COL106
Data Structures and Algorithms

Subodh Sharma and Rahul Garg

Announcements

Announcements

- Students are still copying in the assignments
- Need to face action

Announcements

- Lab examination 2
  - Sunday 29 October, 1-3pm and 3:30-5:30pm

Applications of Trees: Data Compression

Data Compression Problem

- Given a text corpus using 32 symbols
  - 26 alphabets
  - Space and punctuation characters
- How can we efficiently encode it in bits?

Data Compression Problem

- Given a text corpus using 32 symbols
  - 26 alphabets
  - Space and punctuation characters
- How can we efficiently encode it in bits?
- Solution 1: Use 8-bit ASCII codes to represent text
  - Number of bits needed: Number of characters in the corpus (N) * 8
- Can we do better?

Data Compression Problem

- Given a text corpus using 32 symbols
  - 26 alphabets
  - Space and punctuation characters
- How can we efficiently encode it in bits?
  - Solution 2: Since there are only 32 symbols, use a 5-bit code to represent the characters in the text

Data Compression using Fixed Length Code

Symbol	Code
a	00000
b	00001
c	00010
d	00011
e	00100
…	….
z	11000
<space>	11001
…	…

- Number of symbols: 32
- Need 5 bits to represent them
- Number of bits needed to store the corpus = N * 5
- This is called fixed length encoding
- Can we store the corpus using smaller number of bits?

Data Compression Problem

- Given a text corpus using 32 symbols
  - 26 alphabets
  - Space and punctuation characters
- How can we efficiently encode it in bits?
- Solution 2: Use a 5-bit code to represent the characters in the text ➔ 5N bits to store the corpus
- Solution 3: Some letters (e, t, a, o, i, n) appear more often than other letters (x, y, z).
  - Use less bits for more frequent symbols
  - More bits for less frequent symbols

Data Compression: Solution 3

- Use less bits for more frequent symbols
- More bits for less frequent symbols
- Problem: How do we know when a symbol ends and the next one begins?
- Ex. c(a) = 01
- c(b) = 010
- c(e) = 1

What is 0101?

b e = b e

= a a

a a

Data Compression: Solution 3

- Use less bits for more frequent symbols
- More bits for less frequent symbols
- Problem: How do we know when a symbol ends and the next one begins?
- Ex. c(a) = 01 What is 0101?
- c(b) = 010 01 01 = aa
- c(e) = 1 010 1 = be
- Solution: Use another code to mark the end of the symbol – will take too many bits and become expensive

Solution: Prefix Codes

Definition: A prefix code for a set S of symbols is a function c that maps S to a string of 1s and 0s in such a way

For x,y in S, x ≠ y, c(x) is not a prefix of c(y)

c(x) = 10110

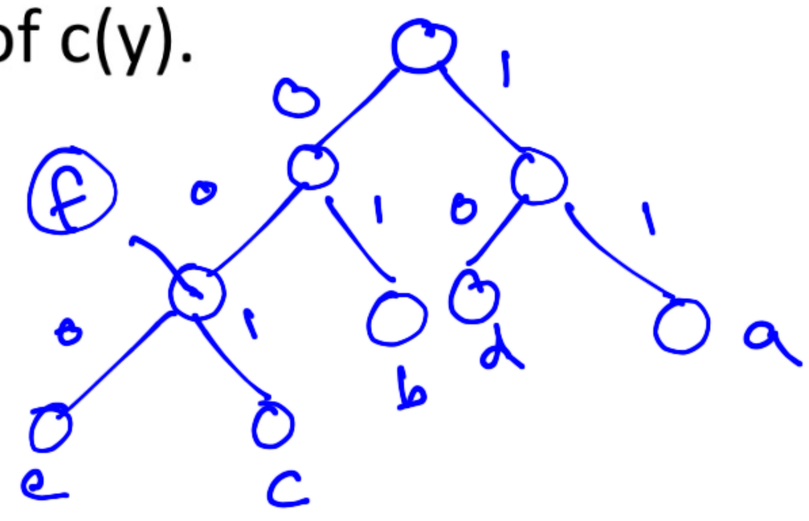
c(y) = ~~10110~~

110110

Prefix Codes: Example

Definition: A prefix code for a set S of symbols is a function c that maps S to a string of 1s and 0s in such a way

For x,y in S, x ≠ y, c(x) is not a prefix of c(y).



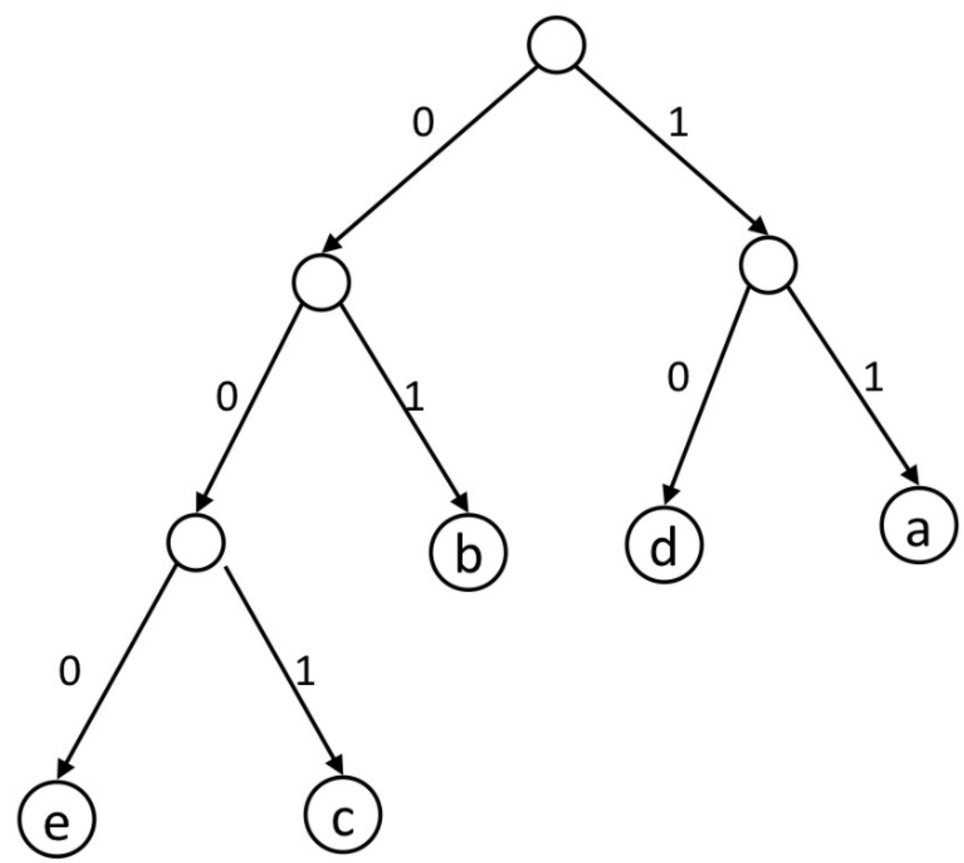
- Ex. c(a) = 11
- c(b) = 01
- c(c) = 001
- c(d) = 10
- c(e) = 000
- Is this a prefix code for S = {a, b, c, d, e}?

