طراحی و تحلیل الگوریتم ها

دکتر امیر لکی زاده

استادیار گروه مهندسی کامپیوتر دانشگاه قم

2. Knapsack

Knapsack کوله پشتی

n شی داریم $O_1, O_2, \ldots, O_n$

یک کوله پشتی با حداکثر ظرفیت M

ارزش شی i : $O_i \to P_i$ وزن O_i : $O_i \to w_i$

هدف: ماکزیمم کردن $\sum x_i P_i$ به شرطی که $\sum x_i w_i \leq M$

$$x_i = \begin{cases} 1 & a_i \text{ انتخاب} \\ 0 & a_i \text{ عدم انتخاب} \end{cases}$$

$\downarrow$

0/1 Knapsack

if $0 \leq x_i \leq 1 \to$ fractional Knapsack (کسری)

$n=3 \quad M=20 \quad P = \langle P_1, P_2, P_3 \rangle = \langle 25, 24, 15 \rangle \quad W = \langle w_1, w_2, w_3 \rangle = \langle 18, 15, 10 \rangle$

x_1	x_2	x_3	$\sum x_i w_i$	$\sum x_i P_i$
$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	16.5	24.25
1	$\frac{2}{15}$	0	20	28.2
0	$\frac{2}{3}$	1	20	31
0	1	$\frac{1}{2}$	20	31.5

ابتدا شی انتخاب می‌شود که نسبت ارزش به وزن آن بیشتر باشد و این روند ادامه می‌یابد تا به حداکثر وزن ممکن یعنی M برسد

$$\frac{P_2}{w_2} > \frac{P_3}{w_3} > \frac{P_1}{w_1}$$

2. Knapsack(Fractional)

```
fractional_knapsack(P, W, M) {
  n <- length(P);
  Sort objects in decreasing order on Pi/wi (Profit/weight)
  for i <- 1 to n x[i] <- 0
  r <- M; // remaining capacity                باقیمانده ظرفیت
  for i <- 1 to n
  { if (w[i] <= r) {
        x[i] <- 1;
        r <- r - w[i];
     }
     else break; }
  if (i <= n)
     x[i] <- r/w[i] ;       از آخرین شی به اندازه (وزن باقیمانده / وزن شی) انتخاب می‌شود.
  return x;
```

2. Knapsack(0/1)

بیشترین ارزش درحالتی که $O_1, O_2, \ldots, O_i$ بررسی شده اند و وزن انتخابی نیز حداکثر j شده است (به j نرسیده است) $P[i, j]$

جواب مسئله $P[n, M]$

$$P[i, j] = \begin{cases} P[i-1, j] & w_i > j \\ \\ \mathrm{Max}\left(P[i-1, j],\ P[i-1, j-w[i]] + P_i\right) & w_i < j \end{cases}$$

2. Knapsack(0/1)

```
0/1 - Knapsack (w, P) {
  for i <- 0 to n   P[i,0] <- 0;          n تعداد اشیا
  for i <- 0 to M   P[0,i] <- 0;          M حداکثر وزن
  for i <- 1 to n
    for j <- 1 to M
      if (P[i-1, j] > P[i-1, j-w[i]] + Pi) {
          P[i,j] <- P[i-1, j];
          x[i] <- 0;
      }
      else { P[i,j] <- P[i-1, j-w[i]] + Pi;
          x[i] <- 1;                         O(n×M)
      }
  return P, x;
}
```

معایب:

۱- اندازه بزرگ ماتریس P

۲- x می تواند پیوسته باشد ولی فرض بر گسسته بودن x شده است.

مثال:

weight / object	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1*	6	7	7	18	19	24*	25	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

$M = 11$ $w = \langle 1, 2, 5, 6, 7 \rangle$ $P = \langle 1, 6, 18, 22, 28 \rangle$

