بخش

آ

روش‌های طراحی الگوریتم‌ها

در این بخش روش‌های مختلف طراحی الگوریتم‌ها مورد بررسی قرار می‌گیرند. به‌طور کلی، برای حل مسئله‌های مختلف از یکی از این روش‌ها استفاده می‌شود:

۱. مبتنی بر استقرا[۱]،
۲. تقسیم‌وحل[۲]،
۳. برنامه‌ریزی پویا[۳]،
۴. حریصانه[۴]، و
۵. جست‌وجوی فضای حالت[۵].

چهار روش اول اغلب برای طراحی کارا یا چندجمله‌ای راه‌حل به‌کار می‌روند که توضیح مفصل آن‌ها در فصل‌های بعد داده خواهد شد. اما مسئله‌های زیادی هستند که به احتمال زیاد راه‌حل چندجمله‌ای ندارند و برای حل آن‌ها به‌ناچار باید فضای حالت مسئله را

---
[۱] induction
[۲] divide and conquer
[۳] dynamic programming
[۴] greedy
[۵] state space search

بررسی و جست‌وجو کرد و راه‌حل مناسب را به‌دست آورد. برای به‌نظم درآوردن این کار می‌توان از روش پسگرد[۶] استفاده کرد. اما مسئله‌ها ممکن است بسیار بزرگ باشند و تولید فضای حالت آن‌ها شاید خیلی کند شود. محدود ساختن فضای جست‌وجو راهی است که موجب کاهش قابل توجه زمان اجرای الگوریتم می‌شود. در این رابطه، استفاده از «درخت بازی[۷]»، به‌خصوص از روش هرس $\alpha - \beta$ در پیاده‌سازی مسئله‌هایی که از جنس بازی‌های دو یا چند نفره هستند معمول است. هم‌چنین روش «انشعاب و کران[۸]» برای کاهش فضای حالت برخی از مسئله‌های بهینه‌سازی بسیار مورد استفاده قرار می‌گیرد.

در این بخش از کتاب این روش‌های حل با جزئیات کافی ارائه و با مثال‌های زیادی دنبال می‌شوند.

البته روش‌های دیگری مانند الگوریتم‌های «تقریبی قابل اثبات[۹]» هستند که برای حل سریع و نادقیق برخی مسئله‌ها بسیار مورد توجه هستند. در این مسئله‌ها میزان تقریب اثبات می‌شود و این کار ارزش علمی بالایی دارد. در برابر آن، روش مکاشفه‌ای[۱۰] است که پاسخ نادقیقی برای مسئله به‌دست می‌آورد که میزان تقریب آن‌ها به‌صورت نظری اثبات نمی‌شود، بلکه به‌صورت تجربی با دیگر روش‌ها مقایسه می‌شود. خیلی از راه‌حل‌ها در عمل این‌چنین‌اند.

backtracking[۶]
game tree[۷]
branch and bound[۸]
approximation algorithms[۹]
heuristic[۱۰]

فصل

۱

طراحی الگوریتم با استقرا

برای حل مسئله‌هایی که ماهیت بازگشتی دارند، استقرا یک روش کارآمد است. به‌کمک استقرا می‌توان برای چنین مسئله‌هایی الگوریتم‌های مناسب طراحی کرد، درستی آن‌ها را اثبات و تحلیل کرد.

در این فصل چند مسئله را توضیح می‌دهیم که به این روش حل می‌شوند.

۱–۱ مسئله‌ی «ستاره‌ی مشهور»

می‌خواهیم با انجام تعداد کمی پرسش از n نفر، فردی که ویژگی‌های یک «ستاره‌ی مشهور[۱]» را دارد، در صورت وجود، بیابیم. یک نفر ستاره است اگر تک‌تک افراد، او را از پیش بشناسند ولی او هیچ‌کس را نشناسد. گراف «شناخت» بین این افراد گرافی جهت‌دار است با n گره و دستِ‌بالا با $n(n-1)$ یال که ما از پیش اطلاعی از آن نداریم، اما در واقع با انجام این پرسش‌ها، بخش‌هایی از آن‌را می‌سازیم. ما مجاز هستیم از یک فرد a بپرسیم که آیا b را می‌شناسد یا خیر؟ این یک پرسش است. پاسخ مثبت به این پرسش یعنی که در

[۱] celebrity

گراف شناخت، یال $a \to b$ وجود دارد و پاسخ منفی می‌گوید که چنین یالی وجود ندارد.

ما البته در بدترین حالت با $n(n-۱)$ پرسش، دو بار از هر دو نفر، می‌توانیم گراف شناخت را به‌طور کامل بسازیم؛ ستاره در صورت وجود، تنها گره‌ای است که درجه‌ی ورودی آن $n-۱$ و درجه‌ی خروجی آن صفر است. نشان می‌دهیم که می‌توانیم این مسئله را در بدترین حالت با $۳(n-۱) - \lceil \frac{\lg n}{۲} \rceil$ پرسش حل کنیم.

راه‌حل اول: فرض می‌کنیم که با استقرا می‌توانیم بود یا نبود ستاره را بین $n-۱$ نفر اول به‌دست آوریم. چون ممکن است تنها یک ستاره باشد، یا این‌که ستاره‌ای نداشته باشیم. یکی از سه حالت زیر ممکن است:

۱. ستاره‌ی نهایی همان ستاره‌ی موجود بین $n-۱$ نفر اول است،
۲. نفر nام ستاره است، یا
۳. ستاره نداریم.

در حالت اول، با فرض وجود ستاره‌ای به‌نام x بین $n-۱$ نفر اول، تنها با دو پرسش دیگر می‌توان مطمئن شد که نفر nام هم x را می‌شناسد و x او را نمی‌شناسد. در آن صورت x ستاره‌ی همه است. اگر $n-۱$ نفر اول ستاره‌ای به‌نام x داشته باشند، اما x ستاره‌ی همه نباشد، مسئله ستاره ندارد. اما اگر $n-۱$ نفر اول ستاره نداشته باشند، باز ممکن است نفر nام ستاره‌ی همه باشد.

برای بررسی حالت دوم، دستِ‌بالا به $۲(n-۱)$ پرسش دیگر نیاز داریم؛ چون باید بپرسیم که آیا هر یک از $n-۱$ نفر اول، نفر nام را می‌شناسد و نیز نفر nام هیچ‌یک از آن‌ها را نمی‌شناسد.