c(f) = 00 ✗

a b c
c 11 01 001

Checking for Prefix Codes

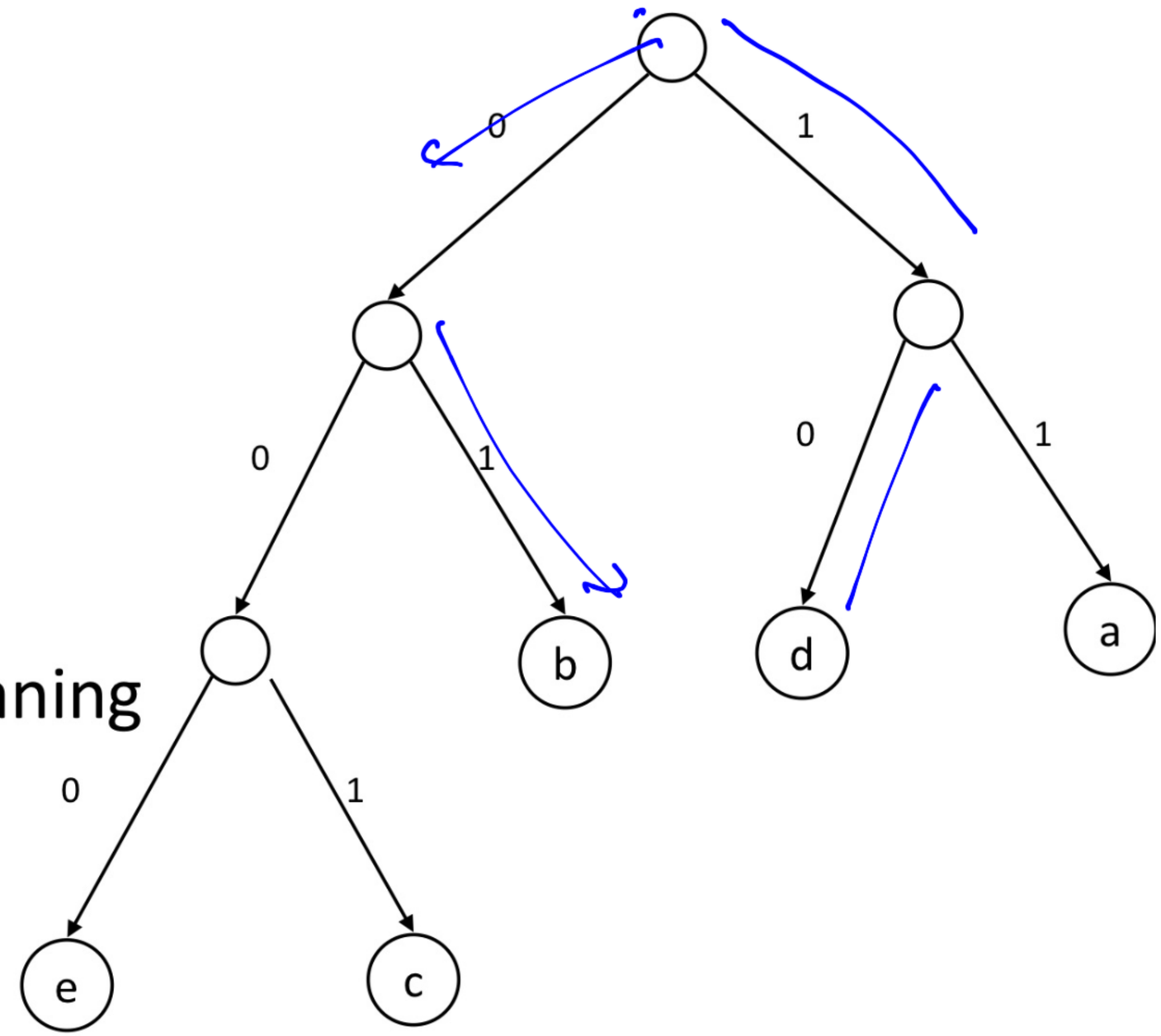
- Ex. c(a) = 11
- c(b) = 01
- c(c) = 001
- c(d) = 10
- c(e) = 000
- Is this a prefix code for S = {a, b, c, d, e}?
- Encoding example:



Prefix Codes: Decoding

- c(a) = 11
- c(b) = 01
- c(c) = 001
- c(d) = 10
- c(e) = 000
- Q. What is the meaning of 1001000001 ?

