CS473-Algorithms I

Lecture 11

Huffman Codes

Huffman Codes

- Widely used and very effective technique for compressing data
- Savings of 20% to 90% are typical
- Depending on the characteristics of the file being compressed Huffman's greedy algorithm
  - uses a table of the frequencies of occurrence of each character
  - to build up an optimal way of representing each character as a binary string

Example: A 100,000-character data file that is to be compressed only 6 characters {a, b, c, d, e, f} appear

	a	b	c	d	e	f
frequency (in thousands)	45K	13K	12K	16K	9K	5K
fixed-length codeword	000	001	010	011	100	101
variable-length codeword	0	101	100	111	1101	1100
variable-length codeword	0	10	110	1110	11110	11111

Huffman Codes

Binary character code:

- each character is represented by a unique binary string

Fixed-length code:

- needs 3 bits to represent 6 characters
- requires 100.000×3=300,000 bits to code the entire file

Variable-length code:

- can do better by giving frequent characters short codewords & infrequent words long codewords
- requires $45\times1+13\times3+12\times3+16\times3+9\times4+5\times4$

  $=224{,}000$ bits

Prefix Codes

Prefix codes: No codeword is also a prefix of some other codeword

It can be shown that:

optimal data compression achievable by a character code can always be achieved with a prefix code

Prefix codes simplify encoding (compression) and decoding

Encoding: Concatenate the codewords representing each character of the file

e.g. 3 char file "abc" $\xrightarrow{\text{encoded}}$ 0.101.100 = 0101100

Prefix Codes

Decoding: is quite simple with a prefix code

the codeword that begins an encoded file is unambigious since no codeword is a prefix of any other

- identify the initial codeword
- translate it back to the original character
- remove it from the encoded file
- repeat the decoding process on the remainder of the encoded file

