IV Advanced Design and Analysis Techniques

Elements of a Greedy Strategy

1. Determine the optimal substructure of the problem.
2. Develop a recursive solution. Show that if we make the greedy choice, then only one subproblem remains.
3. Prove that it is always safe to make the greedy choice.
4. Develop a recursive algorithm that implements the greedy strategy.
5. Convert the recursive algorithm to an iterative algorithm.

(Steps 3 and 4 can occur in either order.)

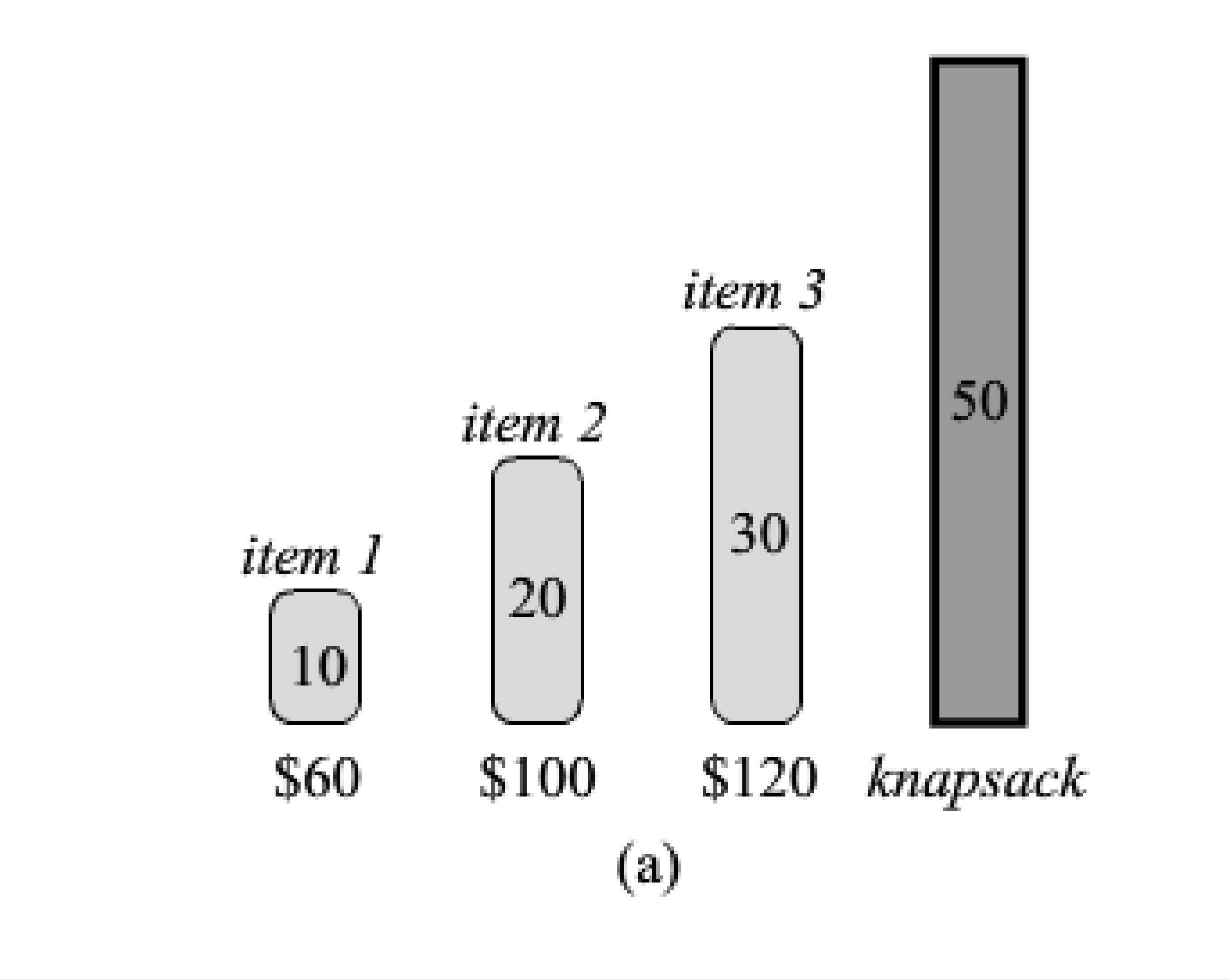
Greedy Algorithm Design

1. Cast the optimization problem as :
   - One in which we make a choice
   - And are left with one subproblem to solve.
2. Prove that there is always an optimal solution to the original problem that makes the greedy choice
   - The greedy choice is always safe.
3. Demonstrate optimal substructure by
   - Showing that by having made the greedy choice,
   - What remains is a subproblem with the property
     - IF we combine an optimal solution to the subproblem with the greedy choice we have made,
     - THEN we arrive at an optimal solution to the original problem.

