به نام یگانه معبود بخشنده مهربان

طراحی الگوریتم ها

Design and Analysis of Algorithms

گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه اصفهان

ترم دوم سال تحصیلی ۹۰–۹۱

ارائه دهنده : پیمان ادیبی

روش حریصانه

Greedy Approach

مفاهیم

- روش حریصانه برای یافتن پاسخ بهینه (بصورت دنباله‌ای از انتخابها) در مسایل بهینه سازی بکار میرود.
- الگوریتمهای حریصانه در هر مرحله انتخابی را انجام میدهند که تنها در آن لحظه از نظر معیار بهینه سازی بهترین انتخاب به نظر میرسد.
- بعبارت دیگر الگوریتمهای حریصانه یک دنباله از انتخابهای بهینه محلی (locally optimal) را به امید دست یافتن به یک پاسخ بهینه سراسری (globally optimal solution) انجام میدهند.
- پاسخ شدنی یا امکان پذیر (feasible solution) :
  - پاسخی که یک یا چند شرط را برآورده میسازد.
- پاسخ بهینه (optimal solution) :
  - پاسخی که در عین برآورده ساختن شرطها (شدنی بودن)، یک معیار را نیز بهینه میسازد.

مفاهیم

- روش حریصانه رسیدن به پاسخ بهینه را تضمین نمیکند!!
  لذا برای هر مسأله باید اثبات درست بودن (بهینه بودن) الگوریتم حریصانه انجام شود.

- اثبات درستی (proof of correctness):
  معمولاً با روشهایی چون برهان خلف و استقراء و رد آن با مثال نقض انجام میپذیرد.

مراحل یک الگوریتم حریصانه

0- با یک مجموعه از انتخابهای ممکن شروع میکند.

1- روال انتخاب (S.P. -- selection procedure): در مرحله جاری ازمیان گزینه های موجود، آنکه بهترین بنظر میرسد را انتخاب میکند.

2- بررسی امکان پذیری (F.C. – feasibility check): تعیین میکند که آیا انتخاب اخیر با در نظر گرفتن وضعیت در این لحظه میتواند یک پاسخ شدنی ایجاد کند؟ اگر جواب مثبت باشد، این انتخاب از مجموعه انتخابهای موجود حذف و در پاسخ گنجانده میشود. در غیراینصورت بطور دائم از پاسخ کنار گذاشته میشود.

3- بررسی پاسخ (S.C. – solution check): آیا به یک پاسخ شدنی برای نمونه مسأله رسیده ایم؟ اگر جواب منفی است به مرحله 1 باز میگردیم.

مسأله کوله پشتی صفر و یک

- یک کوله پشتی با ظرفیت وزنی M و تعداد n شیء $\{I_1,\ldots,I_n\}$ داریم.
  وزن شیء I_i برابر w_i و ارزش آن p_i است.
  هدف یافتن یک دنباله بهینه $(x_1, \ldots, x_n)$ است، بنحویکه $\sum_{i=1}^{n} p_i x_i$ بیشینه بوده و در عین حال محدودیت $\sum_{i=1}^{n} w_i x_i \le M$ نیز برقرار باشد.

$$x_i = \begin{cases} 0 & \text{عدم وجود شیء } I_i \text{ در کوله پشتی} \\ 1 & \text{وجود شیء } I_i \text{ در کوله پشتی} \end{cases}$$

- روش حریصانه:
  - مجموعه انتخابهای ممکن: $C=\{I_1, I_2, \ldots, I_n\}$
  - S.P. : انتخاب بهترین شیء موجود در این لحظه (I_k) و حذف آن از مجموعه C
  - F.C. : آیا با توجه به ظرفیت موجود در کوله پشتی در این لحظه، انتخاب I_k ممکن است؟ اگر پاسخ مثبت است I_k در کوله پشتی قرار گیرد.
  - S.C. : اگر مجموعه C خالی است، خاتمه. در غیر اینصورت به مرحله S.P. برو.

مسأله کوله پشتی صفر و یک

- سئوال: در روال انتخاب بهترین شیء را چه شیئی بگیریم؟
- ایده 1: بهترین شیء را پر ارزش‌ترین شیء بگیریم:
  - مثال: M= 11, n=3

	I1	I2	I3
p	18	20	17
w	3	10	2

× جواب روش حریصانه ≠ جواب بهینه

- ایده 2: بهترین شیء را کم وزن‌ترین شیء بگیریم:
  - مثال: M= 11, n=3

	I1	I2	I3
p	4	20	6
w	5	8	2

× جواب روش حریصانه ≠ جواب بهینه

- ایده 3: بهترین شیء را پر ارزش‌ترین در واحد وزن بگیریم:
  - مثال: M= 11, n=3

	I1	I2	I3
p	16	20	30
w	2	5	6

× جواب روش حریصانه ≠ جواب بهینه

- روش حریصانه برای مسأله کوله پشتی صفر و یک درست عمل نمیکند.