e.g. string 001011101 parses uniquely as

0.0.101.1101 $\xrightarrow{\text{decoded}}$ aabe

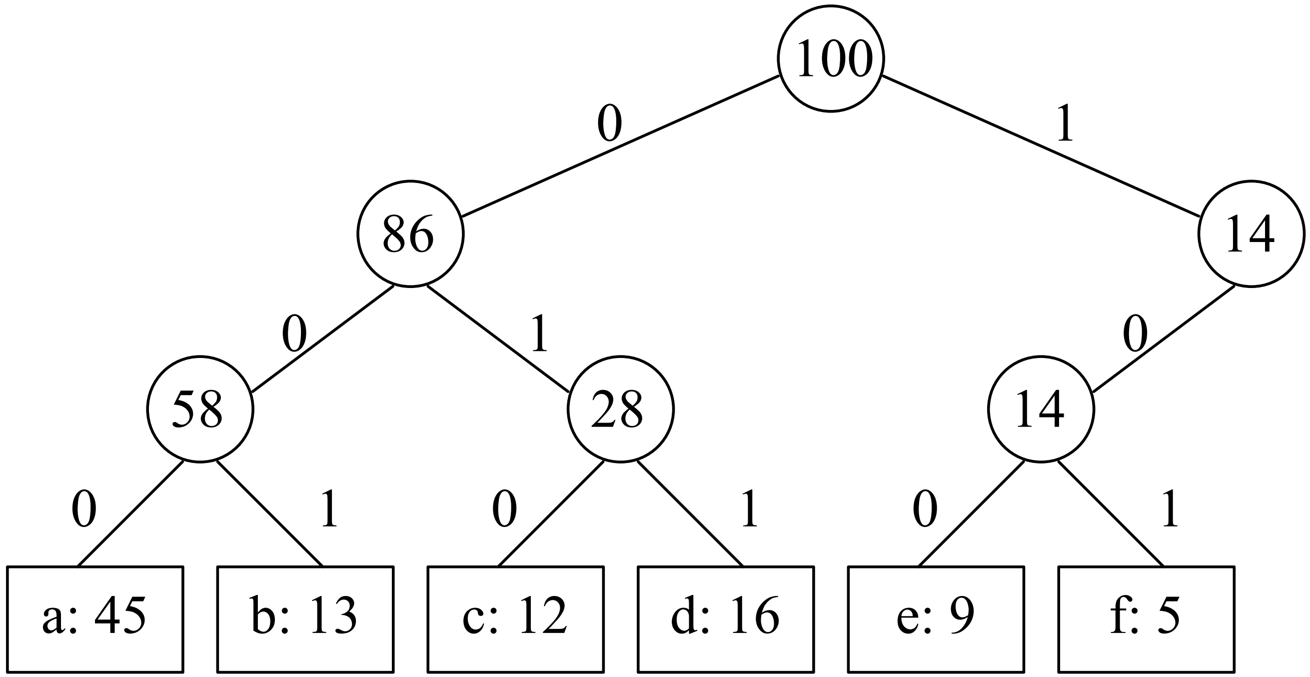
Prefix Codes

Convenient representation for the prefix code:
a binary tree whose leaves are the given characters

Binary codeword for a character is the path from the root to that character in the binary tree

“0” means “go to the left child”
“1” means “go to the right child”

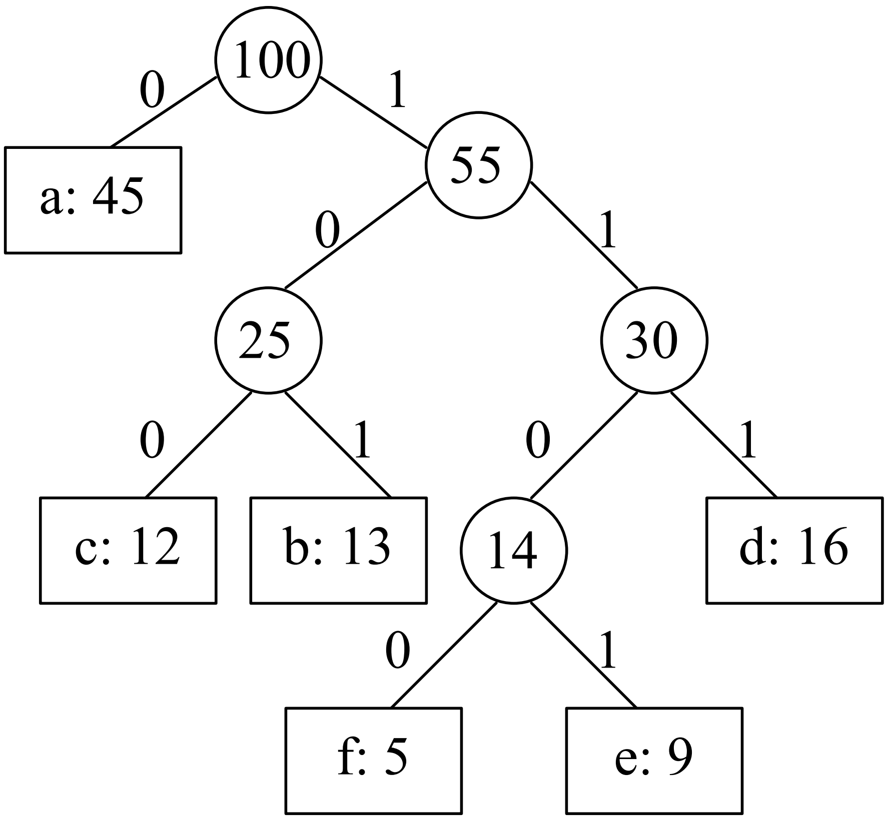
Binary Tree Representation of Prefix Codes



The binary tree corresponding to the fixed-length code

Binary Tree Representation of Prefix Codes

The binary tree corresponding to the optimal variable-length code



An optimal code for a file is always represented by a full binary tree

Full Binary Tree Representation of Prefix Codes

Consider an FBT corresponding to an optimal prefix code

It has $|C|$ leaves (external nodes)