اگر هیچ‌یک از حالت‌های اول و دوم ستاره‌ای را پیدا نکردند، مسئله جواب ندارد.

بیشینه‌ی تعداد پرسش‌ها از رابطه‌ی بازگشتی $T(n) = T(n-۱) + ۲(n-۱)$ و $T(۱) = ۰$ به‌دست می‌آید که پاسخ آن $T(n) = n(n-۱)$ است؛ اگر می‌خواستیم گراف شناخت را به‌طور کامل بسازیم هم به همین تعداد پرسش می‌رسیدیم. اگر ستاره‌ی نهایی همان ستاره‌ی بین $n-۱$ نفر باشد، می‌توانیم دو پرسشی که در دور اول که او درگیرش بوده را دوباره نپرسیم. البته، در حالت اول، کمینه‌ی تعداد پرسش‌ها برای یافتن ستاره $۲(n-۱)$ است و این درصورتی است که شانس بیاوریم و نفر اول که مورد پرسش قرار می‌گیرد ، خود ستاره باشد.

راه‌حل بهتر: به روش حذفی عمل می‌کنیم، به این صورت که در هر پرسش یک نفر را از مجموعه حذف می‌کنیم و با استقرا راه‌حل را دنبال می‌کنیم. برای این کار فرض کنید که از a بپرسیم که آیا b را می‌شناسد. اگر پاسخ او مثبت باشد، a نمی‌تواند

ستاره باشد، و اگر منفی باشد b نمی‌تواند ستاره باشد؛ در هر صورت یکی از این دو حذف می‌شوند. اگر این کار را $n-۱$ بار تکرار کنیم، تنها یک نفر به‌نام c می‌ماند که ممکن است ستاره‌ی همه باشد و تنها c می‌تواند ستاره باشد، که با $۲(n-۱)-۱$ پرسش دیگر روشن می‌شود. از این رو، با این الگوریتم، بیشینه‌ی تعداد کل پرسش‌ها می‌شود $n-۱+۲(n-۱)-۱=۳(n-۱)-۱$. علت وجود ۱− در این فرمول آن است که در پرسش‌های دور دوم، دستِ‌کم یکی از آن‌ها تکراری است.

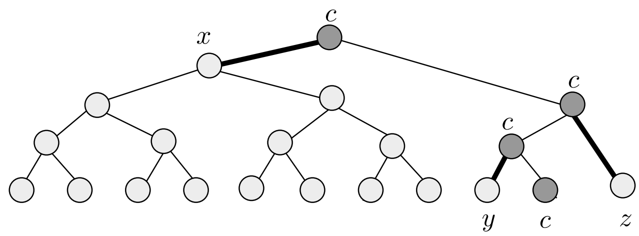
رویه‌ی Celebrity این مسئله را حل می‌کند. در این رویه‌فرض می‌کنیم که اگر i فرد j را بشناسد، مقدار تابع i Knows j مثبت و گرنه منفی است.

```
CELEBRITY (KNOWS, n)
 1  i ← 1
 2  j ← 2
 3  next ← 3
    ▷ in the first phase, eliminate all but one candidate
 4  while next ≤ n + 1
 5     do if i KNOWS j
 6           then i ← next
 7           else j ← next
 8        next ← next + 1
          ▷ one of either i or j is eliminated
 9  if i = n + 1
10     then candidate ← j
11     else candidate ← i
    ▷ Now we check that the candidate is indeed the celebrity
12  k ← 1
13  while k ≤ n and
      k KNOWS candidate and not candidate KNOWS k
14     do k ← k + 1
15  if k = n + 1
16     then celebrity ← candidate
17     else celebrity ← 0    i.e., no celebrity
```

این رویه تا سطر ۹ یک نفر که نامزد یا *candidate* است را به‌دست می‌آورد که ممکن است ستاره باشد که در سطرهای بعد بررسی می‌شود. چنان‌چه گفته شد، این رویه با $۳(n-۱)$ پرسش کار را انجام می‌دهد که این تعداد کمینه نیست و می‌توان بهتر عمل کرد.

می‌توان کاری کرد c که نامزد برای ستاره‌ی مشهور شدن است، در مرحله‌ی اول همیشه به تعداد قابل توجهی مورد پرسش قرار گرفته باشد. اگر نتیجه‌ی همه‌ی پرسش‌ها را ثبت کنیم، در مرحله‌ی دوم الگوریتم دیگر لازم نیست این پرسش‌ها را دوباره بپرسیم. برای این کار توجه کنید که هر پرسش را می‌توان به‌صورت یک گره با دو فرزند در درختی به‌نام

«درخت پرسش» نشان داد که «برنده»ی هر پرسش (کسی که احتمال ستاره بودن او هست) به‌عنوان پدر در درخت بالا برود. نمونه‌ای از یک درخت پرسش در شکل ۱-۱ نشان داده شده است. با این ترتیب، مجموعه‌ی پرسش‌های مرحله‌ی اول الگوریتم یک درخت را تشکیل می‌دهد که ریشه‌ی آن c است. در الگوریتم ارائه‌شده معلوم نیست که در چه مرحله‌ای و چند بار c مورد پرسش قرار گرفته است؛ ممکن است c در آخرین پرسش این مرحله درگیر شده باشد.



شکل ۱-۱ مثالی از درخت پرسش برای $n = ۱۱$. در این مثال، گره ریشه برنده است و پرسش‌های مربوط به یال‌های سیاه را نباید دوباره پرسید.

اگر درخت پرسش‌ها را به‌صورت یک درخت دودویی متوازن در بیاوریم، یعنی کاری کنیم که همه‌ی افراد در دور اول (یا در مرحله‌ی بعد) مورد پرسش قرار بگیرند و بقیه‌ی درخت را همین طور بسازیم، مطمئن خواهیم بود که c پیش از این، در تعدادی پرسش که متناسب با ارتفاع درخت است، قرار داشته است. برای این کار ابتدا میهمانان را به دسته‌های دوتایی تقسیم می‌کنیم (اگر n فرد باشد، یک نفر تنها می‌ماند).

