4.8 Huffman Codes

These lecture slides are supplied by Mathijs de Weerd

Optimal Prefix Codes

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

(its frequency) **x** (the number of bits of its encoding):

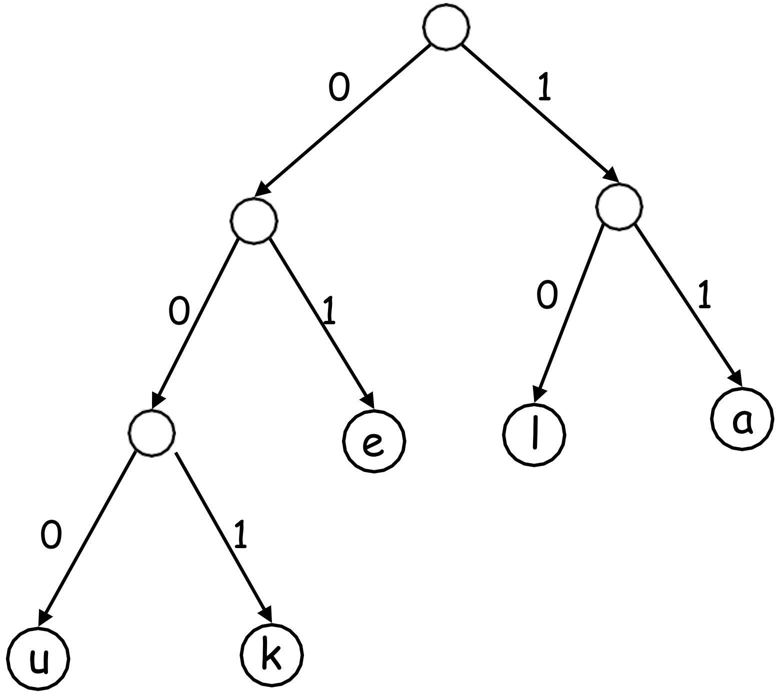
$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

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

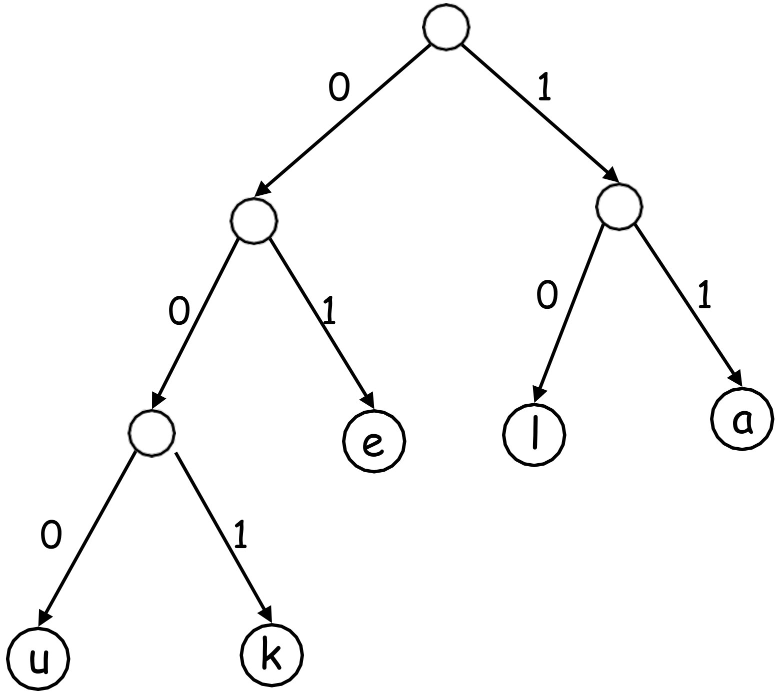
Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000



