4.8 Huffman Codes

- Encoding in bits
  - Prefix codes
  - Tree representation
- Huffman codes:
  - Algorithm
  - Proof

Encoding in bits

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. Encode 2^5 different symbols using 5 bits per symbol.
This is called fixed length encoding.

Ex. c(a) = 00000
c(b) = 00001
c(e) = 00100

What is 0000000100?

TU Delft

Data Compression

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.

Ex. c(a) = 01 What is 0101?
c(b) = 010
c(e) = 1

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

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

TU Delft

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) = 01
c(b) = 010
c(e) = 1

Q. Is this a prefix code?

A. No, c(a) is a prefix of c(b).

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

Q. Is this a prefix code?

A. Yes.

Q. What is the meaning of 1001000001 ?

A. "leuk"

TU Delft

Towards Optimal Prefix Codes

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

Optimal code: small representation (on average)

Suppose frequencies are known in a text:

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

Q. What is the *average* size in bits of letters in this text given c() above?

A. $2*f_a + 2*f_e + 3*f_k + 2*f_l + 3*f_u = 2.3$

Definition. The average bits per letter of a prefix code c is the sum over all symbols x in S of its frequency f_x times the number of bits of its encoding:

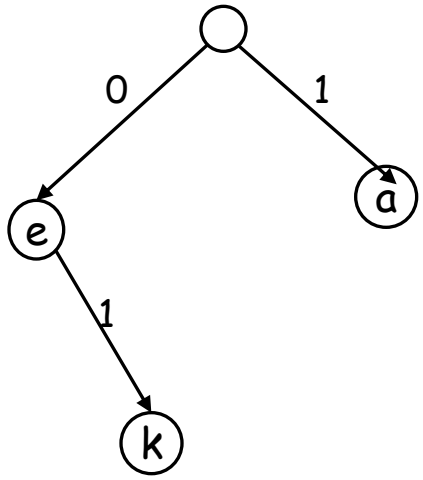
$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

We would like to find a prefix code that has the lowest possible average bits per letter.