از یک عضو هر دسته می‌پرسیم که آیا او فرد دیگرِ آن دسته را می‌شناسد. «برندگان» این پرسش‌ها (یعنی کسی که می‌ماند) به مرحله‌ی بعد می‌روند و این پرسش‌ها با آن‌ها انجام می‌شوند. اگر فردی در مرحله‌ای تنها ماند، کاری می‌کنیم که در مرحله‌ی بعد، در یک دسته‌ی دوتایی قرار بگیرد. این کار را تکرار می‌کنیم تا به یک نفر برسیم. این فرد همان نامزد ستاره‌شدن است.

عمق هر برگ این درخت، نشان‌دهنده‌ی تعداد پرسش‌هایی است که از او شده است. ارتفاع این درخت برابر $\lceil \lg n \rceil$ است و با توجه به احتمال تنها افتادن نامزد، یک بار در هر دو مرحله از دسته‌بندی‌ها، مطمئن هستیم که این فرد پیش از این دستِ‌کم در $k = \lceil \frac{\lg n}{۲} \rceil$ پرسش درگیر بوده است. یعنی در مرحله‌ی دوم الگوریتم، از $۲(n - ۱)$ پرسش، مطمئنیم که دستِ‌کم k تای آن‌را در مرحله‌ی اول پرسیده‌ایم. پس تعداد کل پرسش‌ها دستِ‌بالا برابر $۳(n - ۱) - \lceil \frac{\lg n}{۲} \rceil$ می‌شود.

تمرین‌های ۱-۱

۱-۱.۱ (خبرپراکنی)
n نفر به نام‌های $a_۱, a_۲, \cdots, a_n$ هر کدام خبری در اختیار دارند. خبر فرد iام را b_i می‌نامیم. توجه کنید که در حالت کلی b_iها با هم متفاوتند. دو نفر ممکن است با هم تماس بگیرند. اگر a_i با a_j تماس بگیرد و a_i پیش از تماس، خبرهای β_i و a_j خبرهای β_j را داشته باشند (β_i و β_j مجموعه‌هایی از خبرهای اولیه‌ی b_iها هستند)، پس از تماس، هر دو دارای خبرهای $\beta_i \cup \beta_j$ خواهند شد. به‌عبارتی خبرهای خود را به یکدیگر منتقل می‌کنند. یک مرحله از خبرپراکنی، تماس هم‌زمان و دوبه‌دوی این n نفر با هم است. توجه کنید که در یک مرحله از خبرپراکنی، یک فرد نمی‌تواند با بیش از یک نفر دیگر تماس بگیرد. هدف مسئله این است که پیدا کنیم در چند مرحله و چگونه می‌توان خبرهای اولیه‌ی b_iها را در اختیار همه‌ی a_iها قرار داد.

الف) اگر $n = ۷$ باشد، کمینه‌ی تعداد مرحله‌ها برای خبرپراکنی را به‌دست آورید و نیز نشان دهید که در هر مرحله چه افرادی باید با هم تماس بگیرند.

ب) کمینه‌ی تعداد مرحله‌ها را برای $n = ۲^k$ به‌دست آورده اثبات نمایید.

۱-۱.۲ (مهمانی سیاسی)
در یک مهمانی سیاسی n نماینده شرکت دارند که هر کدام عضو تنها یک حزب سیاسی است. شما نمی‌توانید از یک نماینده بپرسید که عضو کدام حزب است و فقط از برخورد دو نماینده با هم متوجه می‌شوید که آیا عضو یک حزب هستند یا خیر (اگر با هم صمیمی بودند عضو یک حزب هستند و گرنه از دو حزب متفاوتند). با فرض این‌که بیش از نیمی از نمایندگان متعلق به یک حزب هستند، الگوریتمی از $\mathcal{O}(n)$ ارائه کنید تا یک نماینده از حزب اکثریت را پیدا کند.

۱-۱.۳ (دعوت به مهمانی)
n نفر در یک مهمانی گرد هم آمده‌اند. میزبان می‌خواهد بیشترین تعداد از مهمانان را به صرف شام دعوت کند به‌طوری که هر یک از دعوت شدگان دستِ‌کم k آشنا در میان سایر دعوت شدگان داشته باشد. آشنایی یک رابطه‌ی دوطرفه است و ما همه‌ی اطلاعات آشنایی‌ها را از پیش داریم. برای این کار یک الگوریتم چندجمله‌ای ارائه دهید و آن‌را اثبات نمایید.

۱-۱.۴ (شام به افراد متنفر از هم)
قرار است n نفر را به شام دعوت کنید که در بین آن‌ها R زوج هست و می‌دانیم که برای زوج $(u, v) \in R$، u و v از هم متنفرند و حاضر نیستند با هم سر یک میز بنشینند. می‌خواهیم تعداد کمینه‌ی میزها و محل نشستن هر فرد را تعیین کنیم تا افراد متنفر از هم سر یک میز ننشینند. فرض کنید که میزها گنجایش نشاندن تعداد زیادی مهمان را دارند. آیا یک الگوریتم کارا برای این مسئله هست؟

۱-۲ محاسبه‌ی عدد nام فیبوناچی

nامین عدد فیبوناچی[۲] یا F_n، از رابطه‌ی بازگشتی زیر تعریف می‌شود:

$$F_n = \begin{cases} ۰ & \text{if } n = ۰, \\ ۱ & \text{if } n = ۱, \\ F_{n-۱} + F_{n-۲} & \text{if } n > ۱. \end{cases} \qquad (۱-۱)$$

به‌عنوان مثال، $۰, ۱, ۱, ۲, ۳, ۵, ۸, ۱۳, ۲۱, ۳۴$ اولین تا دهمین عددهای فیبوناچی هستند.

هدف از این قسمت ارائه و تحلیل روش‌های مختلفِ محاسبه‌ی F_n برای هر $n \geq ۰$ دل‌خواه (شاید خیلی بزرگ) است. هم‌چنین می‌خواهیم روشی بیان کنیم که با دقت و در زمانِ چندجمله‌ای F_n را محاسبه کند.

۱-۲-۱ روش استقرایی

از رابطه‌ی بازگشتی استفاده می‌کنیم.

```
FIBONACCI (n)
    ▷ Computing the nth Fibonacci number
    if n = 0 then return 0
    if n = 1 then return 1
    if n > 1 then return FIBONACCI(n − 1)+FIBONACCI(n − 2)
```

چنان‌چه در کتاب [۷] آمد، با این الگوریتم، محاسبه‌ی F_n به‌طور دقیق با $F_n - ۱$ بار عمل جمع انجام می‌دهد. اما این الگوریتم به‌ظاهر خطی مسئله را در زمان نمایی حل می‌کند، چون اندازه‌ی ورودی برابر $\log F_n$ است.