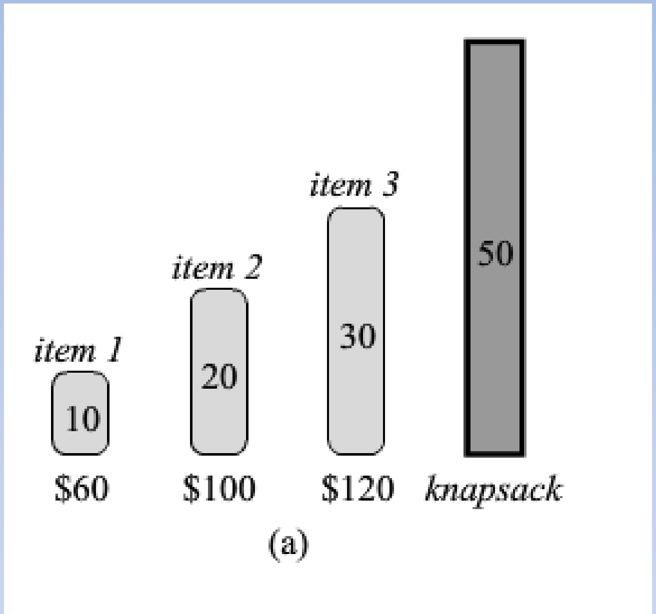
Key Properties for Greedy

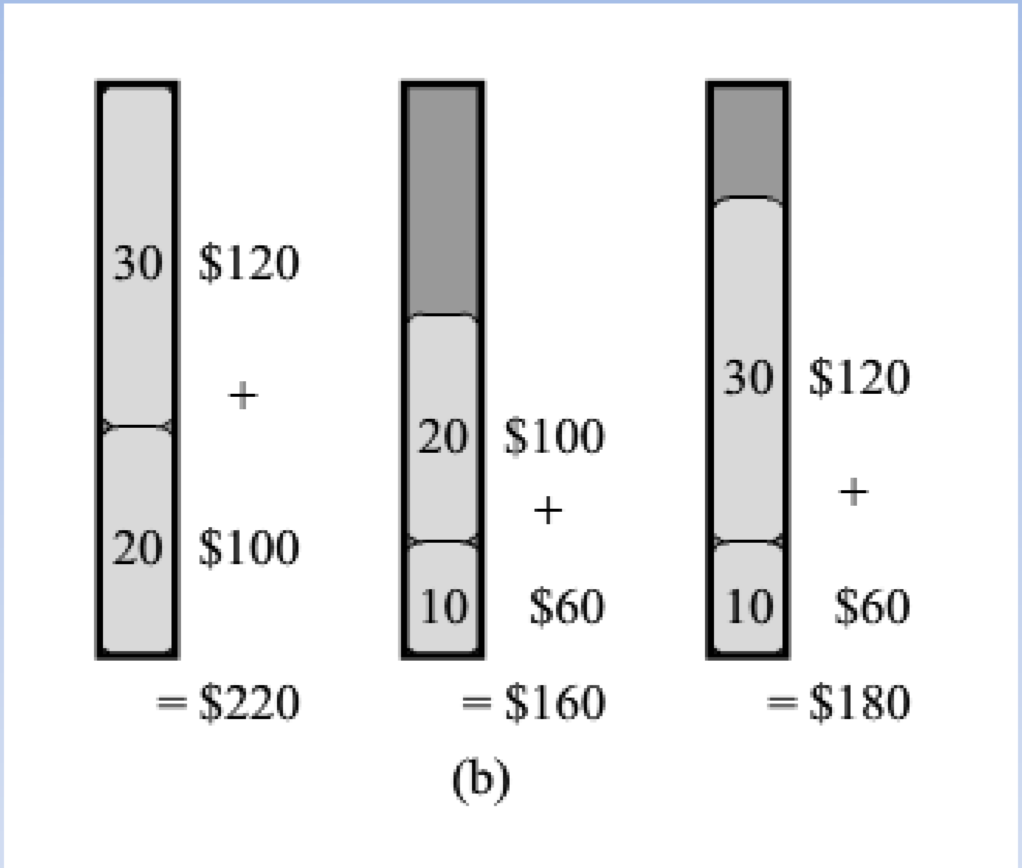
- Greedy-Choice Property:
  - We can assemble a globally optimal solution by making locally optimal (greedy) choices.
- Optimal Substructure:
  - A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.