TU Delft

Representing Prefix Codes using Binary Trees

Ex. c(a) = 1
c(e) = 0
c(k) = 01



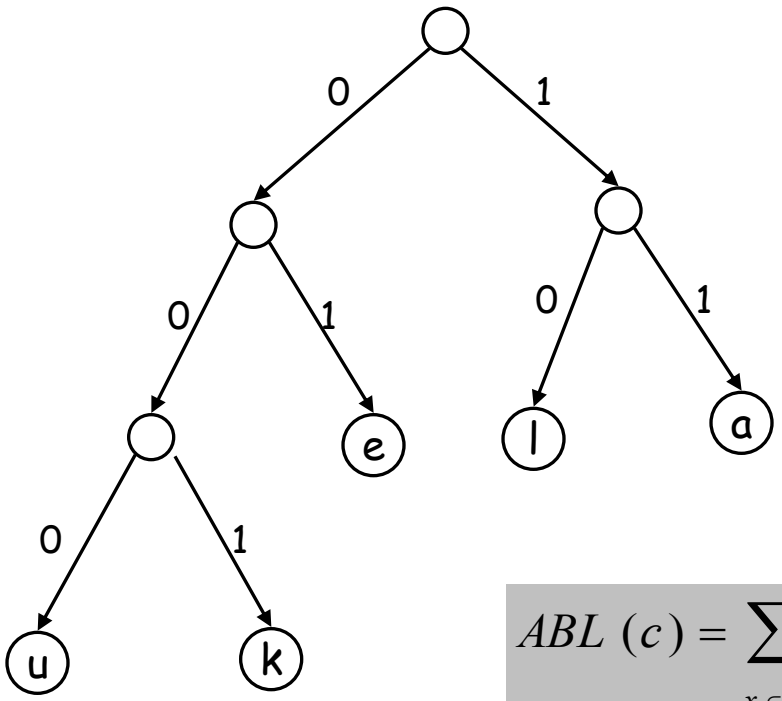
depth of x

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

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

Representing Prefix Codes using Binary Trees

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

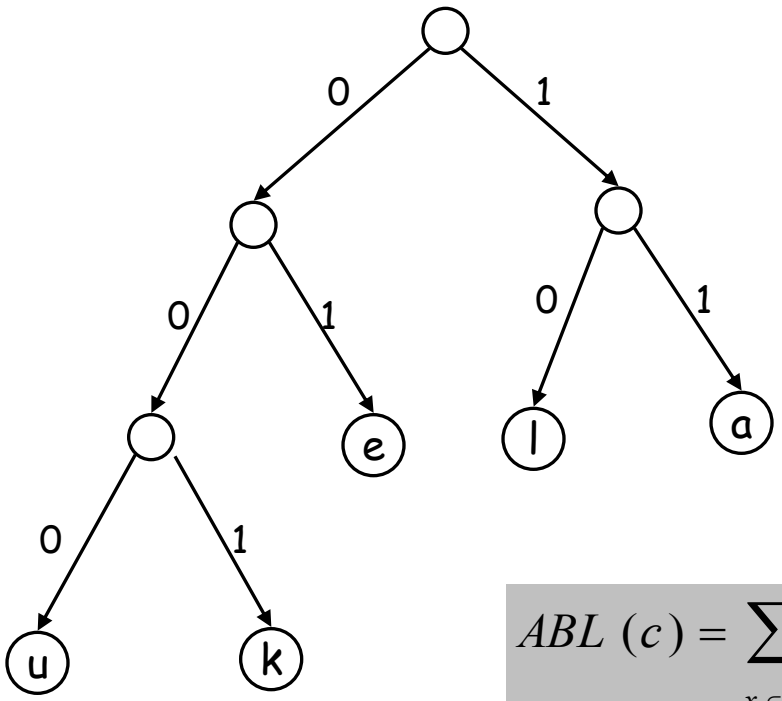


$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

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

Representing Prefix Codes using Binary Trees

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



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

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

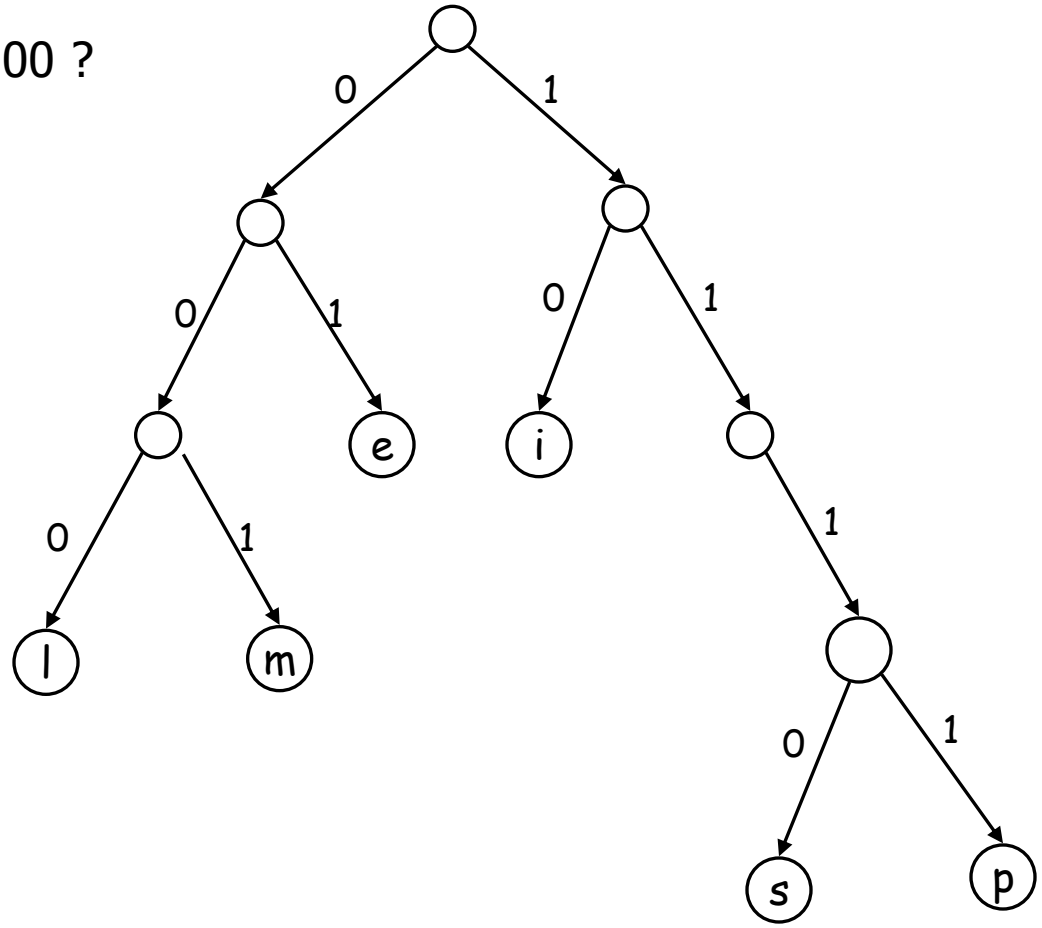
A. Only the leaves have a label.

Pf. An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.

