南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

http://faculty.sustech.edu.cn/yusq/

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Greedy Algorithms

Encoding and Huffman

Slides are supplied by Mathijs de Weerd

Encoding

- Why we need encoding?
- Difference between human beings and computers
- Post code/the address system in China

Encoding

- An example
  - ➢ Pay a gold bullion to a worker who will work 7 days for you
  - ➢ You should pay him per day
  - ➢ The gold bullion can only be cut two times

Encoding

- 64 samples (1 poison)
- How many guinea pigs at least to find the poison?

	?	?	?	?	?	?
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	0	0	1	0	0
…						
63	1	1	1	1	1	1

Data Compression

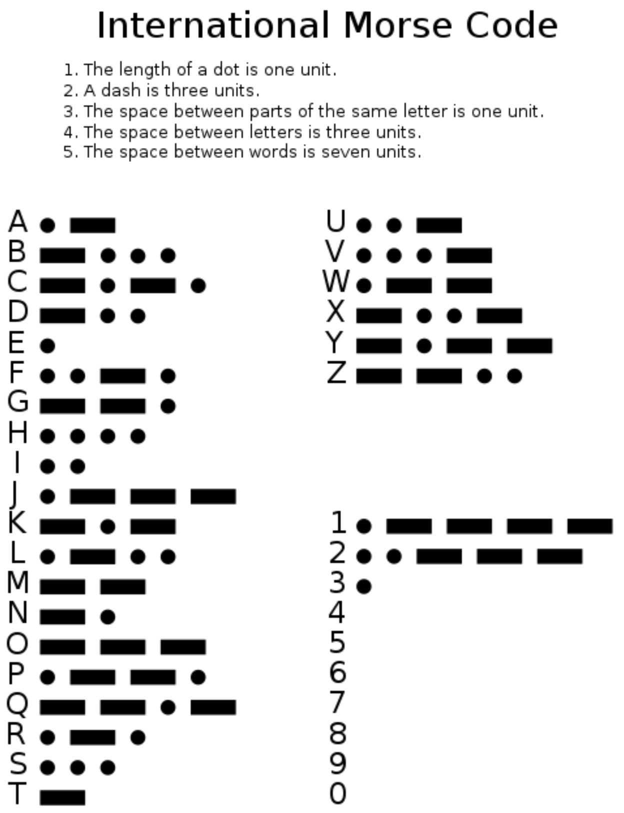
- Q. Given a text that uses 32 symbols (26 different letters, space, and some punctuation characters), how can we encode this text in bits?
- A. We can encode 2^5 different symbols using a fixed length of 5 bits per symbol. This is called fixed length encoding.

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.

Data Compression

- 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.



International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A • ▬
B ▬ • • •
C ▬ • ▬ •
D ▬ • •
E •
F • • ▬ •
G ▬ ▬ •
H • • • •
I • •
J • ▬ ▬ ▬
K ▬ • ▬
L • ▬ • •
M ▬ ▬
N ▬ •
O ▬ ▬ ▬
P • ▬ ▬ •
Q ▬ ▬ • ▬
R • ▬ •
S • • •
T ▬
U • • ▬
V • • • ▬
W • ▬ ▬
X ▬ • • ▬
Y ▬ • ▬ ▬
Z ▬ ▬ • •
1 • ▬ ▬ ▬ ▬
2 • • ▬ ▬ ▬
3 •
4
5
6
7
8
9
0

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

- What is 0101?

https://morsecode.world/international/translator.html

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:

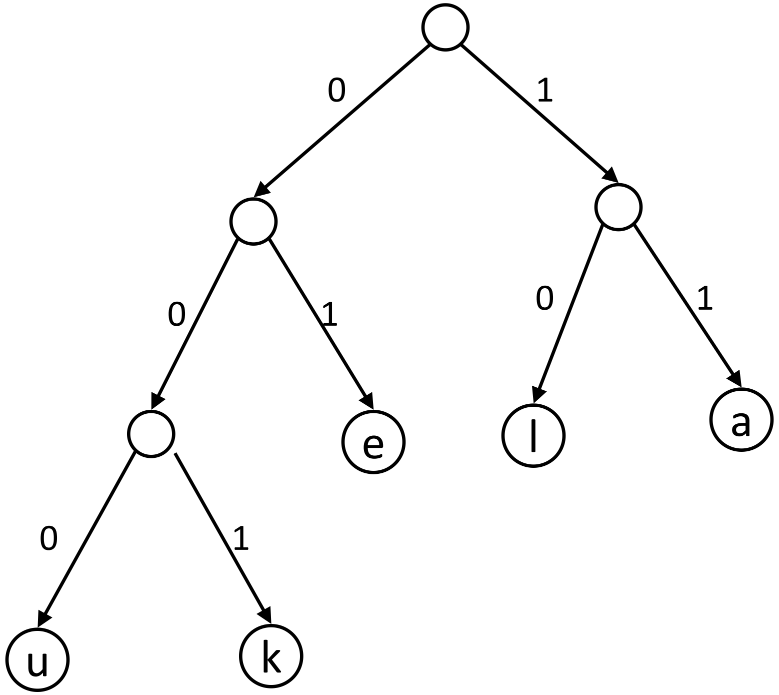
$$ABL(\gamma) = \sum_{x \in S} f_x |\gamma(x)|$$

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

- Suppose we model a code in a binary tree…

Representing Prefix Codes using Binary Trees

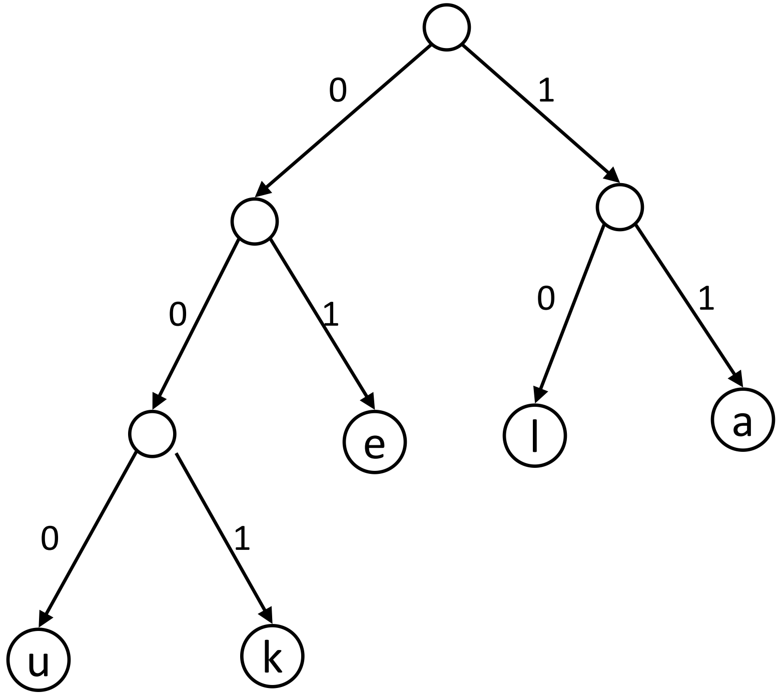
- Ex. c(a) = 11
- c(e) = 01
- c(k) = 001
- c(l) = 10
- c(u) = 000



