# Lecture 29: Greedy Algorithms II

2023/10/19

詹博华（中国科学院软件研究所）

# Huffman encoding

- Suppose we wish to compress a file. There are $n$ kinds of characters in the file, each character has a given frequency.
- For example: there are $6$ kinds of characters labeled $\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}$. They have frequency 45, 13, 12, 16, 9, 5, respectively.
- Assign a binary code (a unique binary string) for each character, in order to minimize the total size of coding for the file.

# Huffman encoding: example

| | 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 |

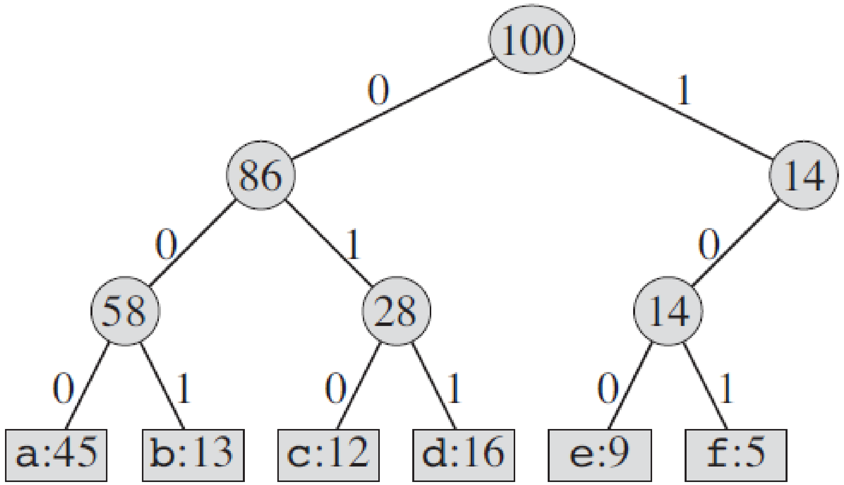
- **Fixed-length codeword:** assign each letter to the same number of bits.
  - With the fixed-length encoding above, the string $\mathbf{abd}$ is encoded as $\mathbf{000001011}$.
- **Variable-length codeword:** assign each letter a codeword with possibly different number of bits.
  - With the variable-length encoding above, the string $\mathbf{abd}$ is encoded as $\mathbf{0101111}$.

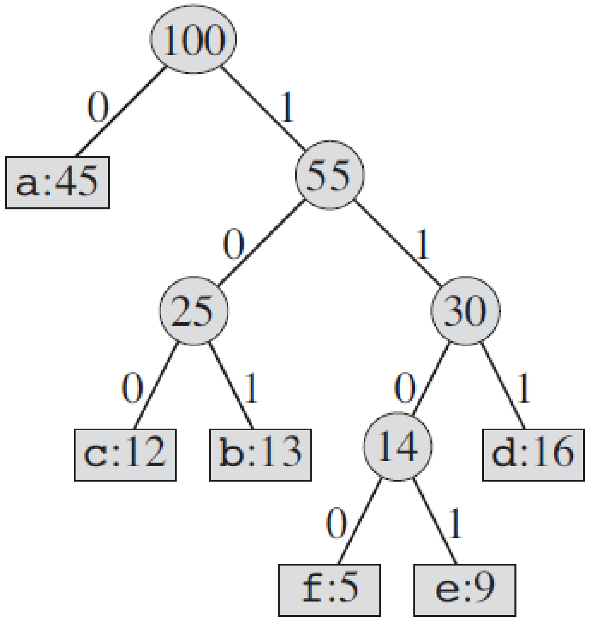
# Prefix code

- **Prefix code:** a code is a prefix code if no codeword is a prefix of another. This simplifies decoding of a file.
- Example: given the prefix code
$$\mathrm{a} \rightarrow 0, \mathrm{b} \rightarrow 101, \mathrm{c} \rightarrow 100, \mathrm{d} \rightarrow 111, \mathrm{e} \rightarrow 1101, \mathrm{f} \rightarrow 1100$$
- The codeword
$$001011101$$
can be easily decoded as $\mathrm{aabe}$.