Representing Prefix Codes using Binary Trees

Q. What is the meaning of

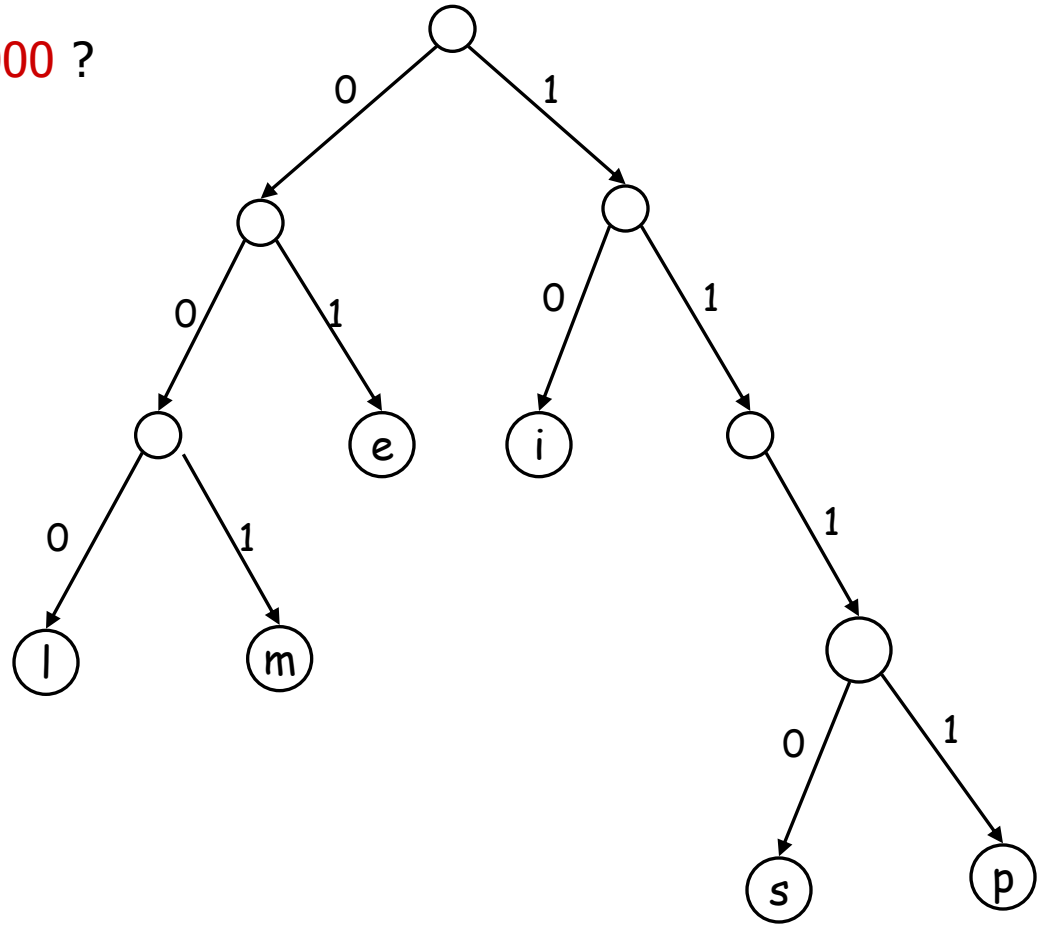
11101000111101000 ?



Representing Prefix Codes using Binary Trees

Q. What is the meaning of

11101000111101000 ?



