

4.8 Huffman Codes

These lecture slides are supplied by Mathijs de Weerd

Data Compression

Q. Given a text that uses **32** symbols (26 different letters, space, and some punctuation characters), how can we encode this text in *bits*?

Q. Some symbols (**e, t, a, o, i, n**) are used far **more often** than others. How can we use this to **reduce** our encoding?

Q. How do we know when the next symbol begins?

Ex. $c(a) = 01$
 $c(b) = 010$
 $c(e) = 1$

What is **0101**?

Data Compression

Q. Given a text that uses 32 symbols (26 different letters, space, and some punctuation characters), how can we encode this text in bits?

A. We can encode 2^5 different symbols using a **fixed** length of **5** bits per symbol. This is called **fixed length encoding**.

Q. Some symbols (*e, t, a, o, i, n*) are used far **more often** than others. How can we use this to **reduce** our encoding?

A. Encode these characters with **fewer** bits, and the others with **more** bits.

Q. How do we know when the next symbol begins?

A. Use a separation symbol (like the pause in Morse), or make sure that there is no ambiguity by ensuring that **no** code is a **prefix** of another one.

Ex. $c(a) = 01$

$c(b) = 010$

$c(e) = 1$

What is **0101**?

Prefix Codes

Definition. A **prefix code** for a set S is a function c that maps each $x \in S$ to 1s and 0s in such a way that for $x, y \in S$, $x \neq y$, $c(x)$ is not a prefix of $c(y)$.

Ex. $c(a) = 11$

$c(e) = 01$

$c(k) = 001$

$c(l) = 10$

$c(u) = 000$

Q. What is the meaning of 1001000001 ?

Suppose **frequencies** are known in a text of 1G:

$f_a=0.4$, $f_e=0.2$, $f_k=0.2$, $f_l=0.1$, $f_u=0.1$

Q. What is the **size** of the encoded text?

Prefix Codes

Definition. A **prefix code** for a set S is a function c that maps each $x \in S$ to 1s and 0s in such a way that for $x, y \in S$, $x \neq y$, $c(x)$ is not a prefix of $c(y)$.

Ex. $c(a) = 11$

$c(e) = 01$

$c(k) = 001$

$c(l) = 10$

$c(u) = 000$

Q. What is the meaning of 1001000001 ?

A. “leuk”

Suppose frequencies are known in a text of 1G:

$f_a=0.4$, $f_e=0.2$, $f_k=0.2$, $f_l=0.1$, $f_u=0.1$

Q. What is the size of the encoded text?

A. $2*f_a + 2*f_e + 3*f_k + 2*f_l + 4*f_u = 2.4G$

Optimal Prefix Codes

Definition. The **average bits per letter** of a **prefix** code c is the **sum** over all symbols of:

(its frequency) $\times$ (the number of bits of its encoding):

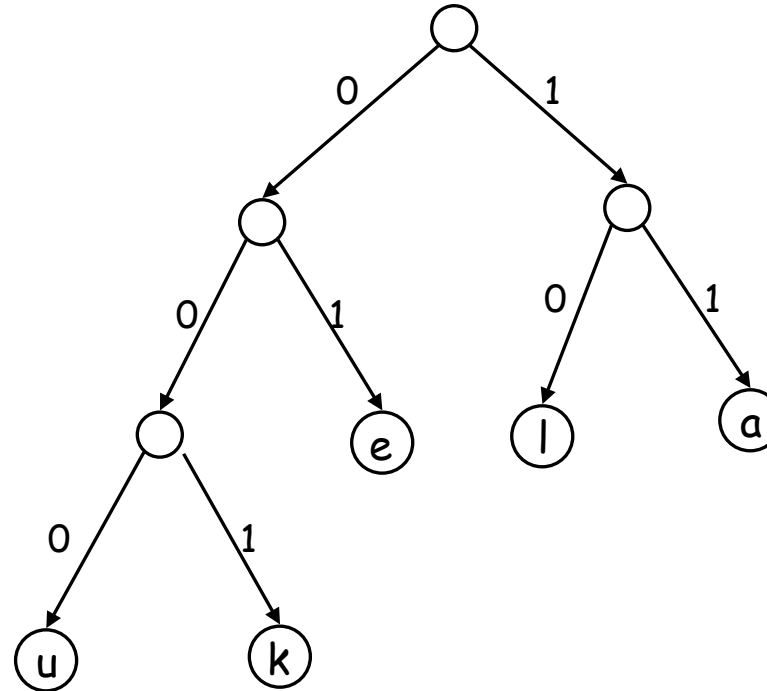
$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

GOAL: find a prefix code that is has the **lowest** possible **average bits** per letter.