# Prefix code as binary trees

- Represent each codeword as a path from root to leaf of the tree.
- Example: the fixed-length code $\mathrm{a} \rightarrow \mathbf{000}, \mathrm{b} \rightarrow \mathbf{001} \ldots$ on the left, the variable-length code $\mathrm{a} \rightarrow \mathbf{0}, \mathrm{b} \rightarrow \mathbf{101} \ldots$ on the right.



# Cost of a binary tree

- Cost of a binary tree computes the average depth weighted by frequency:

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

- This corresponds to the size of the compressed file.

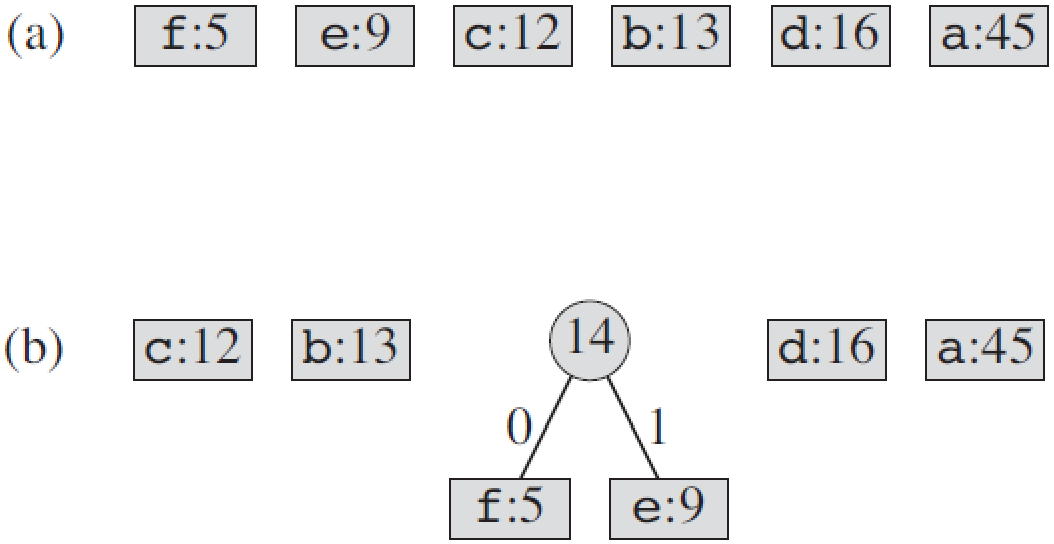
# Greedy algorithm for optimal encoding

- **Huffman encoding:** start from the leaves of the tree labeled by frequencies, gradually build the tree by combining nodes. Each time, combine the two nodes with smallest total frequency.

```
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
```