d | b | e | c



Prefix Codes: Encoding Length

- c(a) = 11
- c(b) = 01
- c(c) = 001
- c(d) = 10
- c(e) = 000

Suppose the frequencies of letters in the corpus are given as

- f(a) = 50
- f(b) = 5
- f(c) = 8
- f(d) = 12
- f(e) = 25

Q. What is the size of the encoded text?

Size = 50 * 2 + 5 * 2 + 8 * 3 + 12 * 2 + 25 * 3

Prefix Codes: Encoding Length

- c(a) = 11
- c(b) = 01
- c(c) = 001
- c(d) = 10
- c(e) = 000

Suppose the frequencies of letters in the corpus are given as

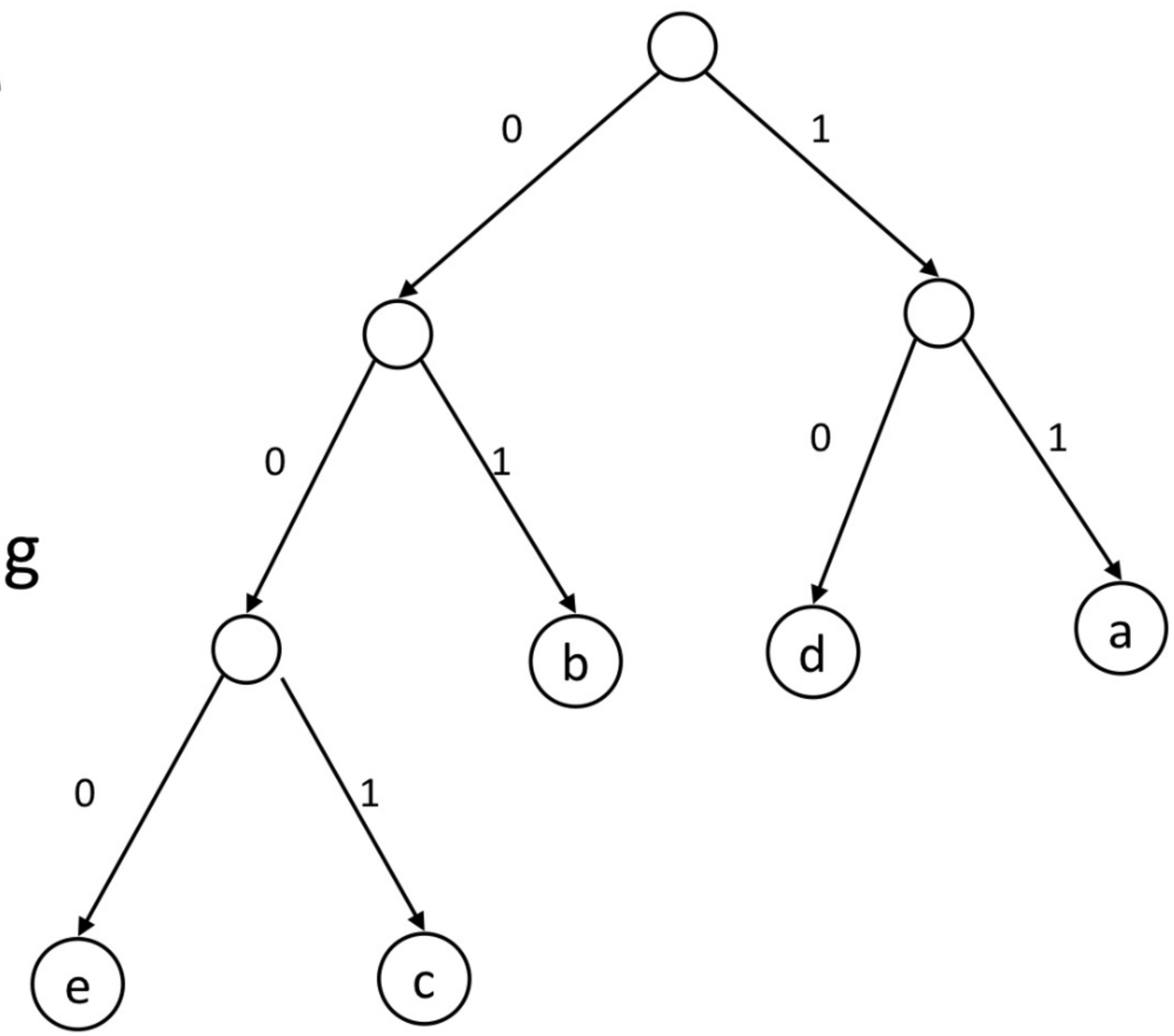
- f(a) = 50
- f(b) = 5
- f(c) = 8
- f(d) = 12
- f(e) = 25

Q. What is the size of the encoded text?

Ans: 2 * f(a) + 2 * f(b) + 3 * f(c) + 2 * f(d) + 3 * f(e)

A General Prefix Code Tree

- Only the leaf nodes have labels corresponding to symbols
- Encoding of a symbol is given by the path from root to the corresponding leaf
- Path to a symbols x can never be a prefix of path to another symbol y