√484

مسأله کوله پشتی کسری
(Fractional Knapsack Problem)

- تنها تفاوت با مسأله کوله پشتی صفر و یک اینست که در اینجا مجاز به انتخاب کسر دلخواهی از هر شیء هستیم (یعنی $0 \le x_i \le 1$).
- روش حریصانه قبل با ایده 3 (بهترین شیء را پر ارزش‌ترین در واحد وزن بگیریم) جواب بهینه را خواهد یافت:
  - مثال: M= 11, n=3

	I1	I2	I3
p	16	20	30
w	2	5	6
p/w	8	4	5

جواب = (1، 0.6، 1) ، سود= 58

√ جواب روش حریصانه = جواب بهینه

- اثبات درستی: با استفاده از برهان خلف ثابت کنید روش حریصانه فوق برای مسأله کوله پشتی کسری همیشه به جواب بهینه میرسد (تمرین).

√480

مسأله خرد کردن پول

- هدف پرداخت یک مبلغ مشخص با حداقل تعداد سکه:

Minimize n

subject to the constraint $\sum_{i=1}^{n} p_i x_i = A$

n = تعداد سکه پرداختی

p_i = ارزش سکه i ام

A = مبلغ کل

$$x_i = \begin{cases} 0 & \text{عدم انتخاب سکه i ام} \\ 1 & \text{انتخاب سکه i ام} \end{cases}$$

- الگوریتم حریصانه: هر بار سکه ای با بیشترین مبلغ برداشته شود. اگر با این سکه مبلغ کل تا این زمان از A بیشتر نشود، آن سکه به مجموعه سکه‌ها اضافه شود. در غیر اینصورت انتخاب از سکه ها یی با یک مبلغ پایین‌تر انجام شود. این مراحل تا رسیدن به A تکرار شود .
  - مثال:

با سکه های 1، 5، 10، 25، و 50 تومانی روش حریصانه همیشه درست است.

با سکه های 1، 5، 10، 12، و 25 تومانی روش حریصانه درست نخواهد بود.

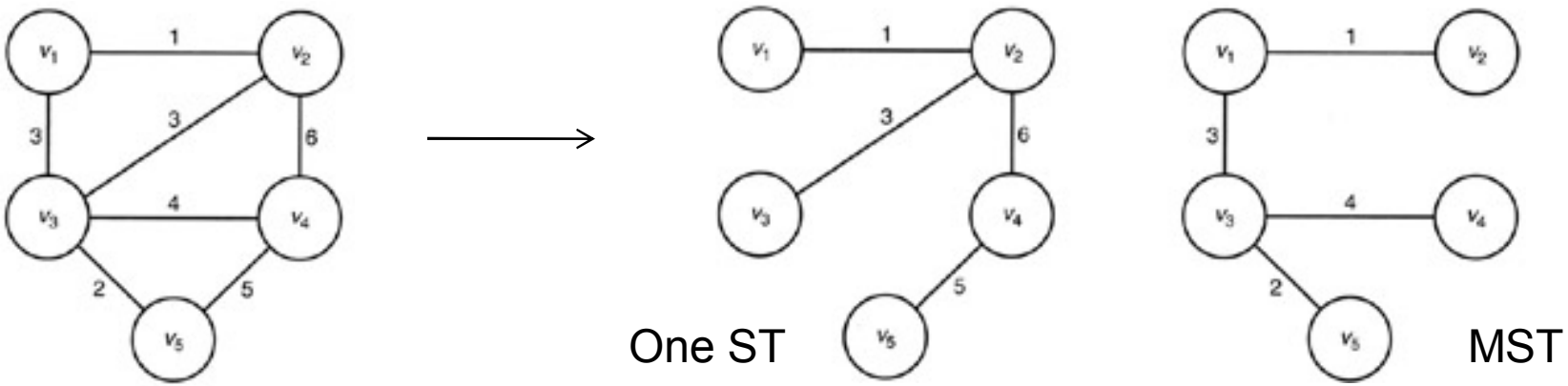
مثلا برای 46 تومان: جواب حریصانه: 25+12+5+1+1+1+1 ← 7 سکه

جواب بهینه: 25+10+10+1 ← 4 سکه

√508

یافتن درخت پوشای کمینه برای یک گراف (Minimal Spanning Tree)

- درخت پوشا برای یک گراف : درختی که شامل تمام رئوس و زیرمجموعه ای از یالهای گراف باشد.
- درخت پوشای کمینه (MST) برای یک گراف وزن دار: درخت پوشایی که مجموع وزنهای یالهای آن کمینه باشد.