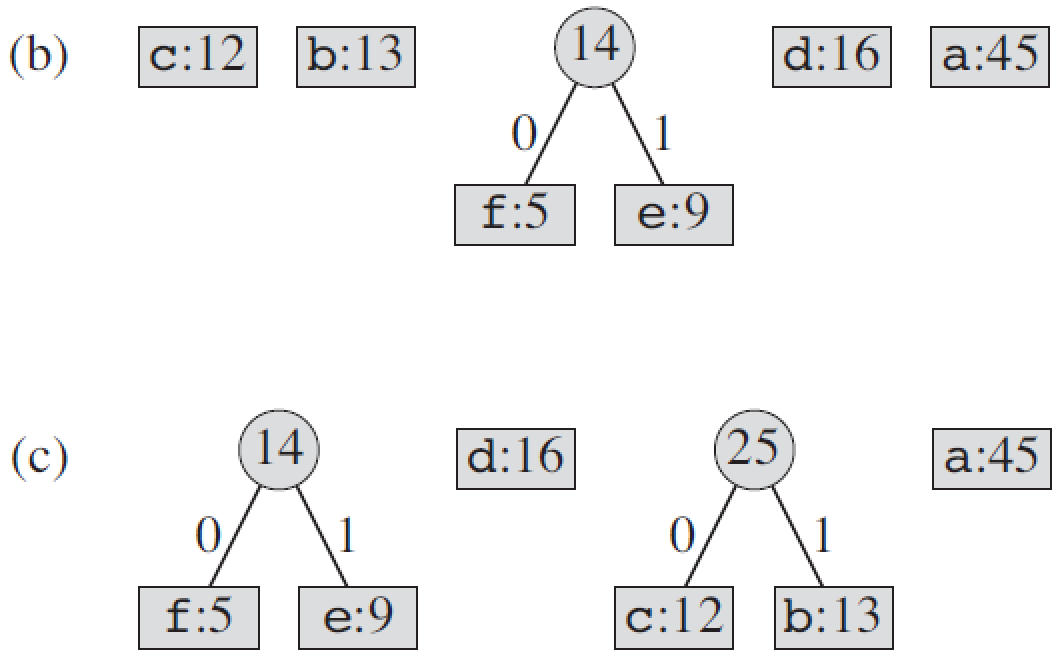
# Huffman encoding: example

- **Step 1:** $\{f\}$ and $\{e\}$ are the smallest, with frequency $5$ and $9$, respectively. They are combined to form node with frequency $14$.



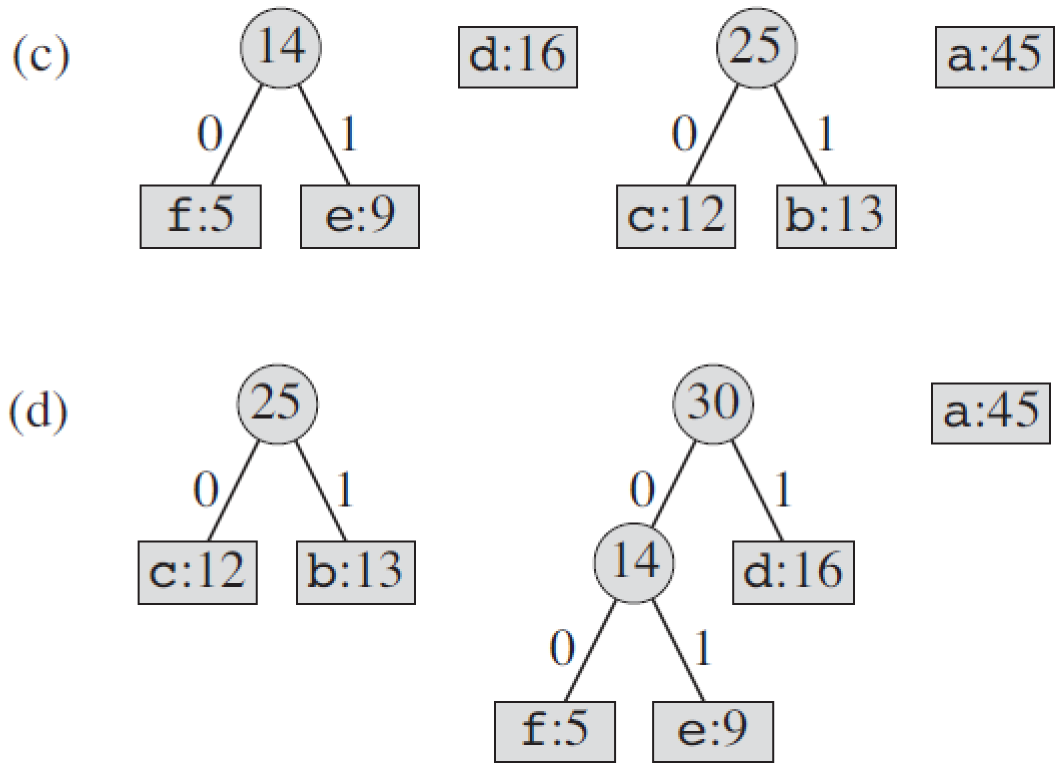
# Huffman encoding: example

- **Step 2:** $\{b\}$ and $\{c\}$ are now the smallest, with weight $12$ and $13$, respectively. They are combined to form node with frequency $25$.



# Huffman encoding: example

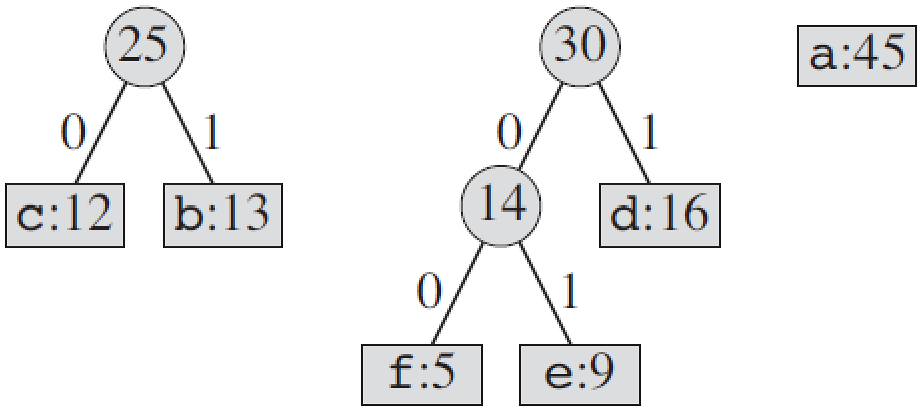
- **Step 3:** $\{e, f\}$ and $\{d\}$ are now the smallest, with frequency $14$ and $16$, respectively.



