Australian National University

COMP3600/6466 Algorithms

Lecture 14

S2 2016

Dr. Hassan Hijazi

Prof. Weifa Liang

Application 5: The Activity-Selection

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Activity i has a **start time** s_i and a **finish time** f_i

Problem: select a maximum-size subset of *mutually compatible* activities.

Activities a_i and a_j are **compatible** if $s_i \geq f_j$ or $s_j \geq f_i$.

Example: The subset $\boxed{\{a_3, a_9, a_{11}\}}$ consists of mutually compatible activities

The subset $\boxed{\{a_1, a_4, a_8, a_{11}\}}$ is a better solution.

The Activity-Selection Problem

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Increasing finish time

Problem: select a maximum-size subset of *mutually compatible* activities.

Optimal substructure?

Introduce activities a_0 with $s_0 = f_0 = 0$ and a_{n+1} with $s_{n+1} = f_{n+1} = f_n$.
Let $S_{i,j}$ denote the set of activities that start after the end of activity a_i and finish before the beginning of activity a_j.

Example: $S_{0,4} = \{a_1\}$, $S_{0,n+1} = \{a_1, \ldots, a_n\}$, $S_{1,11} = \{a_4, a_6, a_7, a_8, a_9\}$.

The Activity-Selection Problem

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Increasing finish time

Problem: select a maximum-size subset of *mutually compatible* activities.

Optimal substructure?

Let A_{ij} denote the maximum set of mutually compatible activities in S_{ij}.

Consider $a_k \in A_{ij}$

It is easy to see that $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$.

$$\implies |A_{ij}| = |A_{ik}| + |A_{kj}| + 1.$$

The Activity-Selection Problem

a_k is not know in advance!

Let $c[i,j]$ denote the size of an optimal solution to S_{ij}

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

$$\text{Optimal solution value} = c[0, n+1]$$

A Greedy Algorithm

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Increasing finish time

No recursive or iterative computation, take a decision NOW.
The decision that maximises your objective function.

I will pick a valid meeting that has the smallest finishing time.

Is this a locally optimal decision?

Yes, if I replace my meeting with any other valid meeting,
I can only reduce my chances of scheduling more meetings.

A Greedy Algorithm

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Increasing finish time

No recursive or iterative computation, take a decision NOW.
The decision that maximises your objective function.

I will pick a valid meeting that has the smallest finishing time.

Will the greedy local decision always lead to a globally optimal solution?

Yes, but we need a formal proof!

Proving that Greedy is Optimal

What is a formal proof?

A proof based on mathematical reasoning

Theorem 1. The subset obtained by always picking the valid activity that finishes first is globally optimal

Proof. Let $S_k = \{a_i \in \{a_1, \ldots, a_n\} : s_i \geq f_k\}$.
S_k is the set of activities that can be scheduled after a_k.
Let $A_k = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$ be an optimal solution for S_k, that is A_k is a maximum-size subset of mutually compatible activities in S_k.
Let $a_{i_1^*}$ be the activity that finishes first in S_k. If $a_{i_1^*}$ is not in A_k, then the set $A_k^* = \{a_{i_1^*}, a_{i_2}, \ldots, a_{i_k}\}$ is also an optimal solution for S_k.
Observe that $A_k^* = \{a_{i_1^*}\} \cup A_{i_1^*} = \{a_{i_2}, \ldots, a_{i_k}\}$. Since the same argument can be applied with $A_{i_1^*}, \ldots A_{i_k^*}$, the proof is complete.

Greedy Algorithms: How to

Formulate your optimisation problem such as:

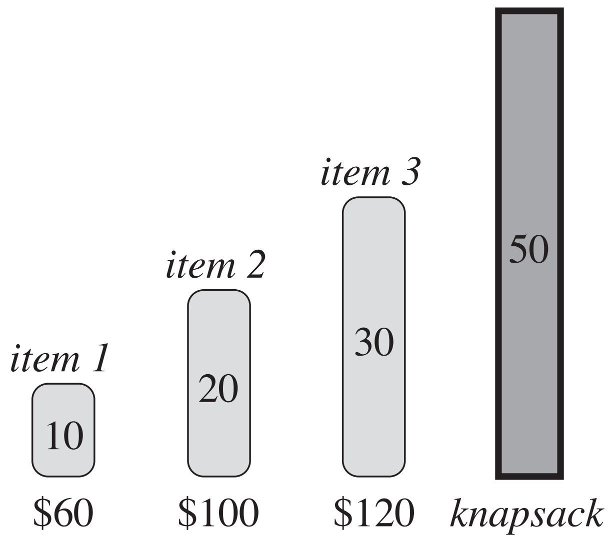
1. If you make a decision NOW, you are left with ONE subproblem to solve.

2. Define your greedy decision

3. Prove that the greedy decision leads to a globally optimal solution

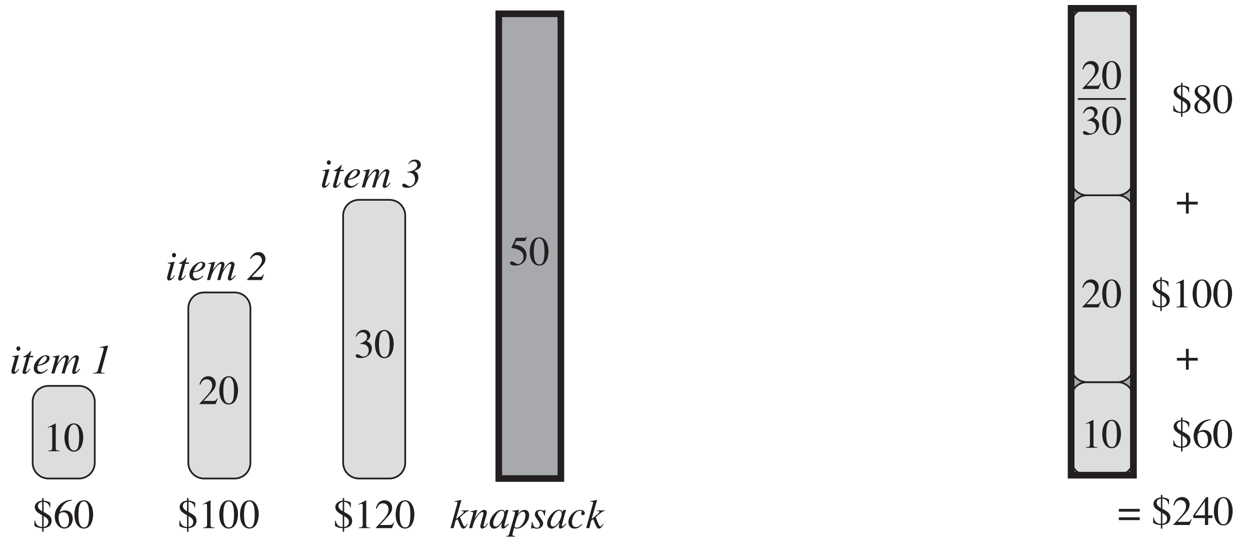
Application 6: The Thief

A thief robbing a store finds n items. The ith item is worth v_i dollars and weighs w_i kilos, where v_i and w_i are integers. The thief wants to take as valuable a load as possible, but he can carry at most W kilos in his knapsack, for some integer W. Which items should he take?



(a)

Case 1. Diamonds (Fractional Items)



(a) (c)

Greedy Algorithms: How to

Formulate your optimisation problem such as:

1. If you make a decision NOW, you are left with ONE subproblem to solve.

$\boxed{\text{Pick ONE element NOW}}$

2. Define your greedy decision

$\boxed{\text{Pick THE MOST valuable element}}$

3. Prove that the greedy decision leads to a globally optimal solution

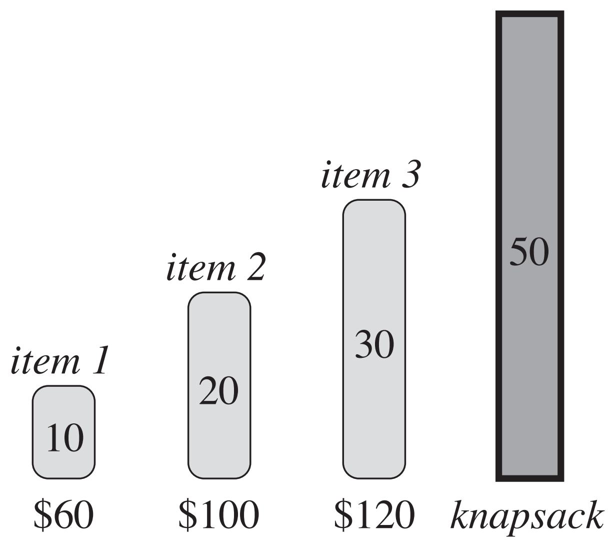
Case 1. Diamonds (Fractional Items)

Pick THE MOST valuable element

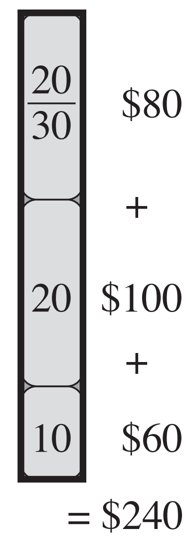
FAIR comparison!

Price per kilo

$$
\begin{aligned}
pu_1 &= 60/10 = 6 \\
pu_2 &= 100/20 = 5 \\
pu_3 &= 120/30 = 4
\end{aligned}
$$



(a)



(c)

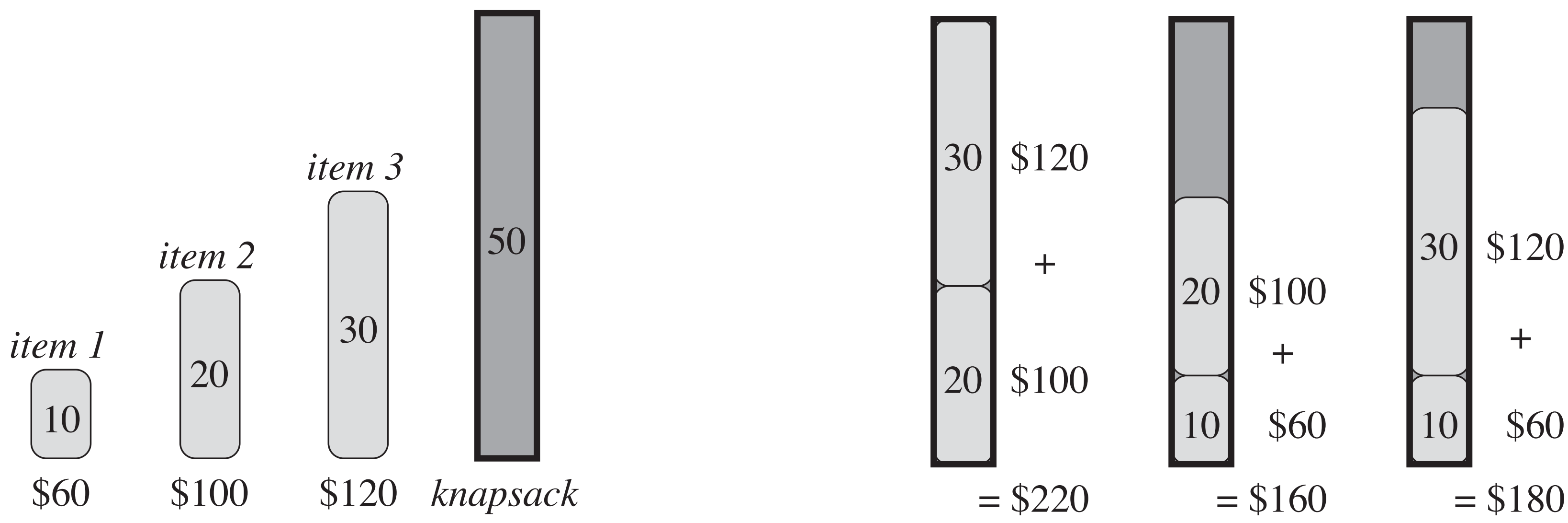
Case 1. Diamonds (Fractional Items)