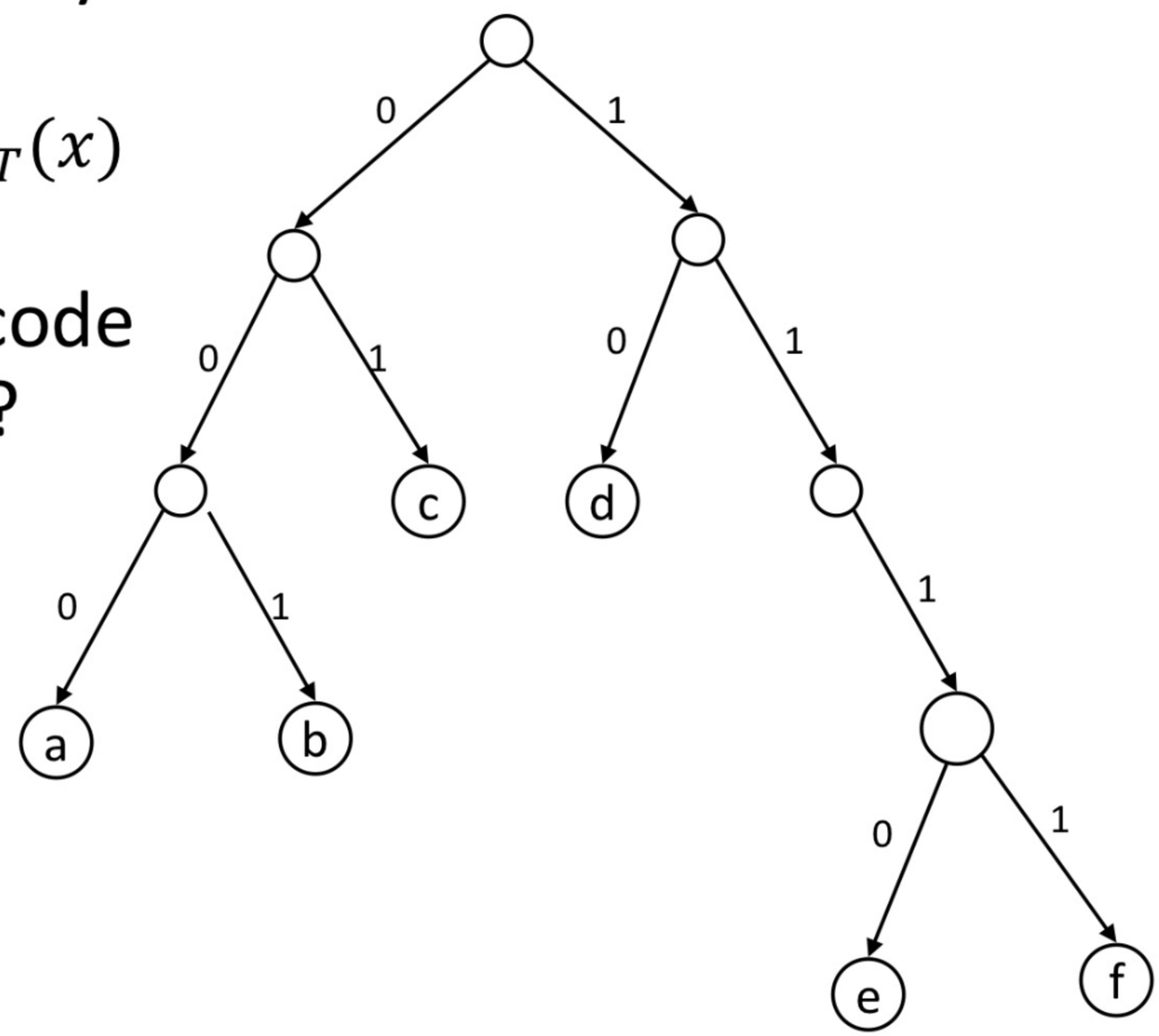


Prefix Codes: Encoding Length

In general, the encoding length for a tree is given by:

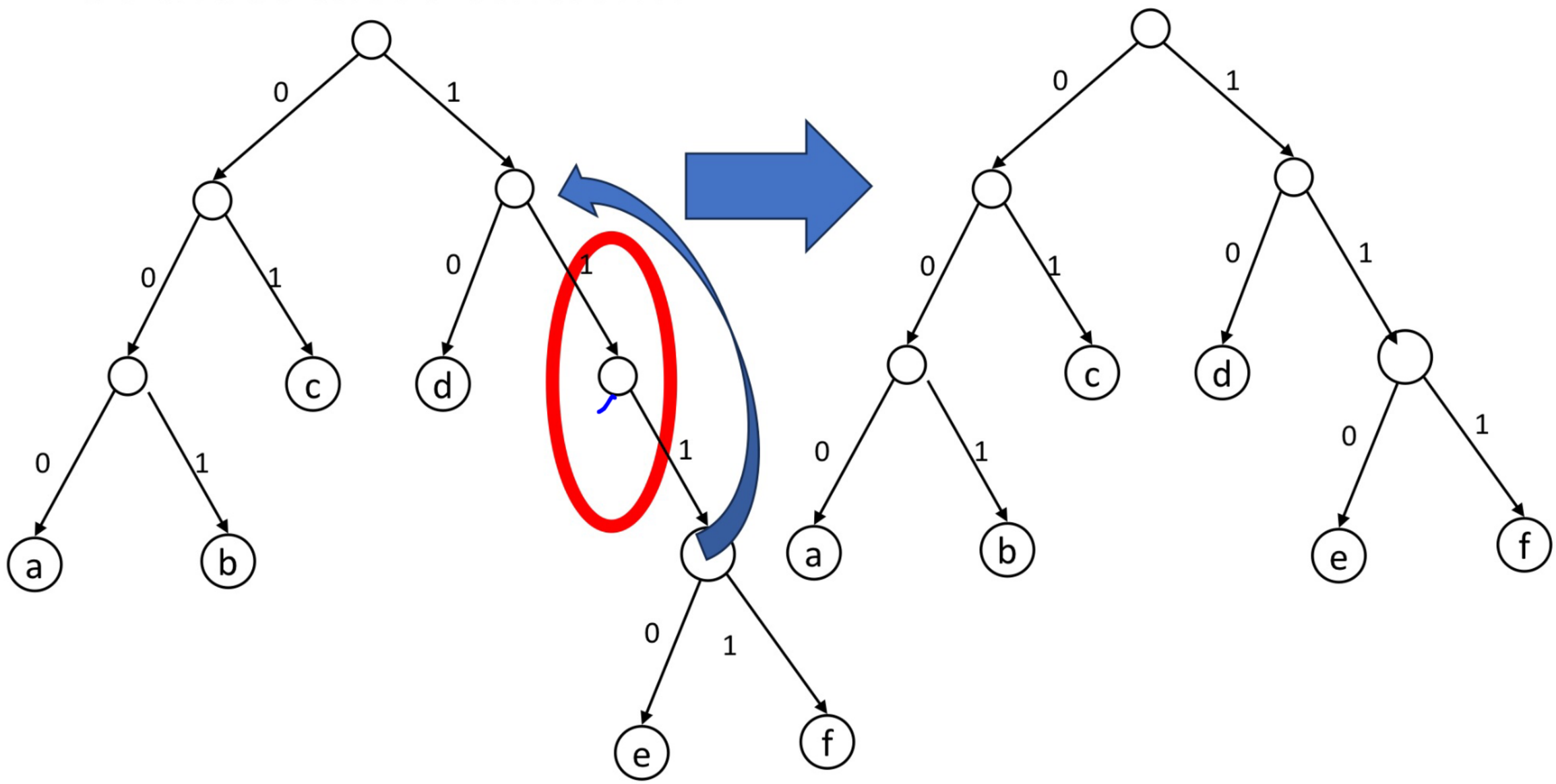
$$\text{LEN}(T) = \sum_{x \in S} f(x) \cdot \text{depth}_T(x)$$