One for each letter of the alphabet where C is the alphabet from which the characters are drawn

Lemma: An FBT with $|C|$ external nodes has exactly $|C|-1$ internal nodes

Full Binary Tree Representation of Prefix Codes

Consider an FBT T corresponding to a prefix code

How to compute, $B(T)$, the number of bits required to encode a file

$f(c)$: frequency of character c in the file

$d_T(c)$: depth of c's leaf in the FBT T

note that $d_T(c)$ also denotes length of the codeword for c

$$B(T) = \sum_{c \in C} f(c)\, d_T(c)$$

which we define as the cost of the tree T

Prefix Codes

Lemma: Let each internal node i is labeled with the sum of the weight $w(i)$ of the leaves in its subtree

Then $B(T) = \sum_{c \in C} f(c)\, d_T(c) = \sum_{i \in I_T} w(i)$ where

I_T denotes the set of internal nodes in T

Proof: Consider a leaf node c with $f(c)$ & $d_T(c)$
Then, $f(c)$ appears in the weights of $d_T(c)$ internal node along the path from c to the root
Hence, $f(c)$ appears $d_T(c)$ times in the above summation

Constructing a Huffman Code

Huffman invented a greedy algorithm that constructs an optimal prefix code called a Huffman code

The greedy algorithm

- builds the FBT corresponding to the optimal code in a bottom-up manner
- begins with a set of $|C|$ leaves
- performs a sequence of $|C|-1$ "merges" to create the final tree

Constructing a Huffman Code

A priority queue Q, keyed on f, is used
to identify the two least-frequent objects to merge

The result of the merger of two objects is a new object

- inserted into the priority queue according to its frequency
- which is the sum of the frequencies of the two objects merged

Constructing a Huffman Code

HUFFMAN(C)

  $n \leftarrow |C|$
  $Q \leftarrow C$
  for $i \leftarrow 1$ to $n - 1$ **do**
    $z \leftarrow$ ALLOCATE-NODE()
    $x \leftarrow$ left[z] $\leftarrow$ EXTRACT-MIN(Q)
    $y \leftarrow$ right[z] $\leftarrow$ EXTRACT-MIN(Q)
    $f[z] \leftarrow f[x] + f[y]$
    INSERT(Q, z)
  return EXTRACT-MIN(Q) Δ only one object left in Q

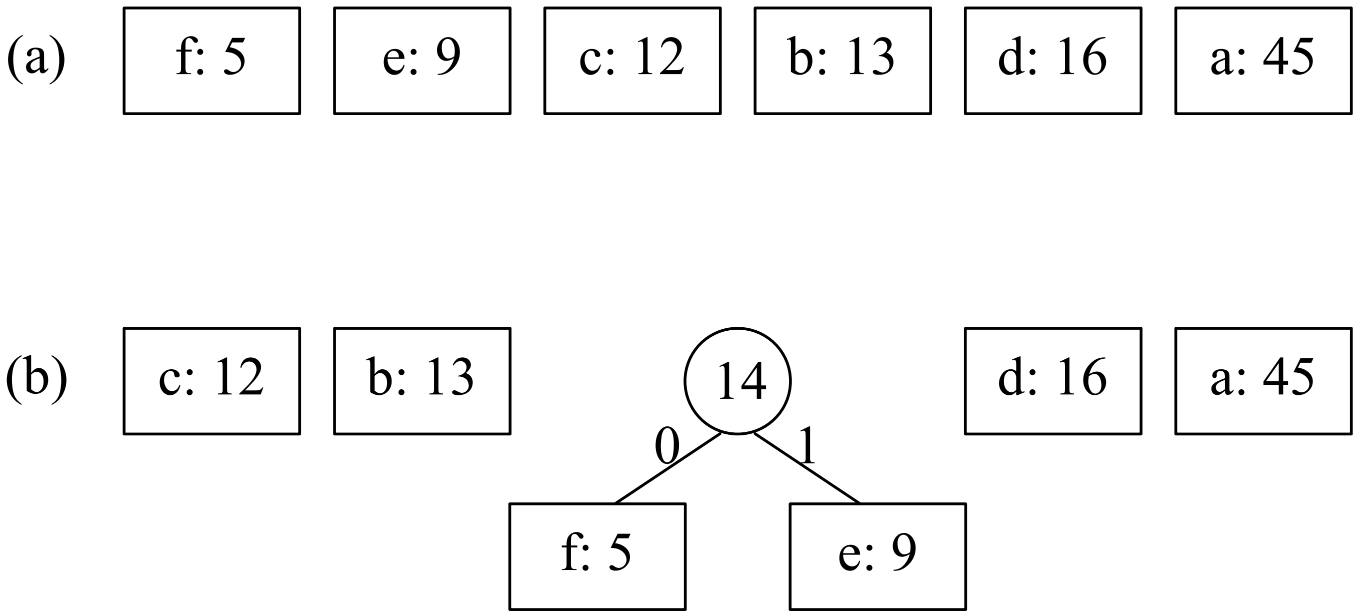
Priority queue is implemented as a binary heap

