------------------------------ 16 ------------------------------

Greedy Algorithms

貪 心

Greedy algorithm: one always makes a locally optimal choice. (in the hope that this choice will lead to a globally optimal solution)

(Not always find an optimal solution)

16.1 An activity-selection problem

Input: n activities with start time s_i and finish time f_i

Output: a maximum set of compatible activities

$\begin{cases} O(n \lg n)\text{: real} \\ O(n)\text{: integer sort OK} \end{cases}$

Step 0: Sort the input according to f_i increasingly.

Step 1: Schedule the activities one by one. $O(n)$

(according to f_i's)

Time: $T(n) = \begin{cases} O(n) & \text{if the input is sorted,} \\ O(n \lg n) & \text{otherwise.} \end{cases}$ or integer sort can be applied

Correctness:

(Assume that $A=\{a_1, a_2, ..., a_n\}$ is sorted)

16-1a

(1) There is an optimal solution contains a_1.

(If Y is an optimal solution, $Y-\{\text{first in } Y\} \cup \{a_1\}$ is also an optimal solution.)

和 a_1 沒衝突

(2) Let $Y=\{a_1\} \cup Y'$ be an optimal solution to A.

Then, Y' is an optimal solution to $A'=\{a_i \mid s_i \geq f_1\}$.

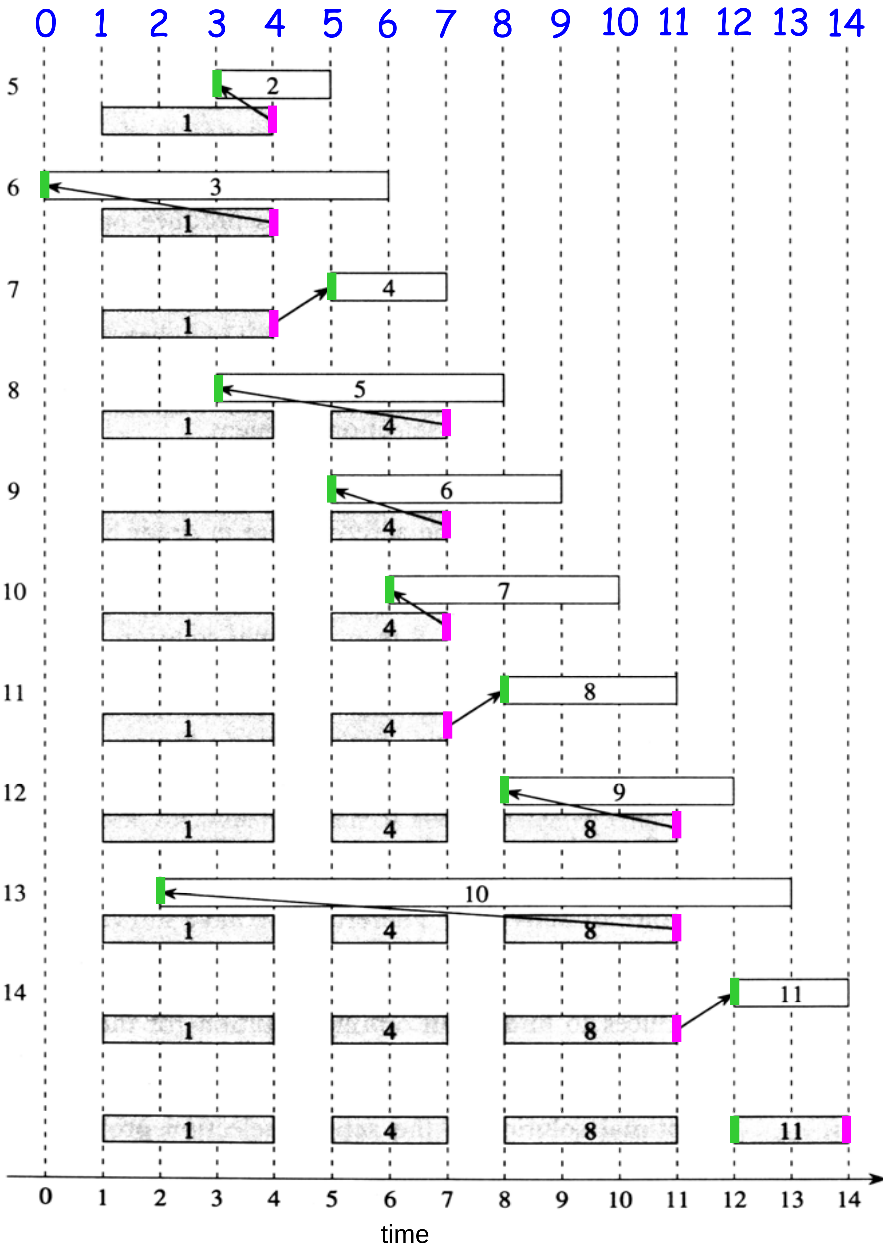
after a choice → same problem of smaller size