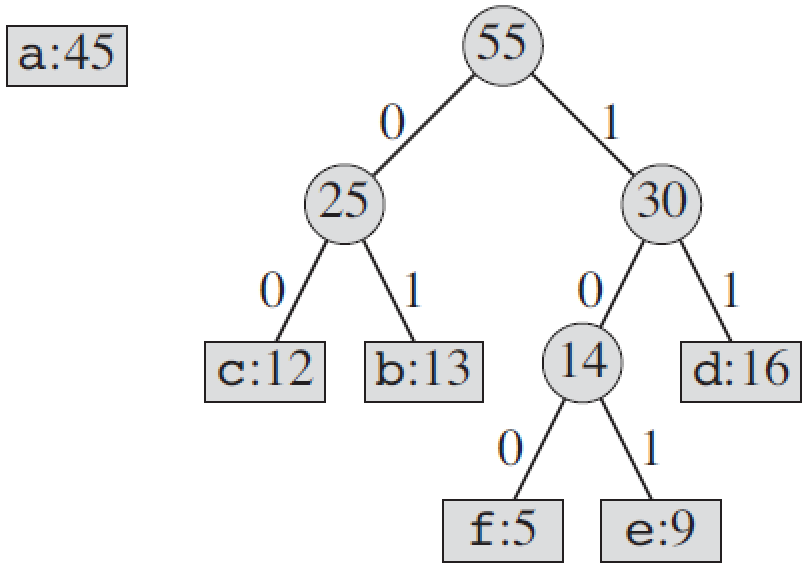
# Huffman encoding: example

- **Step 4:** The nodes $\{b, c\}$ and $\{d, e, f\}$ are the smallest, with frequency $25$ and $30$, respectively.

(d)

