Huffman coding

Problem: loss-less compression

- Suppose we have a 100,000-character data file to store
- Suppose our data contains only four possible characters A, B, C, D
- A binary code encodes each character with a binary string (or codeword)
- Goal: find a binary code so that the file is encoded with as few bits as possible

Fixed length code

- A simple binary code that use the same number of bits to represent each character
- In our previous example, we have 4 characters
- We can encode each letter use $\log_2 4 = 2$ bits

A	00
B	01
C	10
D	11

- Message: A A B B C D
- Code:

- Code: 11 01 00 10 01
- Message:

Coding efficiency

Suppose we looked at a sample of the data we need to encode, and the frequency of different characters vary significantly

Total	130
A	70
B	30
C	25
D	5

- What is the length of the fixed length code for this sample of 130 letters?
  - $2 \times 130 = 260$
- Can we achieve better efficiency?
  - Noticing that D is very rare, and A is frequent
  - Can we use longer code for D and shorter code for A?

Variable length codes

Symbol	Codeword
A	0
B	100
C	101
D	11

Total	130
A	70
B	30
C	25
D	5

- Message: A A B B C D
- Code:

- Code: 11 01 00 10 01
- Message:

- Total # of bits for encoding this sample?

Consider an alternative code

letter	code
A	0
B	01
C	001
D	111

- But does this code work?
  - Code: 001111
  - Message: ABD or CD?
- The code is ambiguous
- Issue:
  - The code for letter A is a prefix of the code for letter B

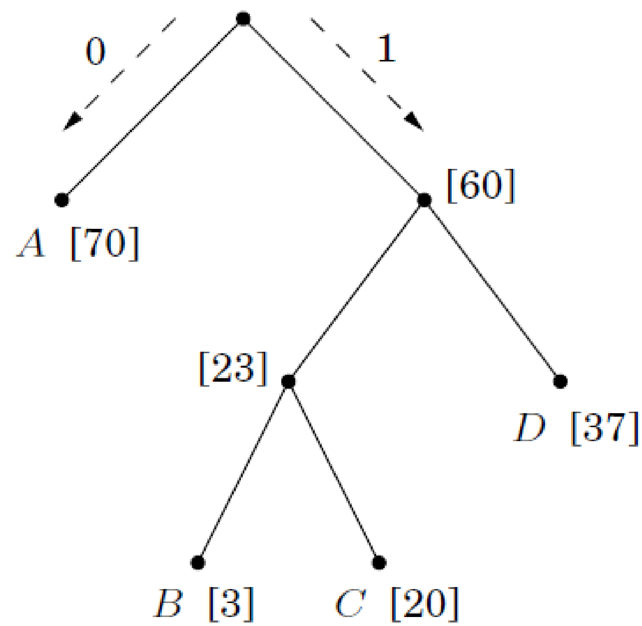
To avoid this ambiguity, the code must be **prefix free**:

- No code could be the prefix of another code

Binary tree representation

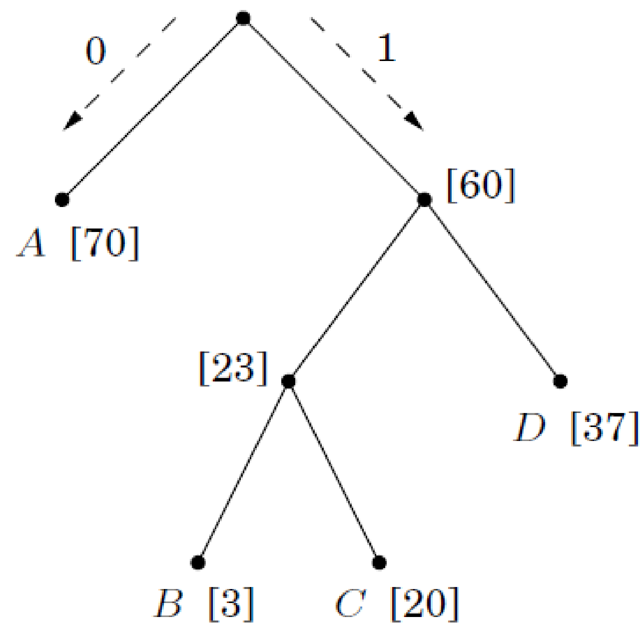
- Prefix-free code can be represented using a <u>binary tree</u>
- Characters are leaf nodes
- Each edge is labeled as either 0 or 1
- The code is defined by the path from root to the node
- Code length = path length (depth of the node)