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



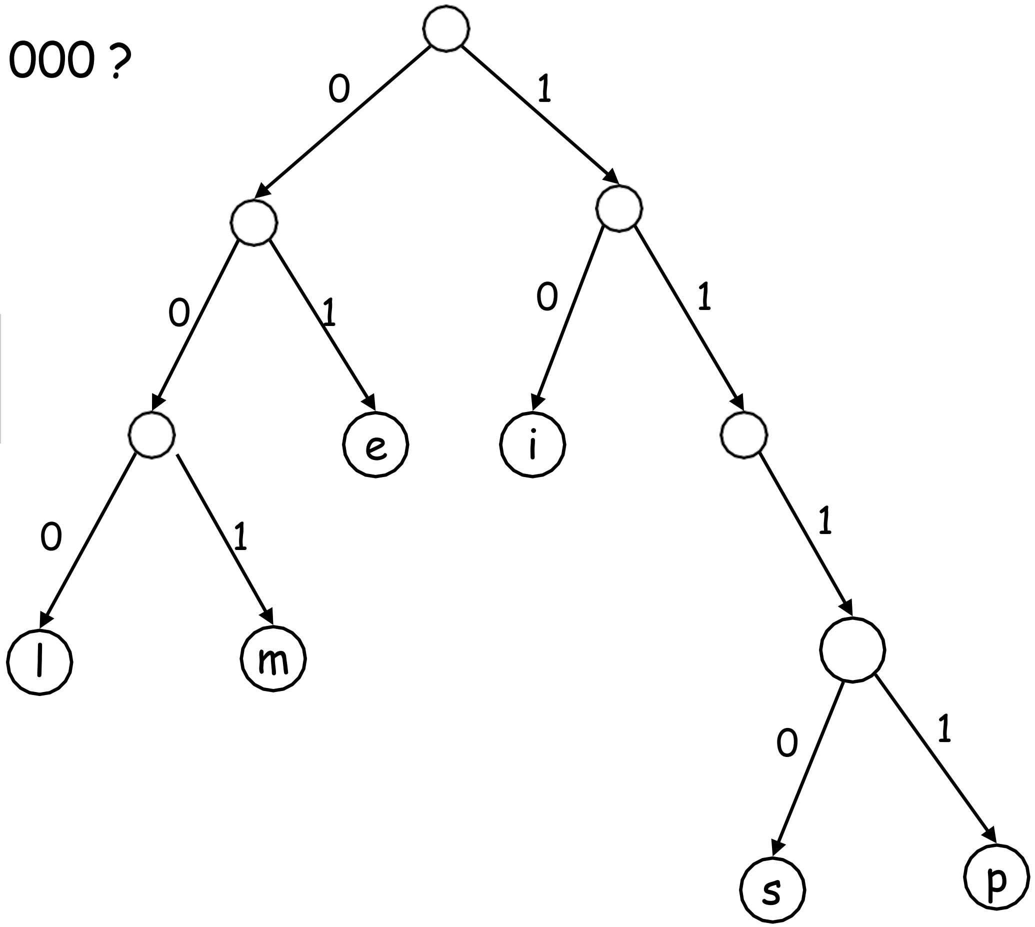
Q. How does the **tree** of a **prefix code** look?

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 ?

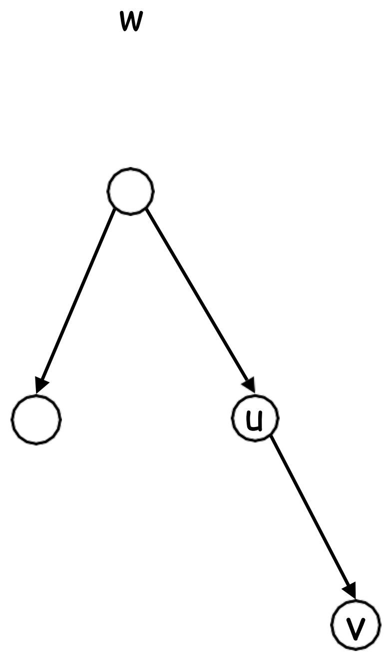


$$ABL(T) = \sum_{x \in S} f_x \cdot \mathrm{depth}_T(x)$$

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**.



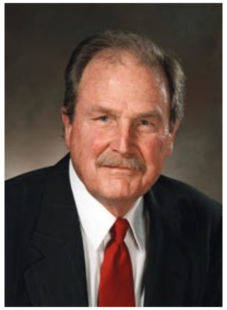
Optimal Prefix Codes: Huffman Encoding

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 the tree **bottom-up**.

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

b) **Recursively** build tree for the rest: replacing **y** and **z** with

a **meta-letter** for ***yz***. with frequency $f_{yz} = f_y + f_z$

c) **Consider** the new alphabet $S' = S - \{y,z\} + \{yz\}$

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 ω in S' with fω=fy+fz
      T' = Huffman(S')
      T = add two children y and z to leaf ω from T'
      return T
   }
}
```

Q. What is the time complexity?

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

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

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

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' where
y and z removed, ω=yz added (see next page)

Claim. **ABL(T') = ABL(T) - f_ω**
Proof.

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

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.
Ind. Hypothesis: Huffman tree *T'* for **S'** with ω instead of **y** and **z** is **optimal:**

ABL(T') <= ABL(Z'), for any feasible Z' for S'

Step: (by contradiction)

- Suppose Huffman tree T for S is **not** optimal.
- So there is some tree Z_1 such that **ABL(Z_1) < ABL(T)**.
- Then there is also a tree Z for which leaves *y* and *z* exist that are **brothers** and have the **lowest level** **and ABL(Z_1) < ABL(T) (see Claim 1)**.
- Let Z' **obtained** from Z with and **z and y** deleted, and their **former parent** labeled ω.
- Similar T' is derived from **S'** in our 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)* --> ABL(Z') < ABL(T')**.

TEST DI AUTO-VERIFICA

Rispondere in modo rigoroso a queste domande/esercizi:

- Data una istanza generica del problema minimum prefix code, cosa si deve minimizzare? A cosa corrisponde in termini di Labeled Trees.
- Quali sono le proprietà fondamentali di un Labeled Tree corrispondente ad un prefix code ottimale?
- Eseguire passo per passo (facendo la traccia di ogni variabile) l'algoritmo di Huffman su diverse istanze di almeno 8 simboli e capire bene la struttura ricorsiva vedendo quando si riassegnano i sottoalberi corrispondenti ai metasimboli costruiti
- Individuare in ogni passo della dimostrazione di ottimalità quali proprietà specifiche della struttura ottimale si sta usando e quando viene applicata l'ipotesi induttiva sull'istanza di dimensione inferiore