طراحی الگوریتم‌ها — الگوریتم‌های حریصانه

Introduction to Algorithm

مهدی جوانمردی

۱۴۰۱

فصل ۱۶ کتاب – الگوریتم‌های حریصانه

- ۱۶.۱: مسئله انتخاب فعالیت
- ۱۶.۲: المان‌های استراتژی حریصانه
- ۱۶.۳: کد هافمن

Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step. For many optimization problems, using dynamic programming to determine the best choices is overkill; simpler, more efficient algorithms will do. A ***greedy algorithm*** always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

مسئله انتخاب فعالیت

- مجموعه ای از n فعالیت داریم $S = \{a_1, a_2, \ldots, a_n\}$
- که هر فعالیت دارای زمان شروع و پایان است:

$$a_i \begin{cases} \textit{start time } s_i \\ \textit{finish time } f_i \end{cases} \qquad 0 \le s_i < f_i < \infty \qquad [s_i, f_i)$$

- فعالیت‌های سازگار:

$$a_i \text{ and } a_j \text{ are } \textbf{\textit{compatible}} \longrightarrow [s_i, f_i) \text{ and } [s_j, f_j) \text{ do not overlap} \begin{cases} s_i \ge f_j \\ \text{or} \\ s_j \ge f_i \end{cases}$$

- هدف: پیدا کردن بزرگترین زیرمجموعه از فعالیت‌های باهم سازگار
- فرض: فعالیت‌های بر حسب زمان پایان مرتب شده اند $f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n$
- مثال:

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

$\{a_3, a_9, a_{11}\}$

$\boxed{\begin{array}{l} \{a_1, a_4, a_8, a_{11}\} \\ \{a_2, a_4, a_9, a_{11}\} \end{array}}$

راه حل برنامه نویسی پویا مسئله انتخاب فعالیت

- زیرساختار بهینه

S_{ij} set of activities that start after activity a_i finishes and that finish before activity a_j starts

find a maximum set of mutually compatible activities in S_{ij}

A_{ij}

includes some activity a_k

S_{ik} a_k S_{kj}

$A_{ik} = A_{ij} \cap S_{ik}$ $A_{kj} = A_{ij} \cap S_{kj}$

$$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$$

$$|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$$

the size of an optimal solution

$c[i, j]$

$$c[i, j] = c[i, k] + c[k, j] + 1$$

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

examine all activities in S_{ij}

انتخاب حریصانه!

- چی می‌شد اگر بدون حل همه انتخاب‌های ممکن فقط با یک انتخاب به جواب می‌رسیدیم؟
- انتخاب شهودی و حریصانه:
- انتخاب فعالیتی که منابع را برای فعالیت‌های بیشتری آزاد میگذارد
- یعنی انتخاب فعالیتی در S که زودترین زمان پایان را دارد!

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

- انتخاب حریصانه در این حالت یعنی انتخاب a_1
- بعد از انتخاب حریصانه فقط فعالیت‌های باقی می‌ماند که بعد از پایان آن شروع شوند

$S_k = \{a_i \in S : s_i \geq f_k\}$ the set of activities that start after activity a_k finishes

اثبات راه حل حریصانه برای مسئله انتخاب

تئوری:

- زیرمسئله S_k را در نظر بگیرید و فرض کنید a_m فعالیتی در آن با سریع‌ترین زمان پایان باشد
- قطعا a_m در بزرگترین زیرمجموعه باهم سازگار فعالیت‌های مجموعه S_k حاضر خواهد بود

اثبات:

- فرض کنید A_k بزرگترین زیرمجموعه باهم سازگار فعالیت‌های مجموعه S_k باشد
- همچنین a_j فعالیتی در A_k باشد با سریع‌ترین زمان پایان
- فرض: $a_j \neq a_m$ $\longleftarrow$ $f_m \leq f_j$
- در اینصورت فرض کنید $A'_k = A_k - \{a_j\} \cup \{a_m\}$ باشد
- خواهیم داشت که فعالیت‌های A'_k باهم سازگارند و اندازه آن برابر با A_k است
- در نتیجه A'_k نیز یک بزرگترین زیرمجموعه باهم سازگار از S_k است و شامل a_m نیز می‌باشد

راه حل حریصانه مسئله انتخاب فعالیت

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

$$S_k = \{a_i \in S : s_i \geq f_k\}$$

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$ // find the first activity in S_k to finish
    $m = m + 1$
if $m \leq n$
    return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return $\emptyset$