3. Huffman Codes

fixed-length code

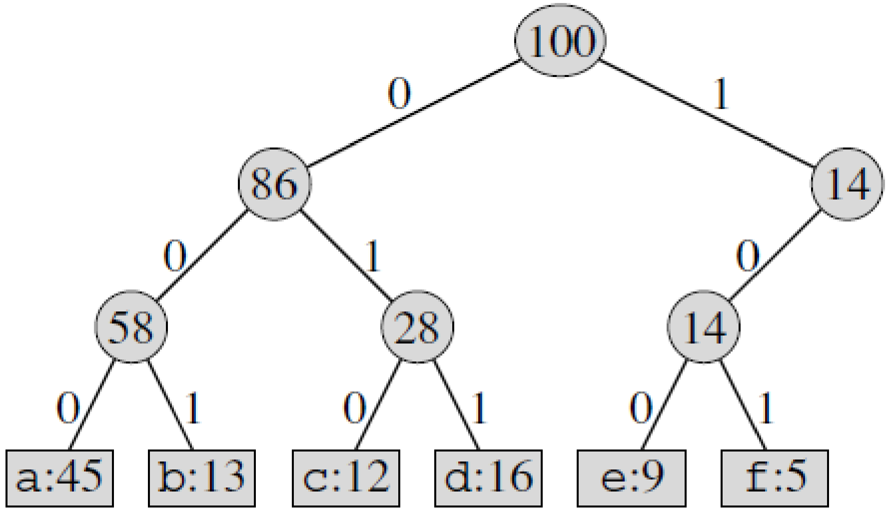
binary character code

variable-length code

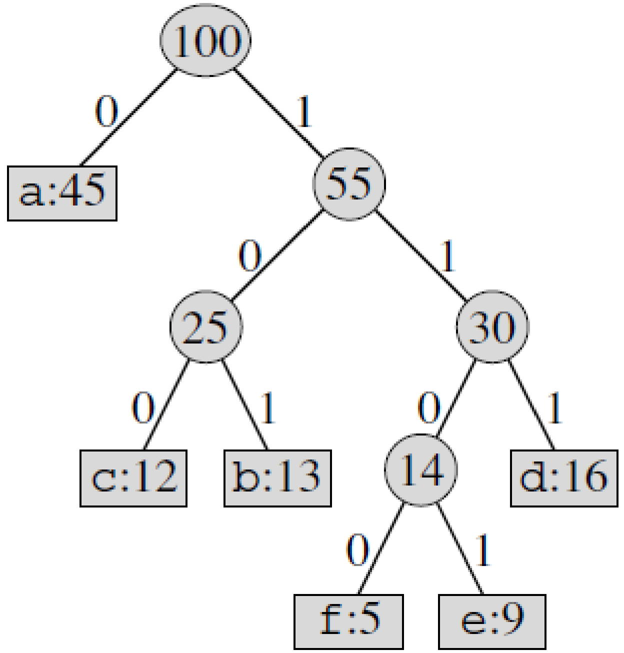
Prefix codes

3. Huffman Codes

	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



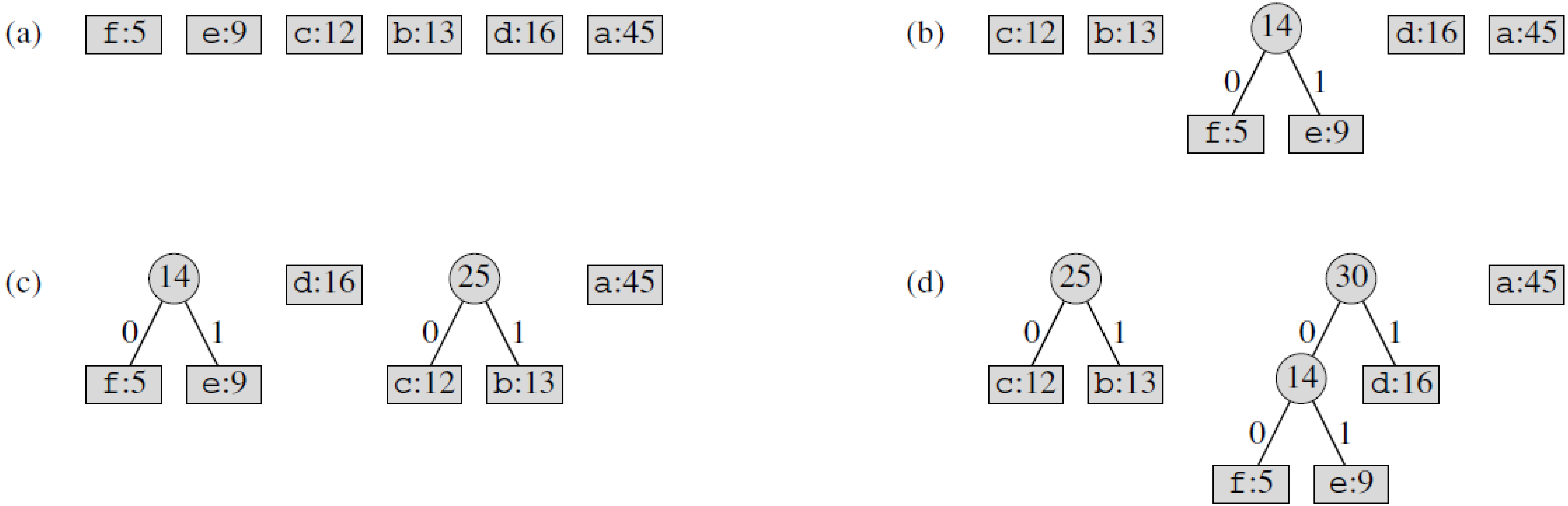
(a)