اما می‌دانیم که $F_n = \left\langle \frac{\phi^n}{\sqrt{۵}} \right\rangle$ که $\langle x \rangle$ نزدیک‌ترین عدد صحیح به x و $\phi = (۱ + \sqrt{۵})/۲$ نسبت طلایی[۳] است.

به‌عبارت دیگر با این فرمول این مسئله را می‌توان از مرتبه‌ی $\Omega(\phi^n)$ هم حل کرد.

البته ϕ^n را می‌توان با $\mathcal{O}(\lg n)$ بار مجذور کردن متوالی ϕ به‌دست آورد که از نظر زمان اجرا یک الگوریتم خطی است ولی یک مشکل جدی محاسباتی دارد. مشکل این‌جاست که

[۲] Fibonacci
[۳] golden ration

محاسبات در عمل بر روی عددهای با ممیز شناور انجام می‌شود و حاصل کار حاوی خطای محاسباتی است. پس برای nهای بزرگ، عدد به‌دست آمده ممکن است درست نباشد.

۱-۲-۲ روش خطی زمانی و بدون خطای محاسباتی

لم ۱-۱ عددهای فیبوناچی در رابطه‌ی زیر صدق می کنند.

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n. \qquad (۱-۲)$$

اثبات: با استقرا اثبات می‌کنیم. برای $n = 1$ واضح است که

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

برای $n \geq 2$ و طبق تعریف اصلی رابطه‌های زیر برقرارند؛

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n.$$

□

با استفاده از رابطه‌ی ۱-۲، مشخص است که برای به‌دست آوردن F_n باید ماتریس $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ را n بار در خودش ضرب کرد و در $\Theta(\lg n)$ و بدون خطای محاسباتی به‌طور دقیق محاسبه کرد. این یک الگوریتم خطی است. اما عددهای میانی ممکن است خیلی بزرگ و نسبت به n، نمایی شوند. پس این الگوریتم برای عددهای بزرگ هم‌چنان نمایی است، اما اگر عددها طوری باشند که بر روی آن‌ها بتوان در زمان ثابت عملیات حسابی انجام داد، الگوریتم خطی است.

تمرین‌های ۱-۲

۱-۲.۱ (نسبت طلایی)
اثبات کنید که نسبت طلایی ϕ و مزدوج آن $\hat{\phi}$ ریشه‌های معادله‌ی $x^2 + x + 1 = 0$ هستند.

۱-۲.۲ (محاسبه با فرمول)

با استقرا اثبات کنید که iامین عدد فیبوناچی در معادله‌ی $f_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$ صدق می‌کند.

۱-۳.۲ $*$

ثابت کنید که به‌طور یکتا، هر عدد طبیعی N به‌صورت جمع عددهای متفاوتی از دنباله‌ی فیبوناچی که شامل دو جمله‌ی متوالی نیستند قابل نمایش است. یعنی، $N = \sum_{i=1}^{m} F_{i_j}$ که $|i_j - i_{j-1}| \geq 2$. البته در مورد عددهای فیبوناچی می‌دانیم: $F_1 = 1$، $F_2 = 2$ و $F_{n+2} = F_{n+1} + F_n$.

مثال:

$10 = F_5 + F_2$ represented by 10010
$11 = F_5 + F_3$ represented by 10100
$12 = F_5 + F_3 + F_1$ represented by 10101.

۱-۴.۲ (محاسبه در پیمانه) $*$

بگویید با چه الگوریتم کارایی می‌توان iامین عدد فیبوناچی را به پیمانه‌ی یک عدد صحیح r به‌دست آورد.

۱-۳ کوچک‌ترین دایره‌ی محاطی

می‌خواهیم کوچک‌ترین دایره‌ی محاطی[۴]ای را بیابیم که همه‌ی n نقطه‌ی داده‌شده، یا بر روی محیط یا در داخل آن باشند. کاربرد این مسئله زیاد است؛ فرض کنید می‌خواهیم تعدادی بسته را از روی زمین به وسیله‌ی یک رُبات با بازوی متحرک برداریم. پایه‌ی رُبات را در کدام نقطه قرار دهیم تا با کمترین اندازه‌ی بازو، این کار امکان‌پذیر باشد؟ در این مثال می‌خواهیم فاصله‌ی پایه‌ی رُبات تا دورترین بسته کمینه شود.

یک راه‌حل «کورکورانه» بر این اساس است که در جواب بهینه بدون شک دو یا سه تا از نقطه‌های ورودی بر روی محیط دایره‌ی بهینه هستند. پس، همه‌ی سه نقطه‌ها را در نظر می‌گیریم (دستِ‌بالا به تعداد $\binom{n}{3}$) و دایره‌ای را که از سه نقطه می‌گذرد رسم و بررسی می‌کنیم که آیا بقیه‌ی نقطه‌ها در داخل یا بر روی محیط این دایره هستند یا خیر. هم‌چنین، همه‌ی زوج-نقطه‌ها (به تعداد $\binom{n}{2}$) را در نظر می‌گیریم و به قطر هر زوج-نقطه دایره‌ای رسم و بررسی می‌کنیم که آیا بقیه‌ی نقطه‌ها را در برمی‌گیرد. کوچک‌ترین این دایره‌های محاطی جواب است. بدیهی است که این راه‌حل از $\mathcal{O}(n^4)$ است.

این مسئله یک راه‌حل $\mathcal{O}(n \lg n)$ دارد که مبتنی بر ساخت «نمودار ورونوی دورترین

[۴] smallest enclosing circle

فاصله[۵]» است که در زمینه‌ی «هندسه‌ی محاسباتی[۶]» است و الگوریتم آن هم چندان ساده نیست.

ما در این قسمت یک الگوریتم ساده‌ی تصادفی[۷] و مبتنی بر استقرا ارائه می‌کنیم که در زمان میانگین $\mathcal{O}(n)$ این مسئله را حل می‌کند و پیاده‌سازی آن هم بسیار ساده است. در این قسمت با این الگوریتم‌ها و نحوه‌ی تحلیل آن‌ها آشنا می‌شوید.