GREEDY-ACTIVITY-SELECTOR(s, f)

$n = s.length$
$A = \{a_1\}$
$k = 1$
for $m = 2$ **to** n
    if $s[m] \geq f[k]$
        $A = A \cup \{a_m\}$
        $k = m$
return A

$S_0 = S$ RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, n)$

المان‌های الگوریتم حریصانه

1. Determine the optimal substructure of the problem.
2. Develop a recursive solution. (For the activity-selection problem, we formulated recurrence (16.2), but we bypassed developing a recursive algorithm based on this recurrence.)
3. Show that if we make the greedy choice, then only one subproblem remains.
4. Prove that it is always safe to make the greedy choice. (Steps 3 and 4 can occur in either order.)
5. Develop a recursive algorithm that implements the greedy strategy.
6. Convert the recursive algorithm to an iterative algorithm.

المان‌های الگوریتم حریصانه

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, so that the greedy choice is always safe.
3. Demonstrate optimal substructure by showing that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.

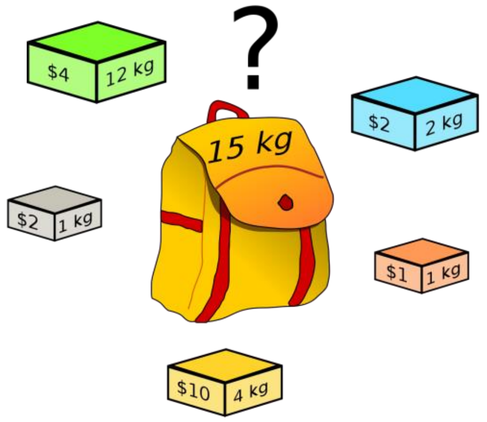
مقایسه برنامه نویسی پویا با روش حریصانه

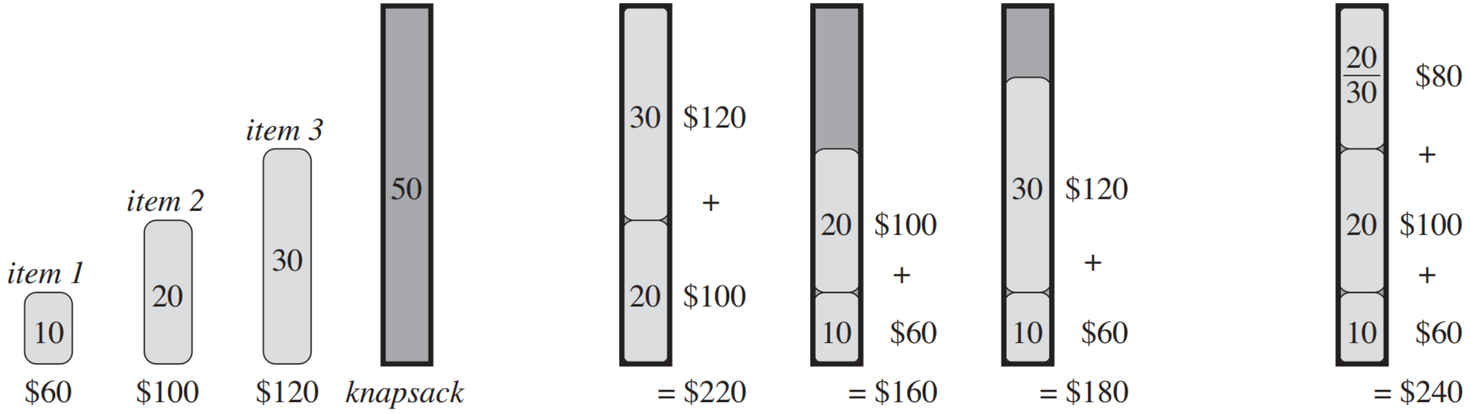
0-1 knapsack problem

fractional knapsack problem

Items	1	2	3	4	5
Weights	12	1	4	1	2
Values	4	2	10	1	2

Capacity: 15





(a) (b) (c)

Huffman Code کد هافمن

- کد هافمن: یک روش برای فشرده سازی داده (معمولا ۲۰ الی ۹۰ درصد)
- دارای جدول نرخ تکرار هر کاراکتر