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

Representing Prefix Codes using Binary Trees

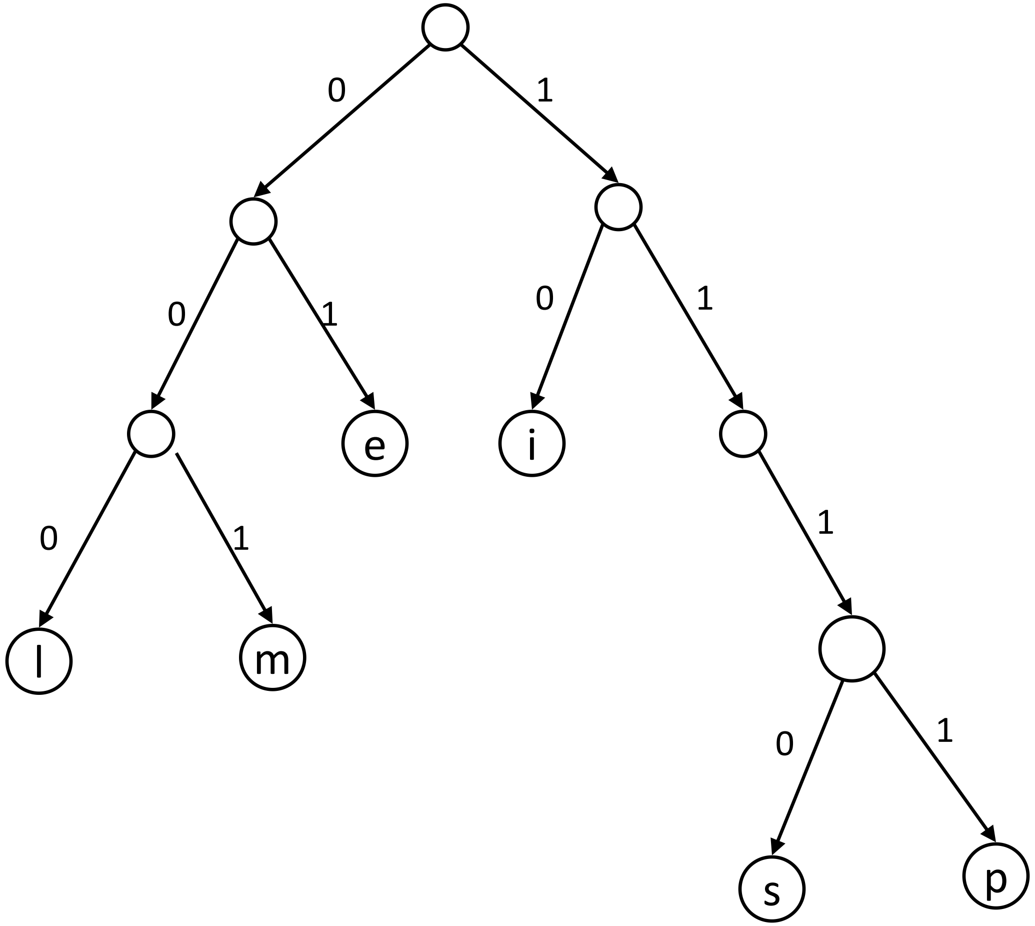
- Ex. c(a) = 11
- c(e) = 01
- c(k) = 001
- c(l) = 10
- c(u) = 000



- Q. How does the tree of a prefix code look?
- A. Only the leaves have a label.
- 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.