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



Prefix Codes: Encoding Length

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



Efficient Prefix Trees

- Cut short any non-leaf node with only one child
- This will reduce the depth of symbols in the corresponding subtree
- Without changing the depths of other leaves
- Overall reduction in the length of the encoding

Finding Optimal Prefix Codes

Definition: A binary tree is full if every node that is not a leaf, has two children

Claim: The binary tree corresponding to the optimal prefix-tree is full

Finding Optimal Prefix Codes

Definition: A binary tree is full if every node that is not a leaf, has two children

Claim: The binary tree corresponding to the optimal prefix-tree is full

Proof: By contradiction

Suppose T is the binary tree for the optimal prefix code and T is not full

Therefore, there is at least one node u with only one child v.

Finding Optimal Prefix Codes

Definition: A binary tree is full if every node that is not a leaf, has two children

Claim: The binary tree corresponding to the optimal prefix-tree is full

Proof: By contradiction

Suppose T is the binary tree for the optimal prefix code and T is not full

Therefore, there is at least one node u with only one child v.

Remove u and make v child of the parent of u (make v root if u was the root)

Depth of all the leaves in the subtree of v decreases by 1

All other depths stay the same

Thus, the new tree has smaller encoding length

Finding Optimal Prefix Codes

Definition: A binary tree is full if every node that is not a leaf, has two children

Claim: The binary tree corresponding to the optimal prefix-tree is full

Proof: By contradiction

Suppose T is the binary tree for the optimal prefix code and T is not full

Therefore, there is at least one node u with only one child v.

Remove u and make v child of the parent of u (make v root if u was the root)

Depth of all the leaves in the subtree of v decreases by 1

All other depths stay the same

Thus, the new tree has smaller encoding length

Note: This is a necessary condition, not a sufficient condition

How to Construct Optimal Prefix Codes?

- Basic idea: Keep symbols with high frequency closer to the root and keep symbols with low frequency at higher depths
- How to build the tree corresponding to the optimal prefix code?
- How to prove that it is optimal?

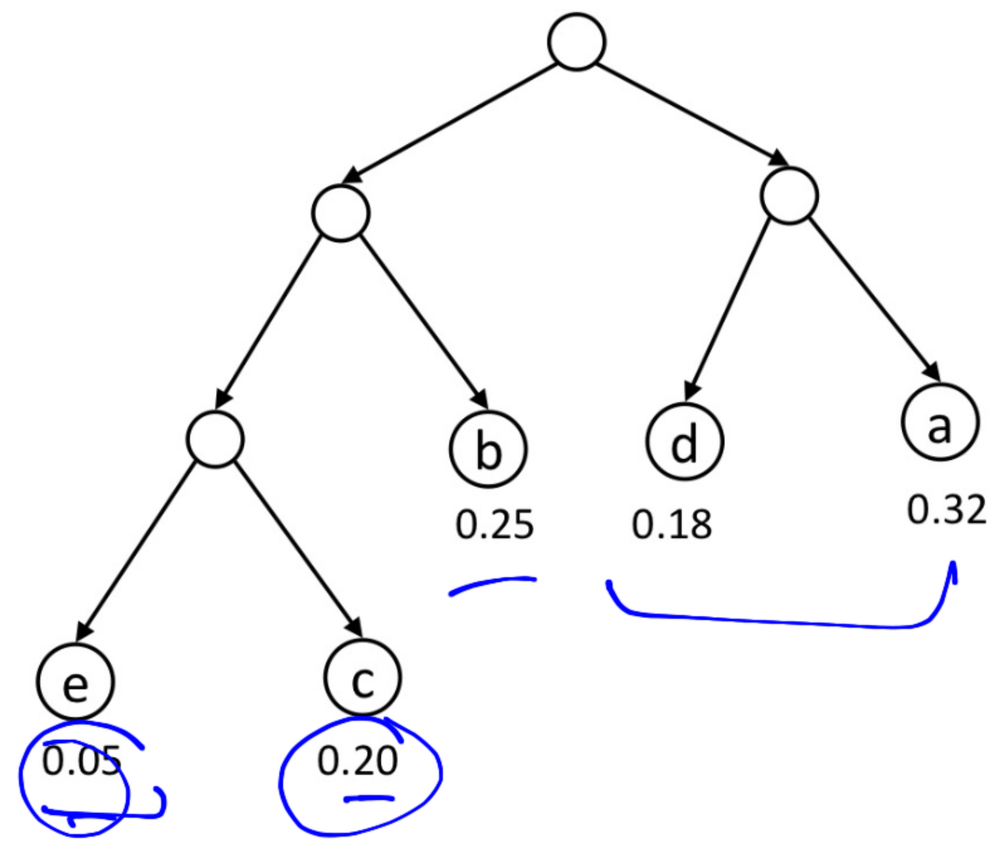
History of Huffman Codes

- Prof. Robert M. Fano
- MIT
- Worked with Shannon, father of Information theory to find optimal prefix codes
- Unable to find the optimal solution
- Gave as a term-paper in a course on Information Theory

Image source: Robert M. Fano - Engineering and Technology History Wiki (ethw.org)

Shannon-Fano Codes

- Top-down approach
- Split the set S into two S1 and S2 with almost equal frequencies
- Recursively build trees on S1 and S2
- Problem is equal splitting is very hard
- Balanced split at the root may not lead to balanced splits later
- f_a=0.32, f_b=0.25, f_c=0.20, f_d=0.18, f_e=0.05