We can model a code in a **binary tree**...

Representing Prefix Codes using Binary Trees

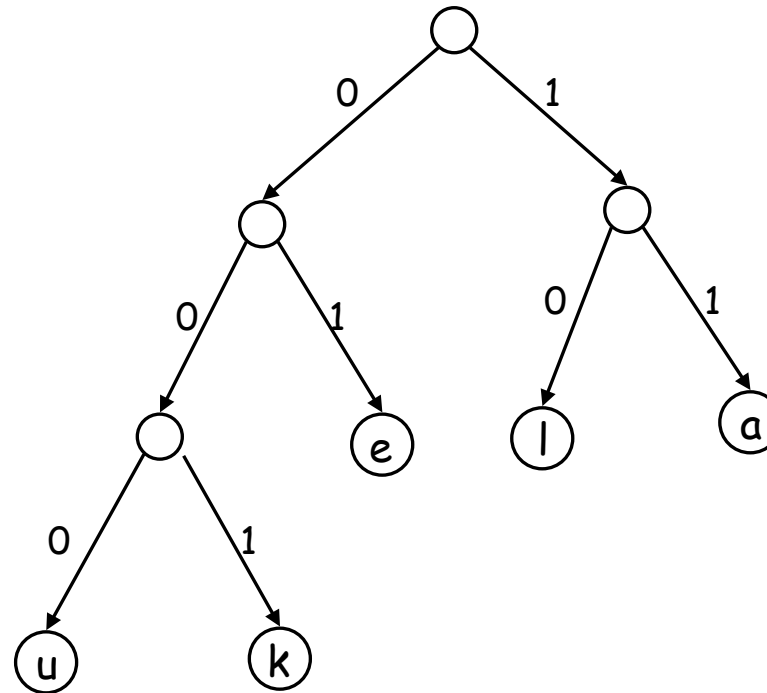
Ex. $c(a) = 11$
 $c(e) = 01$
 $c(k) = 001$
 $c(l) = 10$
 $c(u) = 000$