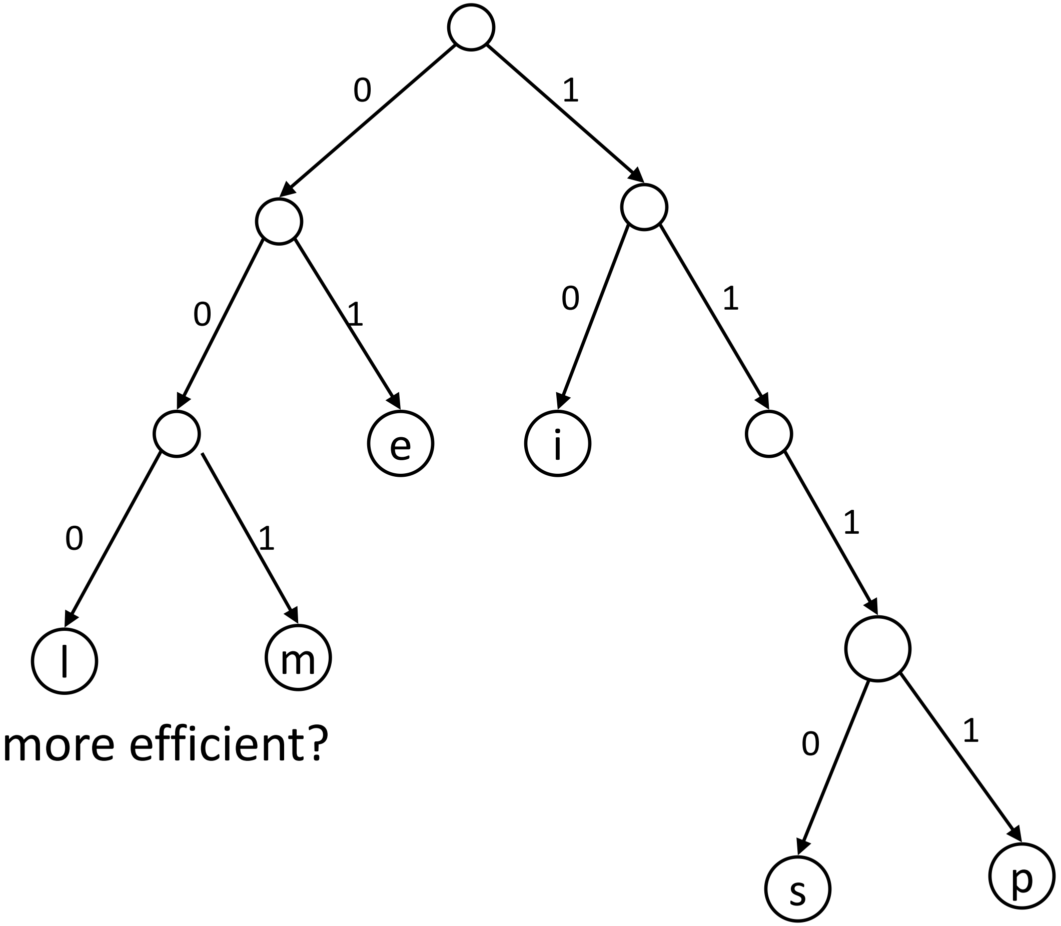
Representing Prefix Codes using Binary Trees

- Q. What is the meaning of
- 111010001111101000 ?