3. Prove that the greedy decision leads to a globally optimal solution

Theorem 1. The solution obtained by always picking the maximum amount of the most valuable item is globally optimal

Proof. Let $O = \{o_1, o_2, \ldots, o_k\}$ be an optimal solution.
O is a selection of items (can be fractional) that maximises the thief's bounty.
Let o^* be the item with the highest value per kilo.
Let w^* denote the total weight of o^*.
If o^* is not in O, let O^* denote the solution obtained by removing $\min(w^*, W)$ worth of items from O and replacing them by o^*.
By construction, O^* cannot have a total value less than O.
This argument can be applied with the item having the second highest value per kilo, then third highest value, etc.
At every step, I discount the weight inserted in W during the previous step.

Case 2. Cars (Indivisible Items)



(a) (b)

Application 7: Data Compression

Fun Facts

In 1952, David Huffman, a PhD candidate at MIT introduced the Huffman code

Allows for data compression with savings up to 90%

Huffman Coding is based on a greedy algorithm!

Basic Idea

Huffman observed that in a given file, some letters or symbols might appear more than others

IDEA: Encode the high frequency symbols with short binary strings

	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

"abaabef" is encoded as
000001000000001100101 using the fixed-length encoding
01010010111011100 using the variable-length encoding

Prefix-Free Code

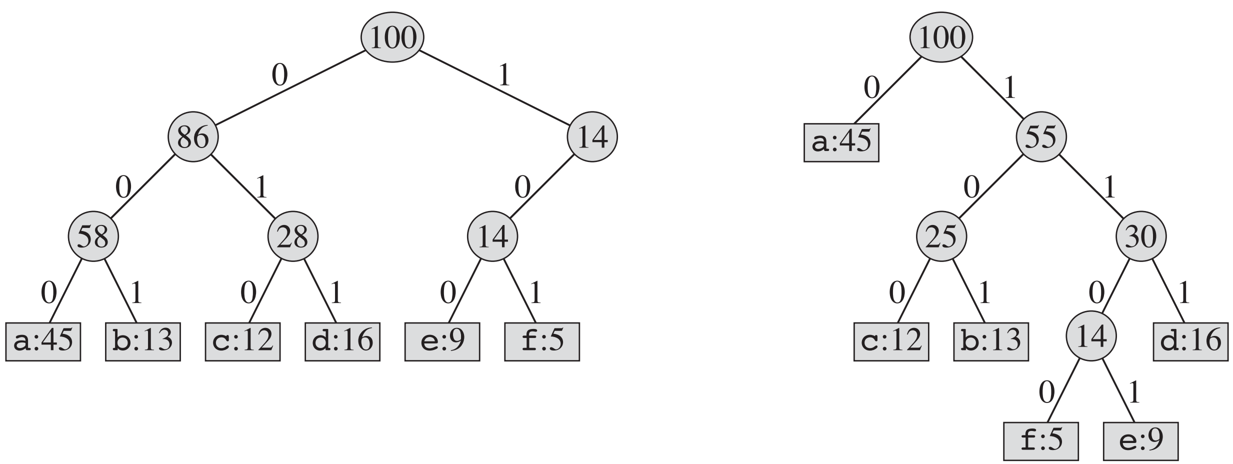
Huffman observed that in order to avoid ambiguous encoding, no codeword can be a prefix of another.

For example, if the codewords are $\{0, 01, 11, 001\}$, the decoding of a string like 001 is ambiguous. You can interpret it as 0-01, or 001.

Prefix-Free Code

Prefix codes are easily representable as binary trees.