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5

- ارائه نمایش بهینه برای حرف کاراکتر بصورت یک کد باینری با طول متغیر

- مثال: یک فایل ۱۰۰،۰۰۰ کاراکتری با ۶ کاراکتر مجزا داریم با نرخ تکرار زیر

	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

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1{,}000 = 224{,}000 \text{ bits}$$

کد هافمن و کد پیشوندی

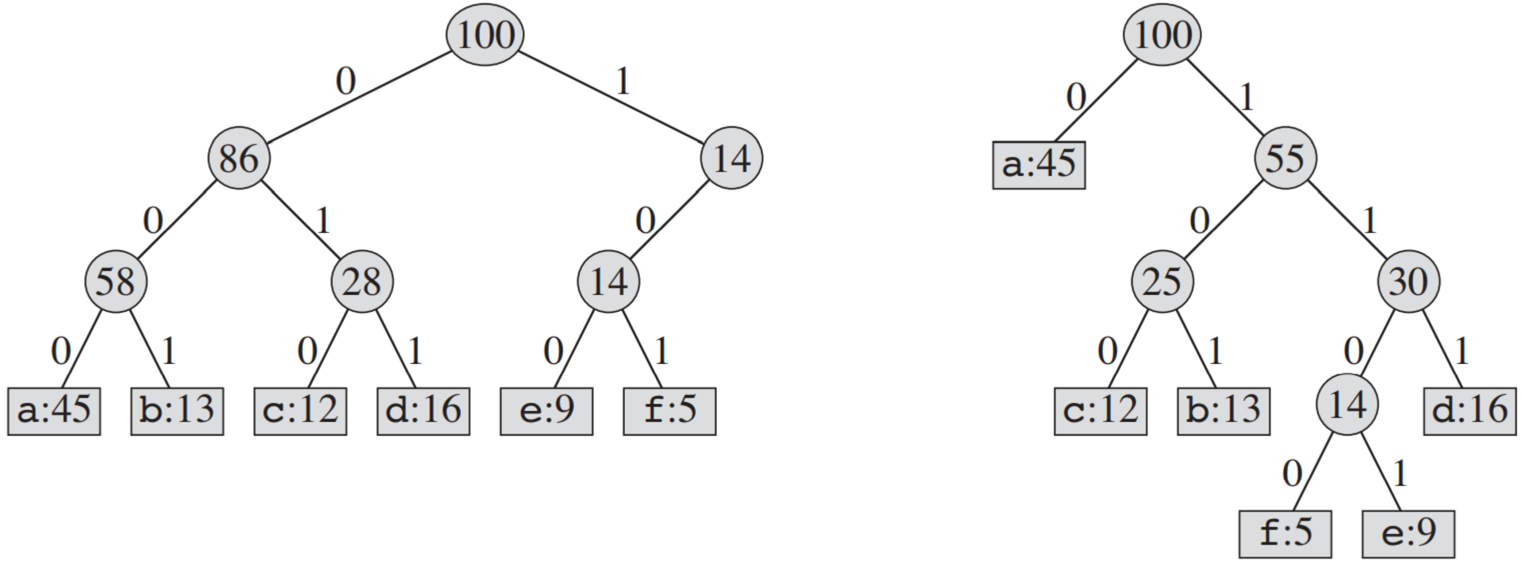
- کد پیشوندی یا Prefix-code یا بهتر است بگوییم Prefix-free-code:

به کدی گفته می شود که هیچ codeword ای در آن پیشوند codeword دیگری نیست

	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

نمایش درخت دودویی برای کدگذاری

	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)

کدگذاری هافمن

- هدف: پیدا کردن کدگذاری بهینه که کمترین تعداد بیت را برای یک فایل استفاده کند
- با فرض داشتن نرخ تکرار هر کاراکتر در کل فایل