Symbol	Codeword
A	0
B	100
C	101
D	11



- Decoding with the prefix tree is easy

Symbol	Codeword
A	0
B	100
C	101
D	11



- Start from the root, follow the path, when you reach a leaf node, output the character, restart from the root
- Decoding: 0111010100

Optimal Prefix Coding Problem

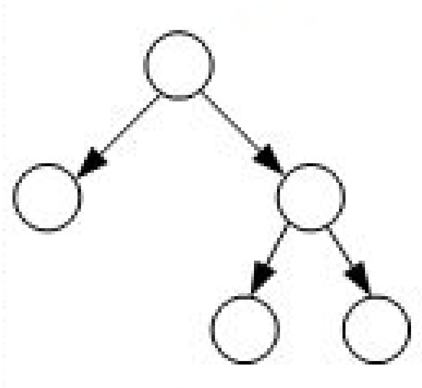
- Input: Given a set of *n* letters (c_1,…, c_n) with frequencies (f_1,…, f_n)

- Construct a binary tree *T* to define a prefix code that minimizes the average code length

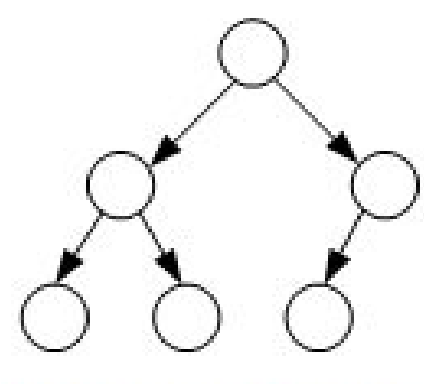
$$cost(T) = \sum_{i=1}^{n} f_i \times depth_T(c_i)$$

Optimal code must be a full binary tree

Definition: A **full binary tree** is a **tree** in which every node other than the leaves has two children.