(b)

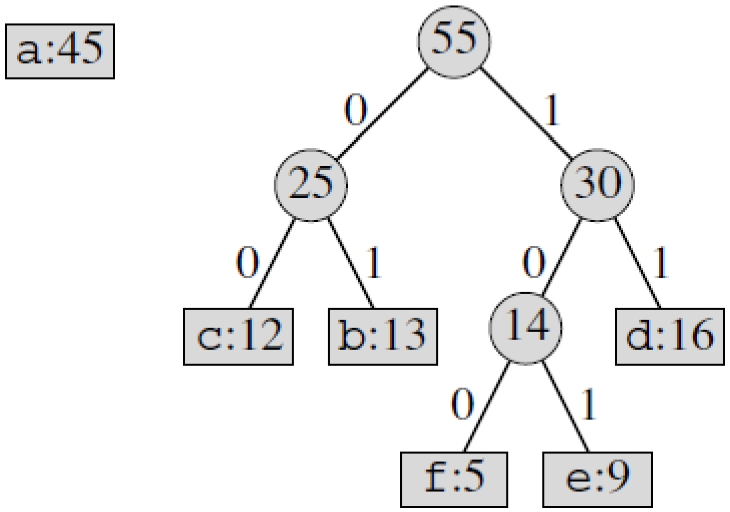
$$B(T) = \sum_{c \in C} f(c) d_T(c) ,$$

3. Huffman Codes

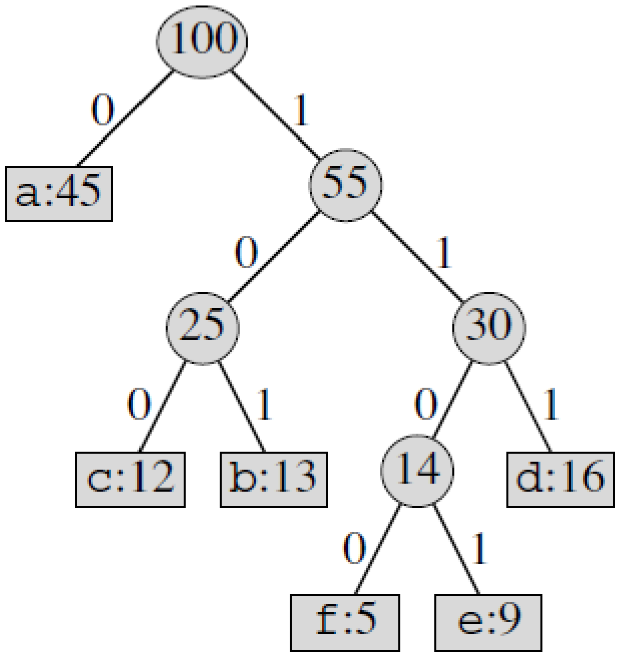


3. Huffman Codes

(e)



(f)



3. Huffman Codes

```
HUFFMAN(C)
n ← |C|
Q ← C
for i ← 1 to n − 1
      do allocate a new node z
         left[z] ← x ← EXTRACT-MIN(Q)
         right[z] ← y ← EXTRACT-MIN(Q)
         f[z] ← f[x] + f[y]
         INSERT(Q, z)
return EXTRACT-MIN(Q)            ▷ Return the root of the tree.
```

3. Huffman Codes

Correctness of Huffman's algorithm

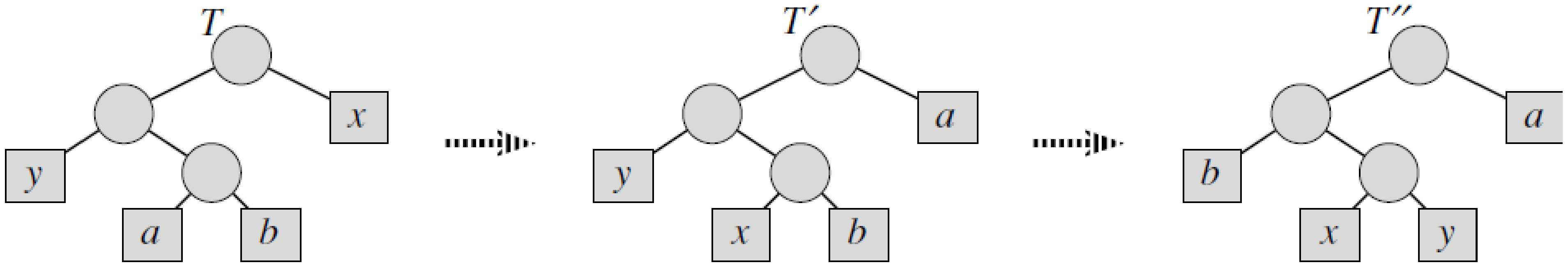
To prove that the greedy algorithm HUFFMAN is correct, we show that the problem of determining an optimal prefix code exhibits the greedy-choice and optimal-substructure properties. The next lemma shows that the greedy-choice property holds.

greedy-choice property

Lemma 16.2

Let C be an alphabet in which each character $c \in C$ has frequency $f[c]$. 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 codewords for x and y have the same length and differ only in the last bit.