Q. How does the tree of a prefix code look?

Representing Prefix Codes using Binary Trees

Ex. $c(a) = 11$
 $c(e) = 01$
 $c(k) = 001$
 $c(l) = 10$
 $c(u) = 000$



Q. How does the **tree** of a **prefix code** look?

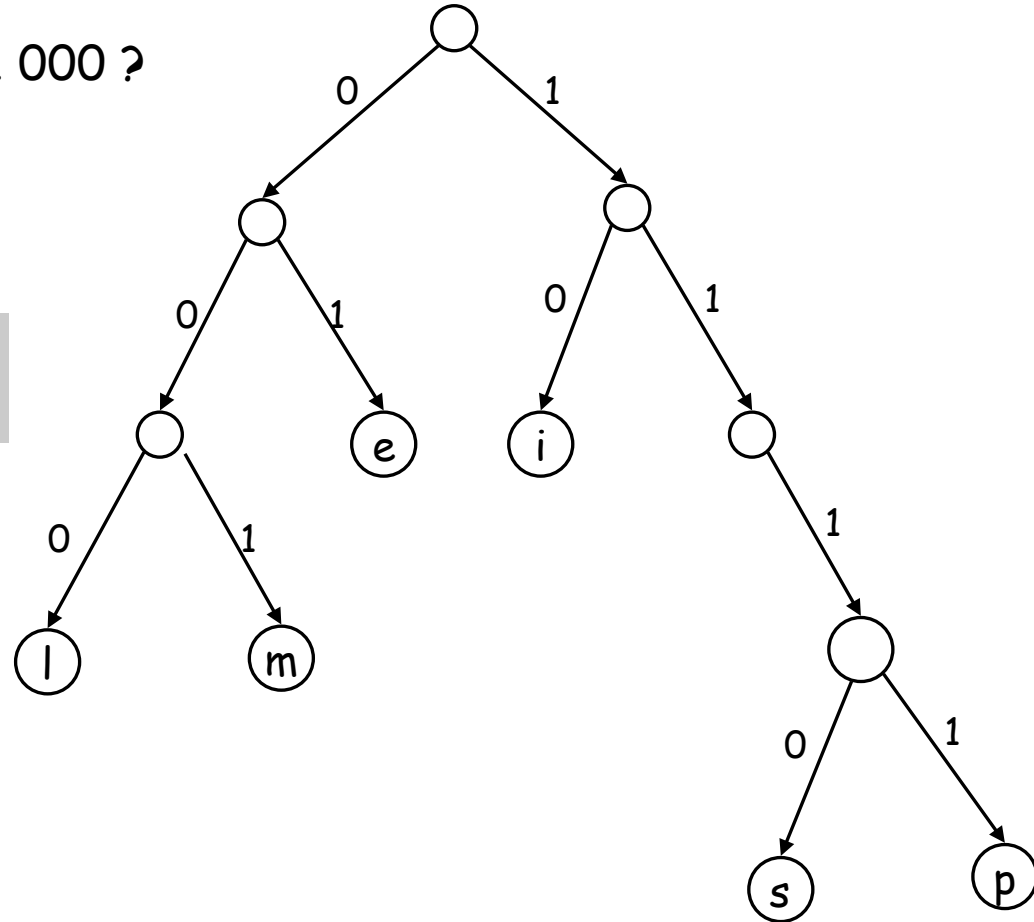
A. Only the **leaves** have a **label**.

Proof. An encoding of x is a prefix of an encoding of y iff the path of x is a prefix of the path of y .

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
1110 10 001 1111 01 000 ?