Initiation of Q (BUILD-HEAP): O(n) time

EXTRACT-MIN & INSERT take O(lg$n$) time on Q with n objects

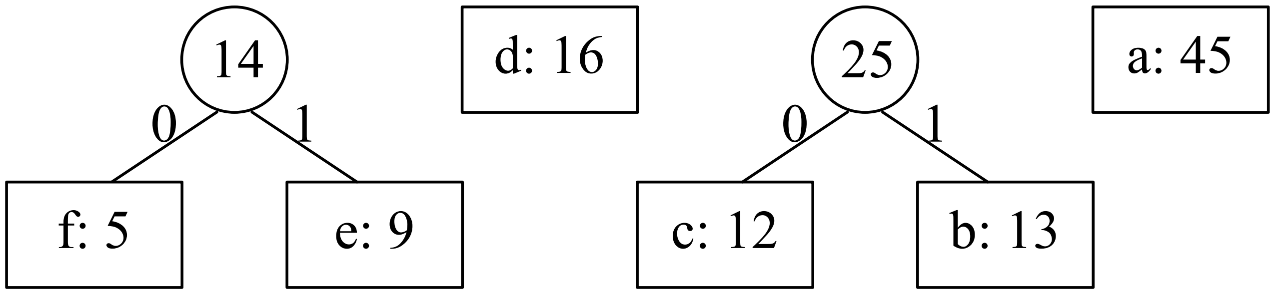
$$T(n) = \sum_{i=1}^{n} \lg i = O(\lg(n!)) = O(n \lg n)$$