Knapsack Problem



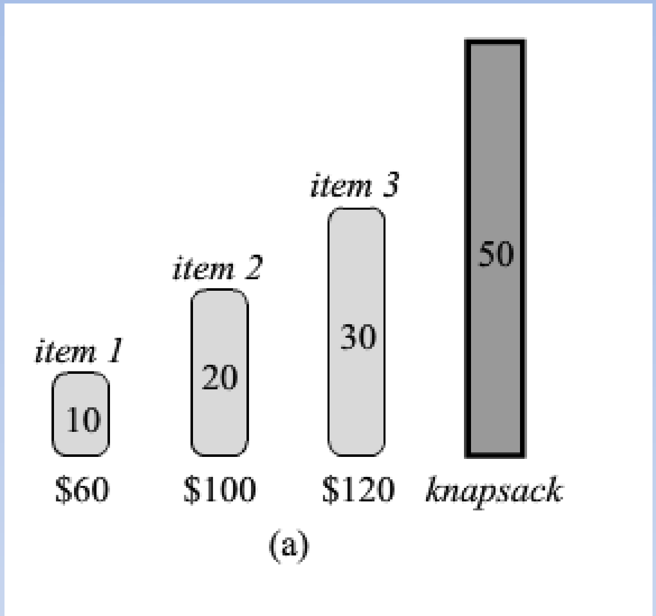
0-1 Knapsack Problem

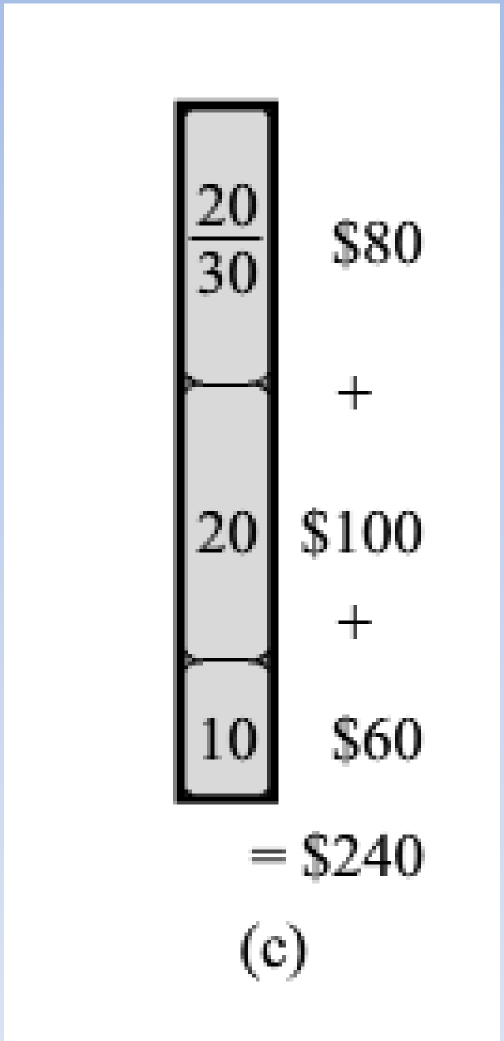




- Item 1
  - Per/lb Most Expensive
- Optimal Solution does not use Item 1

Fractional Knapsack





Huffman Codes
Ch#16.3

- Compress data well
- Used frequently because of simplicity
- Also combined with other methods.

Example Problem & Huffman Codes

- 100,000-Character File
- 6 different characters appear
- ‘a’ appears 45,000 times.

- Need to design a Binary Character Code.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Figure 16.3 A character-coding problem. A data file of 100,000 characters contains only the characters `a`–`f`, with the frequencies indicated. If we assign each character a 3-bit codeword, we can encode the file in 300,000 bits. Using the variable-length code shown, we can encode the file in only 224,000 bits.

- Fixed length code requires 300,000 bits!
  - Can we do better!!
- Variable length code provide uses only 224,000 bits.
- How can we develop a variable length code?

Prefix Codes

- No code word is also a prefix of any other code word.
  - A=0
  - B=101
  - C=100
  - D=111
  - E=1101
  - F=1100

Decoding Prefix Codes

- Code String: 001011101
- Code:
  - A=0,
  - B=101, C=100, D=111
  - E=1101, F=1100
- Uniquely Decodes As:
  - 0 => A
  - 0 => A
  - 101 => B
  - 1101 => E