- الگوریتم بدیهی: بررسی تمام درختهای پوشا در بدترین حالت بدتر از نمایی است

الگوریتم راشال برای یافتن درخت پوشای کمینه

- گراف $G=(V,E)$ با مجموعه رئوس V و مجموعه یالهای E مفروض است. تعداد رأسها $|V|=n$ و تعداد یالها $|E|=m$ میباشد.
- یک درخت پوشا بصورت $T=(V,F)$ نمایش داده میشود که $F \subseteq E$
- الگوریتم راشال (Kruskal):
  - ابتدا $F=\emptyset$؛ و V به n زیرمجموعه مجزا هریک شامل یک رأس افراز میشود $(\{v_1\}, \{v_2\}, \ldots, \{v_n\})$؛ و یالهای E بترتیب غیرنزولی وزنهایشان مرتب میشوند.
  - **S.P.** : کم وزن ترین یال را از لیست مرتب خارج کن (e).
  - **F.C.** : اگر e رأسهایی از دو زیرمجموعه مجزا را بهم وصل میکند، یعنی دور ایجاد نمیکند و شدنی است. در اینصورت دو زیرمجموعه را در هم ادغام کرده و e را به F اضافه کن.
  - **S.C.** : اگر تمام زیرمجموعه‌ها با هم ادغام شده‌اند خاتمه وگرنه به قدم **S.P.** برو.
- این الگوریتم از نوع داده "مجموعه مجزا" استفاده میکند که در پیوست سوم کتاب تشریح شده است. جزئیات الگوریتم 4.2 از کتاب مطالعه شود
- مثال: ...

الگوریتم راشال برای یافتن درخت پوشای کمینه

- تحلیل الگوریتم راشال:
  - ابتدا $F=\emptyset$؛ و V به n زیرمجموعه مجزا هریک شامل یک رأس افراز میشود $(\{v_1\}, \{v_2\}, \ldots, \{v_n\})$؛ و یالهای E بترتیب غیرنزولی وزنهایشان مرتب میشوند.
  - **S.P.** : کم وزن ترین یال را از لیست مرتب خارج کن (e).
  - **F.C.** : اگر e رأسهایی از دو زیرمجموعه مجزا را بهم وصل میکند، دو زیرمجموعه را در هم ادغام کرده و e را به F اضافه کن.
  - **S.C.** : اگر تمام زیرمجموعه‌ها با هم ادغام شده‌اند خاتمه وگرنه به قدم S.P. برو.

زمان مرتب سازی یالها: $\theta(m\lg m)$ → زمان غالب $(m \geq n-1)$

زمان حلقه اصلی در بدترین حالت: $\theta(m\lg m)$

کار با ساختمان داده مجموعه مجزا از مرتبه $\lg m$ بوده (پیوست سوم کتاب) و تعداد تکرار حلقه در بدترین حالت (بررسی همه یالها) برابر m است.

زمان مقدار دهی اولیه به مجموعه های مجزا: $\theta(n)$

- بدترین حالت (پریال‌ترین گراف $m=n(n-1)/2$): $W(m,n)=\theta(n^2\lg n)$
- بهترین حالت (کم یال‌ترین گراف $m=(n-1)$): $B(m,n)=\theta(n\lg n)$

الگوریتم راشال برای یافتن درخت پوشای کمینه

- اثبات درستی الگوریتم راشال:
  لم 4.2 و قضیه 4.2 کتاب مطالعه شود.

- همانطور که پیشتر گفته شد، پیچیدگی الگوریتمها گاهی وابسته به ساختمان داده‌های بکار رفته هستند.

- در الگوریتم راشال مشاهده شد که استفاده از ساختمان داده مجموعه مجزا (پیوست سوم کتاب) باعث شد که مقداردهی اولیه و حلقه اصلی الگوریتم پرهزینه نباشند، و لذا مرتبه در حد همان مرتبه مرتب سازی باقی میماند.

الگوریتم پریم برای یافتن درخت پوشای کمینه

فرض: گراف $G(V,E)$ همبند، بدون جهت، و دارای وزنهای مثبت.

- $F=\{\}$، $Y=\{v_1\}$ (از رأس v_1 شروع میکنیم).
- S.P. و F.C. : رأسی در $V-Y$ که نزدیکترین به Y است را انتخاب کن (v). یالی که از طریق آن v بعنوان نزدیکترین رأس به Y متصل میشود را تعیین کن (e). سپس v را به Y و e را به F اضافه کن.
- S.C.: اگر $Y==V$ به پاسخ رسیده ایم، وگرنه تکرار مراحل S.P. و F.C.

مثال عددی : ....

اثبات درستی: لم 4.1 و قضیه 4.1 کتاب

الگوریتم 4.1 کتاب. تحلیل در هر حالت: $T(n) = 2(n-1)(n-1)$

مقایسه:
- راشال: $\theta(mlgm)$ ← $\theta(n^2lgn)$ برای پریال ترین گراف، $\theta(nlgn)$ برای کم یال ترین گراف
- پریم: $\theta(n^2)$

نتیجه: برای گراف نسبتا کم یال از راشال و برای گراف نسبتا پریال از پریم استفاده شود

الگوریتم دایکسترا برای یافتن کوتاهترین مسیر از یک مبدأ در گراف (مسأله SSSP)

هدف: یافتن کوتاهترین مسیر از یک رأس مشخص v_1 به سایر رئوس در یک گراف وزندار جهت دار $G(V,E)$ با وزنهای مثبت.