۱-۳-۱ ساختار افزایشی راه‌حل

فرض کنید $P = \{p_۱, p_۲, \cdots, p_n\}$ مجموعه‌ی n نقطه‌ی داده‌شده باشد که به‌صورت تصادفی شماره‌گذاری شده‌اند. هم‌چنین فرض کنید $P_i = \{p_۱, p_۲, \cdots, p_i\}$ و D_i کوچک‌ترین دایره‌ی محاطی برای P_i است.

الگوریتم به‌صورت افزایشی[۸]، رفتار می‌کند، یعنی در مرحله‌ی iام فرض می‌شود $D_{i-۱}$ وجود دارد و ما نقطه‌ی p_i را در آن درج می‌کنیم. اگر p_i درون یا بر روی محیط $D_{i-۱}$ بود (یعنی $p_i \in D_{i-۱}$) که $D_i = D_{i-۱}$ و گرنه D_i دایره‌ای است که بر اساس لم ۱-۲، p_i بر روی محیط آن قرار دارد.

لم ۱-۲ برای $i = ۲, \ldots, n$ داریم:

- اگر $p_i \in D_{i-۱}$، $D_i = D_{i-۱}$.
- اگر $p_i \notin D_{i-۱}$، p_i بر روی محیط D_i است.

این لم بخشی از لم قوی‌تر ۱-۳ است که اثبات آن در زیر می‌آید.

الگوریتم

رویه‌ی $\textsc{MinDisc}(P)$ کوچک‌ترین دایره‌ی محاطی برای مجموعه نقاط P را پیدا می‌کند. برای این کار، ابتدا نقطه‌ها را به صورت تصادفی شماره‌گذاری می‌کند و آن‌ها را تک‌تک در نظر می‌گیرد. بدیهی است که $D_۲$ دایره‌ای است به قطر $\overline{p_۱p_۲}$. الگوریتم نقطه‌ی p_i برای $۳ \leq i \leq n$ و $D_{i-۱}$ را در نظر می‌گیرد. چنان‌چه در لم ۱-۲ آمد، اگر $p_i \in D_{i-۱}$، در

[۵] farthest point Voronoi diagram
[۶] computational geometry
[۷] randomized
[۸] incremental

آن‌صورت $D_i = D_{i-1}$. و گرنه D_i کوچک‌ترین دایره‌ی محاطی نقطه‌های $\{p_1, \cdots, p_{i-1}\}$ است با این شرط که p_i بر روی محیط آن باشد.

<u>MinDisc</u> (P)
    ▷ Input: A set P of n points on the plane
    ▷ Output: Smallest enclosing circle for P
Compute a random permuatation $\{p_1, \cdots, p_n\}$ of P
Let D_2 be the smallest enclosing circle for $P_2 = \{p_1, p_2\}$
for $i \leftarrow 3$ **to** n
  do if $p_i \in D_{i-1}$
      then $D_i \leftarrow D_{i-1}$
      else $D_i \leftarrow$ MinDisc1Point$(\{p_1, \cdots, p_{i-1}\}, p_i)$
return D_n

این کار با فراخوانی رویه‌ی MinDisc1Point انجام می‌شود.

<u>MinDisc1Point</u> (P, q)
    ▷ Input: A set P of n points and a point q such that,
    ▷ there exits an enclosing disk for P with q on its boundary
    ▷ Output: Smallest enclosing circle for P with q on its boundary
Compute a random permuatation $\{p_1, \cdots, p_n\}$ of P
Let D_1 be the smallest circle with p_1 and q on its boundary
for $j \leftarrow 2$ **to** n
  do if $p_j \in D_{j-1}$
      then $D_j \leftarrow D_{j-1}$
      else $D_j \leftarrow$ MinDisc2Points$(\{p_1, \cdots, p_{j-1}\}, p_j, q)$
return D_n

توجه کنید که رویه‌ی دوم هنگامی فراخوانی می‌شود که مسئله‌ی مورد نظر جواب دارد. رویه‌ی MinDisc1Point (P, q) هم با همین روش، کوچک‌ترین دایره‌ی محاطی P را به‌دست می‌آورد که q بر روی محیط آن باشد. با فرض آن‌که می‌دانیم مسئله جواب دارد، نقطه‌های $P \ni p_j$ را با یک ترتیب تصادفی و برای $1 \leq j \leq n$ مورد بررسی قرار می‌دهیم و از D_{j-1} دایره‌ی D_j را می‌سازیم. D_1 شروع کار است که به‌سادگی ساخته می‌شود. مشابه الگوریتم پیشین، اگر $p_j \in D_{j-1}$ باشد، در آن صورت $D_j = D_{j-1}$ وگرنه D_j دایره‌ای است که هم q و هم p_j بر روی محیط آن هستند. این کار با فراخوانی MinDisc2Points

(P, q, p_j) انجام می‌شود، با این فرض که مسئله جواب دارد.

رویه‌ی (P, q_1, q_2) MinDisc2Points هم مانند قبل است، با این تفاوت که D_1 دایره‌ای است که از سه نقطه‌ی p_1، q_1 و q_2 می‌گذرد. اگر $p_k \notin D_{k-1}$، در آن‌صورت D_k دایره‌ای است که از p_k، q_1 و q_2 می‌گذرد. توجه کنید که D_k ممکن است دایره‌ی محاطی $\{p_1, \ldots, p_k\}$ نباشد (چند نقطه خارج از آن قرار گرفته باشند)، ولی با توجه به فرض اولیه که مسئله برای P جواب دارد، در انتهای حلقه D_n به‌درستی کوچک‌ترین دایره‌ای است که P را در بر می‌گیرد و q_1 و q_2 بر روی محیط آن هستند.

```
MinDisc2Points (P, q1, q2)
    ▷ Input:  A set P of n points and two points q1 and q2
    ▷         such that, there exits an enclosing disk for P
    ▷         with q1 and q2 on its boundary
    ▷ Output: Smallest enclosing circle for P
    ▷         with q1 and q2 on its boundary
Compute a random permuatation {p1, ···, pn} of P
Let D0 be the smallest circle with q1 and q2 on its boundary
for k ← 1 to n
  do if pk ∈ Dk−1
       then Dk ← Dk−1
       else Dk ← the circle on pk, q1, and q2 on its boarder
return Dn
```

اثبات درستی الگوریتم

درستی الگوریتم با توجه به لم ۱-۲ تا حدود زیادی روشن است. در این‌جا یک لم قوی‌تر بیان و آن‌را اثبات می‌کنیم.