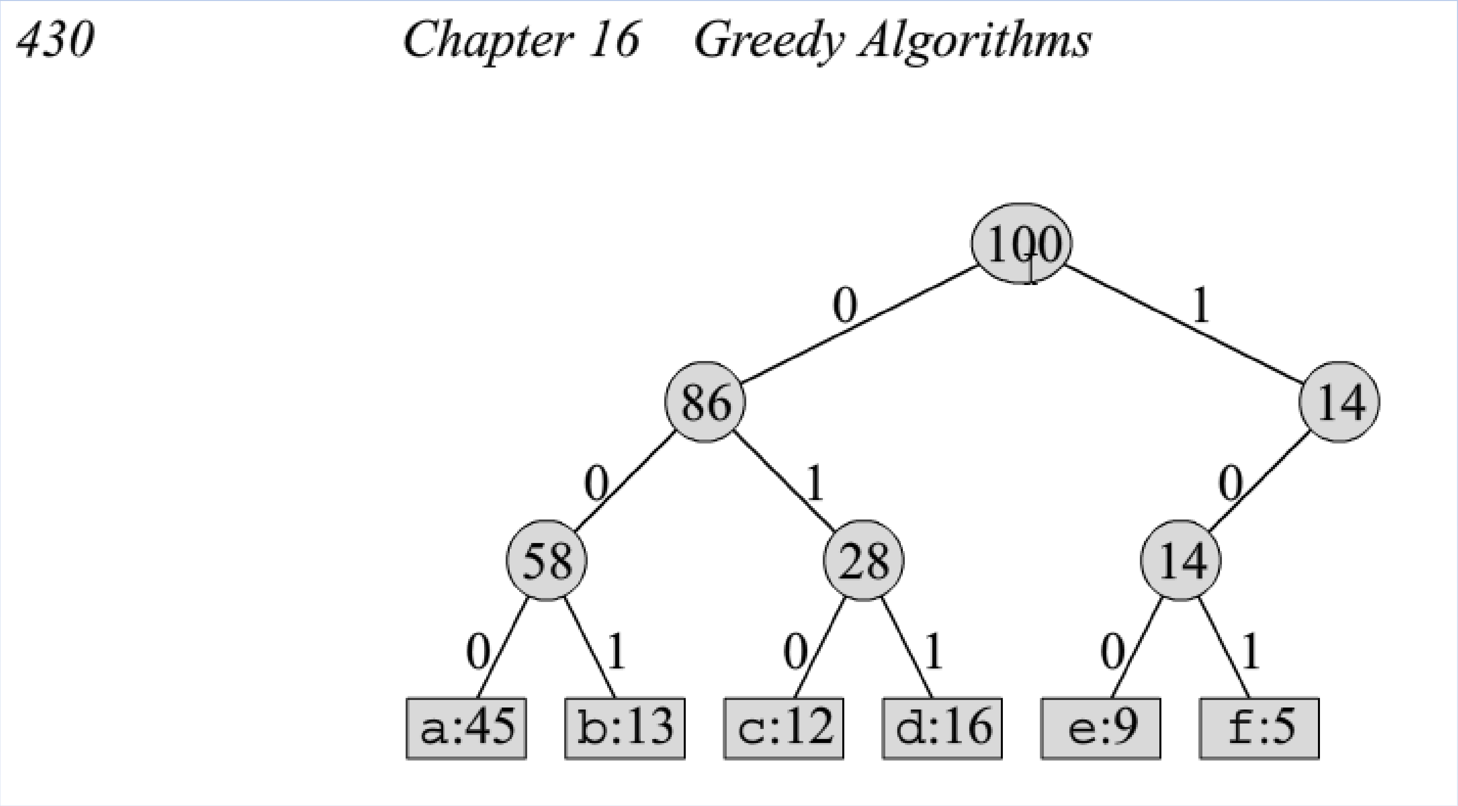
Decoding Prefix Codes

- Code String: 001011101
- Code:
  - A=0,
  - B=101, C=100, D=111
  - E=1101, F=1100
- Uniquely Decodes As:
  - 0 => A
  - 0 => A
  - 101 => B
  - 1101 => E

- Need an Efficient Data Structure to support decoding!