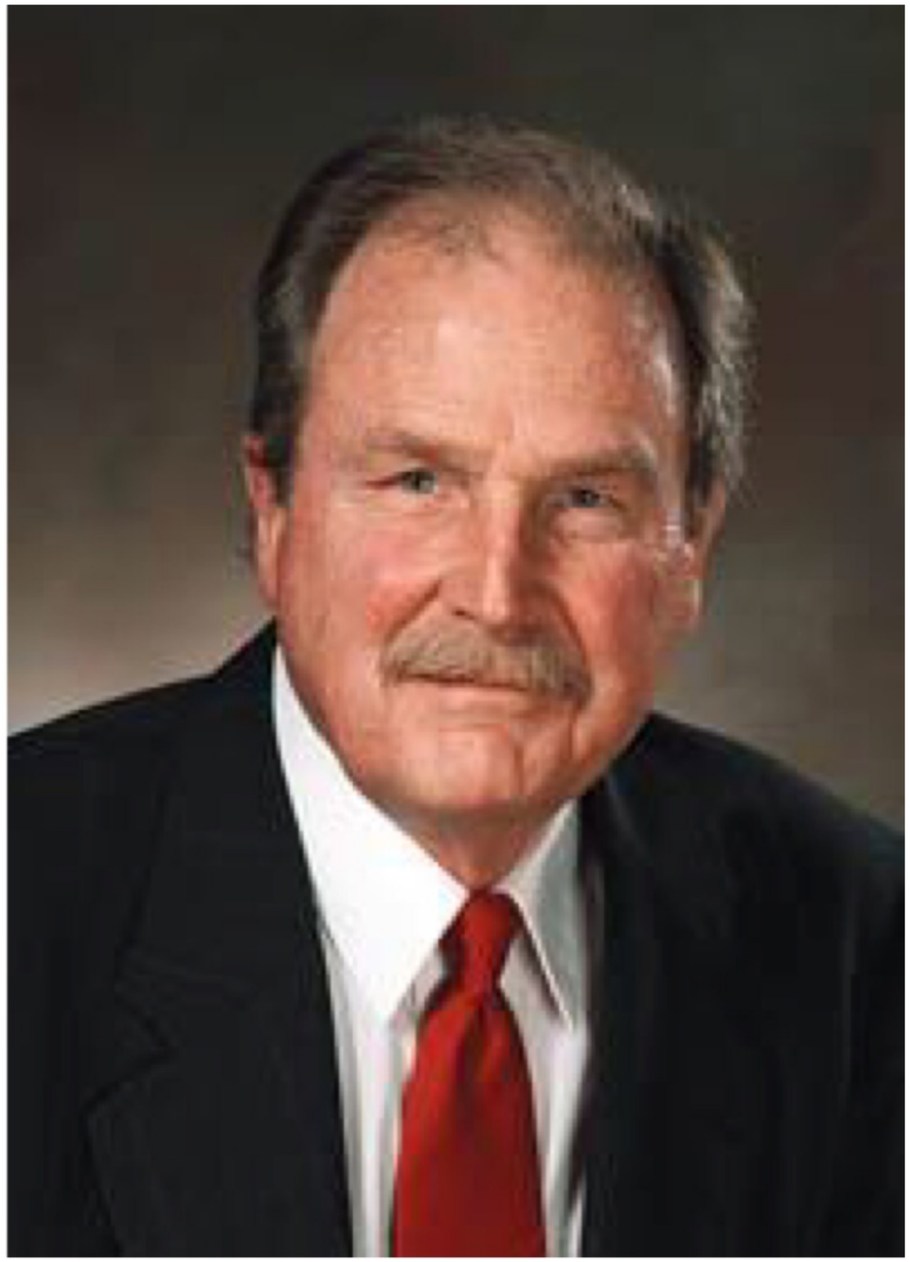


Huffman Codes

Huffman was a student at MIT in the Information Theory course

Solved the problem

Basic idea was to build the tree starting with the leaves instead of the root



David Huffman

Image source: [Discovery of Huffman Codes | Mathematical Association of America (maa.org)]

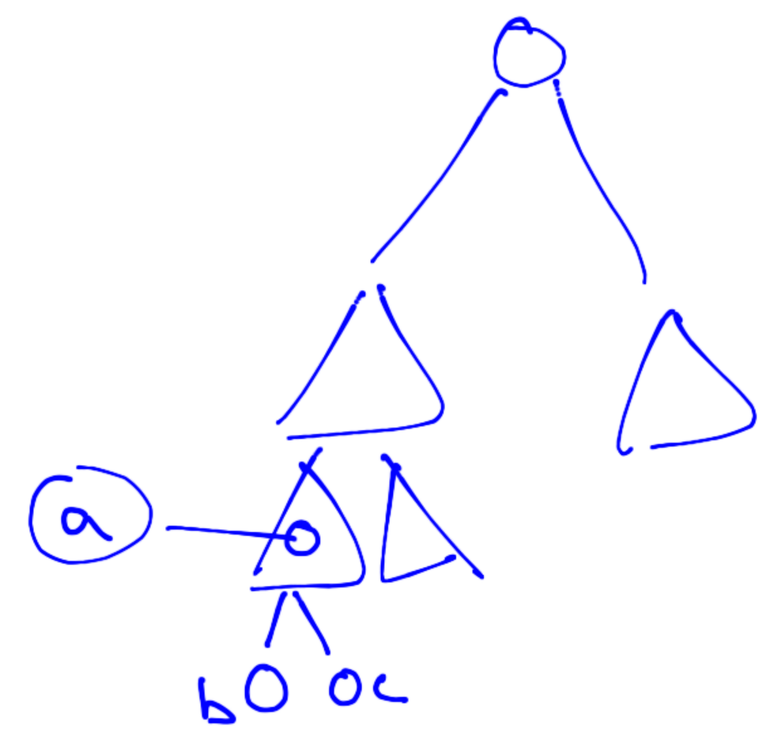
Huffman Codes

Claim 1: The lowest frequency symbol should be at the lowest level in tree of optimal prefix code.

Proof: Consider a tree T where the lowest frequency symbol (say a) is not at the lowest level.

Swap that symbol with symbol (say b) at the lowest level. The encoding length improves.

