CS 473: Algorithms

Chandra Chekuri
chekuri@cs.uiuc.edu
3228 Siebel Center

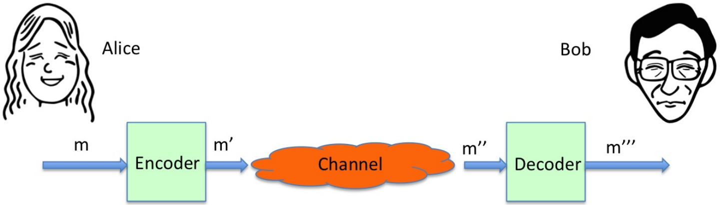
University of Illinois, Urbana-Champaign

Fall 2008

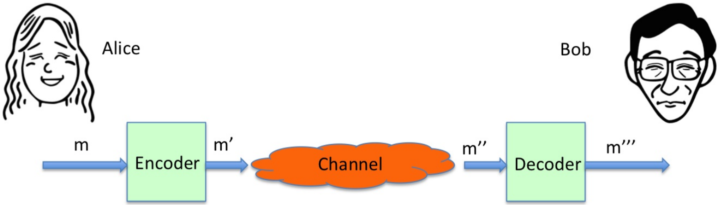
Part I

Information Transmission

Information Transmission



Information Transmission



- compression
- error correction
- cryptography/security

(En)Coding and Decoding

- input alphabet Σ (letters)
- output/channel alphabet Δ
- message m: string in Σ^*

(En)Coding

A function that maps strings $m \in \Sigma^*$ to strings $m' \in \Delta$:
$C : \Sigma^* \rightarrow \Delta^*$.

Decoding

A function that maps strings in Δ to strings in Σ: $D : \Delta^* \rightarrow \Sigma^*$.

Error Correction

- input message m, coded message $m' = C(m)$
- m' corrupted by channel, received message is m''
- Decoded message is $D(m'')$
- Goal: want $D(m'') = m$ if not too many errors (different models)
    - maximum k errors
    - maximum α fraction of errors
    - each bit randomly flipped with some probability
    - some bits not received (erasures)

Requires length of $C(m)$ to be longer than m.

Cryptography

- input message m, coded message $m' = C(m)$
- Decoded message is $D(m')$
- Goal: want $D(m') = m$ *and* eavesdropper should not be able to infer m from m'. Many different scenarios.

Typically requires length of $C(m)$ to be longer than m.

Compression

- input message m, coded message $m' = C(m)$
- Decoded message is $D(m')$
- Goal: want $D(m') = m$ *and* m' is as "short" as possible

Single Use Compression

Comression of a file: example Unix compress, gzip, WinZip, pkzip ...

- m is (usually) very large
- tailor made code C that works only for m
- the endecoding mechanism/decoding algorithm stored as part of m'!
- m is large enough that above does not increase size of m' too much.

Compression in Information Transmission

- m may not be very big
- many different messages sent over time
- sender and receiver may have to agree on C apriori

Compression in Information Transmission

- m may not be very big
- many different messages sent over time
- sender and receiver may have to agree on C apriori

Requirement: some assumption on distribution of messages

Compression in Information Transmission

- m may not be very big
- many different messages sent over time
- sender and receiver may have to agree on C apriori

Requirement: some assumption on distribution of messages
Example: messages are English text (emails)
Knowledge: frequencies of various letters, words, phrases etc.

A Simple Distributional Model

Knowledge about *typical* frequency of letters from Σ.

A Simple Distributional Model

Knowledge about *typical* frequency of letters from Σ.

Example: English text

A Simple Distributional Model

Knowledge about *typical* frequency of letters from Σ.

Example: English text
What is the most frequent letter?

A Simple Distributional Model

Knowledge about *typical* frequency of letters from Σ.

Example: English text
What is the most frequent letter? "e"

A Simple Distributional Model

Knowledge about *typical* frequency of letters from Σ.

Example: English text
What is the most frequent letter? "e"

- let $|\Sigma| = n$
- know probability of occurence of each letter: $p_1, p_2, \ldots, p_n$
- for $1 \leq i \leq n$, $p_i \in [0, 1]$ and $\sum_{i=1}^{n} p_i = 1$

A Simple Coding Strategy

- Map each letter in Σ to a string in Δ^*, that is $C : \Sigma \to \Delta^*$
- Suppose message $m = a_1 a_2 \ldots a_k$ where $a_i \in \Sigma$. Then $C(a_1 a_2 \ldots a_k) = C(a_1)C(a_2)\ldots C(a_k)$

Fixed Length Codes

Fixed Length Codes

Have same length encoding for each symbol in Σ. That is $|C(a)| = |C(b)|$ for each $a, b \in \Sigma$.

Fixed Length Codes

Fixed Length Codes

Have same length encoding for each symbol in Σ. That is $|C(a)| = |C(b)|$ for each $a, b \in \Sigma$.