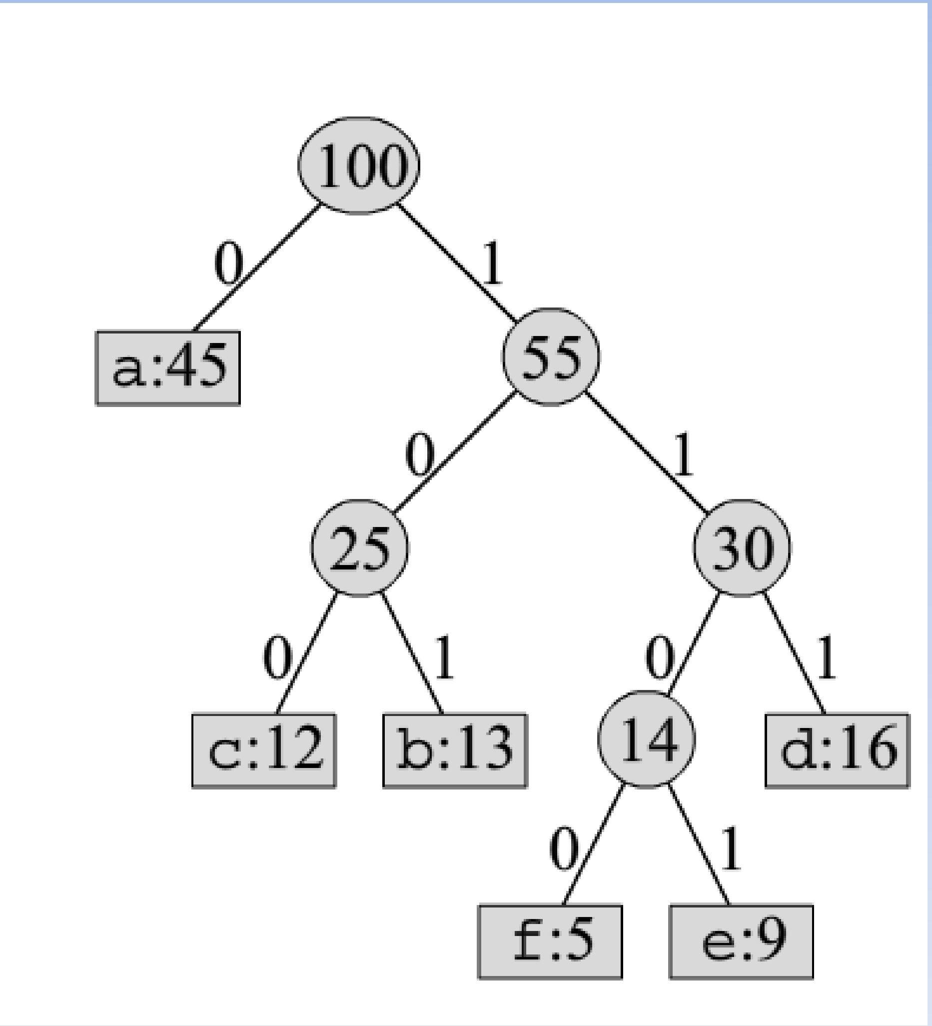
Prefix Codes w/ Binary Tree

- Code 1:
  - A=000, B=001, C=010,
  - D=011 E=100, F=101



Prefix Codes w/ Binary Tree

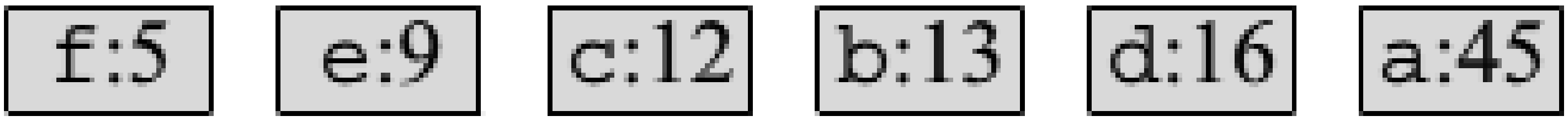
- Code:
  - A=0,
  - B=101, C=100, D=111
  - E=1101, F=1100



Huffman Algorithm

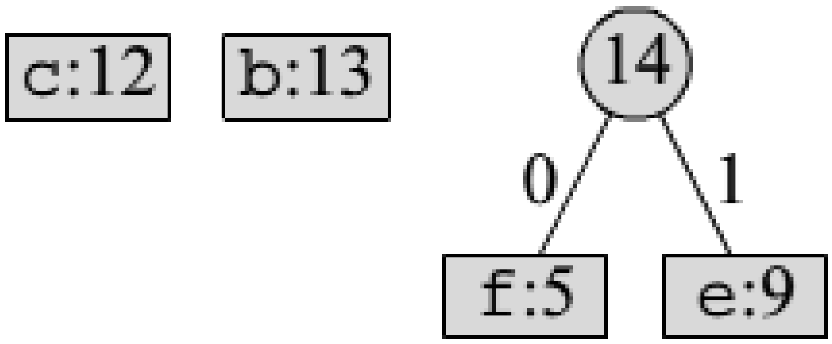
```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n - 1
    allocate a new node z
    z.left = x = EXTRACT-MIN(Q)
    z.right = y = EXTRACT-MIN(Q)
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)    // return the root of the tree
```

Huffman Algorithm Example



- Initial Characters w/ Frequencies

Huffman Algorithm Example



allocate a new node z

$z.left = x = \text{EXTRACT-MIN}(Q)$

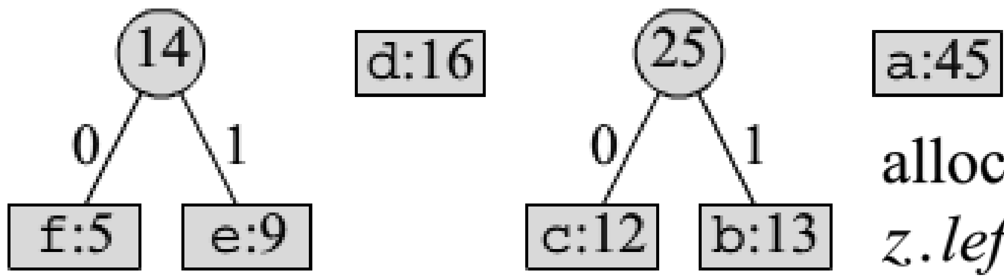
$z.right = y = \text{EXTRACT-MIN}(Q)$

$z.freq = x.freq + y.freq$

$\text{INSERT}(Q, z)$

- First Iteration
  - Allocate Node
  - Extract-Min from Q for Left: [f:5]
  - Extract-Min from Q for Right: [e:9]
  - Sum Frequencies and add New Node to Q
    - [14: [f:5], [e:9]]