- $F=\{\}$، $Y=\{v_1\}$ (از رأس v_1 شروع میکنیم).
- S.P. و F.C. : رأسی در $V-Y$ که کوتاهترین مسیر از v_1 تا آن رأس تنها با رئوس موجود در Y بعنوان رئوس میانی را دارد، انتخاب شود (v). یالی که روی کوتاهترین مسیر، v را به Y متصل میکند، تعیین گردد (e). سپس v را به Y و e را به F اضافه کن.
- S.C.: اگر $Y==V$ به پاسخ رسیده ایم، وگرنه تکرار مراحل S.P. و F.C.

مثال عددی : ....

اثبات درستی: مانند الگوریتم پریم

الگوریتم 4.3 کتاب. تحلیل در هر حالت: $T(n) = 2(n-1)(n-1)$

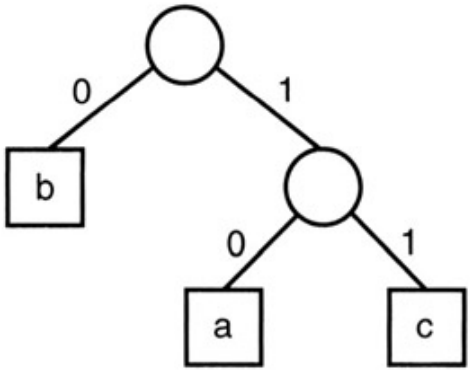
مقایسه:
فلوید ← از هر رأس به هر رأس ← $\theta(n^3)$
دایکسترا ← از یک رأس به سایر رأسها ← $\theta(n^2)$

کد هافمن

- انواع کدگذاری در فایلها:
  - کد دودویی با طول ثابت: بسته به تعداد عناصر (کاراکترهای) موجود یک طول کد بدست آورده و به هر عنصر یک عدد دودودیی منسوب میکنیم ← مثال: اگر سه کاراکتر a ، b ، و c داشته باشیم طول کد باید 2 باشد: a ≡ 00 ، b ≡ 01 ، c ≡ 11 ← کد واژه (codeword)
  - کد دودویی با طول متغیر: اگر بدانیم فراوانی برخی کاراکترها بیشتر است به آنها کدواژه با طول کمتر تخصیص میدهیم تا فضای کمتری اشغال شود. مثال: ababcbbbc ← فراوانی b بیش از a و c است: a ≡ 10 ، b ≡ 0 ، c ≡ 11
  - پس برای رشته ababcbbbc داریم:
    کدگذاری با طول ثابت: 000100011101010111 ← 18 بیت
    کدگذاری با طول متغیر: 1001001100011 ← 13 بیت

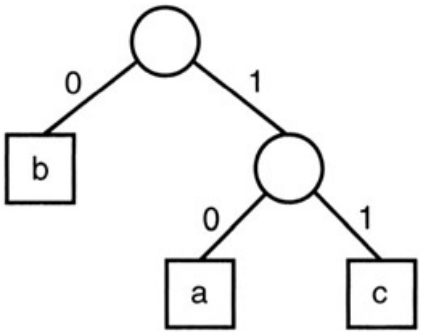
کد هافمن

- کدهای پیشوندی (Prefix Codes):
  - نوعی کد با طول متغیر است که در آن هیچ کدواژهٔ مربوط به یک کاراکتر، آغاز کدواژه کاراکتر دیگر را نمیسازد. مثلاً اگر 01 کد کاراکتری باشد، 011 نمیتواند کد کاراکتر دیگری باشد (مانند مثال اسلاید قبل).
- نمایش کد پیشوندی با درخت دودویی گسترش یافته:
  - هر گره یا دو فرزند دارد یا فرزند ندارد.
  - برگها با مربع و غیربرگها با دایره نمایش داده میشوند.
  - برگها کاراکترها هستند. برای هر گره به شاخه سمت چپ 0 و به شاخه سمت راست 1 منسوب میکنیم.
  - مثال: $a \equiv 10$ ، $b \equiv 0$ ، $c \equiv 11$



کد هافمن

- **کدگشایی برای کدهای پیشوندی:**
  - با شروع از اول رشته (فایل) از ریشهٔ درخت شروع میکنیم. هرگاه در فایل به 0 رسیدیم به سمت چپ و هرگاه به 1 رسیدیم به سمت راست در درخت حرکت میکنیم. وقتی به اولین برگ رسیدیم کاراکتر مورد نظر یافت شده است.
  - مثال : 1001001100011 ← ...