Representing Prefix Codes using Binary Trees

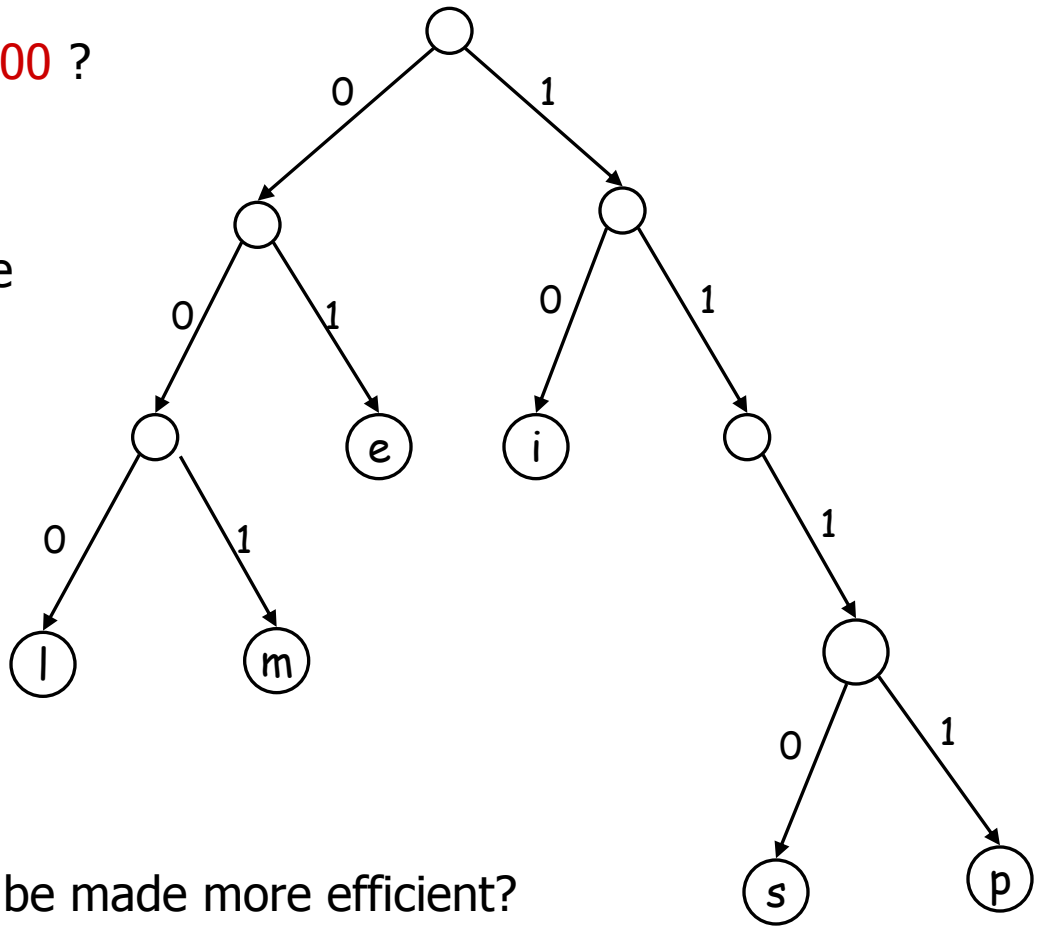
Q. What is the meaning of

11101000111101000 ?

A. "simpel"

Efficiency in terms of the tree representation is defined by

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



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

Representing Prefix Codes using Binary Trees

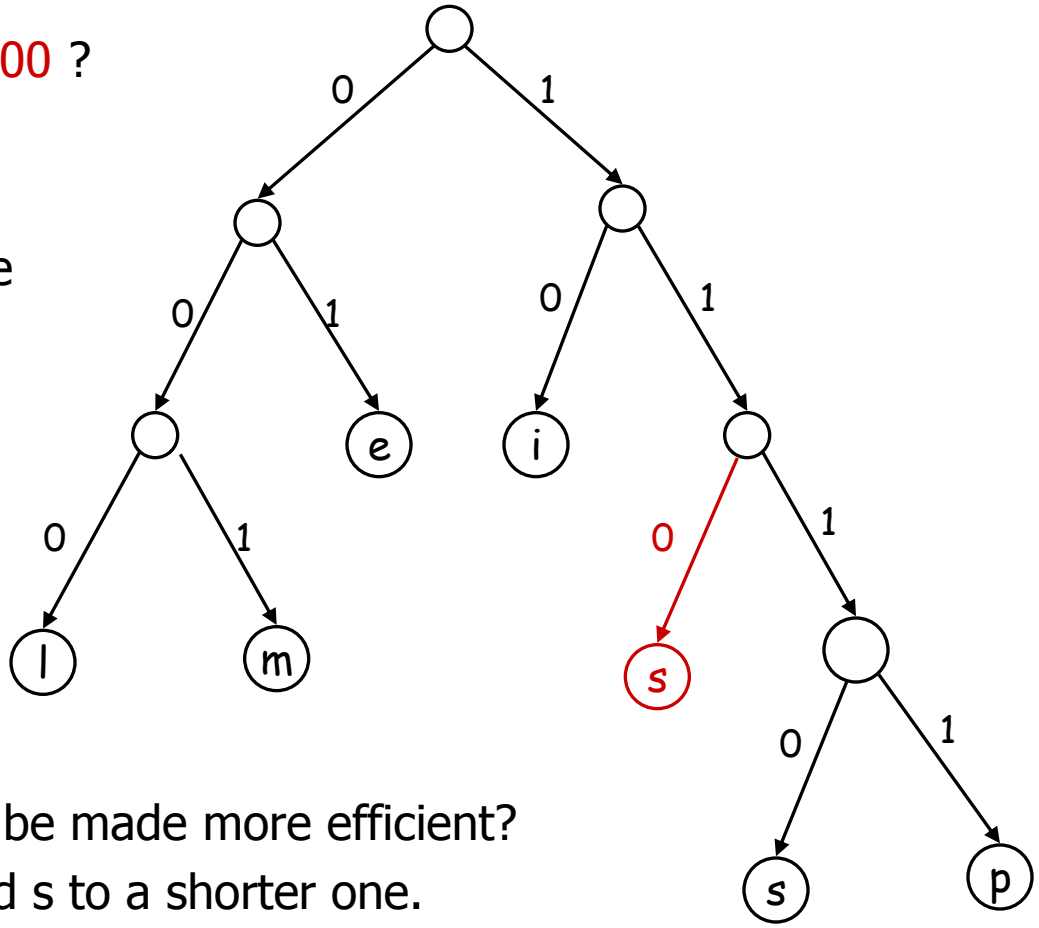
Q. What is the meaning of

11101000111101000 ?