لم ۱-۳ فرض کنید P مجموعه‌ای از نقطه‌های ورودی در صفحه، R مجموعه‌ای از نقطه‌های مجزا از P و $p \in P$. در آن‌صورت گزاره‌های زیر برقرارند:

۱. اگر دایره‌ای باشد که همه‌ی نقطه‌های P را در بر بگیرد و نقطه‌های R بر روی محیط آن باشد، در این صورت کوچک‌ترین چنین دایره‌ای یکتاست و ما آن‌را با $\mathrm{md}(P, R)$ نشان می‌دهیم.

۲. اگر $p \in \mathrm{md}(P \backslash \{p\}, R)$، در آن‌صورت $\mathrm{md}(P, R) = \mathrm{md}(P \backslash \{p\}, R)$، و

۳. اگر $p \notin \mathrm{md}(P \backslash \{p\}, R)$، در آن‌صورت $\mathrm{md}(P, R) = \mathrm{md}(P, R \cup \{p\})$.

اثبات:

۱. در صورت وجود چنین دایره‌ای که آنرا D می‌نامیم، اگر تعداد نقطه‌های R بیشتر از ۲ باشد، D جواب بهینه است. اگر $|R| \leq ۲$، در آن‌صورت D را می‌توان با حفظ نقطه‌های R بر روی آن، آن‌قدر کوچک کرد تا تعداد نقطه‌های بر روی آن ۲ یا ۳ شود و نتوان بیش از آن دایره را کوچک کرد. در این صورت به یک جواب بهینه می‌رسیم و آن یکتاست.

۲. بدیهی است.

۳. فرض کنید $D_۰ = \mathrm{md}(P\backslash\{p\}, R)$ و نیز مرکز $D_۰$ نقطه‌ی $x_۰$ است. هم‌چنین فرض کنید $D_۱ = \mathrm{md}(P, R)$ جواب مسئله است و مرکز آن $x_۱$ می‌باشد ولی p بر روی محیط آن نیست.

چنان‌چه در شکل ۱-۲ دیده می‌شود، $D_۰$ و $D_۱$ باید شامل نقطه‌های مشترک P باشند؛ یعنی با هم در نقطه‌های $z_۱$ و $z_۲$ تلاقی دارند. در شکل، ناحیه‌ی شامل P به‌صورت خاکستری نشان داده شده است.

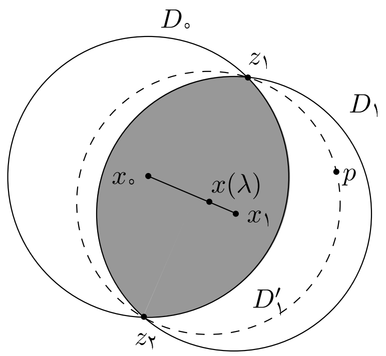
روشن است که ناحیه‌ای که در $D_۱$ هست ولی در $D_۰$ نیست، تنها شامل نقطه‌ی p است و نقطه‌ی دیگری در آن قرار ندارد. پاره‌خط $\overline{x_۰x_۱}$ را رسم می‌کنیم. اگر $x_۱$ را بر روی این خط به‌اندازه‌ی λ به‌سمت $x_۰$ حرکت دهیم تا به $x(\lambda)$ برسیم، دایره‌ای به مرکز $x(\lambda)$ و شعاع $\overline{x(\lambda)z_۰}$ دایره‌ای کوچک‌تر از $D_۱$ خواهد بود که شامل همه‌ی نقطه‌های P است. با زیاد شدن λ، این دایره کوچک‌تر می‌شود. آنرا آن‌قدر بزرگ می‌کنیم تا به دایره‌ی $D'_۱$ برسیم که p بر روی محیط آن قرار دارد. روشن است که $D'_۱$ کوچک‌ترین دایره‌ای است که $R \cup \{p_۱\}$ بر روی محیط آن قرار دارد.

□

تحلیل الگوریتم

بدیهی است که برای P با n نقطه و با فرض آن‌که همه‌ی D_i های مورد نیاز را داریم، هر بار فراخوانی MinDisc2Points به مقدار $\mathcal{O}(n)$ هزینه خواهد داشت. حال فراخوانی MinDisc1Point با همان تعداد نقطه در P را در نظر بگیرید. اگر دستور شماره‌ی ۶ اجرا نشود، زمان اجرای این الگوریتم همیشه $\mathcal{O}(n)$ است. حال مشخص می‌کنیم که این دستور به‌طور میانگین چند بار اجرا می‌شود. برای این کار از روش «تحلیل معکوس[۹]» استفاده

[۹] backward analysis



شکل ۱-۲ اثبات لم ۱-۳. نقطه‌های P در ناحیه‌ی خاکستری مشترک بین $D_\circ$ و D_1 قرار دارند.

می‌کنیم. یعنی فرض می‌کنیم که به‌جای آن‌که نقطه‌ها تک‌تک اضافه شوند آن‌ها را تک‌تک برداریم و احتمالی را به‌دست آوریم که با حذف نقطه‌ی p_j دایره‌ی بهینه کوچک‌تر شود.

این احتمال برابر است با احتمالی که دایره‌ی بهینه بزرگ‌تر شود اگر p_j اضافه شود. می‌دانیم که در هر حال سه نقطه یا دو نقطه بر روی محیط دایره‌ی محاطی بهینه قرار دارد، که در رویه‌ی MinDisc1Point یکی از آن‌ها q است. برای سه نقطه و به‌روش معکوس، در مرحله‌ی j ام (برای $j = ۳ \dots n$) دایره‌ی بهینه کوچک‌تر می‌شود اگر p_j یکی از دو نقطه‌ای باشد که بر روی محیط قرار دارد. احتمال این حالت برابر است با $\frac{۲}{j-۲}$. اگرهم همیشه غیر از q یک نقطه‌ی دیگر در محیط دایره باشد، احتمال $\frac{۱}{j-۱}$ می‌شود. در هر صورت، میانگین زمان اجرای الگوریتم MinDisc1Point برابر است با:

$$\max\{\sum_{j=۳}^{n} \frac{۲}{j-۲}\mathcal{O}(j), \sum_{j=۲}^{n} \frac{۱}{j-۱}\mathcal{O}(j)\} = \mathcal{O}(n).$$

با استفاده از این تحلیل و با همین روش، به‌سادگی می‌توان دید که زمان اجرای رویه‌ی اصلی یعنی MinDisc هم به‌طور میانگین $\mathcal{O}(n)$ است.