Huffman Algorithm Example



allocate a new node z

$z.left = x = \text{EXTRACT-MIN}(Q)$

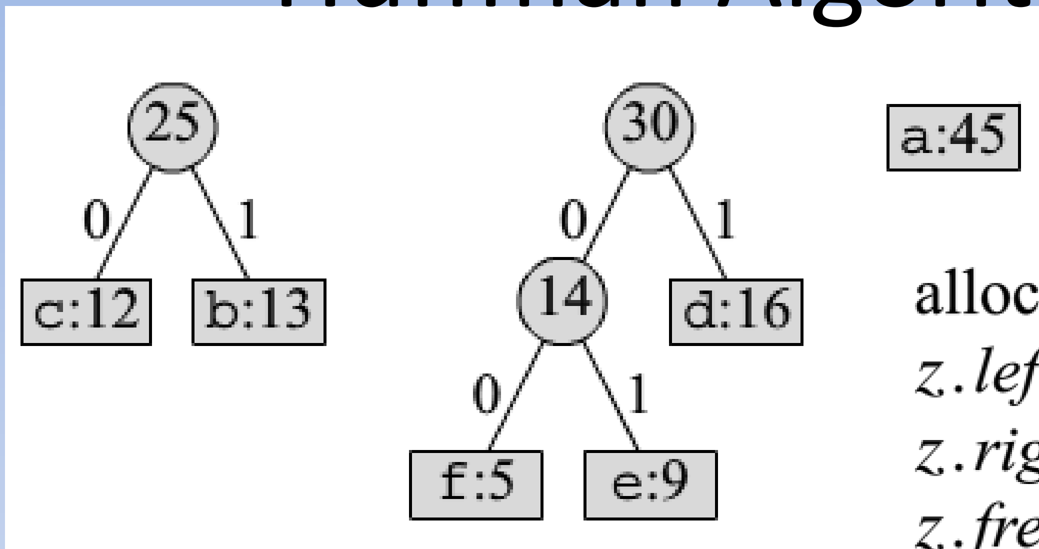
$z.right = y = \text{EXTRACT-MIN}(Q)$

$z.freq = x.freq + y.freq$

$\text{INSERT}(Q, z)$

- Second Iteration
  - – Allocate Node
  - – Extract-Min from Q for Left: [c:12]
  - – Extract-Min from Q for Right: [b:13]
  - – Sum Frequencies and add New Node to Q
    - [25: [c:12], [b:13]]

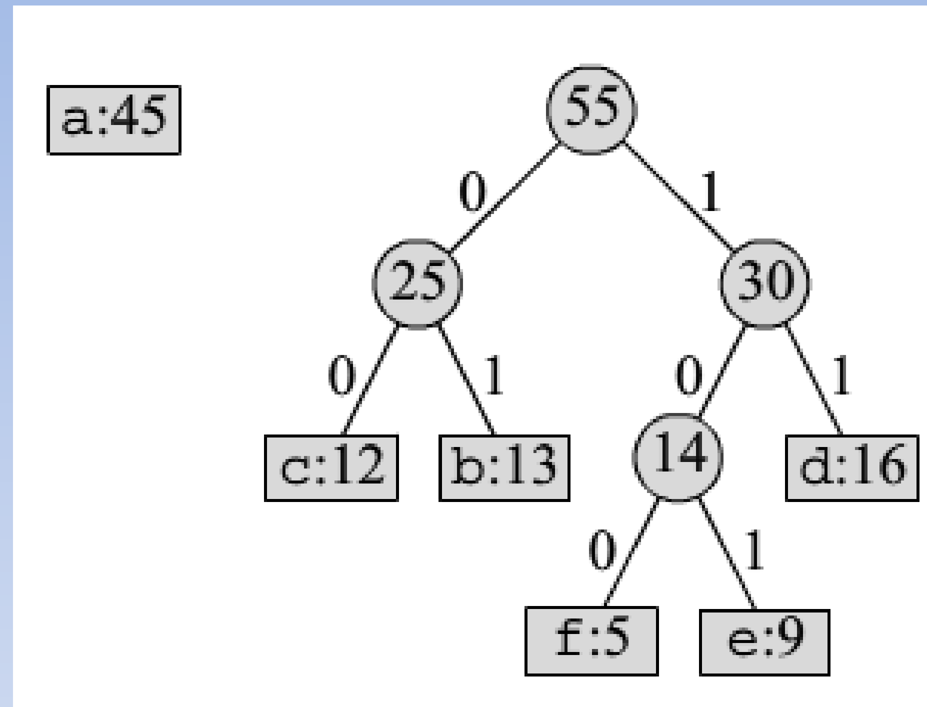
Huffman Algorithm Example



allocate a new node z
$z.left = x = \text{EXTRACT-MIN}(Q)$
$z.right = y = \text{EXTRACT-MIN}(Q)$
$z.freq = x.freq + y.freq$
$\text{INSERT}(Q, z)$