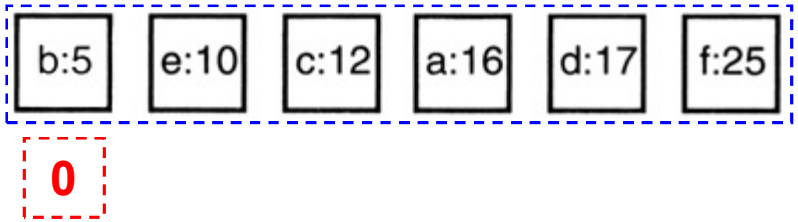
- **کد هافمن:**
  - کد بهینه (دارای کمترین طول) برای یک فایل است.
  - کد هافمن یک کد پیشوندی با طول متغیر است.
- **الگوریتم هافمن:**
  - یک الگوریتم حریصانه برای یافتن درخت دودویی متناظر با کد بهینه (هافمن) با داشتن فراوانی وقوع عناصر

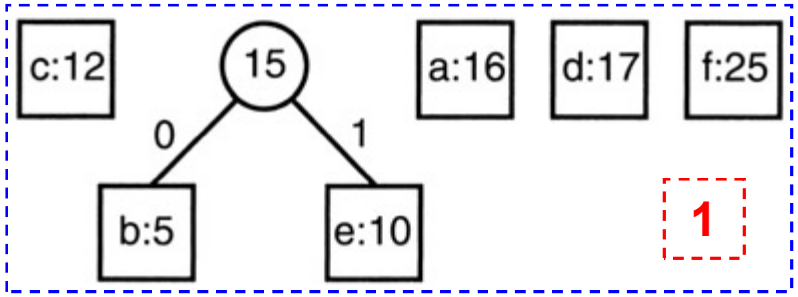
الگوریتم هافمن

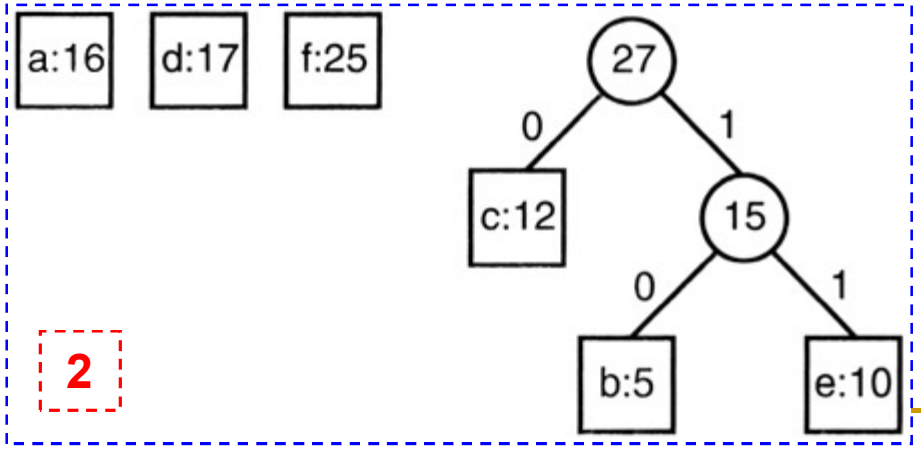
- پیاده سازی با صف اولویت:
  - تمام n عنصر را در یک صف اولویت قرار میدهیم.
  - بیشترین اولویت با کمترین فراوانی است.
  - در یک حلقه (تا رسیدن به صف تک عنصری) هر بار دو عنصر با بیشترین اولویت را برداشته، فراوانی آنها را با هم جمع کرده، و حاصل را بعنوان یک عنصر جدید در صف اولویت قرار میدهیم.
  - این عنصر جدید ریشهٔ درختی دودویی است که عنصر با اولویت بیشتر زیردرخت سمت چپ آن، و عنصر با اولویت کمتر زیر درخت سمت راست آن خواهد بود. ← پیاده سازی با هیپ ← $\Theta(n\lg n)$
  - مثال: فایلی متشکل از شش کاراکتر با فراوانیهای زیر را داریم. کد هافمن را پیدا کنید. تعداد کل بیت مورد نیاز با کد هافمن را با تعداد کل بیت مورد نیاز با کد طول ثابت مقایسه کنید:

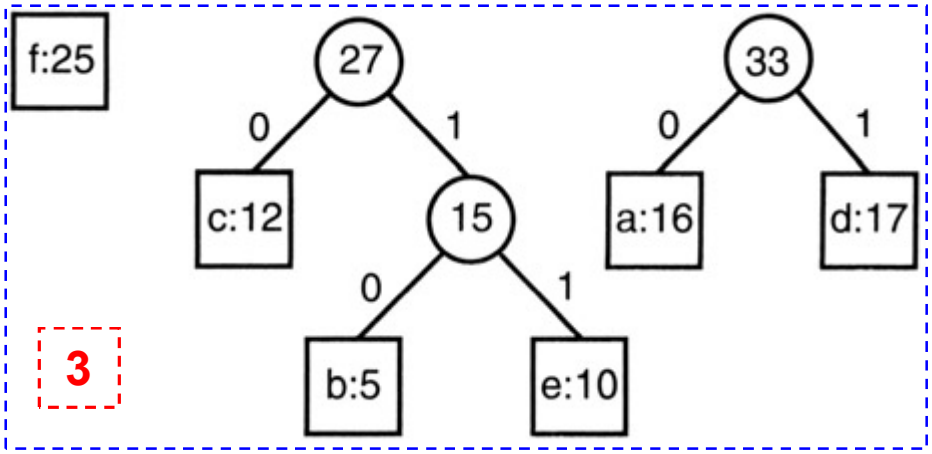
کاراکتر:	a	b	c	d	e	f
فراوانی:	16	5	12	17	10	25