Example:

check s of the next one
f of the last one

sorted

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



16.2 Elements of Greedy strategy

optimization problems

1. ***Greedy-choice property***: a globally optimal solution can be arrived by making a locally optimal (greedy) choice. (top-down, usually)

2. ***Optimal substructure***: an optimal solution to the problem contains optimal solutions to sub-problems. after a choice → same subproblem of smaller size !

打 背 包

***0-1 knapsack problem* (*integer*)**:

weight

Input: n items with weight w_i and value v_i
capacity C (w_i, v_i, and C are integers)

Output: a subset of items with weight≤C and maximum value
(each item must be taken or left behind)

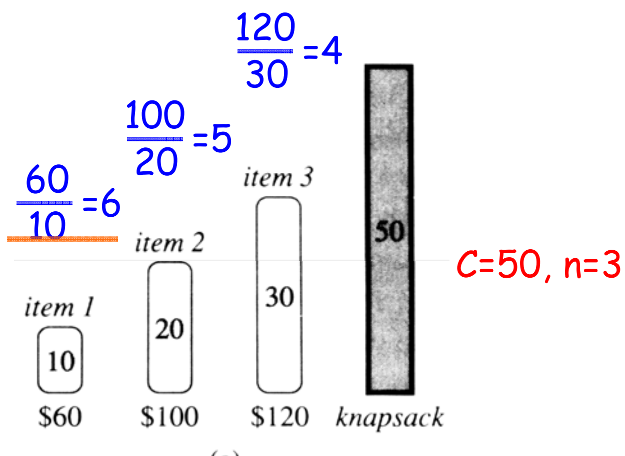
Fractional knapsack problem: Same as above.
But fractions of items can be taken.

16-3x
16-3a

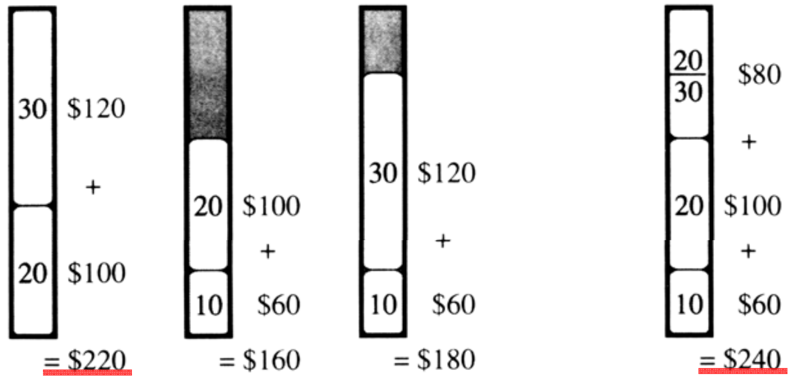
- The fractional problem has both properties. (Greedy according to v_i/w_i, $O(n\lg n)$ time)

- The 0/1 problem only has the optimal-substructure property. (dynamic programming) 16-3b
(integer weights) See Ex. 16.2-2

Example:



(a)



(b) 0-1 (c) fractional

16.3 Huffman codes (for compression)

(total=100) 2 possible code tables

	a	b	c	d	e	f
frequency	45	13	12	16	9	5
fixed-length	000	001	010	011	100	101
variable-length	0	101	100	111	1101	1100

Cost: (fixed) $3\times(45+13+12+16+9+5) = 300$

(var) $1\times45+3\times(13+12+16)+4\times(9+5)$ (√) $= 224$

Coding: abc ⇒ 0·101·100 ⇒ 0101100

Decoding: 001011101 ⇒ 0·0·101·1101 ⇒ aabe

a a b c

字 首 prefix-free : easy for decoding

Prefix codes: no codeword is a prefix of other

(<a:00, b:0, c:10, d:1> are not prefix codes)

000 ⇒ ab , ba , bbb

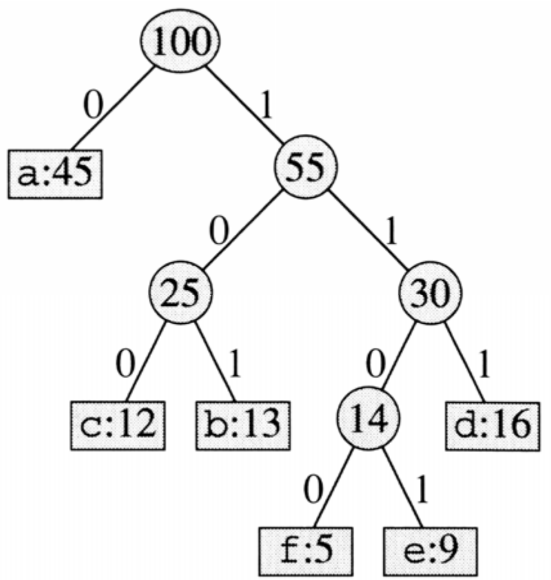
Representing prefix codes by a tree:

16-5x

prefix code

⬍

n-leaf (full) binary tree



- a:0, b:101, c:100, d:111, e:1101, f:1100
- cost of T: $\sum_{c\in C} f(c)\times depth(T,c)$ ← leaves

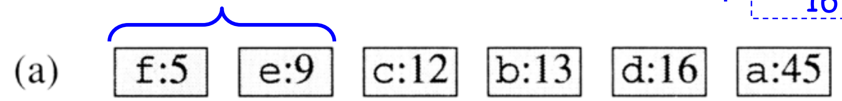
16-5z

Huffman code

16-6x

Build-Heap: O(n)