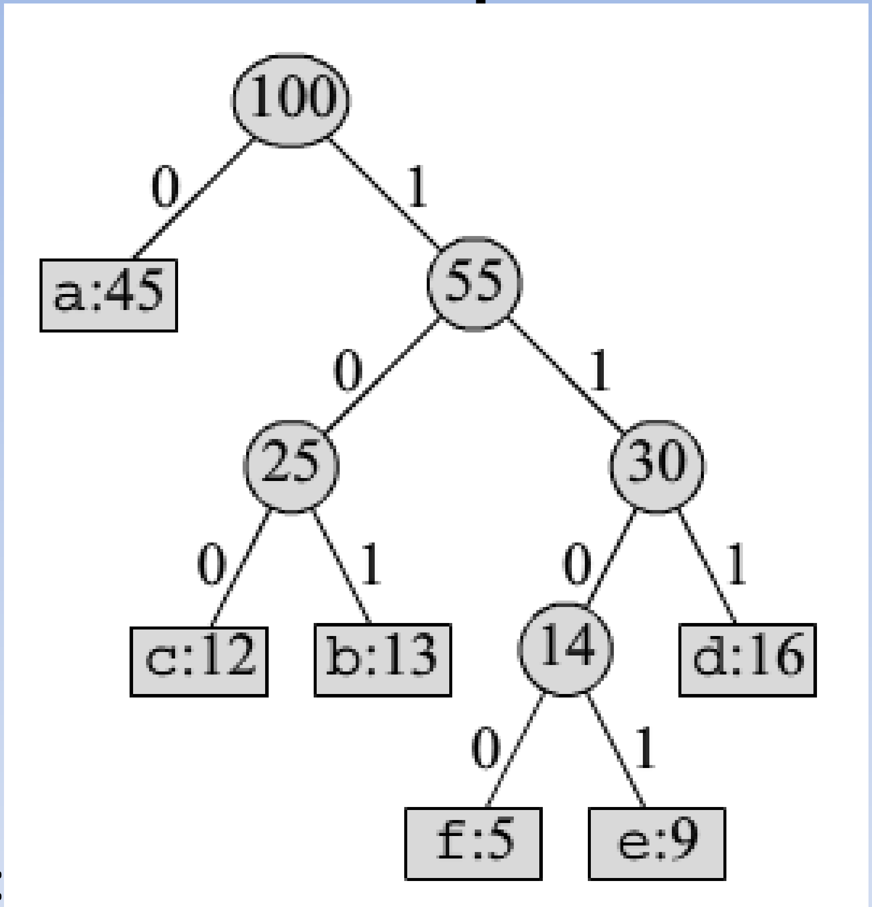
- Third Iteration
  - Allocate Node
  - Extract-Min from Q for Left: [14: [f:5], [e:9]]
  - Extract-Min from Q for Right: [d:16]
  - Sum Frequencies and add New Node to Q
    - [30: [14: [f:5], [e:9]], [d:16]]

Huffman Algorithm Example



- Fourth Iteration
  - Allocate Node
  - Extract-Min from Q for Left:
    - [25: [c:12], [b:13]]
  - Extract-Min from Q for Right:
    - [30: [14: [f:5], [e:9]], [d:16]]
  - Sum Frequencies and add New Node to Q
    - [55: [25: [c:12], [b:13]], [30: [14: [f:5], [e:9]], [d:16]]]

Huffman Algorithm Example



- Final Iteration
  - Allocate Node
  - Extract-Min from Q for Left:
    - [a: 45]
  - Extract-Min from Q for Right:
    - [55: [25: [c:12], [b:13]], [30: [14: [f:5], [e:9]], [d:16]]]
  - Sum Frequencies and add New Node to Q
    - [100: [a:45], [55: [25: [c:12], [b:13]], [30: [14: [f:5], [e:9]], [d:16]]]]

Proof of Correctness

- Lemma 16.2
  - Let C be an alphabet
    - each character $c \in C$ has frequency c.freq.
  - Let x and y be two characters in C having the lowest frequencies.
    - Then there exists an optimal prefix code for C in which the code words for x and y have the same length and differ only in the last bit.

Proof of Correctness

- Proof
  - Take the tree T representing an arbitrary optimal prefix code
  - Modify T to make a tree representing another optimal prefix code such that
    - the lowest frequency characters x and y appear as sibling leaves of maximum depth in the new tree.
  - If we can construct such a tree, then the code words for x and y will have the same length and differ only in the last bit.

Bits required to encode a file

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c) \;, \tag{16.4}$$

- $d_t(c)$ = depth of character c in tree.
- $d_t(c)$ = length of codeword for c.