A full binary tree

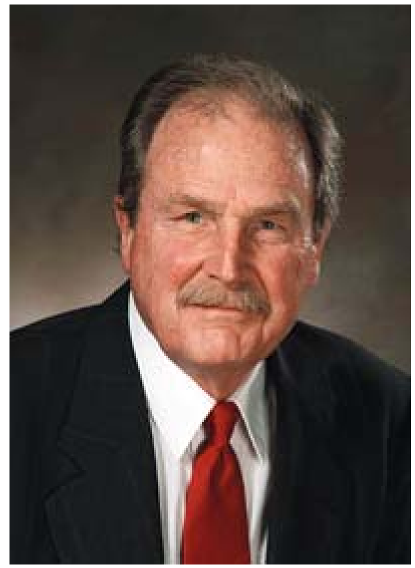


Not a full binary tree

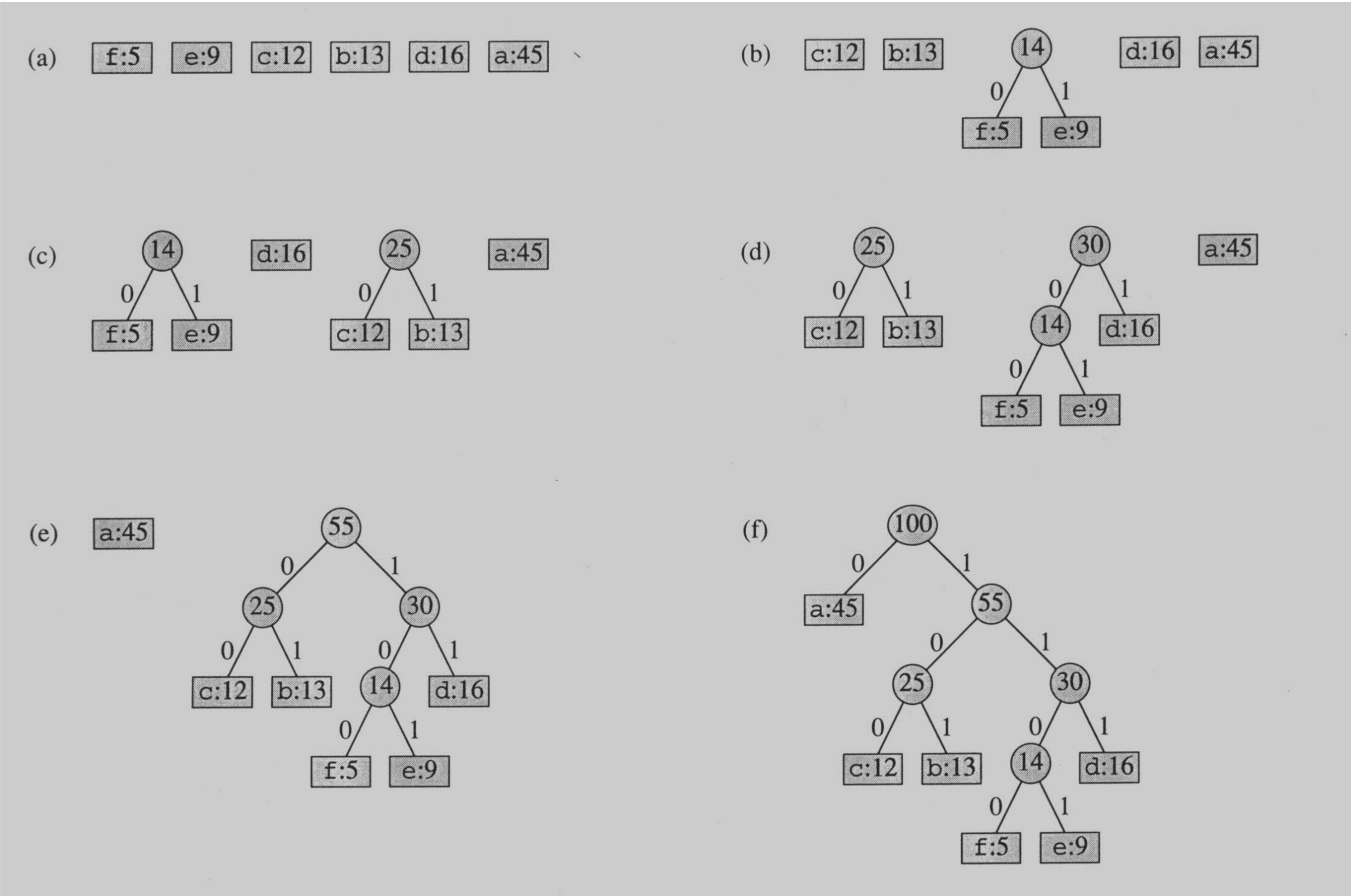
If a binary tree is not full, it can’t represent an optimal code. Why?