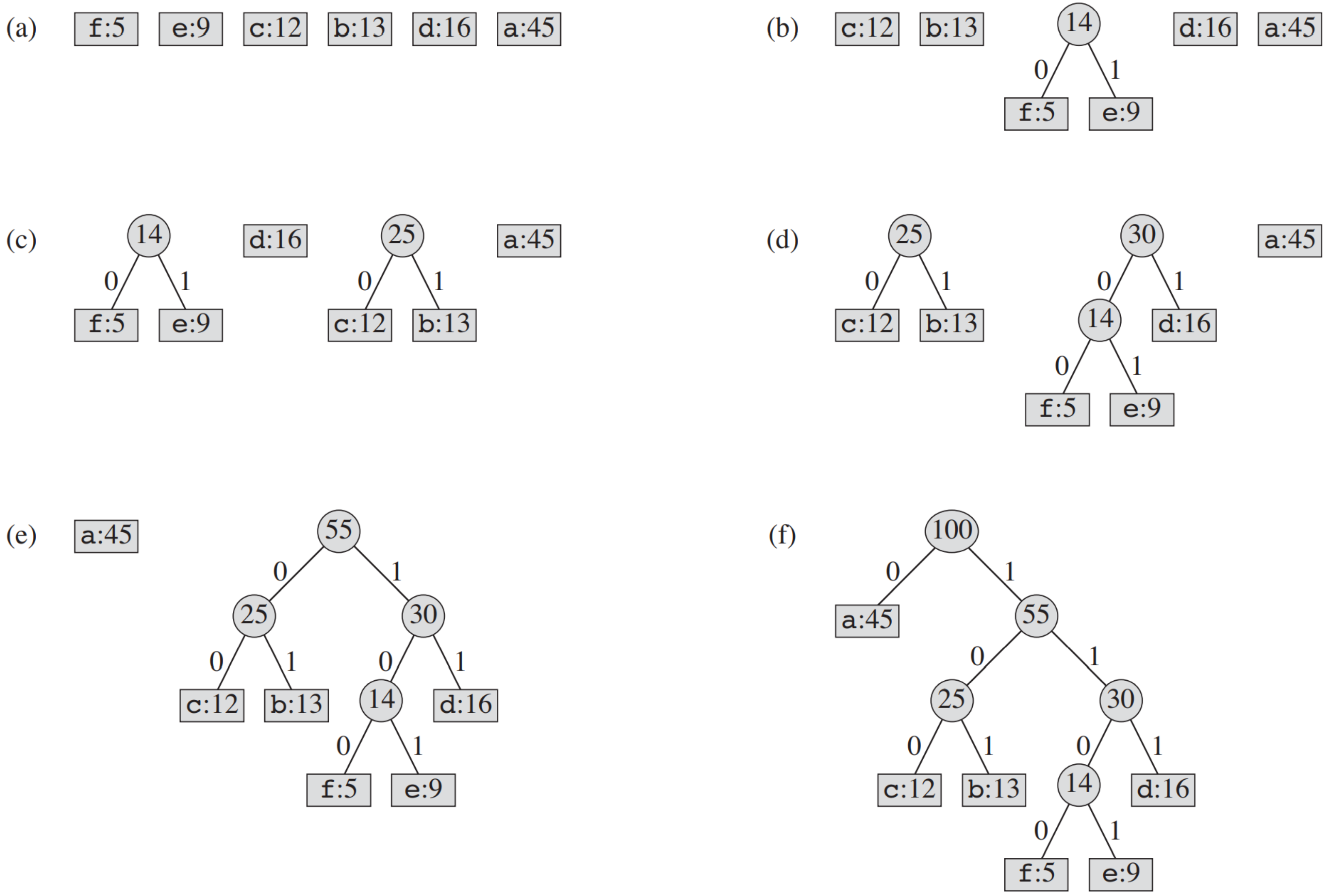
$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$$

$c.freq$ denote the frequency of c

$d_T(c)$ denote the depth of c's leaf in the tree

- آقای هافمن: یک روش حریصانه برای پیدا کردن بهترین کدگذاری را معرفی کرده است
- روش: پیدا کردن درختی با تعداد n برگ که ویژگی کدگذاری بهینه را داشته باشد
- استفاده از min-priority queue یا صف اولویت مینیمم برای بازگرداندن کاراکترهایی با کمترین میزان تکرار