Recall: $\text{LEN}(T) = \sum_{x \in S} f(x) \cdot \mathrm{d}\,(x)$



Huffman Codes

$\sum_{x \in S} f(x) \cdot d(x)$

Proof: Consider a tree T where the lowest frequency symbol (say a) is not at the lowest level.

- Swap that symbol with symbol (say b) at the lowest level. We will show that the encoding length improves
- Old length – New length =

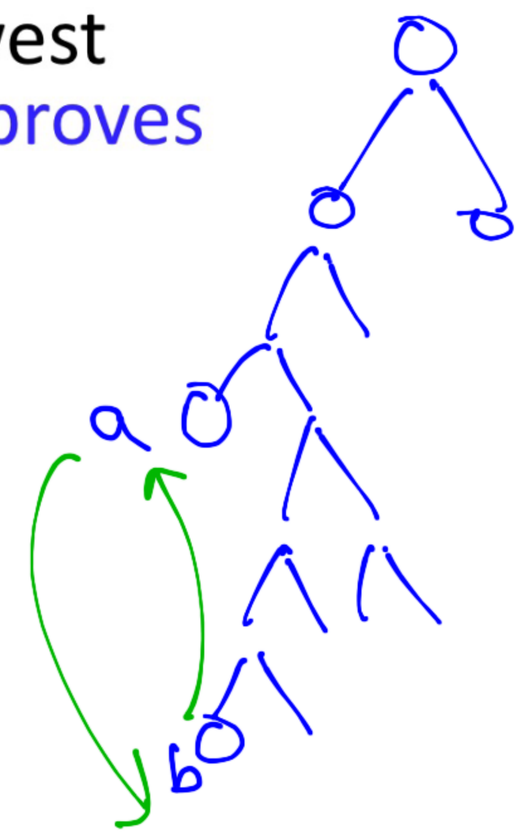
f(a) d(a) + f(b) d(b) – [f(a) d(b) + f(b) d(a)]

= f(b) [d(b) – d(a)] – f(a) [d(b) – d(a)]

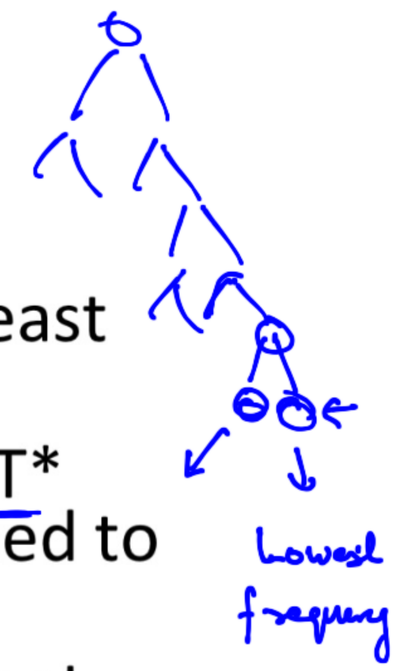
= [f(b) – f(a)] [d(b) – d(a)]

= [> 0] [> 0] > 0

New length is smaller than the old length



Huffman Codes

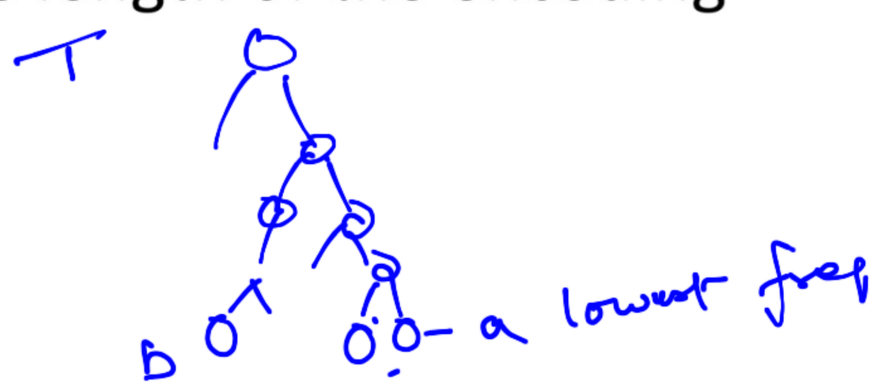


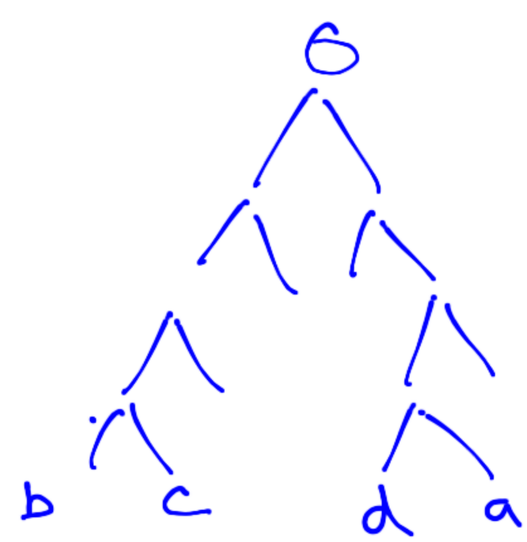
Observation: The lowest level always contains at least two leaves since the optimal tree is full

Claim 2: There is an optimal prefix code with tree T* where the two lowest-frequency letters are assigned to leaves that are siblings in T*.