Example

ASCII Map English letters and keyboard symbols into 7 bits each.
$\Delta = \{0, 1\}$
Decoding: break output string into chunks of 7 bits and map them back to letters.

Fixed Length Codes

Fixed Length Codes

Have same length encoding for each symbol in Σ. That is $|C(a)| = |C(b)|$ for each $a, b \in \Sigma$.

Example

ASCII Map English letters and keyboard symbols into 7 bits each.
$\Delta = \{0, 1\}$
Decoding: break output string into chunks of 7 bits and map them back to letters.

Fixed length codes ignore different frequencies of letters and hence essentially achieve no compression. They are used for information representation.

Variable Length Codes

Variable Length Codes

Have different length encoding for each symbol

- Shorter encodings for more frequent symbols will reduce the *average* bits per symbol.

Variable Length Codes

Variable Length Codes

Have different length encoding for each symbol

- Shorter encodings for more frequent symbols will reduce the *average* bits per symbol.

Example

Morse code is a variable length encoding. Maps *e* to 0 (dot), *t* to 1 (dash), *a* to 01 (dot-dash), ...
What is the text for 0101?

Variable Length Codes

Variable Length Codes

Have different length encoding for each symbol

- Shorter encodings for more frequent symbols will reduce the *average* bits per symbol.

Example

Morse code is a variable length encoding. Maps *e* to 0 (dot), *t* to 1 (dash), *a* to 01 (dot-dash), . . .
What is the text for 0101? Could be *etet*, or *aa* or *eta* or *aet*!
Ambiguity removed by adding pauses between letters.

Variable Length Codes

Variable Length Codes

Have different length encoding for each symbol

- Shorter encodings for more frequent symbols will reduce the *average* bits per symbol.

Example

Morse code is a variable length encoding. Maps *e* to 0 (dot), *t* to 1 (dash), *a* to 01 (dot-dash), . . .
What is the text for 0101? Could be *etet*, or *aa* or *eta* or *aet*!
Ambiguity removed by adding pauses between letters.

- But then encoding is not over 0,1 but over 0,1,2.

Prefix Codes

Definition

A prefix code for a set Σ is function γ such that

1. For $x \in \Sigma$, $\gamma(x)$ is a bit-string
2. For distinct x and y, it is not the case that $\gamma(x)$ is a prefix of $\gamma(y)$, or vice versa.

Prefix Codes

Definition

A prefix code for a set Σ is function γ such that

1. For $x \in \Sigma$, $\gamma(x)$ is a bit-string
2. For distinct x and y, it is not the case that $\gamma(x)$ is a prefix of $\gamma(y)$, or vice versa.

Example

Consider $\Sigma = \{a, b, c, d, e\}$ with encoding γ as follows:

$$\begin{array}{cc} \gamma(a) = 11 & \gamma(b) = 01 \\ \gamma(c) = 001 & \gamma(d) = 10 \\ \multicolumn{2}{c}{\gamma(e) = 000} \end{array}$$

String "bad" encoded as 01 11 10

Decoding Prefix Codes

Decoding Prefix Codes

Algorithm

1. Scan the bit sequence from left to right
2. When a prefix matches code of some symbol, output the symbol

Decoding Prefix Codes

Algorithm

1. Scan the bit sequence from left to right
2. When a prefix matches code of some symbol, output the symbol
   - Justified since no shorter prefix, nor longer extension could encode a symbol

Example

$\Sigma = \{a, b, c, d, e\}$, with

$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

$$0010000011101$$

Decoding Prefix Codes

Algorithm

1. Scan the bit sequence from left to right
2. When a prefix matches code of some symbol, output the symbol
   - Justified since no shorter prefix, nor longer extension could encode a symbol

Example

$\Sigma = \{a, b, c, d, e\}$, with
$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

0010000011101

c

Decoding Prefix Codes

Algorithm

1. Scan the bit sequence from left to right
2. When a prefix matches code of some symbol, output the symbol
   - Justified since no shorter prefix, nor longer extension could encode a symbol

Example

$\Sigma = \{a, b, c, d, e\}$, with
$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

001000011101

$c \quad e$

Decoding Prefix Codes

Algorithm

1. Scan the bit sequence from left to right
2. When a prefix matches code of some symbol, output the symbol
    - Justified since no shorter prefix, nor longer extension could encode a symbol

Example

$\Sigma = \{a, b, c, d, e\}$, with
$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

0010000011101

$c \quad e \quad c$

Decoding Prefix Codes

Algorithm

1. Scan the bit sequence from left to right
2. When a prefix matches code of some symbol, output the symbol
   - Justified since no shorter prefix, nor longer extension could encode a symbol