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$

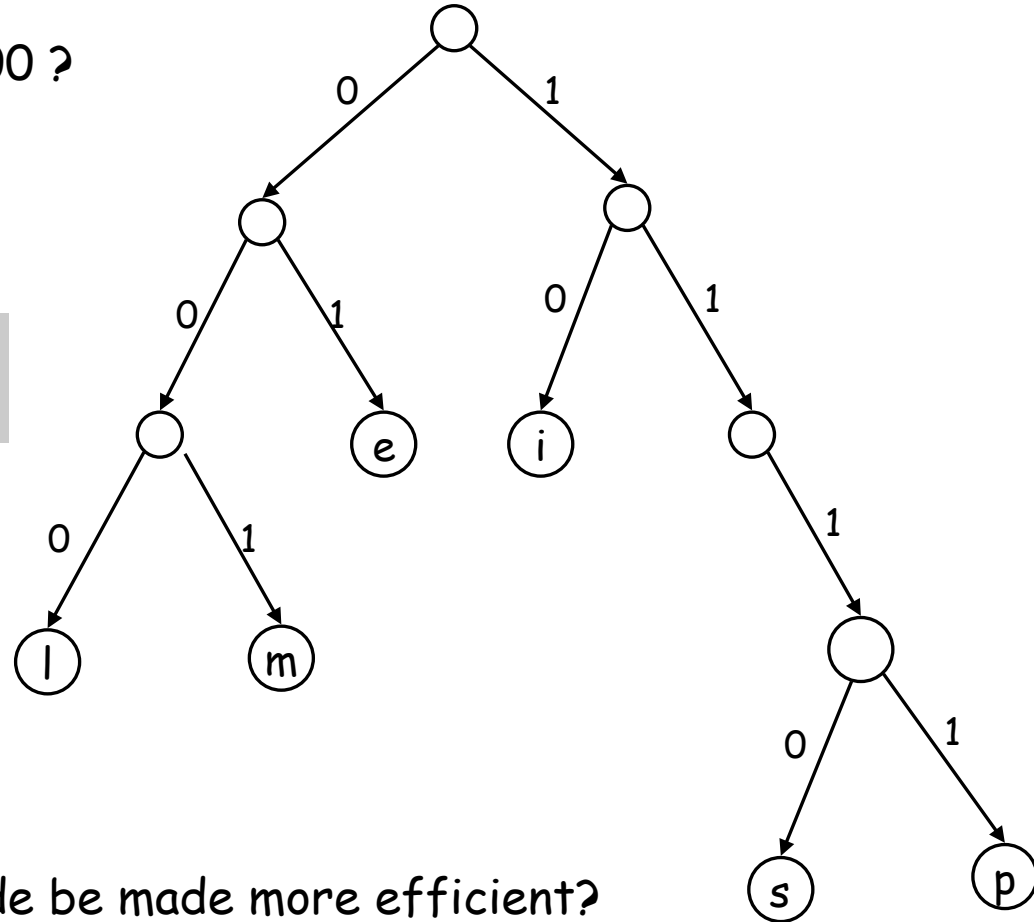


Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?

A. “simpel”

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



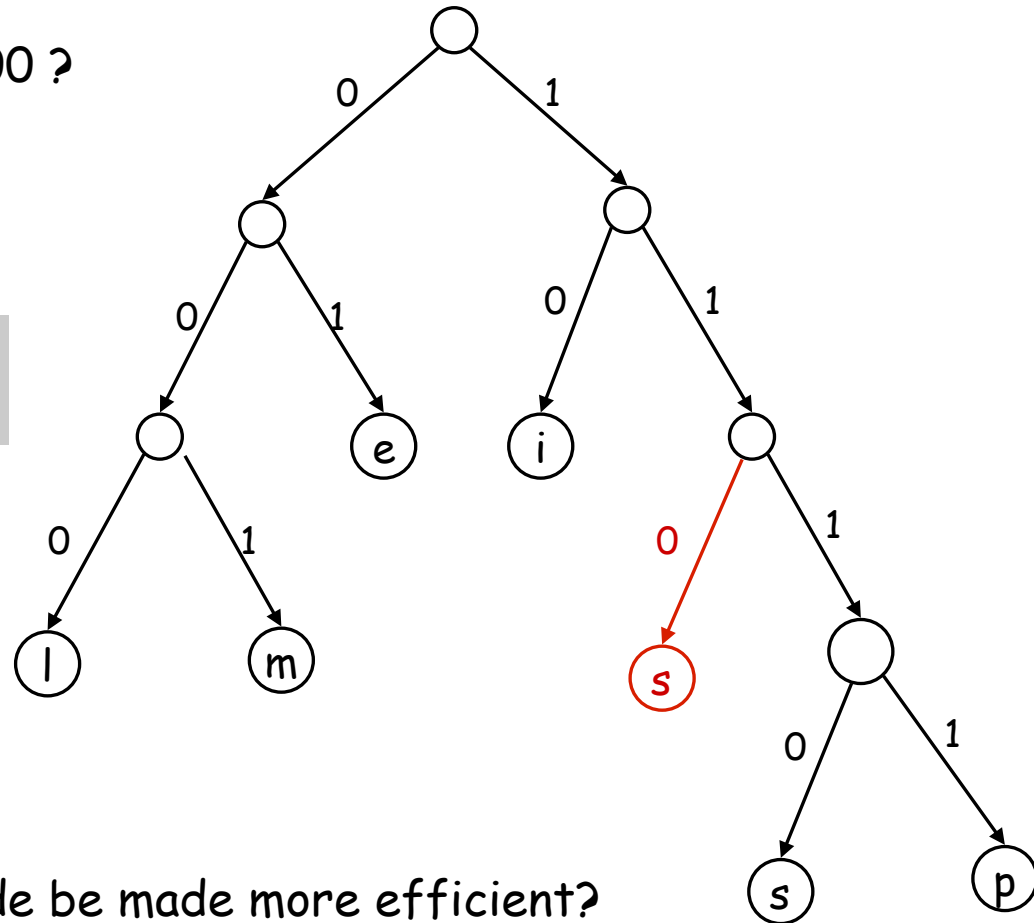
Q. How can this prefix code be made more efficient?