A. "simpel"

Efficiency in terms of the tree representation is defined by

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



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

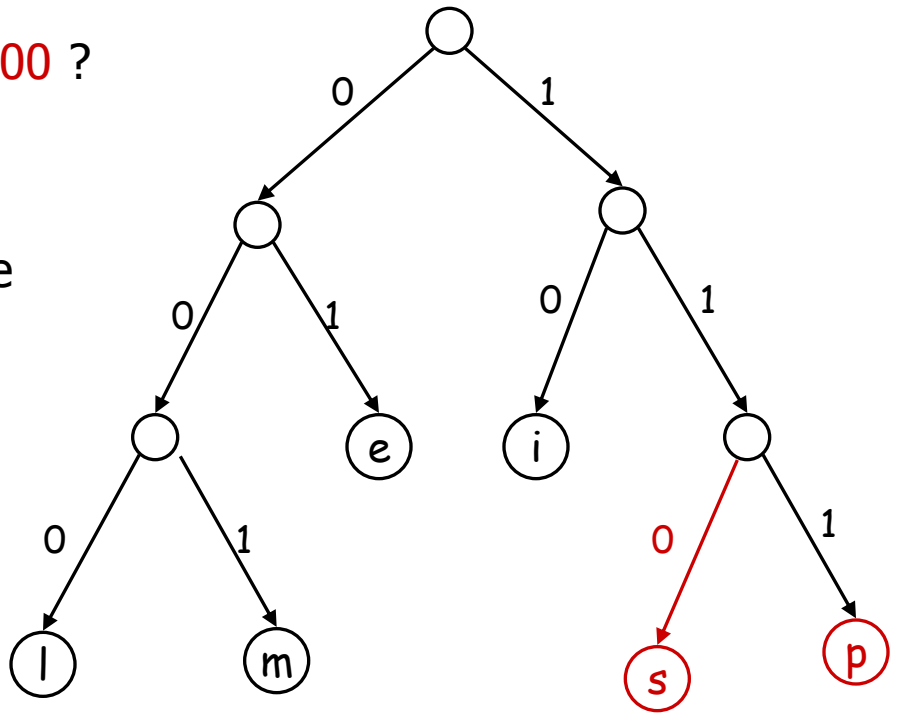
Q. What is the meaning of

11101000111101000 ?

A. "simpel"

Efficiency in terms of the tree representation is defined by

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



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.

remember: only leaves have a label!

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

Pf.

Representing Prefix Codes using Binary Trees

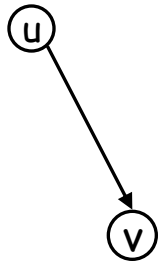
Definition. A tree is full if every node that is not a leaf has two children.

remember: only leaves have a label!

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

Pf. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.
- Thus there is a node u (without label) with only one child v.



- Clearly this new tree T' has a smaller ABL than T. Contradiction.

Representing Prefix Codes using Binary Trees

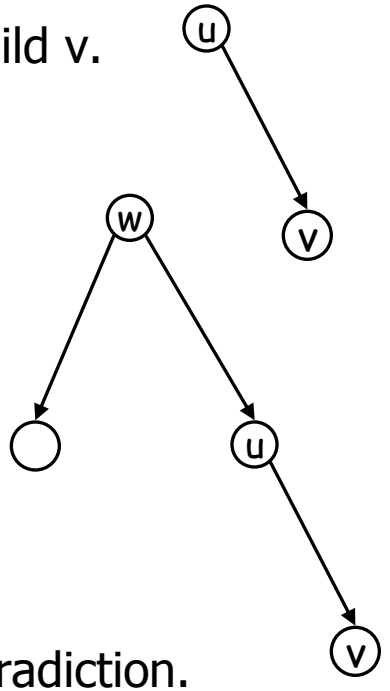
Definition. A tree is full if every node that is not a leaf has two children.

remember: only leaves have a label!

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

Pf. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.
- Thus there is a node u (without label) with only one child v.
- Case 1: u is the root



- Case 2: u is not the root

- Clearly this new tree T' has a smaller ABL than T. Contradiction.

Representing Prefix Codes using Binary Trees

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

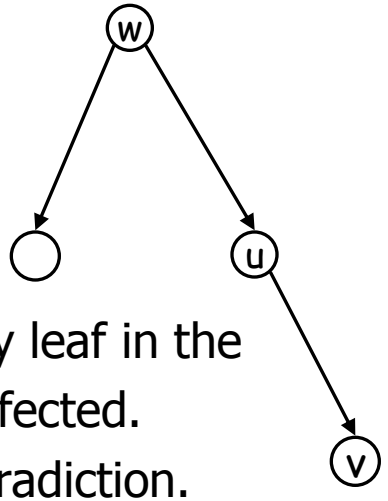
remember: only leaves have a label!

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

Pf. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.
- Thus there is a node u (without label) with only one child v.
- Case 1: u is the root; create T': delete u and use v as the root

- Case 2: u is not the root; create T':
  - 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.

This proof can be found on pages 168-169.

Optimal Prefix Codes: False Start

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

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.