Example

$\Sigma = \{a, b, c, d, e\}$, with

$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

001000001**11**01

$c \quad e \quad c \quad a$

Decoding Prefix Codes

Algorithm

1. Scan the bit sequence from left to right
2. When a prefix matches code of some symbol, output the symbol
   - Justified since no shorter prefix, nor longer extension could encode a symbol

Example

$\Sigma = \{a, b, c, d, e\}$, with
$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

$$0010000011101$$
$$c \quad e \quad c \quad a \quad b$$

Part II

Huffman Codes

Average Bits per Letter

Given:

- input alphabet Σ with $|\Sigma| = n$ and
- letter probabilities $p_1, p_2, \ldots, p_n$
- $\Delta = \{0, 1\}$: binary

Average Bits per Letter

Given:

- input alphabet Σ with $|\Sigma| = n$ and
- letter probabilities $p_1, p_2, \ldots, p_n$
- $\Delta = \{0, 1\}$: binary

Definition

For an alphabet Σ, with probability p_x for symbol x ($\sum_{x \in \Sigma} p_x = 1$), the average number of bits required per letter under the encoding γ

$$\mathrm{ABL}(\gamma) = \sum_{x \in \Sigma} p_x |\gamma(x)|.$$

ABL: Example

Example

For $\Sigma = \{a, b, c, d, e\}$, with probabilities

$$p_a = 0.32 \qquad p_b = 0.25 \qquad p_c = 0.20 \qquad p_d = 0.18 \qquad p_e = 0.05$$

Consider

$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

$$\mathrm{ABL}(\gamma) = 2 \times 0.32 + 2 \times 0.25 + 3 \times 0.2 + 2 \times 0.18 + 3 \times 0.05 = 2.25$$

ABL: Example

Example

For $\Sigma = \{a, b, c, d, e\}$, with probabilities

$p_a = 0.32 \qquad p_b = 0.25 \qquad p_c = 0.20 \qquad p_d = 0.18 \qquad p_e = 0.05$

Consider
$\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

$\mathrm{ABL}(\gamma) = 2 \times 0.32 + 2 \times 0.25 + 3 \times 0.2 + 2 \times 0.18 + 3 \times 0.05 = 2.25$

Consider
$\gamma'(a) = 11,\ \gamma'(b) = 10,\ \gamma'(c) = 01,\ \gamma'(d) = 001,\ \gamma'(e) = 000$
Then $\mathrm{ABL}(\gamma') = 2.23$

Optimal Prefix Codes

Input Given a set Σ and probabilities p_x for each $x \in \Sigma$

Goal Find a prefix code γ for Σ over $\Delta = \{0, 1\}$ such that $\mathrm{ABL}(\gamma)$ is minimum.

Prefix Codes and Binary Trees

Proposition

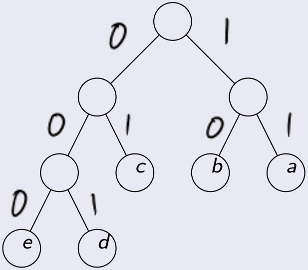
There is a 1-to-1 onto correspondence between prefix codes in Σ and binary trees whose leaves are labelled by $x \in \Sigma$

Proof.

$\gamma(x)$ will be path from root to leaf labelled x in tree, where left child is 0 and right child is 1.

$$\gamma'(a) = 11$$
$$\gamma'(b) = 10$$
$$\gamma'(c) = 01$$
$$\gamma'(d) = 001$$
$$\gamma'(e) = 000$$

$\longleftrightarrow$



Prefix Codes and Binary Trees

Lemma

If T is a rooted binary tree and there is a bijection between the leaves L of T and Σ, then there is a prefix-code $\gamma : \Sigma \to \{0,1\}^*$ where $\gamma(a)$ is given by the path from root of T to a.

Prefix Codes and Binary Trees

Lemma

If T is a rooted binary tree and there is a bijection between the leaves L of T and Σ, then there is a prefix-code $\gamma : \Sigma \to \{0,1\}^*$ where $\gamma(a)$ is given by the path from root of T to a.

Proof Sketch.

- Define $\gamma(a)$ for each a by walking from root to a: output a 0 if the path uses a left child and a 1 if path uses right child. Creates a string of 0's and 1's.
- γ is a prefix code. Why?

Prefix Codes and Binary Trees

Lemma

If T is a rooted binary tree and there is a bijection between the leaves L of T and Σ, then there is a prefix-code $\gamma : \Sigma \to \{0, 1\}^*$ where $\gamma(a)$ is given by the path from root of T to a.

Proof Sketch.

- Define $\gamma(a)$ for each a by walking from root to a: output a 0 if the path uses a left child and a 1 if path uses right child. Creates a string of 0's and 1's.
- γ is a prefix code. Why? If $\gamma(a)$ is a prefix of $\gamma(b)$ then from construction a must be on the path from root to b. But all letters are at leaves of T.