تمرین‌های ۱-۳

۱-۳.۱

برنامه‌ی محاسبه‌ی کوچک‌ترین دایره‌ی محاطی n نقطه را بنویسید.

۱-۳.۲ (پوسته‌ی محدب) *
الگوریتمی از مرتبه‌ی $\mathcal{O}(n\log n)$ برای یافتن پوسته‌ی محدب n نقطه را به‌دست آورید.

۱-۳.۳ (یافتن قطر) *
مجموعه‌ای از n نقطه در فضای ۲- بعدی داده شده است. الگوریتمی ارائه و تحلیل کنید تا قطر این نقطه‌ها را به‌دست آورد که برابر است با کمترین فاصله‌ی دو خط موازی که همه‌ی نقطه‌ها را در بر بگیرند. الگوریتمی از $\mathcal{O}(n\lg n)$ برای این کار پیشنهاد کنید.

۱-۳.۴
مجموعه‌ای از n نقطه در فضای ۲-بعدی داده شده است. الگوریتمی ارائه و تحلیل کنید تا کوچک‌ترین مربع محاطی این نقطه‌ها، با ضلع‌های موازی با محورهای مختصات، را به‌دست آورد.

۱-۳.۵
الگوریتمی از مرتبه‌ی $O(|V|)$ برای یافتن قطر یک درخت آزاد (گراف بدون جهت و بدون دور و وزن) ارائه کنید. قطر درخت بیشینه‌ی طول مسیر بین دو گره در گراف است. الگوریتم خود را توصیف کنید.

تمرین‌های فصل ۱

۱.۱ (رابطه‌ی بازگشتی)
دنباله‌ی f روی اعداد طبیعی به این صورت تعریف می‌شود:

$$\begin{cases} f_i = d_i & 1 \le i \le k \\ f_i = \sum_{j=1}^{k} c_j \times f_{n-j} & i > k \end{cases}$$

که در آن $k \ge 1$، d_i ها و c_i ها عددهای ثابت و صحیحی هستند. الگوریتمی از زمان $\mathcal{O}(\lg n)$ بدهید که جمله‌ی nاُم این دنباله (f_n) را محاسبه کند.

۲.۱
آرایه‌ی $A[0..n-1]$ از عددهای حقیقی داده شده است. در نظر داریم که از A ماتریس $B[0..n-1, 0..n-1]$ را طوری بسازیم که برای $i \le j$ داشته باشیم: $B[i,j] = A[i] + A[i+1] + \cdots + A[j]$. الگوریتم کارایی برای این مسئله پیشنهاد کنید و مرتبه‌ی آن‌را محاسبه کنید.

۳.۱ (هزینه‌ی سرشکنی ۱)
عددهای a_1 تا a_n داده شده‌اند. برای هر i (در صورت وجود) بزرگ‌ترین j کوچک‌تر از i را بیابید

به‌گونه‌ای که $a_i > a_j$ باشد. برای این مسئله، الگوریتمی با زمان اجرای کل $\mathcal{O}(n)$ طراحی کنید (یا این‌که هزینه‌ی سرشکن شده برای هر i برابر $\mathcal{O}(1)$ باشد).

۴.۱ (هزینه‌ی سرشکنی ۲)

n عدد $a_1 \leq a_2 \leq \cdots \leq a_n$ داده شده‌اند. می‌خواهیم k تا سه‌تایی از این عددها با شرط‌های زیر انتخاب کنیم:

- هر عدد دستِ‌بالا در یک سه‌تایی باشد،
- اگر x_i، y_i و z_i، iامین سه‌تایی باشد ($x_i \leq y_i \leq z_i$)، مجموع $\sum_{i=1}^{k}(y_i - x_i)^2$ کمینه شود.

الگوریتمی از $\mathcal{O}(nk)$ برای یافتن این سه‌تایی‌ها ارائه دهید.

۵.۱ (آرایه‌ی پیچیده)

آرایه‌ی $A[1..2n+1]$ را «آرایه‌ی پیچیده» می‌گوییم اگر

$$A[1] \leq A[2] \geq A[3] \leq \cdots A[2n] \geq A[2n+1]$$

آرایه‌ی نامرتب $B[1..2n+1]$ از عددهای حقیقی داده شده است. الگوریتمی کارا پیشنهاد کنید تا با تعویض عنصرها، B را به‌صورت یک آرایه‌ی پیچیده در بیاورد. نوع الگوریتم چیست و هزینه‌ی آن چقدر است؟

۶.۱ (دَوَران رشته)

یک الگوریتم خطی پیشنهاد کنید که تشخیص دهد آیا یک رشته‌ی T یک «دوران» از رشته‌ی T' هست یا خیر. به‌عنوان مثال، `arc` و `car` دورانی از هم هستند.

۷.۱ (داده‌ساختار، هزینه‌ی سرشکنی)

در این مسئله داده‌ساختاری برای ذخیره‌ی n عدد معرفی می‌شود که درج و جست‌وجو در آن سریع انجام شود. این داده‌ساختار از مجموعه‌ای از $\lceil \lg n \rceil$ آرایه تشکیل شده است که طول آرایه‌ی iام، 2^i است. هر یک از آرایه‌ها یا خالی است یا به‌طور کامل پر، اما هر آرایه مرتب است. پر یا خالی بودن آرایه‌ها به نمایش دودویی عدد n بستگی دارد. برای مثال، اگر ۱۱ عدد ذخیره شده باشد، چون 1011 =۱۱ است، داده‌ساختار به صورت زیر است:

A_0: [۵]

A_1: [۴, ۸]

A_2: empty

A_3: [۲, ۶, ۹, ۱۲, ۱۳, ۱۶, ۲۰, ۲۵]

ترتیب درج‌ها در داده‌ساختار تهی مشخص می‌کند که هر آرایه‌ای حاوی چه عنصرهایی است.