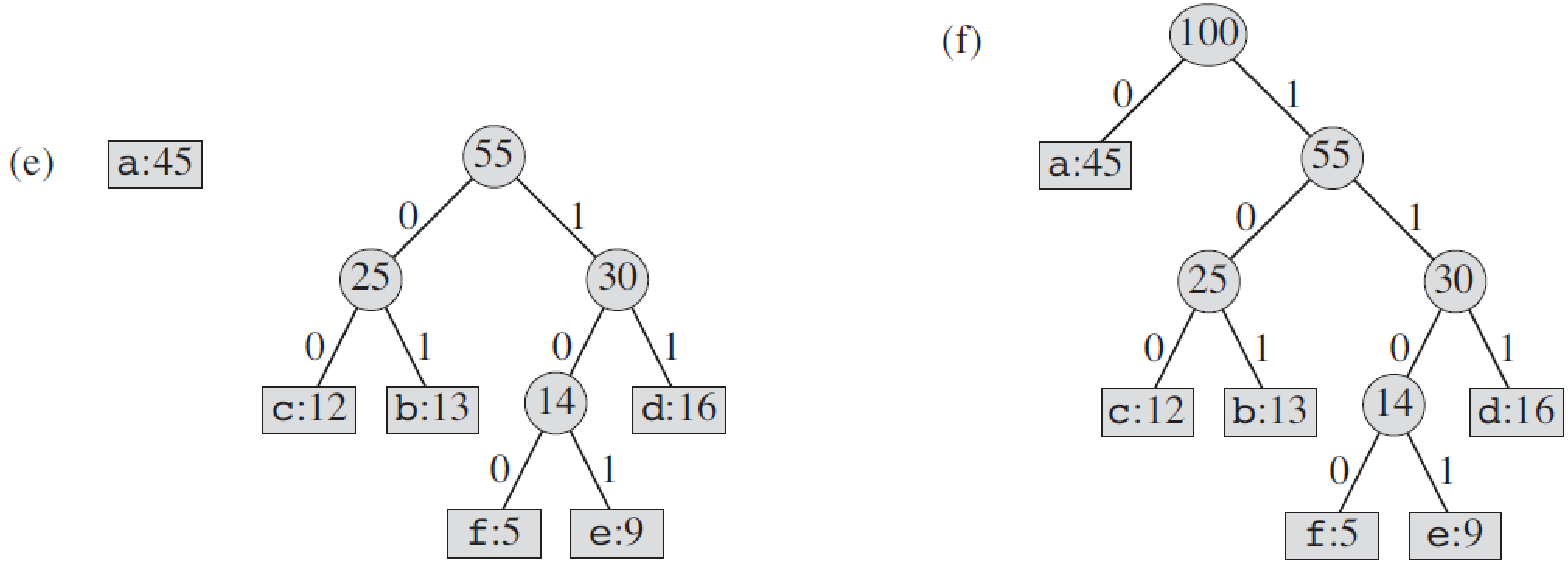



(e)



# Huffman encoding: example

- **Step 5:** finally, combine the node with {a}.



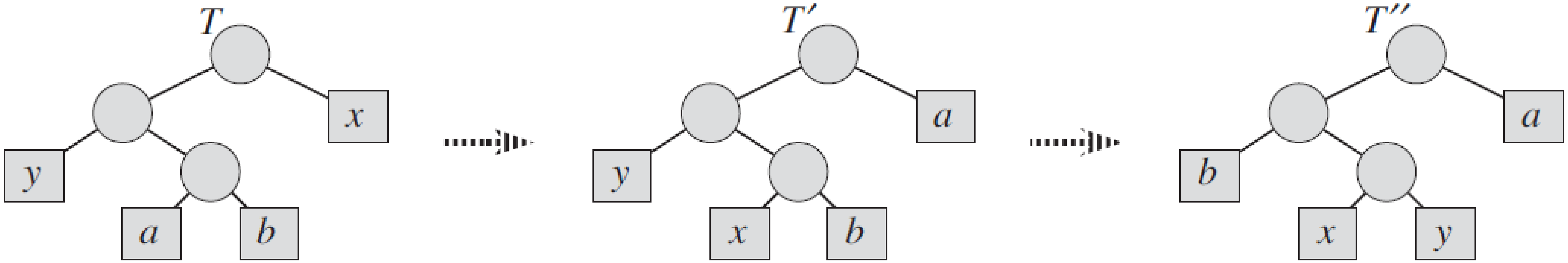
# Why is greedy correct?

**The proof proceeds by the following two steps:**

1. **Greedy-choice property:** it is always optimal to put the two nodes with lowest frequencies together in a single subtree with maximum depth. (Lemma 16.2)
2. **Optimal substructure:** the problem can be reduced to the case after merging the two nodes with lowest frequency into a single node. (Lemma 16.3).

# Greedy-choice property (Lemma 16.2)

- Suppose $a$ and $b$ are nodes with maximum depth, $x$ and $y$ are nodes with lowest frequencies. Assuming $x \neq b$, swap $a$ with $x$ and $b$ with $y$ to yield a tree that has equal or lower cost.




- We get:

$$B(T) - B(T') = (a.\mathrm{freq} - x.\mathrm{freq})\big(d_T(a) - d_T(x)\big) \geq 0,$$

and similarly $B(T') - B(T'') \geq 0$.

# Optimal substructure (Lemma 16.3)

- Consider removing the two characters with lowest frequency $x$ and $y$, replacing with a new character $z$:

$$C' = C - \{x, y\} \cup \{z\},$$

$$z.\mathrm{freq} = x.\mathrm{freq} + y.\mathrm{freq}.$$

- 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 a subtree containing $x$ and $y$ represents an optimal prefix code for the alphabet $C$.
- Key idea: $B(T)$ and $B(T')$ are related by

$$B(T) = B(T') + x.\mathrm{freq} + y.\mathrm{freq}.$$

# Final Theorem

- Huffman encoding algorithm produces an optimal prefix code.
- **Proof by induction:** the first step is optimal (greedy-choice property). The remaining steps can be viewed as carried out on the reduced tree $T'$ and alphabet $C'$, and are optimal by induction hypothesis.