□

Prefix Codes and Binary Trees

Lemma

If $\gamma : \Sigma \to \{0,1\}^*$ *is a prefix-code then there is a rooted binary tree* T *and a bijection from* Σ *to the leaves* L *of* T*.*

Prefix Codes and Binary Trees

Lemma

If $\gamma : \Sigma \rightarrow \{0,1\}^*$ is a prefix-code then there is a rooted binary tree T and a bijection from Σ to the leaves L of T.

Proof Sketch.

- Given γ, create T as follows.
- Let $\Sigma_0 \subset \Sigma$ where $a \in \Sigma_0$ iff $\gamma(a)$ starts with 0. $\Sigma_1 = \Sigma - \Sigma_0$.
- Recursively create tree T_0 for Σ_0 with $\gamma'(a)$ is obtained from $\gamma(a)$ by removing the leading 0. Note: γ' is prefix-code for Σ_0.
- Similarly, T_1 for Σ_1 with leading 1 removed.
- Create T from T_0 and T_1 by adding root r and making T_0 the left sub-tree and T_1 the right sub-tree.

□

$\gamma(a) = 0101$

$\gamma(b) = 010$

Optimal Codes and Full Trees

Definition

A binary tree is full if every internal node has two children.

Optimal Codes and Full Trees

Definition

A binary tree is full if every internal node has two children.

Proposition

The binary tree corresponding to the optimal code is full.

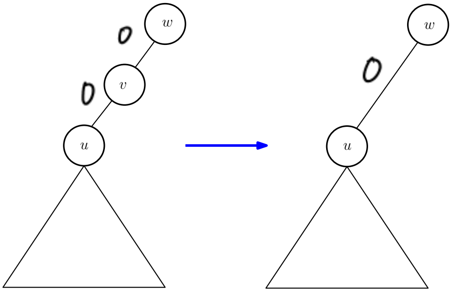
Optimal Codes and Full Trees

Definition

A binary tree is full if every internal node has two children.

Proposition

The binary tree corresponding to the optimal code is full.



Optimal Codes and Full Trees

Definition

A binary tree is full if every internal node has two children.

Proposition

The binary tree corresponding to the optimal code is full.

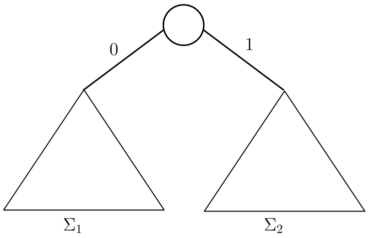
Proof.

- Suppose (for contradiction) T is optimal code, where u has only one child v
- Consider T' where u is removed; if u is the root make v root, otherwise, attach v to parent of u
- T' has a smaller average code, as the code of leaves below u has been shortened by 1 bit. □

Top-Down Approach

Algorithm [Shannon-Fano]

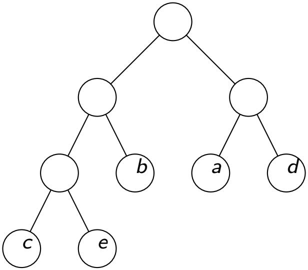
1. Divide Σ into Σ_1 and Σ_2 such that total frequency of Σ_1 and Σ_2 is (if possible) $\frac{1}{2}$
2. Recursively find code for γ_1 for Σ_1 and γ_2 for Σ_2.
3. Code for Σ: $\gamma(x) = 0\gamma_1(x)$, $x \in \Sigma_1$ & $\gamma(x) = 1\gamma_2(x)$, $x \in \Sigma_2$



Example

Example

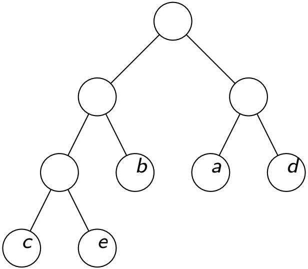
Consider $\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$. First split results in $\{b, c, e\}$ and $\{a, d\}$ and recursively find codes. Resulting code is $\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,$ $\gamma(e) = 000$.



Example

Example

Consider $\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$. First split results in $\{b, c, e\}$ and $\{a, d\}$ and recursively find codes. Resulting code is $\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,$ $\gamma(e) = 000$. γ not optimal; γ' shown earlier is better.

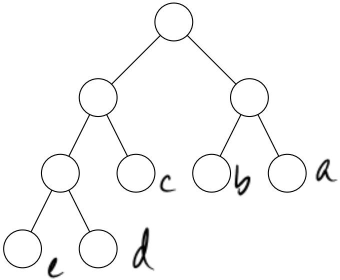


Understanding an Optimal Solution

- Given Σ and p_x for each $x \in \Sigma$
- Suppose we knew the (optimum) tree T but not a labeling of the leaves by Σ. Can we label the leaves?

$$\Sigma = \{a, b, c, d, e\}$$

$$p_a = 0.32$$
$$p_b = 0.25$$
$$p_c = 0.2$$
$$p_d = 0.18$$
$$p_e = .05$$



Depth and Probability

Proposition

Let T^* be an optimal prefix code. For leaves u and v with labels x and y, respectively, if $\mathrm{depth}(u) < \mathrm{depth}(v)$ then $p_x \geq p_y$.