3. Huffman Codes



3. Huffman Codes

we assume that $f[a] \le f[b]$ and $f[x] \le f[y]$.

$f[x] \le f[a]$ and $f[y] \le f[b]$.

$$\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f[x]d_T(x) + f[a]d_T(a) - f[x]d_{T'}(x) - f[a]d_{T'}(a) \\
&= f[x]d_T(x) + f[a]d_T(a) - f[x]d_T(a) - f[a]d_T(x) \\
&= (f[a] - f[x])(d_T(a) - d_T(x)) \\
&\ge 0 ,
\end{aligned}$$

4. Scheduling to Minimize Time

16-2 Scheduling to minimize average completion time

Suppose you are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of tasks, where task a_i requires p_i units of processing time to complete, once it has started. You have one computer on which to run these tasks, and the computer can run only one task at a time. Let c_i be the ***completion time*** of task a_i, that is, the time at which task a_i completes processing. Your goal is to minimize the average completion time, that is, to minimize $(1/n)\sum_{i=1}^{n} c_i$. For example, suppose there are two tasks, a_1 and a_2, with $p_1 = 3$ and $p_2 = 5$, and consider the schedule in which a_2 runs first, followed by a_1. Then $c_2 = 5$, $c_1 = 8$, and the average completion time is $(5 + 8)/2 = 6.5$.

a. Give an algorithm that schedules the tasks so as to minimize the average completion time. Each task must run non-preemptively, that is, once task a_i is started, it must run continuously for p_i units of time. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.

4. Scheduling to Minimize Time

Gready Algorithm for scheduling   الگوریتم greedy برای زمان بندی:

- minimize time in system

$S = \{s_1, s_2, \ldots, s_n\}$ of n services

t_i : s_i زمان سرویس

هدف: مینیمم کردن جمع زمان های سرویس دهی (معطلی + زمان اجرا) برای [illegible] فعالیت ها

$t_1 = 5 \quad t_2 = 10 \quad t_3 = 3$

$1,2,3 : 5+5+10+5+10+3 = 38$

$1,3,2 : 5+5+3+5+3+10 = 31$

$3,1,2 : 3+3+5+3+5+10 = 29$

راه حل: مرتب کردن فعالیت ها برحسب t_i بطور صعودی و سرویس دهی از فعالیت با زمان t_i کوچکتر

$I = (i_1, i_2, \ldots, i_n)$

$$T(I) = t_{i_1} + (t_{i_1} + t_{i_2}) + \cdots + (t_{i_1} + t_{i_2} + \cdots + t_{i_n}) = \sum_{k=1}^{n} (n-k+1)\, t_{i_k}$$

4. Scheduling to Minimize Time

اثبات درستی انتخاب حریصانه :

فرض کنید که $a<b$, $t_{i_a}>t_{i_b}$ و I یک جایگشت بهینه باشد

$$I=\{i_1, i_2, \ldots, i_a, i_{a+1}, \ldots, i_b, \ldots, i_n\}$$

نشان می‌دهیم که با جابجایی i_a و i_b یک جایگشت بهینه‌تر یا زمانبندی بهتر بدست می‌آوریم

$$I'=(i_1, i_2, \ldots, i_{a-1}, i_b, i_{a+1}, \ldots, i_{b-1}, i_a, i_{b+1}, \ldots, i_n)$$

$$T(I')=(n-a+1)t_{i_b}+(n-b+1)t_{i_a}+\sum_{\substack{k=1\\k\neq a\\k\neq b}}^{n}(n-k+1)t_{i_k}$$

$$T(I)=(n-a+1)t_{i_a}+(n-b+1)t_{i_b}+ \quad ''$$

$$T(I)-T(I')=(n-a+1)(t_{i_a}-t_{i_b})-(n-b+1)(t_{i_a}-t_{i_b})$$

$$=(t_{i_a}-t_{i_b})(b-a)\geqslant 0 \Rightarrow T(I)>T(I')$$

Greedy Algorithms

16-1 Coin changing

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

a. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

b. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

c. Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

d. Give an $O(nk)$-time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.