Representing Prefix Codes using Binary Trees

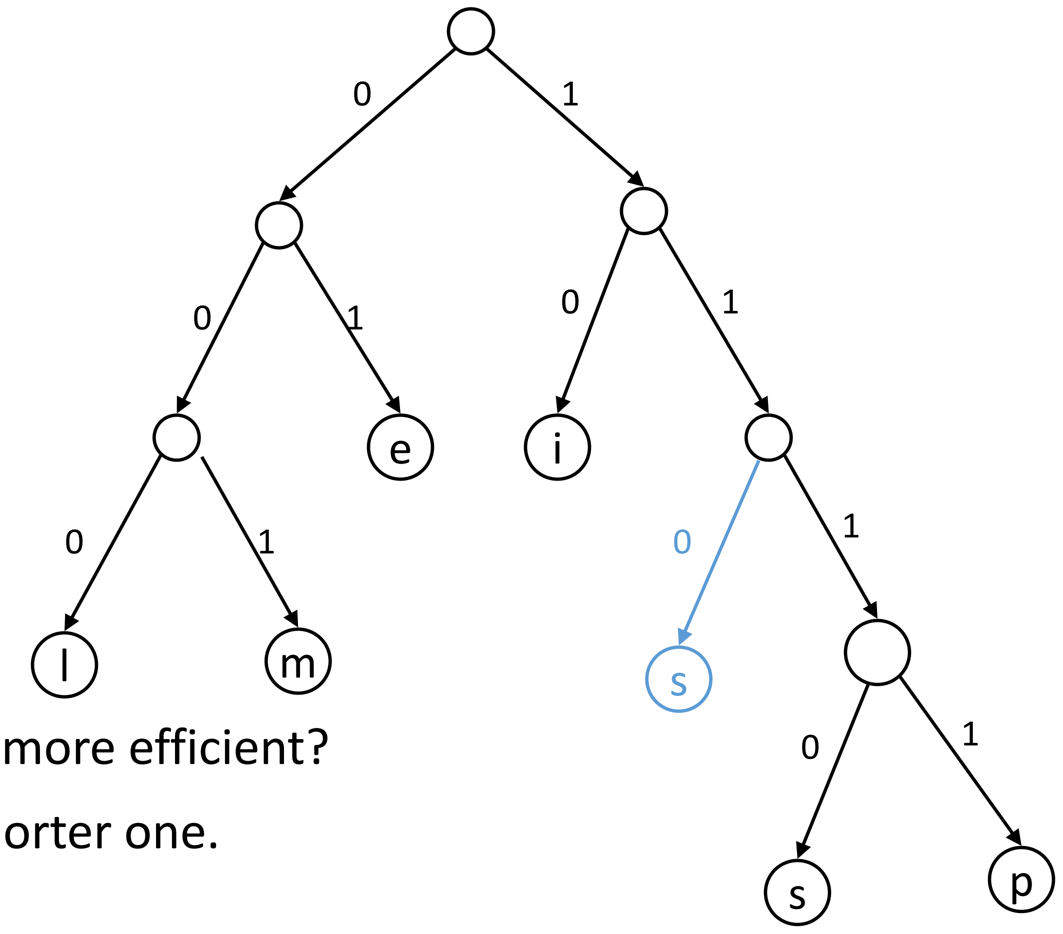
- Q. What is the meaning of
- 111010001111101000 ?
- A. "simpel"



- 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”

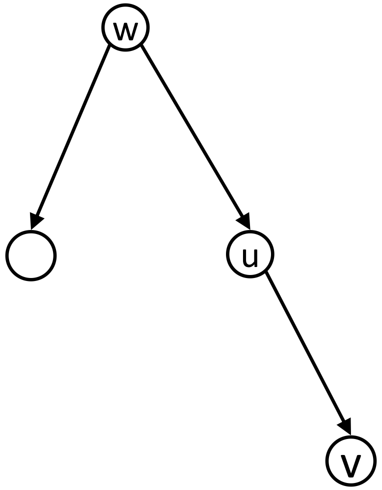


- Q. How can this prefix code be made more efficient?
- A. Change encoding of p and s to a shorter one.
- This tree is now full.

Representing Prefix Codes using Binary Trees

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

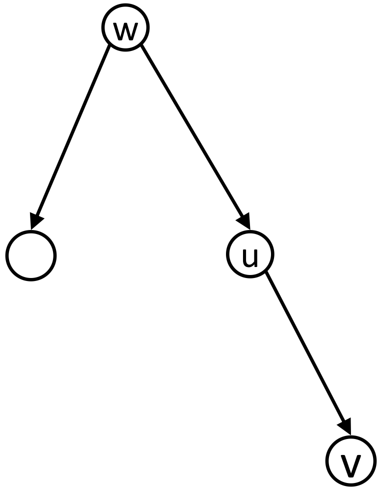
- Claim. The binary tree corresponding to the optimal prefix code is full.
- Pf.