Depth and Probability

Proposition

Let T^* be an optimal prefix code. For leaves u and v with labels x and y, respectively, if $\mathrm{depth}(u) < \mathrm{depth}(v)$ then $p_x \geq p_y$.

Proof.

Depth and Probability

Proposition

Let T^* be an optimal prefix code. For leaves u and v with labels x and y, respectively, if $\mathrm{depth}(u) < \mathrm{depth}(v)$ then $p_x \geq p_y$.

Proof.

- Suppose (for contradiction) $p_x < p_y$

Depth and Probability

Proposition

Let T^* be an optimal prefix code. For leaves u and v with labels x and y, respectively, if $\mathrm{depth}(u) < \mathrm{depth}(v)$ then $p_x \geq p_y$.

Proof.

- Suppose (for contradiction) $p_x < p_y$
- Consider tree T_1^* where the labels of leaves u and v have been exchanged.

$$\mathrm{ABL}(T^*) - \mathrm{ABL}(T_1^*) \quad = \quad \sum_{z \in \Sigma} p_z \mathrm{depth}_{T^*}(z) - \sum_{z \in \Sigma} p_z \mathrm{depth}_{T_1^*}(z)$$

Depth and Probability

Proposition

Let T^* be an optimal prefix code. For leaves u and v with labels x and y, respectively, if $\mathrm{depth}(u) < \mathrm{depth}(v)$ then $p_x \geq p_y$.

Proof.

- Suppose (for contradiction) $p_x < p_y$
- Consider tree T_1^* where the labels of leaves u and v have been exchanged.

$$\begin{aligned}
\mathrm{ABL}(T^*) - \mathrm{ABL}(T_1^*) &= \textstyle\sum_{z \in \Sigma} p_z \mathrm{depth}_{T^*}(z) - \sum_{z \in \Sigma} p_z \mathrm{depth}_{T_1^*}(z) \\
&= (\mathrm{depth}(u) p_x + \mathrm{depth}(v) p_y) \\
&\qquad -(\mathrm{depth}(u) p_y + \mathrm{depth}(v) p_x)
\end{aligned}$$

Depth and Probability

Proposition

Let T^* be an optimal prefix code. For leaves u and v with labels x and y, respectively, if $\mathrm{depth}(u) < \mathrm{depth}(v)$ then $p_x \geq p_y$.

Proof.

- Suppose (for contradiction) $p_x < p_y$
- Consider tree T_1^* where the labels of leaves u and v have been exchanged.

$$\begin{aligned} \mathrm{ABL}(T^*) - \mathrm{ABL}(T_1^*) &= \textstyle\sum_{z \in \Sigma} p_z \mathrm{depth}_{T^*}(z) - \sum_{z \in \Sigma} p_z \mathrm{depth}_{T_1^*}(z) \\ &= (\mathrm{depth}(u) p_x + \mathrm{depth}(v) p_y) \\ &\qquad -(\mathrm{depth}(u) p_y + \mathrm{depth}(v) p_x) \\ &= (\mathrm{depth}(v) - \mathrm{depth}(u))(p_y - p_x) > 0 \end{aligned}$$

Depth and Probability

Proposition

Let T^* be an optimal prefix code. For leaves u and v with labels x and y, respectively, if $\mathrm{depth}(u) < \mathrm{depth}(v)$ then $p_x \geq p_y$.

Proof.

- Suppose (for contradiction) $p_x < p_y$
- Consider tree T_1^* where the labels of leaves u and v have been exchanged.

$$\begin{aligned}
\mathrm{ABL}(T^*) - \mathrm{ABL}(T_1^*) &= \textstyle\sum_{z\in\Sigma} p_z \mathrm{depth}_{T^*}(z) - \sum_{z\in\Sigma} p_z \mathrm{depth}_{T_1^*}(z) \\
&= (\mathrm{depth}(u)p_x + \mathrm{depth}(v)p_y) \\
&\qquad -(\mathrm{depth}(u)p_y + \mathrm{depth}(v)p_x) \\
&= (\mathrm{depth}(v) - \mathrm{depth}(u))(p_y - p_x) > 0
\end{aligned}$$

- T_1^* is better, which contradicts optimality of T^* □

Maximum Depth

Corollary

Least frequent symbol labels the leaf of maximum depth.

Observation

If u and v are leaves of T of same depth d, labeled with x and y then T' has the same ABL as T if labels of u and v are swapped.

Maximum Depth

Corollary

Least frequent symbol labels the leaf of maximum depth.

Observation

If u and v are leaves of T of same depth d, labeled with x and y then T' has the same ABL as T if labels of u and v are swapped.