برای درج عدد x به داده‌ساختار این‌گونه عمل می‌کنیم: ابتدا عدد x را در آرایه‌ی B_0 به‌طول ۱ قرار می‌دهیم. اگر A_0 خالی باشد، آن‌را برابر B_0 قرار می‌دهیم و کار تمام است. و گرنه، A_0 را خالی می‌کنیم و آرایه‌ای به‌طول ۲، حاصل از ادغام A_0 و B_0 می‌سازیم و آن‌را B_1 می‌نامیم. اگر A_1 خالی باشد، آن‌را برابر B_1 قرار می‌دهیم. و گرنه، آن‌را خالی می‌کنیم و B_2 را با ادغام A_1 و B_1 می‌سازیم.

به‌همین ترتیب ادامه می‌دهیم تا زمانی که A_i خالی شود. برای مثال، اگر عدد ۱۰ در این داده‌ساختار درج شود، داده‌ساختار با هزینه‌ای متناسب با ۳ به صورت زیر تغییر خواهد کرد ($n =$ ۱۲ $=1100$):

$A_۰$: empty

$A_۱$: empty

$A_۲$: [۴, ۵, ۸, ۱۰]

$A_۳$: [۲, ۶, ۹, ۱۲, ۱۳, ۱۶, ۲۰, ۲۵]

توجه کنید که هزینه‌ی ادغام دو آرایه به طول m، برابر $۲m$ است. در این‌جا هزینه‌ی یک درج، جمع کل ادغام‌هاست.

الف) در این داده‌ساختار با n عنصر، جست‌وجو برای یافتن یک عنصر چگونه انجام می‌شود و هزینه‌ی آن چقدر است؟

ب) هزینه‌ی یک درج در بدترین حالت چقدر است؟

پ) با استفاده از روش انبوهه (aggregate) اثبات کنید هزینه‌ی سرشکنی عمل درج در این داده‌ساختار $\mathcal{O}(\lg n)$ است.

ت) با استفاده از روش تابع پتانسیل اثبات کنید که هزینه‌ی سرشکنی عمل درج در این داده‌ساختار $\mathcal{O}(\lg n)$ است.

۸.۱

بر روی آرایه‌ی A با N عنصر $A[۱]$ تا $A[N]$ که در ابتدا حاوی جای‌گشتی از عددهای ۱ تا N است، رویه‌ی زیر را اجرا می‌کنیم. اشکال این رویه چیست؟

```
MaybeSort (A, N )
k ← 0
repeat
    k ← k + 1
    for i = 1 to N
        do B[i] ← A[A[i]]
    for i = 1 to N
        do A[i] ← B[i]
until ∀_{1≤i≤N} A[i] = i
return k
```

۹.۱ (بلندقدترین فرد، الگوریتم تصادفی)

n نفر با ترتیب تصادفی به صف وارد یک باجه می‌شوند. مسئول قرار است اندازه‌ی قد بلندقدترین این‌ها را به‌دست آورد. او یک برگه دارد که در ابتدا بر روی آن صفر نوشته است. او قد هر فردی را که وارد باجه می‌شود اندازه‌گیری می‌کند و اگر این قد از عدد نوشته‌شده بر روی برگه بزرگ‌تر باشد، عدد برگه را خط می‌زند و به‌جای آن قد فرد وارد شده را می‌نویسد. به این کار او عمل «جابه‌جایی» می‌گوییم. اگر هر نفر با احتمال یک‌سان بتواند بلندقدترین فرد باشد، در پایان، تعداد میانگین عمل جابه‌جایی چندتاست؟ محاسبه و استدلال کنید.

۱۰.۱
برنامه‌ی زیر، دو عدد صحیح نامنفی A و B را در هم ضرب می‌کند. اگر $A^{(0)}$، $B^{(0)}$ و $P^{(0)}$ مقدارهای متغیرهای A، B و P پیش از اجرای دستور شماره‌ی ۲ و $A^{(k)}$، $B^{(k)}$ و $P^{(k)}$ مقدارهای همین متغیرها در انتهای kامین حلقه باشد، چه رابطه‌ای بین این متغیرها برقرار است؟

```
MULT (A, B)
1  P ← 0
2  while (A ≠ 0)
3     do if A mod 2 = 1
4           then P ← P + B
5        A ← A div 2
6        B ← B × 2
7  return P
```

۱۱.۱ (جست‌وجو در ماتریس)
یک ماتریس A با اندازه‌ی $n \times m$ از عددهای متمایز داده شده است. فرض کنید که هر سطر آن از چپ - به - راست و هر ستون آن از پایین - به - بالا به‌صورت صعودی مرتب هستند. می‌خواهیم با دریافت x مشخص کنیم که آیا x در A هست یا خیر و اگر وجود دارد درایه‌ی مربوطه را برگردانیم. یک الگوریتم با کمینه‌ی تعداد مقایسه‌ها (یعنی مقایسه‌ی x با یک درایه یا مقایسه‌ی دو عنصر از آرایه با هم) برای حل این مسئله ارایه دهید. آن‌را اثبات و تحلیل نمایید. تعداد را دقیق (و نه به‌صورت $\mathcal{O}$) به‌دست آورید.

۱۲.۱ (رشته‌ی k-آینه‌ای)
یک رشته به طول n، k-آینه‌ای است اگر خود رشته آینه‌ای باشد (از دو طرف یک‌سان خوانده شود) و پیشوند و پسوندهای به‌طول $\lfloor \frac{n}{2} \rfloor$ آن نیز k-آینه‌ای باشند. درجه‌ی یک رشته را بیشینه‌ی kای می‌نامیم که آن رشته k-آینه‌ای باشد. الگوریتمی کارا ارائه کنید که به‌ازای رشته‌ی s مجموع درجه‌های همه‌ی پیشوندهای s را به‌دست آورد.

منبع‌ها
قسمت‌های این فصل از منبع‌های زیر تهیه شده‌اند.
مسئله‌ی ستاره‌ی مشهور از کتاب [۱۶] برگرفته شده است. مسئله‌ی محاسبه‌ی عددهای فیبوناچی در کتاب‌های مختلفی از جمله [۵] یافت می‌شود. کوچک‌ترین دایره‌ی محاطی از مسئله‌های هندسه‌ی محاسباتی است که در کتاب‌های این رشته از جمله در [۴] آمده است.