Constructing a Huffman Code

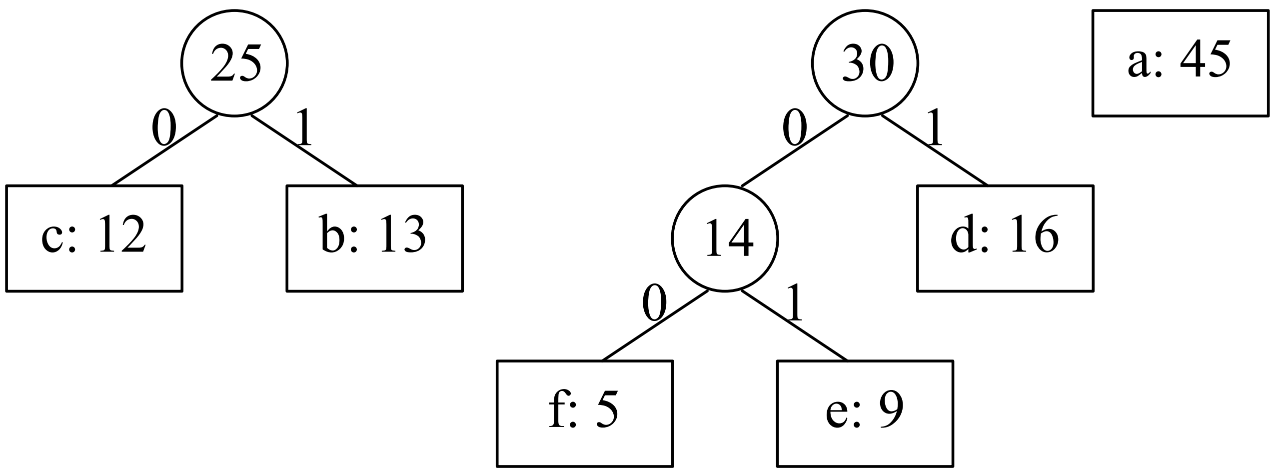


Constructing a Huffman Code

(c)

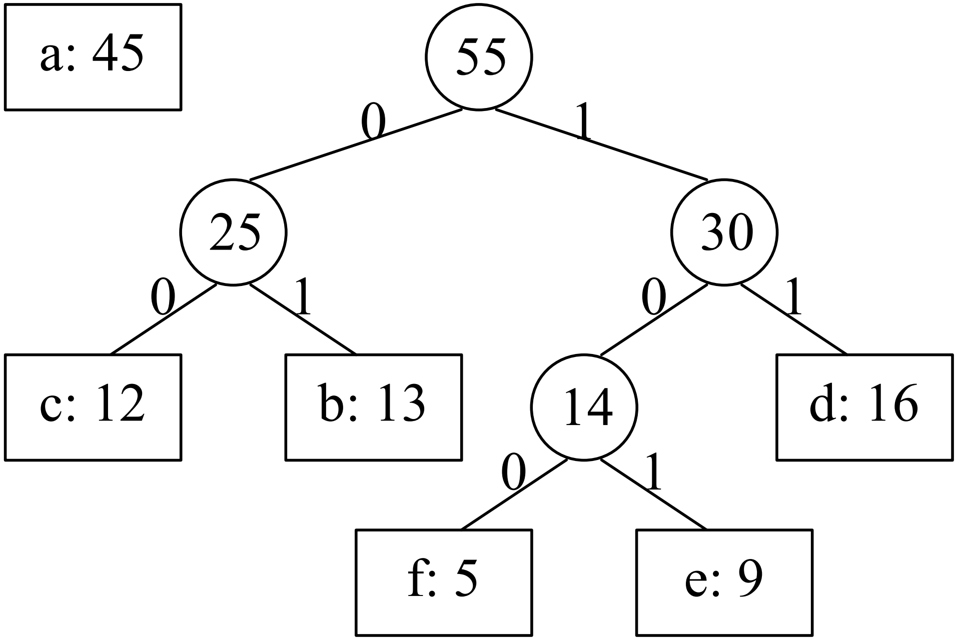


(d)