	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

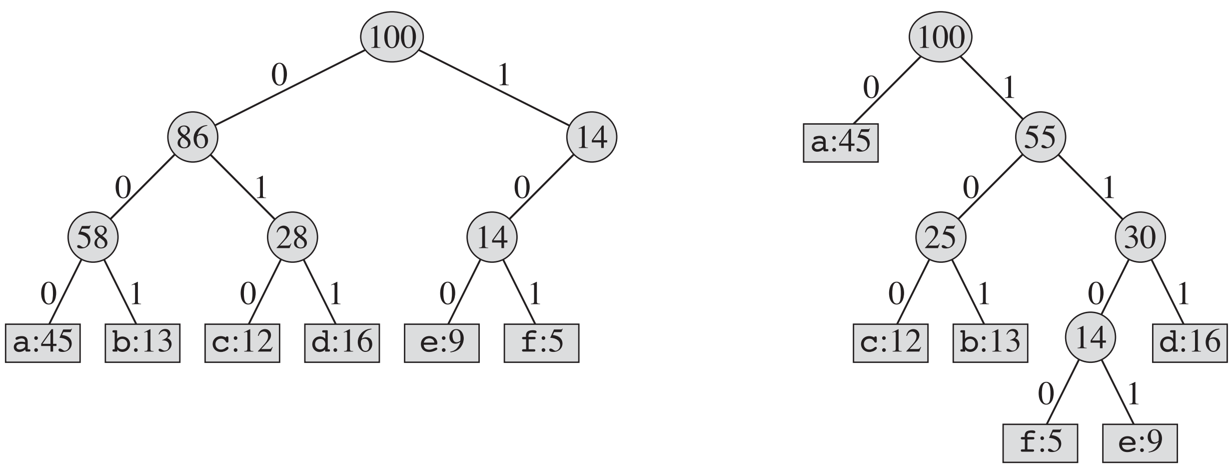


(a) (b)

Prefix-Free Code

Huffman’s idea: Always combine the least frequent pair of elements together

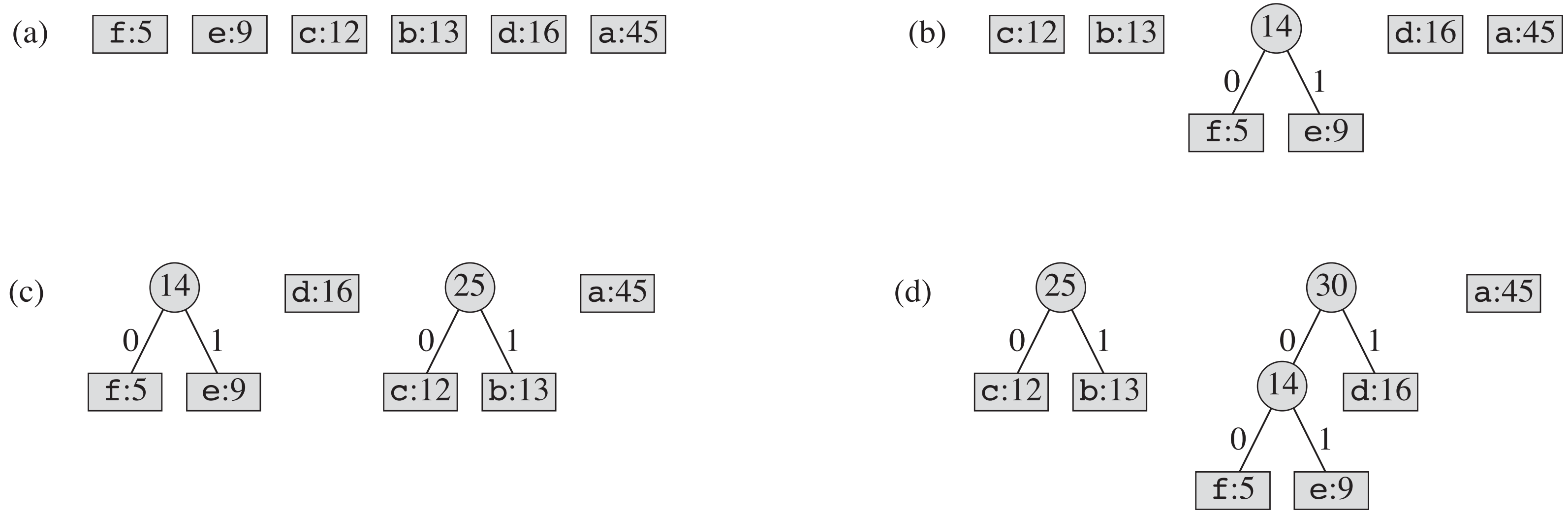
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100