Observation

Any full binary tree with more than two leaves has leaves u and v at maximum depth and which are siblings (share a parent).

Proof.

Let u be a leaf at maximum depth and let w be its parent.
w has another child other than u — this has to be a leaf v since u is at maximum depth. □

Technical Observation

Lemma

Let x and y be the two least frequent elements. Then there is an optimal code T^* such that x, y are siblings.

Technical Observation

Lemma

Let x and y be the two least frequent elements. Then there is an optimal code T^* such that x, y are siblings.

Proof.

Technical Observation

Lemma

Let x and y be the two least frequent elements. Then there is an optimal code T^* such that x, y are siblings.

Proof.

- Let u, v be two sibling leaves of T^* at maximum depth — they exist by previous observation.

Technical Observation

Lemma

Let x and y be the two least frequent elements. Then there is an optimal code T^* such that x, y are siblings.

Proof.

- Let u, v be two sibling leaves of T^* at maximum depth — they exist by previous observation.
- x, y are at maximum depth since they are least frequent.

Technical Observation

Lemma

Let x and y be the two least frequent elements. Then there is an optimal code T^* such that x, y are siblings.

Proof.

- Let u, v be two sibling leaves of T^* at maximum depth — they exist by previous observation.
- x, y are at maximum depth since they are least frequent.
- If x, y do not label u, v, by observation, can swap them to label u, v without increasing ABL. □

Huffman's Algorithm

Algorithm

1. Find x, y with the two lowest probabilities
2. If $|\Sigma| = 2$ return two-leaf tree with x, y as labels.
3. Let $\Delta = (\Sigma \setminus \{x, y\}) \cup \{\omega\}$ with $p_\omega = p_x + p_y$
4. Recursively find optimal code T' for Σ'
5. Code T for Σ is: Add two leaves to leaf labeled ω in T' and label the leaves x and y

Example

$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32$, $p_b = 0.25$, $p_c = 0.2$, $p_d = 0.18$, $p_e = 0.05$

Example

$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$

$\Sigma = \{a, b, c, \omega_1\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_{\omega_1} = 0.23$

Example

$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$

$\Sigma = \{a, b, c, \omega_1\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_{\omega_1} = 0.23$

$\Sigma = \{a, b, \omega_2\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_{\omega_2} = 0.43$

Example

$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$

$\Sigma = \{a, b, c, \omega_1\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_{\omega_1} = 0.23$

$\Sigma = \{a, b, \omega_2\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_{\omega_2} = 0.43$

$\Sigma = \{\omega_2, \omega_3\}$ and $p_{\omega_2} = 0.43,\ p_{\omega_3} = 0.57$

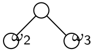
Example

$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$

$\Sigma = \{a, b, c, \omega_1\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_{\omega_1} = 0.23$

$\Sigma = \{a, b, \omega_2\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_{\omega_2} = 0.43$

$\Sigma = \{\omega_2, \omega_3\}$ and $p_{\omega_2} = 0.43,\ p_{\omega_3} = 0.57$

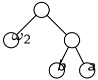


Example

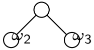
$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$

$\Sigma = \{a, b, c, \omega_1\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_{\omega_1} = 0.23$

$\Sigma = \{a, b, \omega_2\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_{\omega_2} = 0.43$



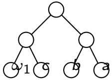
$\Sigma = \{\omega_2, \omega_3\}$ and $p_{\omega_2} = 0.43,\ p_{\omega_3} = 0.57$



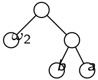
Example

$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$

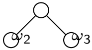
$\Sigma = \{a, b, c, \omega_1\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_{\omega_1} = 0.23$



$\Sigma = \{a, b, \omega_2\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_{\omega_2} = 0.43$

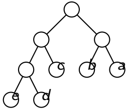


$\Sigma = \{\omega_2, \omega_3\}$ and $p_{\omega_2} = 0.43,\ p_{\omega_3} = 0.57$

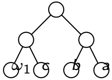


Example

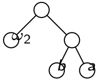
$\Sigma = \{a, b, c, d, e\}$ and $p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_d = 0.18,\ p_e = 0.05$



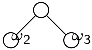
$\Sigma = \{a, b, c, \omega_1\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_c = 0.2,\ p_{\omega_1} = 0.23$



$\Sigma = \{a, b, \omega_2\}$ and
$p_a = 0.32,\ p_b = 0.25,\ p_{\omega_2} = 0.43$



$\Sigma = \{\omega_2, \omega_3\}$ and $p_{\omega_2} = 0.43,\ p_{\omega_3} = 0.57$



Property about Recursive Step

Proposition

Let $\Sigma' = (\Sigma \setminus \{x, y\}) \cup \{\omega\}$, T' *be the Huffman code for* Σ' *and* T *the huffman code for* Σ. *Then,*

$$\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega = \mathrm{ABL}(T') + (p_x + p_y)$$

Property about Recursive Step

Proposition

Let $\Sigma' = (\Sigma \setminus \{x, y\}) \cup \{\omega\}$, T' *be the Huffman code for* Σ' *and* T *the huffman code for* Σ. *Then,*

$$\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega = \mathrm{ABL}(T') + (p_x + p_y)$$

Proof.