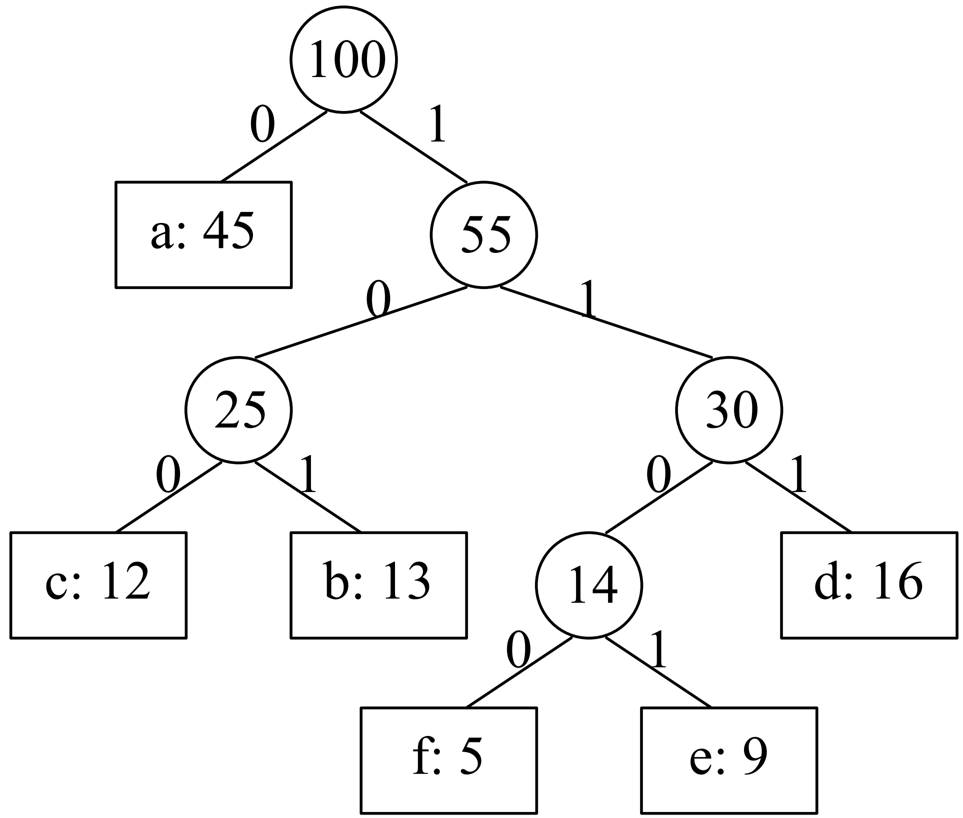
Constructing a Huffman Code

(e)



Constructing a Huffman Code

(f)



Correctness of Huffman's Algorithm

We must show that the problem of determining an optimal prefix code

- exhibits the greedy choice property
- exhibits the optimal substructure property

Lemma 1: Let x & y be two characters in C having the lowest frequencies

Then, ∃ an optimal prefix code for C in which the codewords for x & y have the same length and differ only in the last bit