(a) (b)

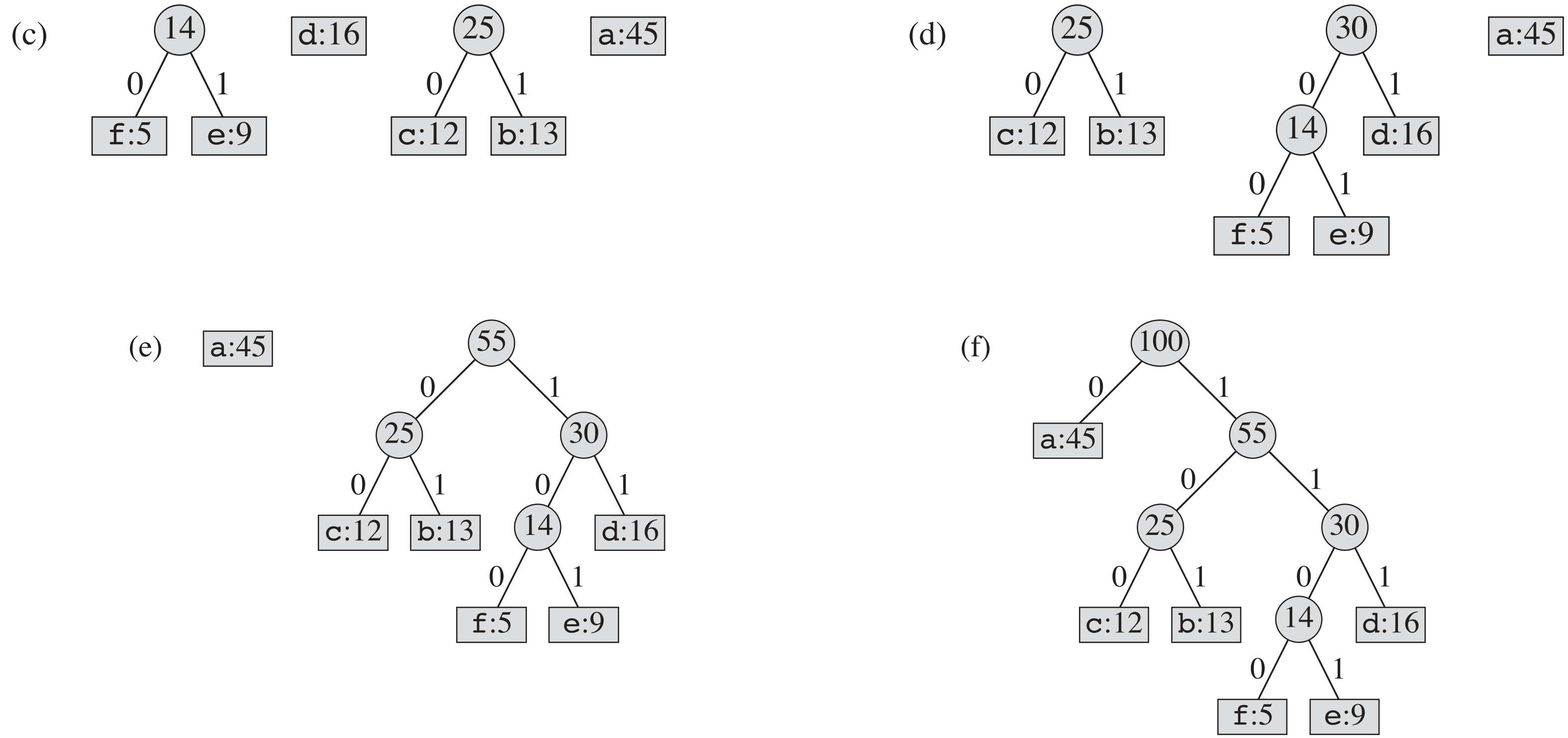
Huffman Greedy Algorithm

Huffman's idea: Always combine the least frequent pair of elements together



Huffman Greedy Algorithm

Huffman's idea: Always combine the least frequent pair of elements together



```
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
```

Sometimes, we have to be greedy!