second least frequent character.
(B) β: lowest leaf in tree. Sibling to x.
(C) Swapping y and β must give yet another optimal code tree.
(D) Final opt code tree, x, y are max-depth siblings. ■

25.2.4.6 Huffman's codes are optimal

Theorem 25.2.3. *Huffman codes are optimal prefix-free binary codes.*

25.2.4.7 Proof...

(A) If message has 1 or 2 diff characters, then theorem easy.
(B) $f[1 \dots n]$ be original input frequencies.
(C) Assume $f[1]$ and $f[2]$ are the two smallest.

(D) Let $f[n+1] = f[1] + f[2]$.
(E) lemma $\implies \exists$ opt. code tree $\mathcal{T}_{\mathrm{opt}}$ for $f[1..n]$
(F) $\mathcal{T}_{\mathrm{opt}}$ has 1 and 2 as siblings.
(G) Remove 1 and 2 from $\mathcal{T}_{\mathrm{opt}}$.
(H) $\mathcal{T}'_{\mathrm{opt}}$: Remaining tree has $3, \ldots, n$ as leafs and "special" character $n+1$ (i.e., parent $1, 2$ in $\mathcal{T}_{\mathrm{opt}}$)

25.2.4.8 La proof continued...

(A) character $n+1$: has frequency $f[n+1]$.
Now, $f[n+1] = f[1] + f[2]$, we have

$$
\begin{aligned}
\mathrm{cost}(\mathcal{T}_{\mathrm{opt}}) &= \sum_{i=1}^{n} f[i] \mathrm{depth}_{\mathcal{T}_{\mathrm{opt}}}(i) \\
&= \sum_{i=3}^{n+1} f[i] \mathrm{depth}_{\mathcal{T}_{\mathrm{opt}}}(i) + f[1] \mathrm{depth}_{\mathcal{T}_{\mathrm{opt}}}(1) \\
&\quad + f[2] \mathrm{depth}_{\mathcal{T}_{\mathrm{opt}}}(2) - f[n+1] \mathrm{depth}_{\mathcal{T}_{\mathrm{opt}}}(n+1) \\
&= \mathrm{cost}\big(\mathcal{T}'_{\mathrm{opt}}\big) + \big(f[1] + f[2]\big) \mathrm{depth}(\mathcal{T}_{\mathrm{opt}}) \\
&\quad - \big(f[1] + f[2]\big)(\mathrm{depth}(\mathcal{T}_{\mathrm{opt}}) - 1) \\
&= \mathrm{cost}\big(\mathcal{T}'_{\mathrm{opt}}\big) + f[1] + f[2].
\end{aligned}
$$

25.2.4.9 La proof continued...

(A) implies min cost of $\mathcal{T}_{\mathrm{opt}} \equiv$ min cost $\mathcal{T}'_{\mathrm{opt}}$.
(B) $\mathcal{T}'_{\mathrm{opt}}$: must be optimal coding tree for $f[3 \ldots n+1]$.
(C) $\mathcal{T}'_H$: Huffman tree for $f[3, \ldots, n+1]$
$\mathcal{T}_H$: overall Huffman tree constructed for $f[1, \ldots, n]$.
(D) By construction:
$\mathcal{T}'_H$ formed by removing leafs 1 and 2 from $\mathcal{T}_H$.
(E) By induction:
Huffman tree generated for $f[3, \ldots, n+1]$ is optimal.
(F) $\mathrm{cost}\big(\mathcal{T}'_{\mathrm{opt}}\big) = \mathrm{cost}(\mathcal{T}'_H)$.
(G) $\implies \mathrm{cost}(\mathcal{T}_H) = \mathrm{cost}(\mathcal{T}'_H) + f[1] + f[2] = \mathrm{cost}\big(\mathcal{T}'_{\mathrm{opt}}\big) + f[1] + f[2] = \mathrm{cost}(\mathcal{T}_{\mathrm{opt}})$,
(H) $\implies$ Huffman tree has the same cost as the optimal tree. ■

25.2.5 What do we get

25.2.5.1 What we get...

(A) A tale of two cities: 779,940 bytes.
(B) using above Huffman compression results in a compression to a file of size 439,688 bytes.
(C) Ignoring space to store tree.
(D) `gzip`: 301,295 bytes
`bzip2`: 220,156 bytes!
(E) Huffman encoder can be easily written in a few hours of work!
(F) All later compressors use it as a black box...

25.2.6 A formula for the average size of a code word

25.2.6.1 Average size of code word

(A) input is made out of n characters.
(B) p_i: fraction of input that is ith char (probability).
(C) use probabilities to build Huffman tree.
(D) Q: What is the length of the codewords assigned to characters as function of probabilities?
(E) special case...

25.2.7 Average length of codewords...

25.2.7.1 Special case

Lemma 25.2.4. $1, \ldots, n$*: symbols.*

Assume, for $i = 1, \ldots, n$*:*
(A) $p_i = 1/2^{l_i}$*: probability for the* i*th symbol*
(B) $l_i \geq 0$*: integer.*

Then, in Huffman coding for this input, the code for i *is of length* l_i*.*

25.2.7.2 Proof

(A) induction of the Huffman algorithm.
(B) $n = 2$: claim holds since there are only two characters with probability $1/2$.
(C) Let i and j be the two characters with lowest probability.
(D) Must be $p_i = p_j$ (otherwise, $\sum_k p_k \neq 1$).
(E) Huffman's tree merges this two letters, into a single "character" that have probability $2p_i$.
(F) New "character" has encoding of length $l_i - 1$, by induction
(on remaining $n - 1$ symbols).
(G) resulting tree encodes i and j by code words of length $(l_i - 1) + 1 = l_i$. ■

25.2.7.3 Translating lemma...

(A) $p_i = 1/2^{l_i}$
(B) $l_i = \lg 1/p_i$.
(C) Average length of a code word is

$$\sum_i p_i \lg \frac{1}{p_i}.$$

(D) X is a random variable that takes a value i with probability p_i, then this formula is

$$\mathbb{H}(X) = \sum_i \mathbf{Pr}[X = i] \lg \frac{1}{\mathbf{Pr}[X = i]},$$

which is the ***entropy*** of X.