heap 45, 13, 12, 16, 9, 5



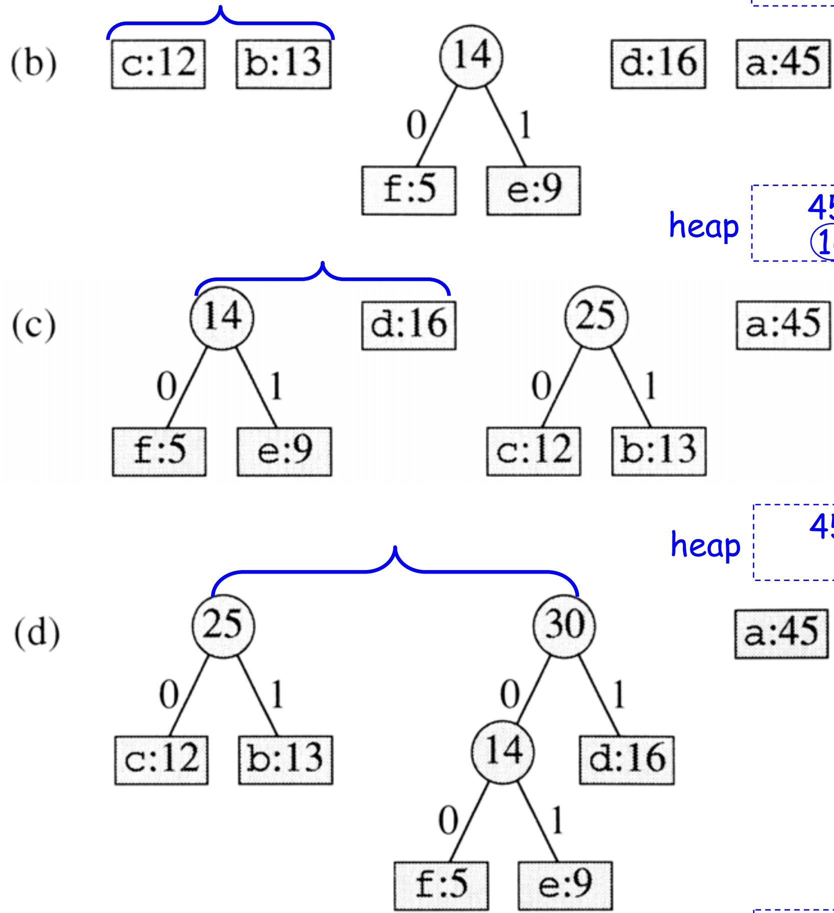
Extract-Min: m_1

Extract-Min: m_2

Insert $m_1 + m_2$

} O(lg n)

heap 45, 13, 12, 16, 14

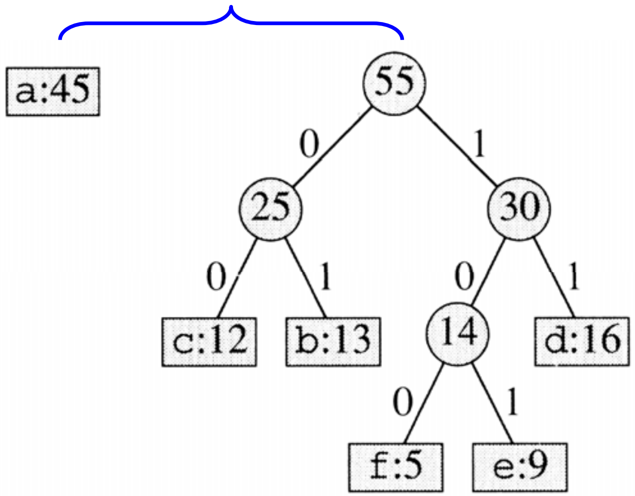


heap 45, 25, 16, 14

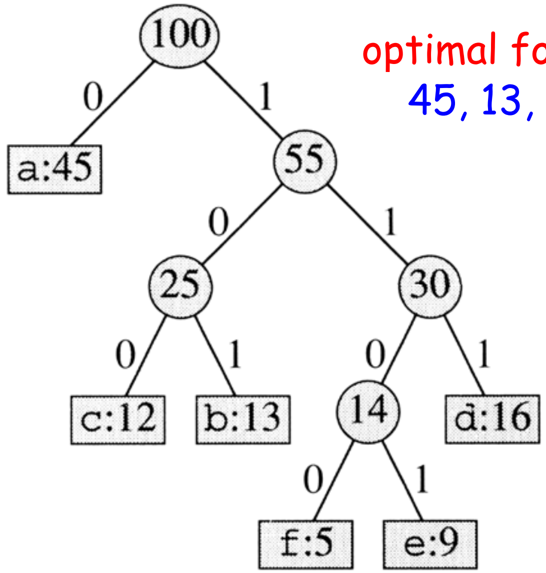
heap 45, 25, 30

heap 45, 55,

(e)



(f)



optimal for
45, 13, 12, 16, 9, 5

- Time: $O(n \lg n)$ /* $(n - 1) \times 3\lg n$
  (using a heap as a priority queue)
- Correctness: omitted

16-7x

16-7y

proof 0-1 knapsack (DP)

Homework: Ex. 16.1-4, 16.2-2, 16.2-3, 16.2-4, 16.2-5, 16.2-7, 16.3-6

proof

(proof: greedy-choice &
optimal substructure)