Property about Recursive Step

Proposition

Let $\Sigma' = (\Sigma \setminus \{x, y\}) \cup \{\omega\}$*,* T' *be the Huffman code for* Σ' *and* T *the huffman code for* Σ*. Then,*

$$\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega = \mathrm{ABL}(T') + (p_x + p_y)$$

Proof.

- $\mathrm{depth}(z)$ for $z \neq x, y$ is the same in both T and T'.

Property about Recursive Step

Proposition

Let $\Sigma' = (\Sigma \setminus \{x, y\}) \cup \{\omega\}$, T' *be the Huffman code for* Σ' *and* T *the huffman code for* Σ. *Then,*

$$\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega = \mathrm{ABL}(T') + (p_x + p_y)$$

Proof.

- $\mathrm{depth}(z)$ for $z \neq x, y$ is the same in both T and T'.
- $\mathrm{depth}_T(x) = \mathrm{depth}_T(y) = \mathrm{depth}_{T'}(\omega) + 1$ and $p_\omega = p_x + p_y$

Property about Recursive Step

Proposition

Let $\Sigma' = (\Sigma \setminus \{x, y\}) \cup \{\omega\}$*,* T' *be the Huffman code for* Σ' *and* T *the huffman code for* Σ*. Then,*

$$\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega = \mathrm{ABL}(T') + (p_x + p_y)$$

Proof.

- $\mathrm{depth}(z)$ for $z \neq x, y$ is the same in both T and T'.
- $\mathrm{depth}_T(x) = \mathrm{depth}_T(y) = \mathrm{depth}_{T'}(\omega) + 1$ and $p_\omega = p_x + p_y$

$$\begin{aligned}
\mathrm{ABL}(T) &= \sum_{z \in S} p_z \mathrm{depth}_T(z) \\
&= p_x \mathrm{depth}_T(x) + p_y \mathrm{depth}_T(y) + \sum_{z \neq x,y} p_z \mathrm{depth}_T(z) \\
&= (p_x + p_y)(1 + \mathrm{depth}_{T'}(\omega)) + \sum_{z \neq x,y} p_z \mathrm{depth}_{T'}(z) \\
&= p_\omega + p_\omega \mathrm{depth}_{T'}(\omega) + \sum_{z \neq x,y} p_z \mathrm{depth}_{T'}(z) \\
&= p_\omega + \mathrm{ABL}(T') \quad \square
\end{aligned}$$

Property about Optimal Encoding

Proposition

Let Z be an optimal tree for Σ and let Z' be an optimal tree for $(\Sigma \cup \{\omega\}) \setminus \{x, y\}$ where x, y are the two least probable letters. Then $\mathrm{ABL}(Z') \leq \mathrm{ABL}(Z) - (p_x + p_y)$.

Property about Optimal Encoding

Proposition

Let Z be an optimal tree for Σ and let Z' be an optimal tree for $(\Sigma \cup \{\omega\}) \setminus \{x, y\}$ where x, y are the two least probable letters. Then $\mathrm{ABL}(Z') \leq \mathrm{ABL}(Z) - (p_x + p_y)$.

Proof.

- From Lemma on optimal trees, assume x, y are siblings in Z.
- Obtain a tree Y for $(\Sigma \cup \{\omega\}) \setminus \{x, y\}$ from Z by removing x, y from Z and labeling parent of x, y with ω.
- Y is a valid tree for $(\Sigma \cup \{\omega\}) \setminus \{x, y\}$
- $\mathrm{ABL}(Y) = \mathrm{ABL}(Z) - (p_x + p_y)$.
- An optimal tree Z' cannot be worse than Y. □

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

- Base case: Huffman code is optimal when $|\Sigma| = 2$. Assume Huffman code is optimal for Σ when $|\Sigma| < k$

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

- Base case: Huffman code is optimal when $|\Sigma| = 2$. Assume Huffman code is optimal for Σ when $|\Sigma| < k$
- Consider Σ ($|\Sigma| = k$). Let x, y be least probable in Σ. $\Sigma' = (\Sigma \cup \{\omega\}) \setminus \{x, y\}$. T, T' be Huffman codes for Σ, Σ'.