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?

A. “simpel”

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



Q. How can this prefix code be made more efficient?

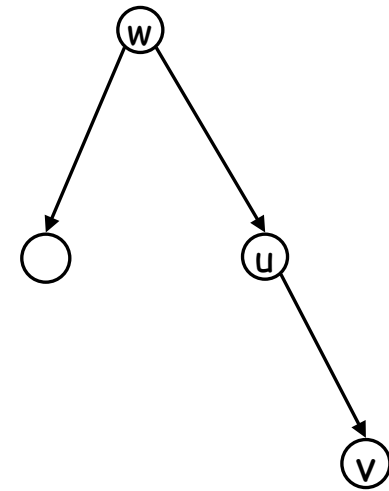
A. Change encoding of **p** and **s** to a shorter one.

This tree is now **full**.

Representing Prefix Codes using Binary Trees

Definition. A tree is **full** if every node that is not a leaf has two children.

Claim. The binary tree corresponding to an **optimal** prefix code is **full**.
Pf.



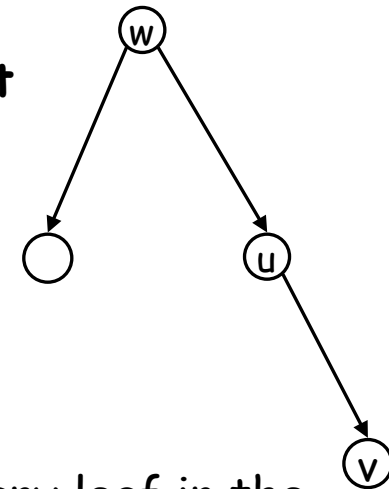
Representing Prefix Codes using Binary Trees

Definition. A tree is **full** if every node that is not a leaf has two children.

Claim. The binary tree corresponding to the **optimal** prefix code is full.

Proof. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.
- This means there is a node U with only one child V .
- **Case 1:** U is the root; delete U and use V as the **root**
- **Case 2:** U is not the root
 - let W be the parent of U
 - delete U and make V be a child of W in place of U
- In both cases the number of bits needed to encode any leaf in the subtree of V is **decreased**. The rest of the tree is not affected.
- Clearly this new tree T' has a smaller ABL than T . Contradiction.