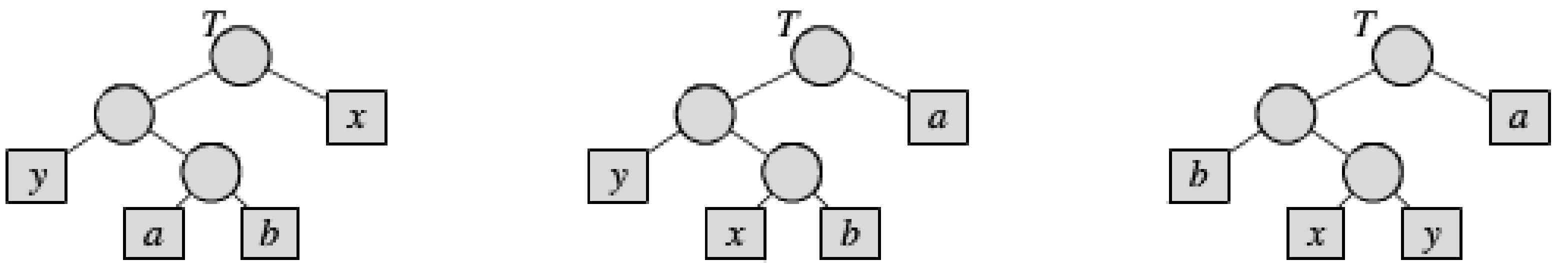
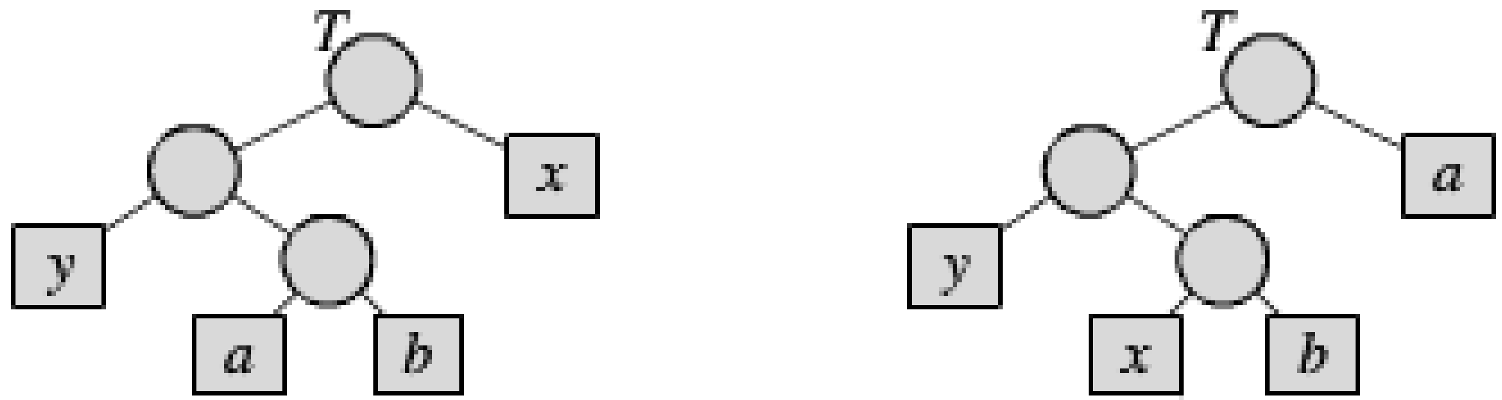


Figure 16.6 An illustration of the key step in the proof of Lemma 16.2. In the optimal tree T, leaves a and b are two siblings of maximum depth. Leaves x and y are the two characters with the lowest frequencies; they appear in arbitrary positions in T. Assuming that $x \neq b$, swapping leaves a and x produces tree T', and then swapping leaves b and y produces tree T''. Since each swap does not increase the cost, the resulting tree T'' is also an optimal tree.

$$
\begin{aligned}
B(T) - B(T') \\
&= \sum_{c \in C} c.\mathit{freq} \cdot d_T(c) - \sum_{c \in C} c.\mathit{freq} \cdot d_{T'}(c) \\
&= x.\mathit{freq} \cdot d_T(x) + a.\mathit{freq} \cdot d_T(a) - x.\mathit{freq} \cdot d_{T'}(x) - a.\mathit{freq} \cdot d_{T'}(a) \\
&= x.\mathit{freq} \cdot d_T(x) + a.\mathit{freq} \cdot d_T(a) - x.\mathit{freq} \cdot d_T(a) - a.\mathit{freq} \cdot d_T(x) \\
&= (a.\mathit{freq} - x.\mathit{freq})(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$



$$
\begin{aligned}
B(T) - B(T') & \\
&= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_{T'}(x) - a.freq \cdot d_{T'}(a) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_T(a) - a.freq \cdot d_T(x) \\
&= (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\geq 0 ,
\end{aligned}
$$

- a is at maximum depth
- Since frequency of x is lower than a
  - Swaping x and a CANNOT increase bits required to encode!

Implications of Lemma 16.2

- Lemma 16.2 implies that building up an optimal tree by mergers can begin with the greedy choice of merging together those two characters of lowest frequency.
- Why is this a greedy choice?
  – We can view the cost of a single merger as being the sum of the frequencies of the two items being merged.

Lemma 16.3

- Let C be a given alphabet with frequency c.freq defined for each character $c \in C$.
- Let x and y be two characters in C with minimum frequency.
  - Let C' be the alphabet C with the characters x and y removed and a new character z added, so that $C' = C - \{x,y\} \cup \{z\}$.
- Define freq for C' as for C, except
  - z.freq = x.freq + y.freq.
- Let T' be any tree representing an optimal prefix code for the alphabet C'.
  - Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C.

Theorem 16.4

- Procedure HUFFMAN produces an optimal prefix code.
- Proof: Follows immediately from Lemma 16.2 & 16.3