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

- Base case: Huffman code is optimal when $|\Sigma| = 2$. Assume Huffman code is optimal for Σ when $|\Sigma| < k$
- Consider Σ ($|\Sigma| = k$). Let x, y be least probable in Σ. $\Sigma' = (\Sigma \cup \{\omega\}) \setminus \{x, y\}$. T, T' be Huffman codes for Σ, Σ'.
- Let Z, Z' be optimal codes for Σ and Σ' respectively

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

- Base case: Huffman code is optimal when $|\Sigma| = 2$. Assume Huffman code is optimal for Σ when $|\Sigma| < k$
- Consider Σ ($|\Sigma| = k$). Let x, y be least probable in Σ. $\Sigma' = (\Sigma \cup \{\omega\}) \setminus \{x, y\}$. T, T' be Huffman codes for Σ, Σ'.
- Let Z, Z' be optimal codes for Σ and Σ' respectively
- By induction T' is optimal for Σ': $\mathrm{ABL}(T') \leq \mathrm{ABL}(Z')$.

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

- Base case: Huffman code is optimal when $|\Sigma| = 2$. Assume Huffman code is optimal for Σ when $|\Sigma| < k$
- Consider Σ ($|\Sigma| = k$). Let x, y be least probable in Σ. $\Sigma' = (\Sigma \cup \{\omega\}) \setminus \{x, y\}$. T, T' be Huffman codes for Σ, Σ'.
- Let Z, Z' be optimal codes for Σ and Σ' respectively
- By induction T' is optimal for Σ': $\mathrm{ABL}(T') \leq \mathrm{ABL}(Z')$.
- By Proposition, $\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega$.

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

- Base case: Huffman code is optimal when $|\Sigma| = 2$. Assume Huffman code is optimal for Σ when $|\Sigma| < k$
- Consider Σ ($|\Sigma| = k$). Let x, y be least probable in Σ. $\Sigma' = (\Sigma \cup \{\omega\}) \setminus \{x, y\}$. T, T' be Huffman codes for Σ, Σ'.
- Let Z, Z' be optimal codes for Σ and Σ' respectively
- By induction T' is optimal for Σ': $\mathrm{ABL}(T') \leq \mathrm{ABL}(Z')$.
- By Proposition, $\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega$.
- By Proposition, $\mathrm{ABL}(Z') \leq \mathrm{ABL}(Z) - p_\omega$.

① $ABL(T') \leq ABL(Z')$

② $ABL(T) = ABL(T') + p_w$

③ $ABL(Z') \leq ABL(Z) - p_w$

$$\begin{aligned} ABL(T) &= ABL(T') + p_w && ② \\ &\leq ABL(Z') + p_w && ① \\ &\leq ABL(Z) - p_w + p_w && ③ \\ &\leq ABL(Z) && ✓ \end{aligned}$$

Optimality Proof

Theorem

The Huffman code is an optimal prefix code.

Proof by Induction.

- Base case: Huffman code is optimal when $|\Sigma| = 2$. Assume Huffman code is optimal for Σ when $|\Sigma| < k$
- Consider Σ ($|\Sigma| = k$). Let x, y be least probable in Σ. $\Sigma' = (\Sigma \cup \{\omega\}) \setminus \{x, y\}$. T, T' be Huffman codes for Σ, Σ'.
- Let Z, Z' be optimal codes for Σ and Σ' respectively
- By induction T' is optimal for Σ': $\mathrm{ABL}(T') \leq \mathrm{ABL}(Z')$.
- By Proposition, $\mathrm{ABL}(T) = \mathrm{ABL}(T') + p_\omega$.
- By Proposition, $\mathrm{ABL}(Z') \leq \mathrm{ABL}(Z) - p_\omega$.
- Implies $\mathrm{ABL}(T) \leq \mathrm{ABL}(Z)$ and hence T is optimal. □

Implementation and Analysis

```
if Σ has two letters then
    encode one as 0 and the other as 1
else
    let x,y be the lowest probability letters
    remove x,y and add ω to get Σ'
    recursively find code T' for Σ'
    code T for Σ is as follows
        for z ≠ x,y T(z) = T'(z)
        T(x) = 0T'(ω) and T(y) = 1T'(ω)
```

Implementation and Analysis

```
if Σ has two letters then
    encode one as 0 and the other as 1
else
    let x,y be the lowest probability letters
    remove x,y and add ω to get Σ'
    recursively find code T' for Σ'
    code T for Σ is as follows
        for z ≠ x,y T(z) = T'(z)
        T(x) = 0T'(ω) and T(y) = 1T'(ω)
```

Implementation and Analysis

```
if Σ has two letters then
    encode one as 0 and the other as 1
else
    let x,y be the lowest probability letters
    remove x,y and add ω to get Σ'
    recursively find code T' for Σ'
    code T for Σ is as follows
        for z ≠ x,y T(z) = T'(z)
        T(x) = 0T'(ω) and T(y) = 1T'(ω)
```

- Store Σ in a priority queue with the probability as key
- Each iteration takes $O(\log n)$
- Total time is $O(n \log n)$ for the $n-2$ iterations