Representing Prefix Codes using Binary Trees

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

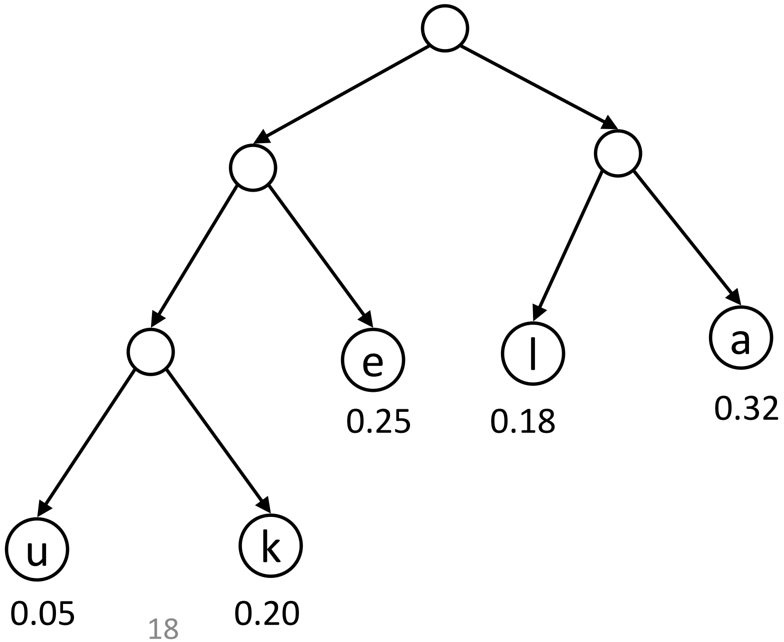
- Claim. The binary tree corresponding to the optimal prefix code is full.
- Pf. (by contradiction)
  - Suppose T is binary tree of optimal prefix code and is not full.
  - This means there is a node u with only one child v.
  - Case 1: u is the root; delete u and use v as the root
  - Case 2: u is not 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.

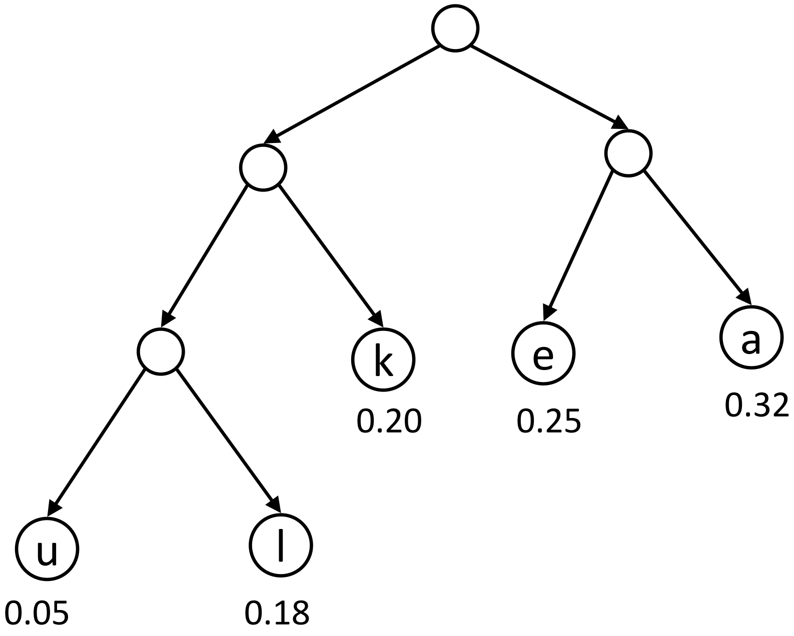


Optimal Prefix Codes: False Start

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

- Greedy template. 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 code, 1949] $f_a=0.32$, $f_e=0.25$, $f_k=0.20$, $f_l=0.18$, $f_u=0.05$





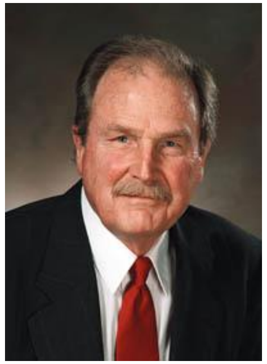
Optimal Prefix Codes: Huffman Encoding

- Observation. 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. 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. [Huffman, 1952] 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.