Correctness of Huffman's Algorithm

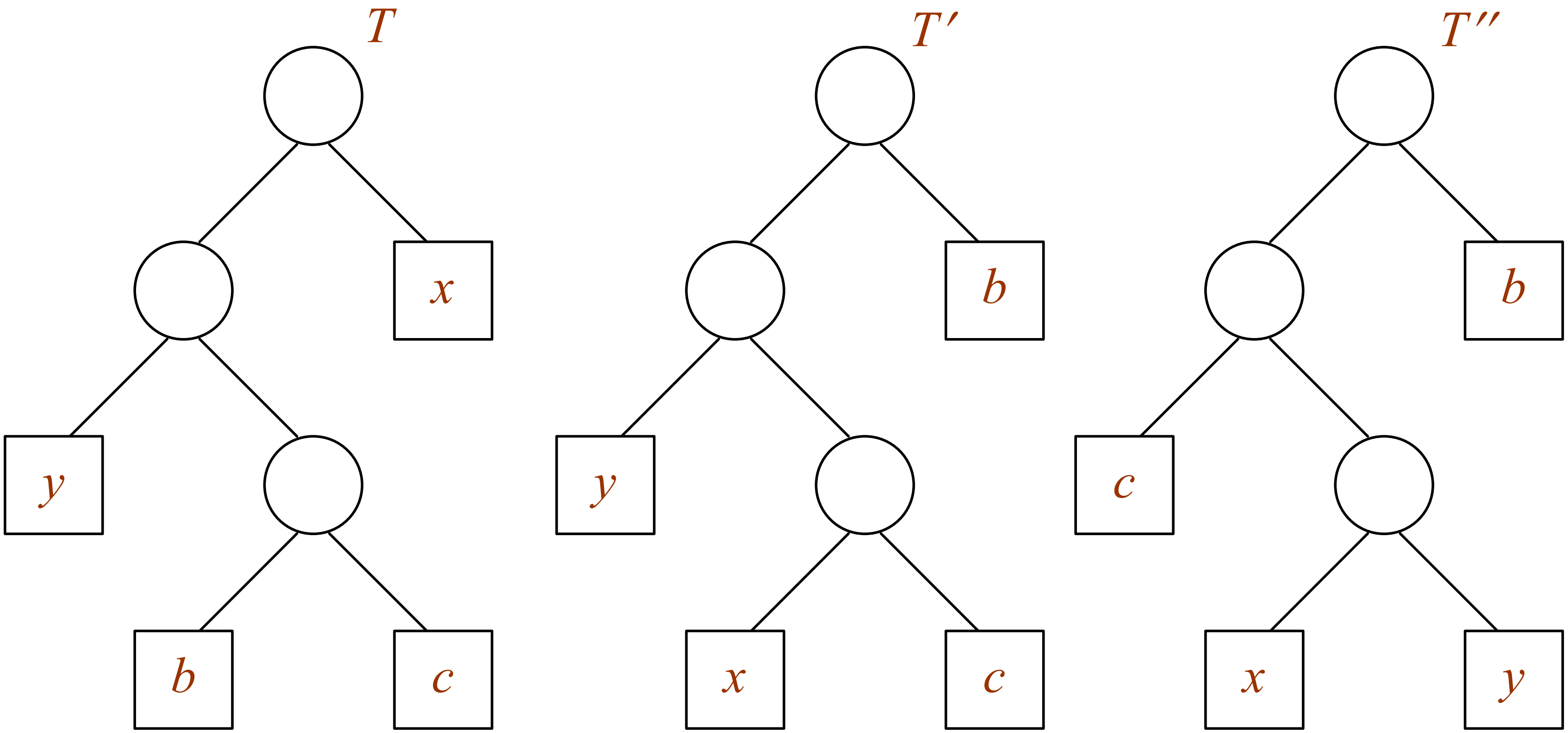
Proof: Take tree T representing an arbitrary optimal code

Modify T to make a tree representing another optimal code such that characters x & y appear as sibling leaves of max-depth in the new tree

Assume that $f[b] \leq f[c]$ & $f[x] \leq f[y]$

Since $f[x]$ & $f[y]$ are two lowest leaf frequencies, in order, and $f[b]$ & $f[c]$ are two arbitrary leaf frequencies, in order, $f[x] \leq f[b]$ & $f[y] \leq f[c]$

Correctness of Huffman's Algorithm



$T \Rightarrow T'$: exchange the positions of the leaves b & x

$T' \Rightarrow T''$: exchange the positions of the leaves c & y

Greedy-Choice Property of Determining an Optimal Code

Proof of Lemma 1 (continued):

The difference in cost between T and T' is

$$B(T) = B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$$

$$= f[x]d_T(x) + f[b]d_T(b) - f[x]d_{T'}(x) - f[b]d_{T'}(b)$$

$$= f[x]d_T(x) + f[b]d_T(b) - f(x)d_T(b) - f[b]d_T(x)$$

$$= f[b](d_T(b) - d_T(x)) - f[x](d_T(b) - d_T(x))$$

$$= (f[b] - f[x])(d_T(b) - d_T(x)) \geq 0$$

Greedy-Choice Property of Determining an Optimal Code

Proof of Lemma 1 (continued):

Since $f[b] - f[x] \geq 0$ and $d_T(b) \geq d_T(x)$
therefore $B(T') \leq B(T)$

We can similarly show that
$B(T') - B(T'') \geq 0 \Rightarrow B(T'') \leq B(T')$
which implies $B(T'') \leq B(T)$