Optimal Prefix Codes: False Start

Q. Where should letters be placed with a high frequency in the tree of an optimal prefix code ?

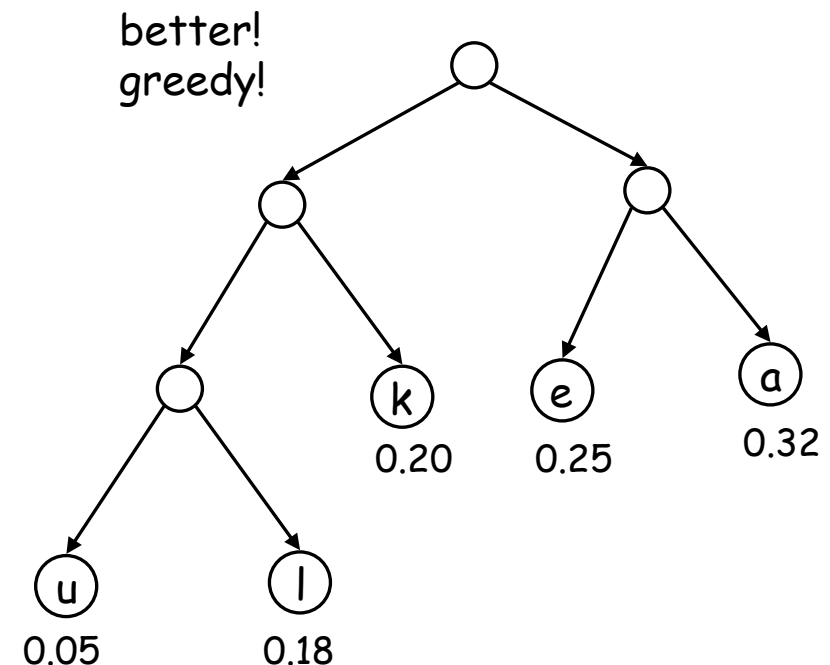
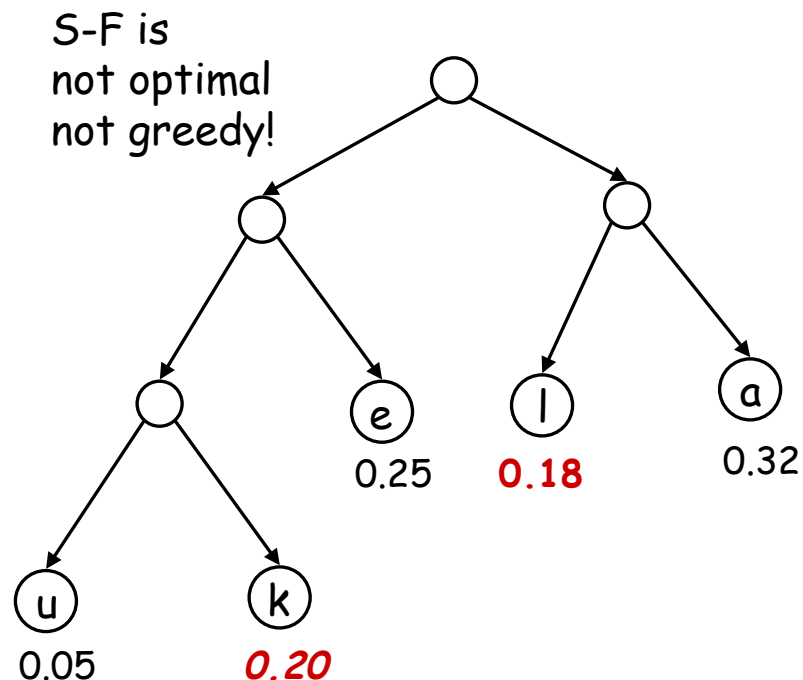
Optimal Prefix Codes: False Start

Q. Where in the tree of an optimal prefix code should letters be placed with a high frequency?

A. Near the top! Use recursive structure of trees.

Greedy template. Create tree **top-down**, split **S** into two sets **S₁** and **S₂** with (almost) **equal frequencies**. Recursively build tree for **S₁** and **S₂**.

[Shannon-Fano, 1949] $f_a=0.32$, $f_e=0.25$, $f_k=0.20$, $f_l=0.18$, $f_u=0.05$



Optimal Prefix Codes: Huffman Encoding

Observation 1. *Lowest frequency items* should be at the *lowest level* in tree of optimal prefix code.

Observation 2. For $n > 1$, the lowest level always contains **at least two leaves** (optimal trees are **full!**).

Observation 3. The **order** in which items appear in a level does not matter.

Claim 1. There is an optimal prefix code with tree T^* where the **two lowest-frequency letters** are assigned to leaves that are brothers in T^* .



Huffman Code

Greedy template. [Huffman, 1952]

Create tree **bottom-up**.

- a) Make **two leaves** for **two lowest-frequency** letters **y** and **z**.
- b) **Recursively** build tree for the rest using a **meta-letter** for **yz**.

Optimal Prefix Codes: Huffman Encoding

```
Huffman(S) {  
  if |S|=2 {  
    return tree with root and 2 leaves  
  } else {  
    let y and z be lowest-frequency letters in S  
    S' = S  
    remove y and z from S'  
    insert new letter  $\omega$  in S' with  $f_{\omega}=f_y+f_z$   
    T' = Huffman(S')  
    T = add two children y and z to leaf  $\omega$  from T'  
    return T  
  }  
}
```

Q. What is the *time complexity*?

Optimal Prefix Codes: Huffman Encoding

```
Huffman(S) {  
  if |S|=2 {  
    return tree with root and 2 leaves  
  } else {  
    let y and z be lowest-frequency letters in S  
    S' = S  
    remove y and z from S'  
    insert new letter  $\omega$  in S' with  $f_{\omega}=f_y+f_z$   
    T' = Huffman(S')  
    T = add two children y and z to leaf  $\omega$  from T'  
    return T  
  }  
}
```

Q. What is the time complexity?

A. $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

Q. How to implement finding *lowest-frequency letters* efficiently?

A. Use *priority queue* for S : $T(n) = T(n-1) + O(\log n) \rightarrow O(n \log n)$

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.