Q. How to create an optimal prefix code (tree) in a greedy way? (1 min)

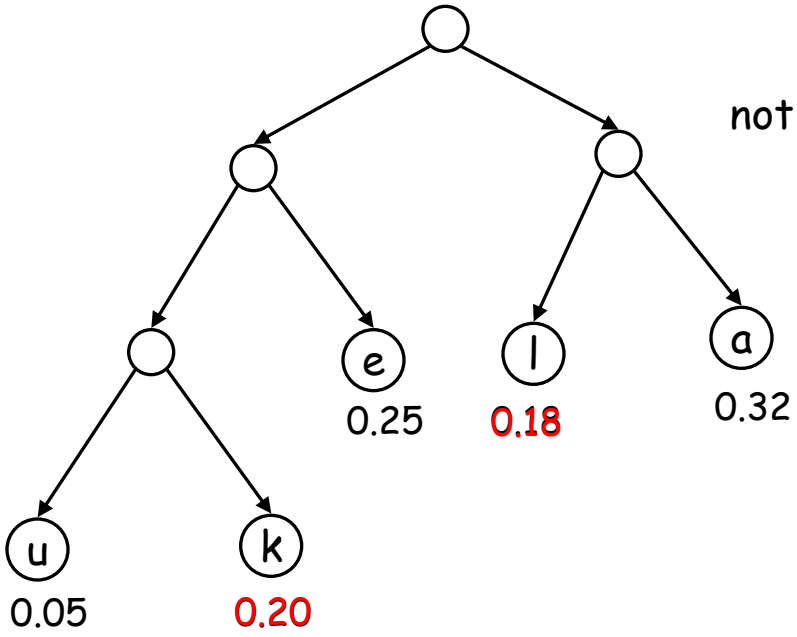
Optimal Prefix Codes: False Start

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

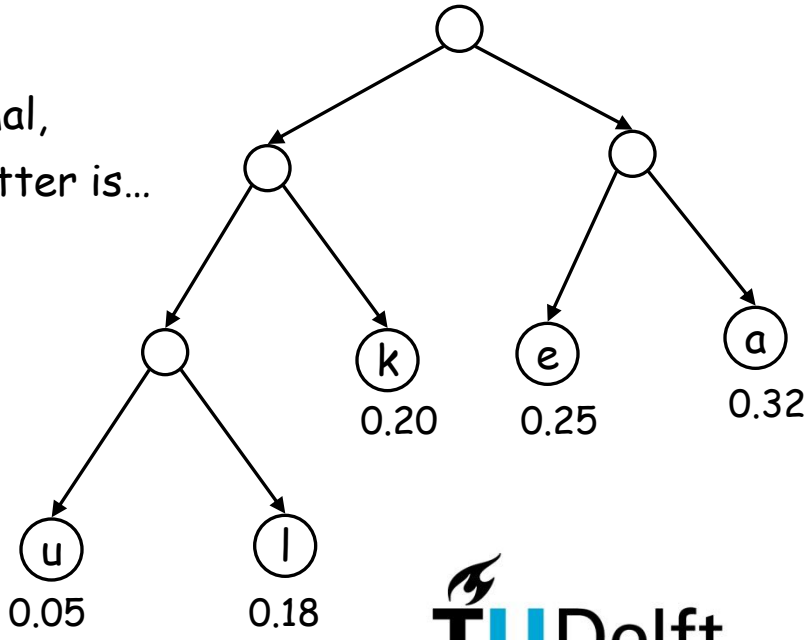
A. Near the top.

Greedy template (1). Create tree top-down, split S into two sets S_1 and S_2 with (almost) equal frequencies. Recursively build tree for S_1 and S_2.

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



not optimal,
better is...



TU Delft

Optimal Prefix Codes: Huffman Encoding

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

Observation. For n > 1, the lowest level always contains at least two leaves.

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

Q. How to find an *optimal* prefix code?

Optimal Prefix Codes: Huffman Encoding

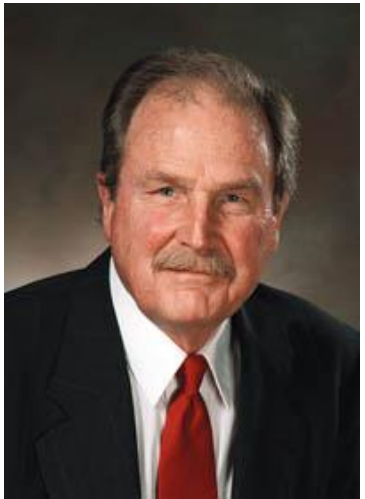
Observation. Lowest frequency items should be at the lowest level in tree of optimal prefix code.

Observation. For n > 1, the lowest level always contains at least two leaves.

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

Claim (Siblings). There is an optimal prefix code with tree T* where the two lowest-frequency letters are assigned to leaves that are siblings in T*.