مثال کدگذاری هافمن



برنامه کد هافمن

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

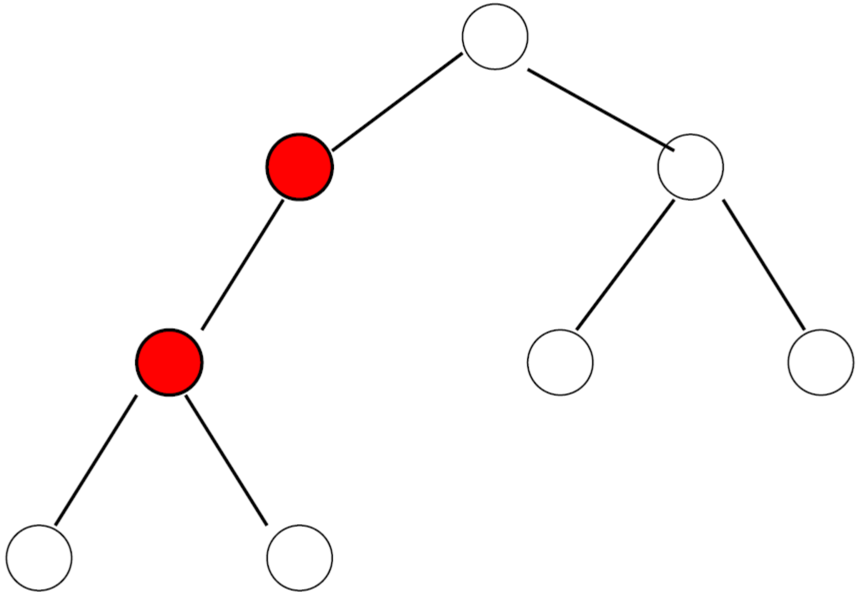
$O(\lg n)$

$O(n \lg n)$

اثبات کد هافمن– ۱

- مشاهده: درخت کدگذاری بهینه حتما یک درخت کامل است!
- به این معنا که هر گره در درخت دقیقا دو فرزند دارد

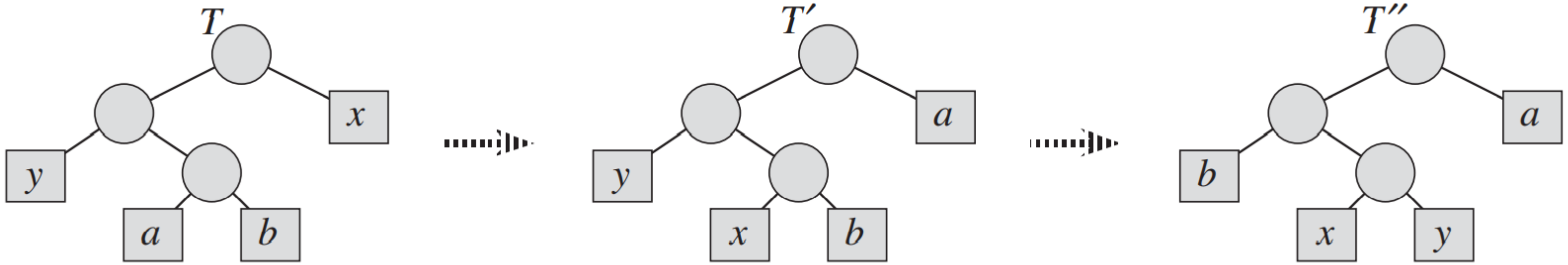
- اثبات:



اثبات کد هافمن–۲

- لم: فرض کنید x و y دو کم تکرارترین کاراکتر باشند. پس وجود خواهد داشت کدگذاری بهینه‌ای که در آن این دو کاراکتر دو برگ برادر در پایین‌ترین سطح درخت باشند

- اثبات: فرض کنید T درخت برای کدگذاری بهینه باشد و b و c به ترتیب دو برگ برادر در پایین‌ترین سطح درخت آن باشد (چون درخت کامل است حتما وجود دارد)
فرض کنید $b.freq \leq c.freq$ و $x.freq \leq y.freq$ باشد. از آنجایی که در فرض x و y کم تکرارترین کاراکترها هستند پس داریم $x.freq \leq b.freq$ و $y.freq \leq c.freq$. از آنجایی که b و c در عمیق‌ترین سطح قراردارند پس داریم $d(b) \leq d(x)$ و همچنین $d(c) \leq d(y)$



اثبات کد هافمن-۳

- لم: فرض کند T یک درخت دودویی کامل معرف کدگذاری بهینه روی الفای C باشد و $f[c]$ تکرار هر کاراکتر عضو C. هر دو کاراتر x و y را که در T بصورت برگ برادر ظاهر شدند را در نظر بگیرید و فرض کنید پدر آن‌ها z. در اینصورت، با در نظر گرفتن z بعنوان کارکتری با تکرار $f[z] = f[x] + f[y]$، در اینصورت درخت $T' = T - \{x,y\}$ معرف کدگذاری بهینه برای الفبای $C' = C - \{x,y\} \cup \{z\}$ است

$$B(T) = B(T') + x.freq + y.freq$$