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 $f_\omega = f_y + f_z$
      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)
- so $O(n^2)$
- Q. How to implement finding lowest-frequency letters efficiently?
- A. Use priority queue for S: T(n) = T(n-1) + O(log n) so O(n log n)

Huffman Encoding: Greedy Analysis

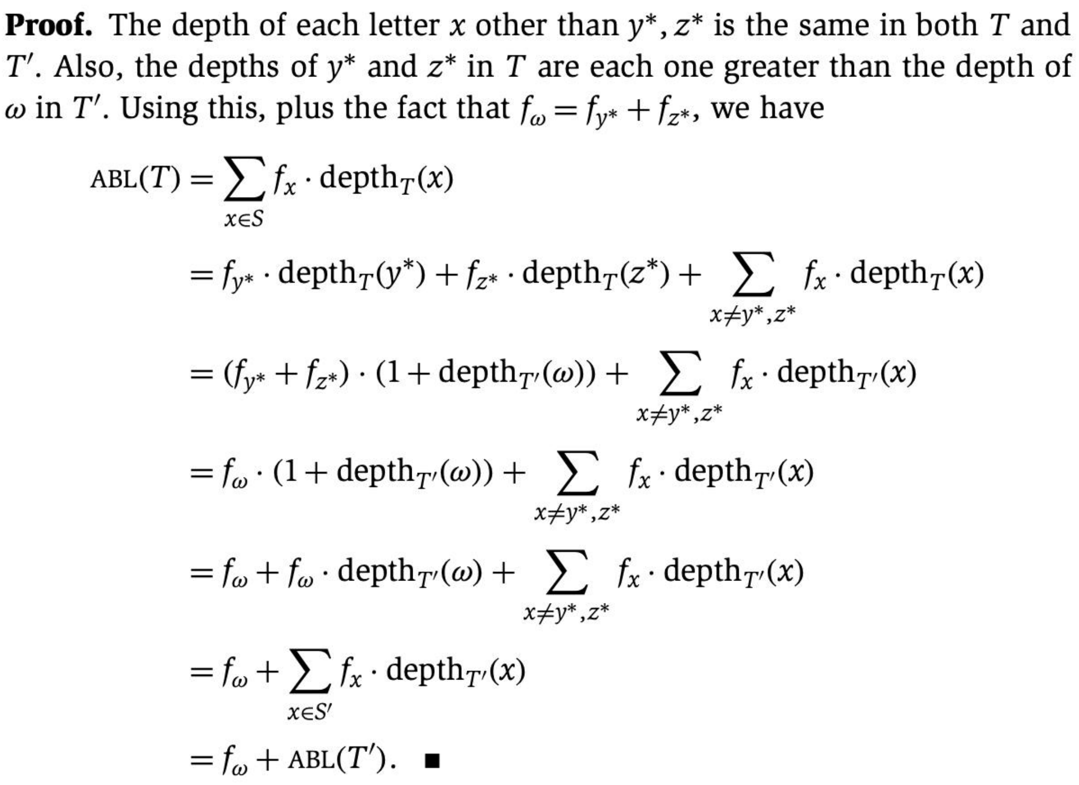
- Claim. Huffman code for S achieves the minimum ABL of any prefix code.
- Pf. by induction, based on optimality of T' (y and z removed, ω added)
- (see next page)

- Claim. ABL(T')=ABL(T)-f_{ω}
- Pf.

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, ω added)
- (see next page)

- Claim. ABL(T’)=ABL(T)-f_ω
- Pf.



Proof. The depth of each letter x other than y^*, z^* is the same in both T and T'. Also, the depths of y^* and z^* in T are each one greater than the depth of ω in T'. Using this, plus the fact that $f_\omega = f_{y^*} + f_{z^*}$, we have

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

Huffman Encoding: Greedy Analysis

- Claim. Huffman code for S achieves the minimum ABL of any prefix code.
- Pf. (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. (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 (see observation).
    - 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_{ω}, as well as ABL(T’)=ABL(T)-f_{ω}.
    - But also ABL(Z) < ABL(T), so ABL(Z’) < ABL(T’).
    - Contradiction with IH.

ZIP file format

- ZIP: an archive file format that supports lossless data compression.

- ZIP File Format Specification
  - ➢ https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.2.0.txt