Greedy template (2). Create tree bottom-up.
Make two leaves for two lowest-frequency letters y and z.
Recursively build tree for the rest using a meta-letter for yz.
[Huffman, 1952]



http://en.wikipedia.org/wiki/David_A._Huffman

Optimal Prefix Codes: Huffman Encoding

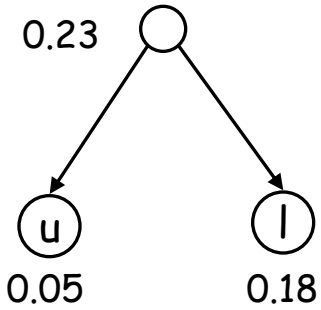
Greedy template (2). Create tree bottom-up.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

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

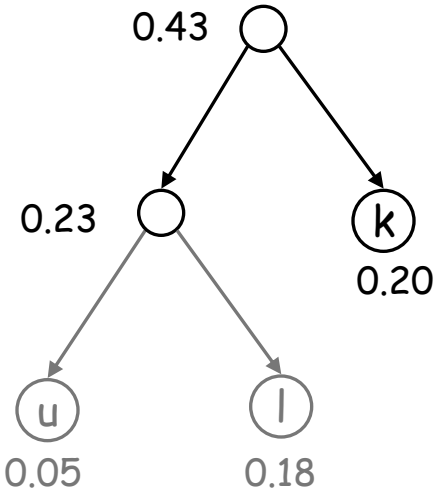
Greedy template (2). Create tree bottom-up.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

f_a=0.32, f_e=0.25, f_k=0.20, f_ω=0.23



Optimal Prefix Codes: Huffman Encoding

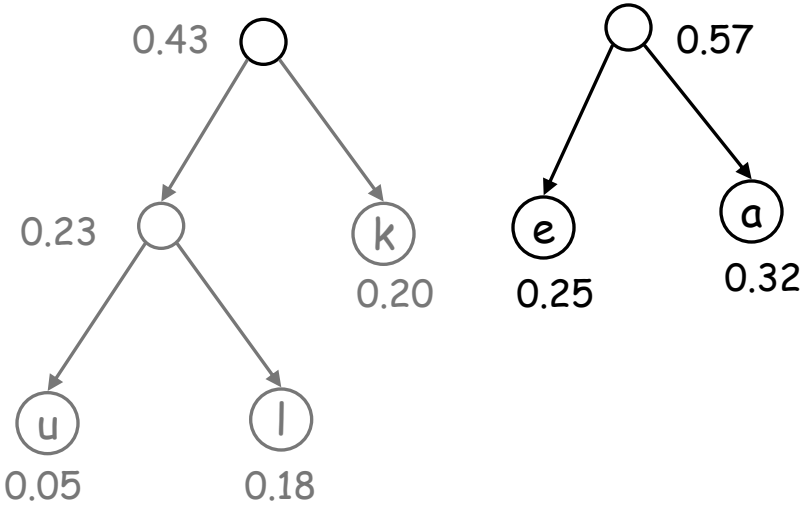
Greedy template (2). Create tree bottom-up.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

f_a=0.32, f_e=0.25, f_ω =0.43



Optimal Prefix Codes: Huffman Encoding

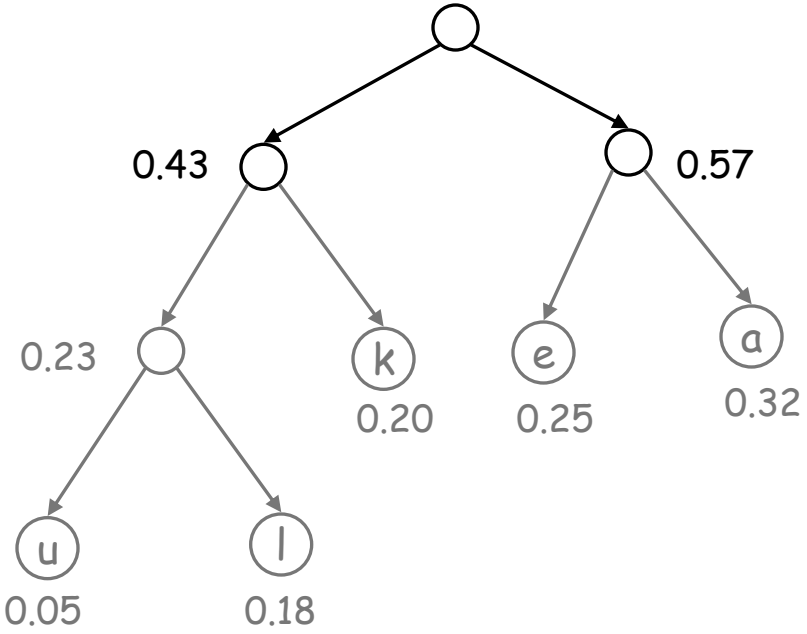
Greedy template (2). Create tree bottom-up.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