David Huffman’s idea

- A Term paper at MIT



- Build the tree (code) bottom-up in a greedy fashion
  - Place the least frequent characters at the bottom of the tree


(a)
f:5
e:9
c:12
b:13
d:16
a:45
(b)
14
0
1
(c)
25
(d)
30
(e)
55
(f)
100

An recursive view of the algorithm

Huffman-recur(C, f)

1. Identify the two least frequent characters x, y in C
2. $C' = C - \{x, y\} + \{\mathrm{z}\}$ //replace x, y with a new character z
3. Assign frequency for z: $f(z) = f(x) + f(y)$
4. T' = Huffman-recur(C', f)
5. Add x and y to T' so that they are the children of z
6. Return T'

Correctness of Huffman's Algorithm

Proof by induction:

1. Base case:

   n=2. (n=1 has no need for code)

   1 bit for each character – code is optimal

2. Inductive hypothesis

   Assume that the algorithm produce an optimal code for all alphabets of size 2, …, k

Correctness of Huffman's Algorithm

3. Inductive step

- For problems with alphabet size $k + 1$.
- Let $C = \{c_1, \dots, c_k, c_{k+1}\}$ be the alphabet, let x and y be its two least frequent characters
- The algorithm removes x and y from the alphabet and introduce a new character z, with $f(z) = f(x) + f(y)$
- The new alphabet C' has size k
- Based on the inductive hypothesis the tree T' is optimal for C'
- So we just need to prove that <u>appending x and y to z will lead to an optimal tree for C</u>

A useful lemma

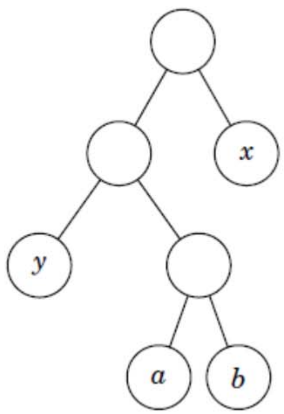
Let x, y be the two least frequent characters of C. There exists an optimal tree in which x and y are siblings.

Proof by **the exchange argument** (start from an optimal tree T in which x and y are not siblings, construct a tree T' in which x and y are not siblings that is no worse)

Proof of Lemma

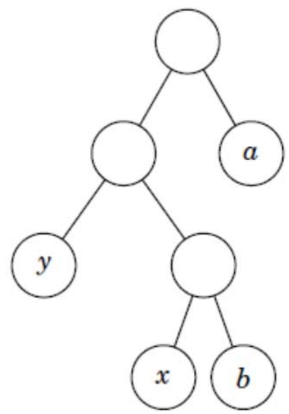
- Let T be an optimal tree where x and y are not siblings.
- Let a be a leaf with maximum depth (i.e., the character with the longest code). Because T is optimal, a must have a sibling b.
- Assume $f(a) \leq f(b)$ and $f(x) \leq f(y)$
- Switch a and x, because x and y have the least frequency, we know $f(x) \leq f(a)$, so switching will not increase the cost
- Switch b and y, because x and y has the least frequency, we have $f(y) \leq f(b)$, so again switching will not increase the cost
- After the two switches, the new tree will have x and y as siblings and it will also be optimal because it is no worse than the optimal tree T

Illustration of the proof of lemma



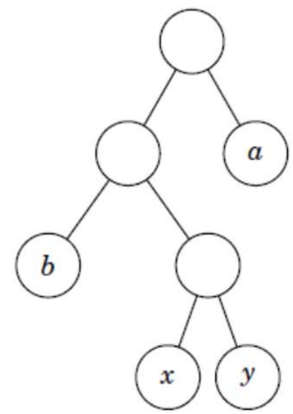
T

Original
tree



T'