Pf. by induction, based on optimality of T' (y and z removed, w added)
(see next page)

Claim. $ABL(T') = ABL(T) - f_w$

Pf.

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.

Proof. by induction, based on optimality of T' (y and z removed, ω added)

(see next page)

Claim. $ABL(T') = ABL(T) - f_\omega$

Proof.

$$\begin{aligned} ABL(T) &= \sum_{x \in S} f_x \cdot \text{depth}_T(x) \\ &= f_y \cdot \text{depth}_T(y) + f_z \cdot \text{depth}_T(z) + \sum_{x \in S, x \neq y, z} f_x \cdot \text{depth}_T(x) \\ &= (f_y + f_z) \cdot (1 + \text{depth}_T(\omega)) + \sum_{x \in S, x \neq y, z} f_x \cdot \text{depth}_T(x) \\ &= f_\omega \cdot (1 + \text{depth}_T(\omega)) + \sum_{x \in S, x \neq y, z} f_x \cdot \text{depth}_T(x) \\ &= f_\omega + \sum_{x \in S'} f_x \cdot \text{depth}_{T'}(x) \\ &= f_\omega + ABL(T') \end{aligned}$$

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.

Prooff. (by induction over $n=|S|$)

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.

Pf. (by induction over $n=|S|$)

Base: For $n=2$ there is no *shorter* code than root and two leaves.

Hypothesis: Suppose Huffman tree T' for S' of size $n-1$ with ω instead of y and z is optimal.

Step: (by contradiction)

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.

Pf. (by induction)

Base: For $n=2$ there is no shorter code than root and two leaves.

Hypothesis: Suppose Huffman tree T' for S' of size $n-1$ with ω instead of y and z is optimal. (IH)

Step: (by contradiction)

- *Idea of proof:*

- Suppose other tree Z of size n is better.
- Delete lowest frequency items y and z from Z creating Z'
- Z' cannot be better than T' by IH.

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.

Pf. (by induction)

Base: For $n=2$ there is no shorter code than root and two leaves.

Hypothesis: Suppose Huffman tree T' for S' with ω instead of y and z is **optimal**. (Inductive Hyp.)

Step: (by contradiction)

- Suppose Huffman tree T for S is not optimal.
- So there is some tree Z such that $ABL(Z) < ABL(T)$.
- Then there is also a tree Z for which leaves y and z exist that are **brothers** and have the **lowest frequency** (see Claim 1).
- Let Z' be Z with y and z deleted, and their **former parent** labeled ω .
- Similar T' is derived from S' in our algorithm.
- We know that $ABL(Z') = ABL(Z) - f_\omega$, as well as $ABL(T') = ABL(T) - f_\omega$.
- But also $ABL(Z) < ABL(T) \implies ABL(Z') < ABL(T')$.
- Contradiction with IH.

Steps of the Proof

Step: (by contradiction)

- Suppose Huffman tree T for S is not optimal.
- So there is some tree Z such that $ABL(Z) < ABL(T)$.
- Then there is also a tree Z for which leaves y and z exist that are brothers and have the lowest frequency (see Obs. 1-2: fullness!).
- Let Z' be Z with y and z deleted, and their former parent labeled ω .
- Similar T' is derived from S' in our algorithm.
- We know that $ABL(Z') = ABL(Z) - f_{\omega}$, as well as $ABL(T') = ABL(T) - f_{\omega}$.
- But also (Absurd Hyp) $ABL(Z) < ABL(T)$, so $ABL(Z') < ABL(T')$.
- Contradiction with IND IH.