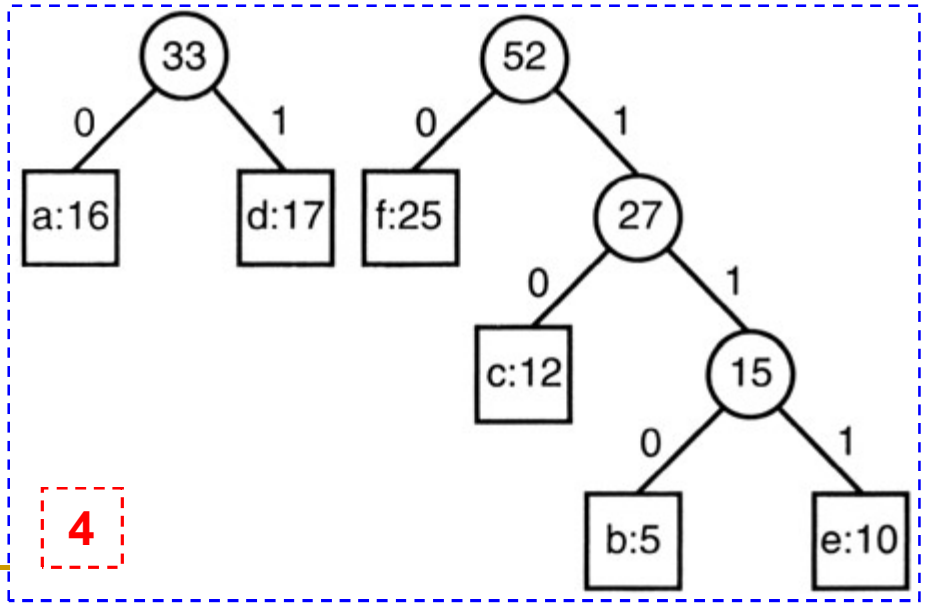
الگوریتم هافمن



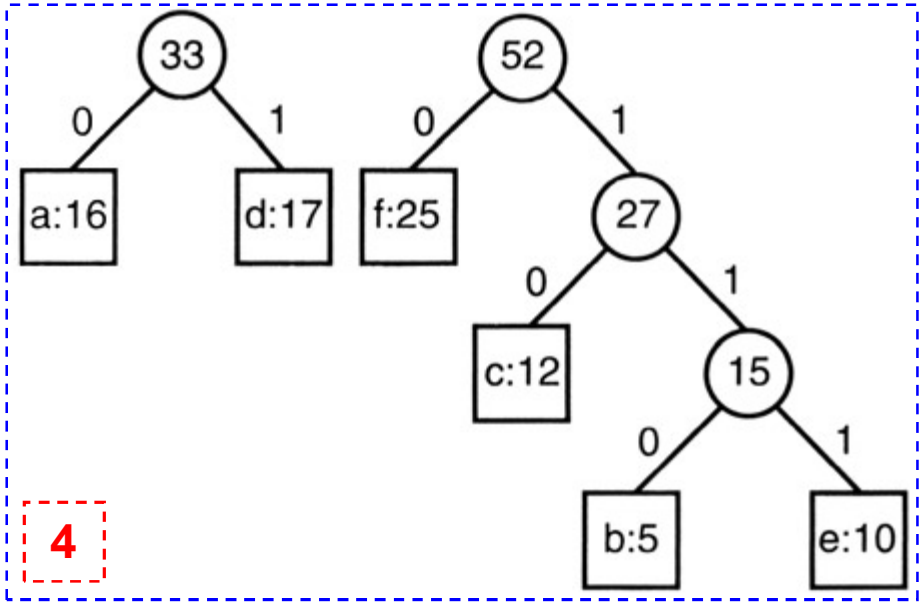


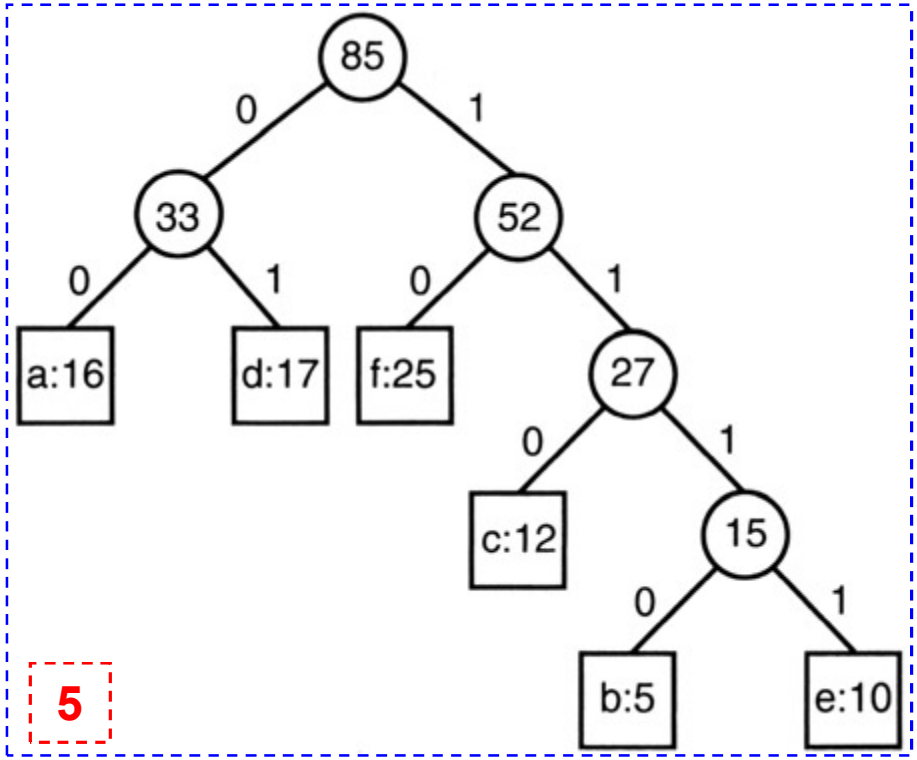






الگوریتم هافمن





کد هافمن:

$a \equiv 00$ ، $b \equiv 1110$ ، $c \equiv 110$ ، $d \equiv 01$ ، $e \equiv 1111$ ، $f \equiv 10$

حداقل بیت مورد نیاز = $16(2) + 5(4) + 12(3) + 17(2) + 10(4) + 25(2) = 212$

بیت مورد نیاز برای کد طول ثابت = $16(3) + 5(3) + 12(3) + 17(3) + 10(3) + 25(3) = 255$

الگوریتم هافمن

- صف اولویت ← PQ :
- برای هر اشاره گر p در صف اولویت:

$p\text{->}\ symbol$ = a distinct character in the file;
$p\text{->}\ frequency$ = the frequency of that character in the file;
$p\text{->}\ left = p\text{->}\ right$ = NULL;

```
for (i = 1; i < = n-1; i++) {    // There is no solution check; rather,
    remove (PQ, p);              // solution is obtained when i = n - 1.
    remove (PQ, q);              // Selection procedure.
    r = new nodetype;            // There is no feasibility check.
    r->left = p;
    r->right = q;
    r->frequency = p->frequency + q->frequency;
    insert (PQ, r);
}
remove (PQ, r);
return r;
```

- اثبات بهینگی الگوریتم هافمن: قضیه 4.5 کتاب

زمانبندی برای کمینه کردن زمان کل در سیستم

- تعداد n کار در انتظار دریافت خدمات هستند.
- هر کار زمان مورد تقاضای مشخصی دارد.
- میخواهیم زمان کل در سیستم (شامل زمان انتظار و زمان دریافت خدمات کلیه کارها) حداقل شود.
- مثال:
  - زمان خدمات برای سه فرآیند برابر $t_1=5$، $t_2=10$، و $t_3=4$ میباشد.
  - اگر آنها را به ترتیب [1, 2, 3] زمانبندی کنیم، زمان کل صرف شده در سیستم بقرار زیر خواهد بود:

$$\underbrace{5}_{\text{فرآیند 1}} + \underbrace{(5+10)}_{\text{فرآیند 2}} + \underbrace{(5+10+4)}_{\text{فرآیند 3}} = 39$$

  - از میان تمام $3!=6$ زمانبندی ممکن کمترین زمان مصرفی مربوط به [3, 1, 2] میباشد، که برابر با 32 است.