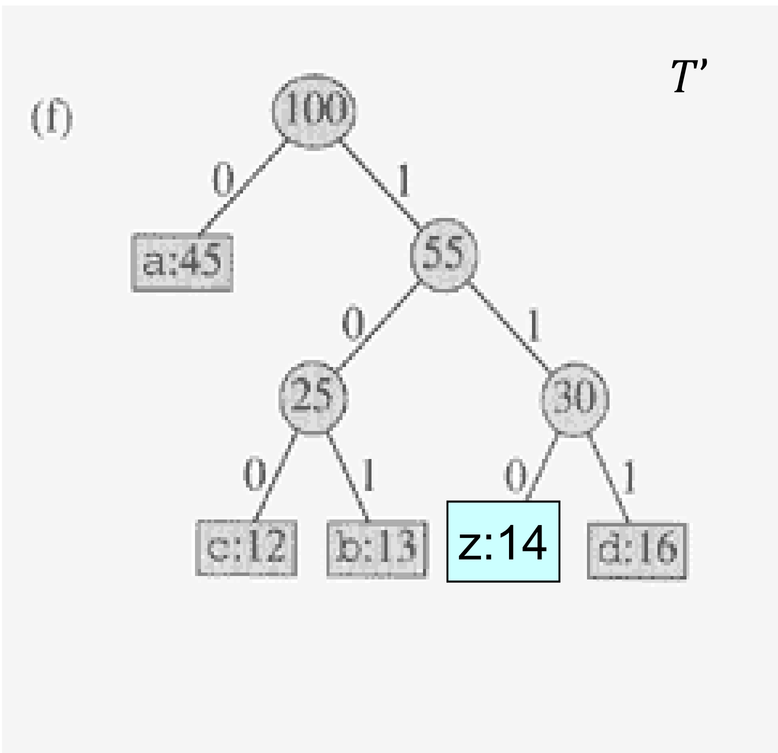
After switching
a and x

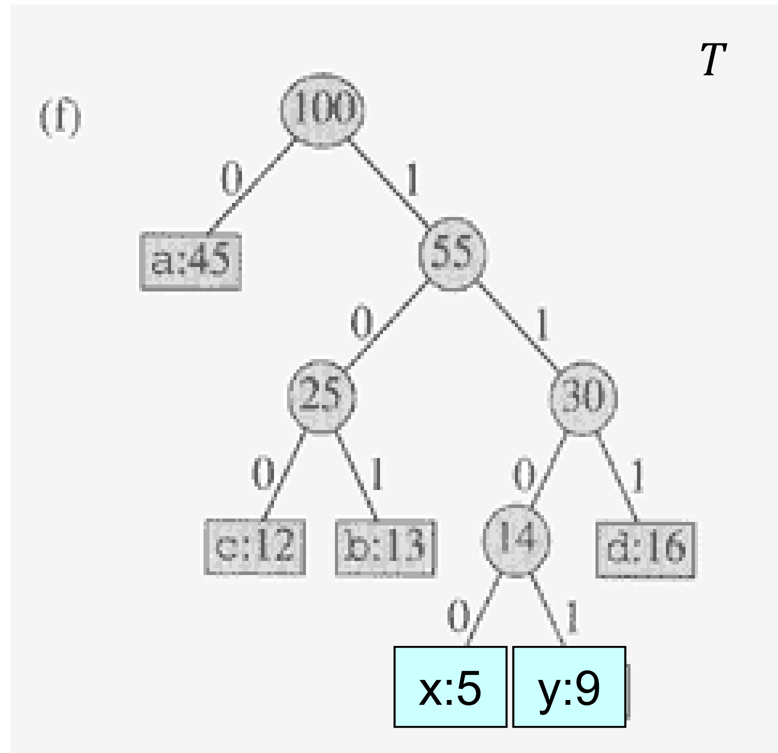


T''

After switching
b and y

Back to the inductive proof





We know that

$$Cost(T) = Cost(T') + f(x) + f(y)$$

Inductive step Cont.

Assume (for contradiction) T is not optimal.

By the lemma, we know there is an optimal tree $\hat{T}$ in which x and y are siblings.

Now if we delete x and y from $\hat{T}$ and replace the internal node with z, we get a new tree $\hat{T}'$. We know that

$$\begin{aligned} cost(\hat{T}') &= cost(\hat{T}) - f(x) - f(y) \\ &< cost(T) - f(x) - f(y) = cost(T') \end{aligned}$$

Now both T' and $\hat{T}'$ are trees for alphabet C' and $cost(\hat{T}') < cost(T')$ --- this contradicts with the fact that T' is optimal for C'(inductive assumption)

So T must be optimal for C

QED