Since T is optimal $\Rightarrow B(T'') = B(T) \Rightarrow T''$ is also optimal

Greedy-Choice Property of Determining an Optimal Code

Lemma 1 implies that
process of building an optimal tree by mergers can begin with the greedy choice of merging those two characters with the lowest frequency

We have already proved that $B(T) = \sum_{i \in I_T} w(i)$, that is,
the total cost of the tree constructed is the sum of the costs of its mergers (internal nodes) of all possible mergers

At each step Huffman chooses the merger that incurs the least cost

Greedy-Choice Property of Determining an Optimal Code

Lemma 2: Consider any two characters x & y that appear as sibling leaves in optimal T and let z be their parent

Then, considering z as a character with frequency $f[z] = f[x] + f[y]$

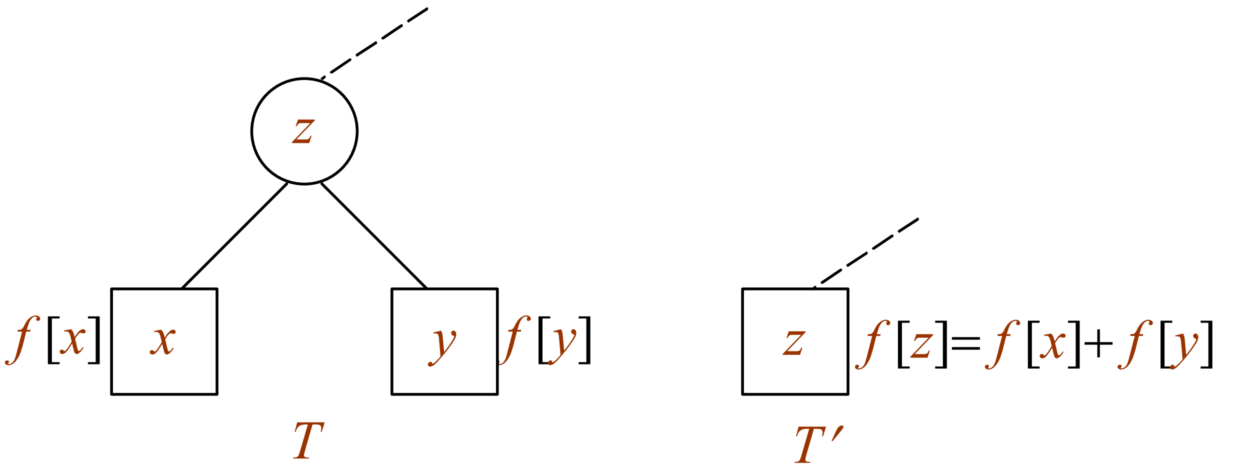
The tree $T' = T - \{x, y\}$ represents an optimal prefix code for the alphabet $C' = C - \{x, y\} \cup \{z\}$

Greedy-Choice Property of Determining an Optimal Code

Proof: Try to express cost of T in terms of cost of T'

For each $c \in C' = C - \{x, y\}$ we have

$$d_T(c) = d_{T'}(c) \Rightarrow f(c)d_T(c) = f(c)d_{T'}(c)$$



$$B(T) = B(T') + f[x](d_T(z)+1) + f[y](d_T(z)+1) + f[z]d_T(z)$$
$$= B(T') + f[z](d_T(z)+1) - f[z]d_T(z)$$
$$= B(T') + f[z] = B(T') + f[x] + f[y]$$

Greedy-Choice Property of Determining an Optimal Code

Proof (continued): If T' represents a nonoptimal prefix code for the alphabet C'

Then, $\exists$ a tree T'' whose leaves are characters in C' such that $B(T'') < B(T')$

Since z is a character in C', it appears as a leaf in T''

If we add x & y as children of z in T'' then we obtain a prefix code for x with cost

$$B(T'') + f[x] + f[y] < B(T') + f[x] + f[y] = B(T)$$

contradicting the optimality of T