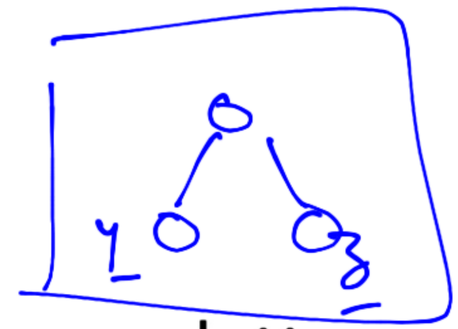
Proof: The two lowest frequency letters must be at the maximum depth of the tree (from claim 1 and a similar reasoning for the other letter)

- If these two letters are not siblings, then swap and make them siblings
- Swapping leaves at the same levels does not change the length of the encoding





Huffman Codes

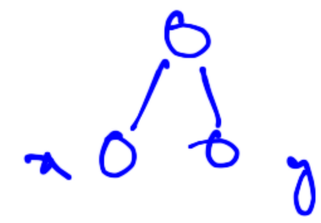


w

$f(w) = f(y) + f(z)$

- Create tree bottom-up.
- Make two leaves for two lowest-frequency letters y and z.
- Recursively build tree for the rest of the letters add a meta-letter for yz (size of the set S has now decreased by 1)

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
    }
}
```

- Q. What is the time complexity?

Slide adapted from: Mathijs de Weerdt, Delft University of Technology

Huffman Coding: Analysis

$T(|S|) = T(|S|-1) + O(|S|)$

$T(|S|) = O(|S|^2)$

Can we do better?

Huffman Coding: Analysis

$T(|S|) = T(|S|-1) + O(|S|)$

$T(|S|) = O(|S|^2)$

Can we do better?

Use min-heaps for finding the lowest frequency letters

$T(|S|) = T(|S|-1) + O(\log(|S|))$

$T(|S|) = O(|S| \log(|S|)$

Proof of Optimality

Theorem: Huffman code for S achieves the minimum encoding length of any prefix code

Proof of Optimality

Theorem: Huffman code for S achieves the minimum encoding length of any prefix code

Proof: By induction on n = |S|

Base case: n = 2. There cannot be a shorter code than root with two leaves.

Induction hypothesis: Suppose the tree T' of size n – 1, for S' obtained by replacing y, z with the symbol ω is optimal for S'

Induction step: The optimal tree T for S is obtained by replacing the leaf ω with a subtree containing y and z.

Proof of Optimality

Induction step: The optimal tree T for S is obtained by replacing the leaf ω in T' with a subtree containing y and z.

Proof: By contradiction.

Proof sketch: Suppose another tree Z is better for S

In Z, y and z must be maximum depth leaf nodes (claim 1)

- Modify Z so that y, z are siblings (the encoding length does not change)
- Replace y, z by symbol ω to obtain Z'
- Now Z' cannot be better than T' by induction hypothesis

Proof of Optimality

Induction step: The optimal tree T for S is obtained by replacing the leaf ω in T' with a subtree containing y and z.

Proof: Suppose the optimal tree Z is better than T

LEN(Z) < LEN(T)

In Z, y and z must be maximum depth leaf nodes (claim 1)

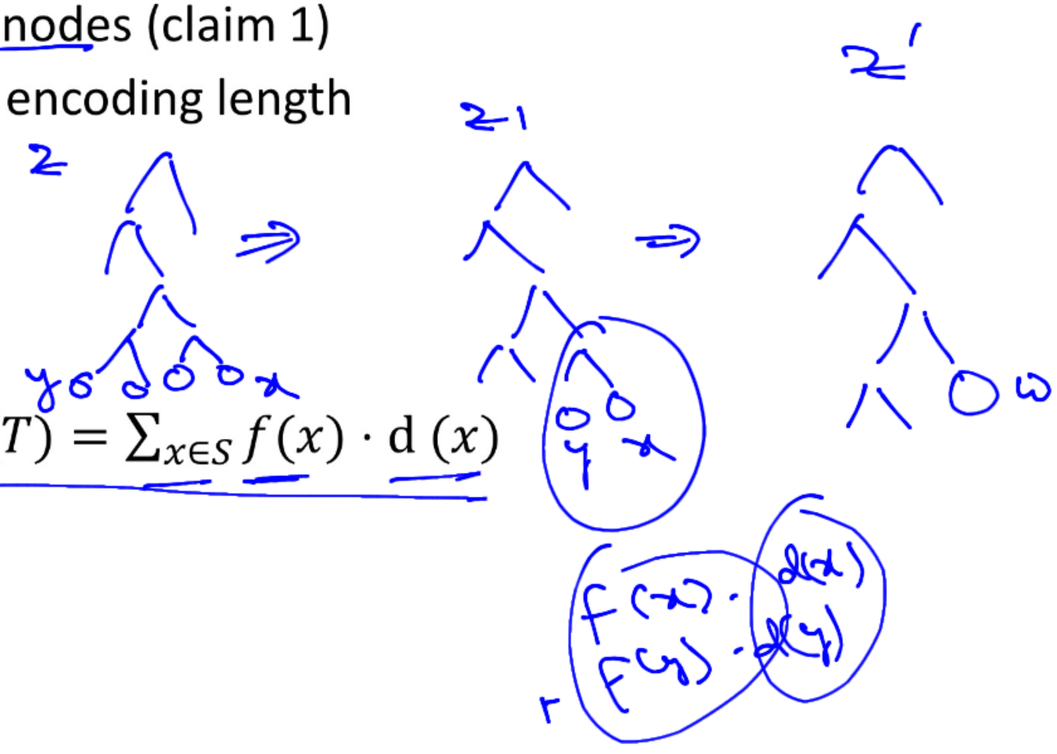
Modify Z to Z1 so that y, z are siblings (the encoding length does not change)

LEN(Z1) = LEN(Z) < LEN(T)

Replace y, z in Z1 by symbol ω to obtain Z'

What is LEN(Z')?

LEN(Z') = LEN(Z1) – f(ω)



$$\text{LEN}(T) = \sum_{x \in S} f(x) \cdot \text{d}(x)$$

Proof of Optimality

Proof: Suppose the optimal tree Z is better than T

- LEN(Z) < LEN(T)
- In Z, y and z must be maximum depth leaf nodes (claim 1)
- Modify Z to Z1 so that y, z are siblings (the encoding length does not change)
- LEN(Z1) = LEN(Z) < LEN(T)
- Replace y, z in Z1 by symbol ω to obtain Z'
- What is LEN(Z')?
  - LEN(Z') = LEN(Z1) – f(ω)
  - LEN(T') = LEN(T) – f(ω)

$$\mathrm{LEN}(T) = \sum_{x \in S} f(x) \cdot \mathrm{d}\,(x)$$

Therefore, $\boxed{\mathrm{LEN}(Z') < \mathrm{LEN}(T').}$

But T' was optimal for S' according to IH

Contradiction to induction hypothesis.

Thank You