$f_{\omega 1} = 0.43$ $f_{\omega 2} = 0.57$



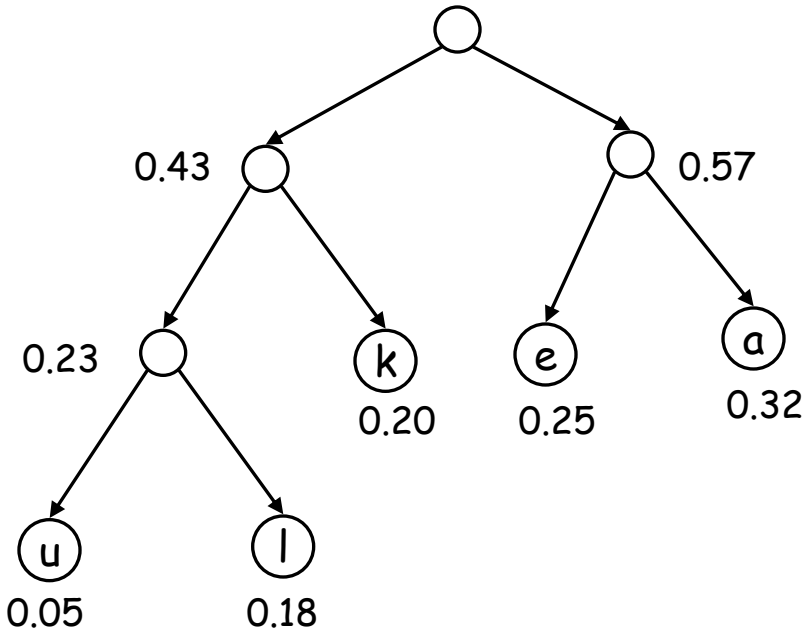
Optimal Prefix Codes: Huffman Encoding

Greedy template (2). Create tree bottom-up.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]



TU Delft

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 ω in S′ with f_ω=f_y+f_z
        T′ = Huffman(S′)
        T = add two children y and z to leaf ω from T′
        return T
    }
}
```

O(1)

O(log*n*)

O(1)
O(log*n*)
T(*n*-1)
O(1)

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

A. Use priority queue for S

Q. What is the time complexity?

A. T(n) = T(n-1) + O(log n) so O(n log n)

Huffman Encoding: Greedy Analysis

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

Pf.

Q. Which proof technique to use?

TU Delft

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. (IH)

Step:

Q. Which proof technique to use?

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. (IH)

Step: (by contradiction)

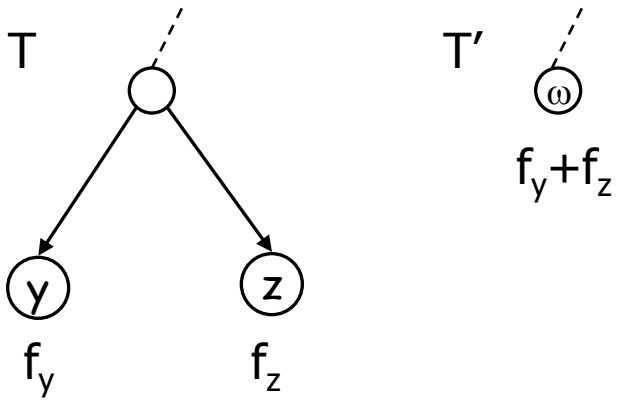
- *Idea of proof:*
  - Suppose other tree Z of size n is better (i.e. smaller ABL).
  - Delete lowest frequency items y and z from Z creating Z′
  - Idem for T and T′.
  - … Q. What is ABL(T′) compared to ABL(T)? => separate proof
  - Then ABL(Z′) < ABL(T′). Contradiction with IH.

Huffman Encoding: Greedy Analysis

Claim. ABL(T')=ABL(T)-f_ω when T' is T with y and z replaced by ω

Pf.

$$
\begin{aligned}
\mathrm{ABL}(T) &= \sum_{x \in S} f_x \cdot \mathrm{depth}_T(x) \\
&= \\
&= \\
&= \\
&= f_\omega + \sum_{x \in S'} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega + \mathrm{ABL}(T')
\end{aligned}
$$



Huffman Encoding: Greedy Analysis

Claim. ABL(T')=ABL(T)-f_ω when T' is T with y and z replaced by ω

Pf.

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

This proof can be found on page 174.

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. (IH)

Step: (by contradiction)

- *Idea of proof:*
  - Suppose other tree Z of size n is better (i.e. smaller ABL).
  - Delete lowest frequency items y and z from Z creating Z′
  - Idem for T and T′.
  - Compute ABL(T′) and ABL(Z′)
  - Then ABL(Z′) < ABL(T′). Contradiction with IH.

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. (IH)

Step: (by contradiction)

- Suppose Huffman tree T for S is not optimal.
- So there is some tree Z such that ABL(Z) < ABL(T).

- Contradiction with IH.

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. (IH)

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 siblings and have the lowest frequency (using Siblings Claim).
- Let Z' be Z with y and z deleted, and their former parent labeled ω.
- Similarly T' is derived for S' (with ω instead of y and z) by algorithm.

- Contradiction with IH.

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. (IH)

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 siblings and have the lowest frequency (using Siblings Claim).
- Let Z' be Z with y and z deleted, and their former parent labeled ω.
- Similarly T' is derived for S' (with ω instead of y and z) by algorithm.
- We know that ABL(Z')=ABL(Z)-f_ω, as well as ABL(T')=ABL(T)-f_ω.
- But also ABL(Z) < ABL(T), so ABL(Z') < ABL(T').
- Contradiction with IH.

This proof can be found on pages 174-175.

TU Delft