زمانبندی برای کمینه کردن زمان کل در سیستم

- الگوریتم بدیهی: در نظر گرفتن تمام زمانبندیها ← $\theta(n!)$
- الگوریتم حریصانه:
  - مرتب سازی کارها به ترتیب غیر نزولی زمان خدمات آنها
  - ارائه خدمات به کارها به ترتیب فوق (اول به کوچکترین زمان خدمات رسانی شود). ← $w(n)=\theta(n\lg n)$
- اثبات بهینگی: قضیه 4.3 کتاب
- تعمیم به چند سرویس دهنده:
  - تعداد n کار و تعداد m سرویس دهنده داریم.
  - مرتب سازی کارها و ارائه به سرویس دهنده ها به ترتیب غیر نزولی

$t_1, \ \ldots \ , t_n$

کوچکترین — بزرگترین

کار 1 ← سرویس دهنده 1

⋮

کار m ← سرویس دهنده m

کار m+1 ← سرویس دهنده 1

⋮

زمانبندی با مهلت معین برای دستیابی به بیشترین سود

- هر کار در یک واحد زمانی انجام میگیرد و دارای یک سود است، که سود تنها درصورت انجام کار قبل از یا در زمان مهلت معین مربوطه قابل حصول خواهد بود.
  - مثلاً مهلت 2 یعنی کار میتواند در زمان 1 یا 2 انجام شود.
- مثال:

Job	Deadline	Profit
1	2	30
2	1	35
3	2	25
4	1	40

Schedule	Total Profit	
[1, 3]	30+25 = 55	زمانبندیهای
[2, 1]	35+30 = 65	ممکن (شدنی)
[2, 3]	35+25 = 60	
[3, 1]	25+30 = 55	
[4, 1]	40+30 = $\boxed{70}$	سود بیشینه
[4, 3]	40+25 = 65	

- الگوریتم بدیهی: $\theta(n!)$

زمانبندی با مهلت معین برای دستیابی به بیشترین سود

- الگوریتم حریصانه:
  - مرتب سازی کارها به ترتیب غیر صعودی سود آنها
  - قرار میدهیم : S={}
  - تا زمانی که نمونه حل نشده است:
    - کار بعدی را انتخاب کن
    - اگر با افزودن این کار ، S شدنی باقی بماند، این کار را به S اضافه کن.
    - اگر کار دیگری وجود ندارد، نمونه حل شده است.
- مثال:

Job	*Deadline*	*Profit*
1	3	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

1. *S* is set to Ø.
2. *S* is set to {1} because the sequence [1] is feasible.
3. *S* is set to {1,2} because the sequence [2,1] is feasible.
4. {1,2, 3} is rejected because there is no feasible sequence for this set.
5. *S* is set to {1,2,4} because the sequence [2,1,4] is feasible.
6. {1,2,4,5}is rejected because there is no feasible sequence for this set.
7. {1,2,4,6}is rejected because there is no feasible sequence for this set.
8. {1,2,4,7}is rejected because there is no feasible sequence for this set.

زمانبندی با مهلت معین برای دستیابی به بیشترین سود

- **بررسی شدنی بودن:** یک مجموعه از کارها بنام S شدنی است اگر و تنها اگر ترتیب حاصل از مرتب سازی کارهای S بصورت **غیرنزولی مهلت‌ها** شدنی باشد.
- مثال: بررسی شدنی بودن S={1,2,4,7} در مثال قبل:
  - مرتب سازی بترتیب غیر نزولی مهلت‌ها: [2, 7, 1, 4]

    √ √ √ ×

    چون کار 4 خارج از مهلتش اجرا میشود شدنی نیست

Job	Deadline	Profit
1	3	40
2	1	35
3	1	30
4	3	25
5	1	20
6	3	15
7	2	10

- **اثبات بهینگی:** قضیه 4.4 کتاب
- الگوریتم 4.4 کتاب:

```
J = [1];
for (i = 2; i <= n; i++){
        K = J with i added according to nondecreasing
            values of deadline [i];
        if (K is feasible)
                    J = K;
}
```

زمانبندی با مهلت معین برای دستیابی به بیشترین سود

- تحلیل پیچیدگی زمانی:
  - اندازه ورودی: n (تعداد کارها)
  - عمل اصلی: عملیات داخل حلقه

- در آغاز مرتب سازی برحسب سود $\theta(n\lg n)$ زمان میبرد.
- در هر تکرار حلقهٔ i برای بررسی شدنی بودن، حداکثر i-1 مقایسه جهت درج کار i در لیست مرتب فعلی برحسب مهلت ها نیاز است.
- همچنین حداکثر i مقایسه برای تعیین شدنی بودن لازم است.
- پس زمان تمام تکرارهای حلقه میشود:
- زمان غالب در بدترین حالت : $w(n)= \theta(n^2)$

```
J = [1];
for (i = 2; i <= n; i++){
        K = J with i added according to nondecreasing
            values of deadline [i];
        if (K is feasible)
                    J = K;
}
```

$$\sum_{i=2}^{n}[(i-1)+i] = n^2 - 1 \in \Theta(n^2)$$

√508