دانشکده‌ی علوم ریاضی

داده‌ساختارها و الگوریتم‌ها — دی ۹۳

جلسه‌ی ۱۴: میانه‌ها و آماره‌های ترتیبی

مدرّس: دکتر شهرام خزائی — *نگارنده: حسام رهنما-نوید مشایخی*

۱ مقدمه

منظور از iاُمین آماره‌ی ترتیبی مجموعه‌ای از n عدد، عددی از آن مجموعه است که دقیقا از $i - ۱$ عدد دیگر در آن مجموعه بزرگتر است یا به عبارت دیگر در لیست مرتب شده‌ی (صعودی) این اعداد، در مکان iاُم قرار داشته باشد (iاُمین عضو کوچک مجموعه‌ی آن اعداد باشد). بنابراین به طور مثال اولین آماره‌ی مجموعه‌ای از n عدد همان مینیمم مجموعه‌ی اعداد می‌باشد.
در این بخش از درس، هدف ما پیدا کردن iاُمین آماره می‌باشد و سعی می‌کنیم الگوریتمی بهینه برای آن ارائه دهیم. پس می‌توانیم مسئله‌ی مطرح شده را به صورت زیر ذکر کنیم:

- ورودی: مجموعه‌ای از n عدد (متمایز) و عدد i که $۱ \leq i \leq n$ می‌باشد.
- خروجی: عددی از مجموعه‌ی اعداد ورودی که iاُمین عنصر کوچک می‌باشد.

قبل از حل مسئله‌ی اصلی که پیدا کردن iاُمین آماره است، به مسئله‌ی دیگری می‌پردازیم.

۲ پیدا کردن همزمان مینیمم و ماکزیمم

می‌خواهیم مقادیر مینیمم و ماکزیمم مجموعه‌ای از n عدد را به طور همزمان پیدا کنیم. ابتدا واضح است که اگر بخواهیم هر یک از مقادیر مینیمم و ماکزیمم را به طور جداگانه پیدا کنیم، به راحتی با $n - ۱$ مقایسه می‌توان این مقادیر را یافت، پس در نهایت با $۲n - ۲$ مقایسه به راحتی می‌توانیم هر دو مقدار مینیمم و ماکزیمم را پیدا کنیم.
حال می‌خواهیم دو مقدار مینیمم و ماکزیمم مجموعه‌ای از اعداد را با کمتر از $۲n - ۲$ مقایسه به طور همزمان پیدا کنیم. در اینجا راه حلی ارائه می‌دهیم که با حداکثر $\lfloor \frac{۳n}{۲} \rfloor$ مقایسه این دو مقدار را پیدا کنیم.
روش کلی کار به این صورت است که ابتدا اعداد آرایه را به دسته‌های دوتایی کنار هم تقسیم می‌کنیم. سپس به ازای هر یک از این دسته‌های دو تایی، با ۳ مقایسه مقادیر مینیمم و ماکسیممی را که تا به حال به دست آورده‌ایم، آپدیت () می‌کنیم. ابتدا دو مقدار که در یک دسته قرار دارند را با هم مقایسه کرده، بعد از آن مقدار کوچکتر این دسته را با مینیمم فعلی (که تا الان به دست آورده‌ایم) و مقدار بزرگتر دسته را با ماکسیمم فعلی مقایسه کرده و مقدار آن‌ها را آپدیت می‌کنیم و در نهایت واضح است که مقادیر مینیمم و ماکسیمم را یافته‌ایم.

اگر بخواهیم تعداد مقایسه‌های انجام شده را حساب کنیم، ما به ازای هر دسته‌ی دوتایی، ۳ مقایسه انجام داده‌ایم. بنابراین در کل $\lfloor \frac{۳n}{۲} \rfloor$ مقایسه برای پیدا کردن دو مقدار مینیمم و ماکسیمم انجام شده است که کمتر از $۲n - ۲$ مقایسه‌ای است که در اولین راه حل به آن اشاره شد.

۳ پیدا کردن آماره‌ی iاُم

در این بخش راه حلی برای پیدا کردن iاُمین عضو کوچکتر ارائه می‌دهیم که زمان اجرای آن در بدترین حالت $\Theta(n^۲)$ می‌باشد ولی در حالت میانگین در $\Theta(n)$ کار می‌کند. ابتدا این الگوریتم را معرفی می‌کنیم و سعی می‌کنیم آن را بهبود دهیم که در بدترین حالت هم، در زمان خطی اجرا شود.
ایده‌ی کلی استفاده از تابع Randomized-Partition است که در الگوریتم Randomized-QuickSort از آن استفاده کردیم. پس در اینجا الگوریتمی که ارائه می‌دهیم تصادفی می‌باشد. به تابع Randomized-Select که در زیر آمده است توجه کنید.

Algorithm 1 Randomized-Select

function Randomized-Select(Array $A[1...n]$, Index p, Index r, Index i)
  if $p = r$ **then**
    return A[p]
  $q \leftarrow$ Randomized-Partition(A, p, r)
  $k \leftarrow q - p + 1$
  if $i = k$ **then**
    return A[q]
  if $i < k$ **then**
    return Randomized-Select$(A, p, q-1, i)$
  if $i > k$ **then**
    return Randomized-Select$(A, q+1, r, i-k)$

تابع Randomized-Select آرایه‌ی A را به همراه اندیس‌های $p,\ r,\ i$ را به عنوان ورودی دریافت می‌کند و iاُمین عضو کوچک زیرآرایه‌ی $A[p...q]$ را به عنوان خروجی برمی‌گرداند.
در اولین خط و در حلقه‌ی این شرط چک می‌شود که اگر آرایه به طول ۱ باشد، همان عنصر آرایه برگردانده شود. در خط سوم با استفاده از تابع Randomized-Partition آرایه‌ی $A[p...r]$ را به دو زیرآرایه‌ی $A[p...q-۱]$ و $A[q+۱...r]$ تقسیم کردیم که تمام اعداد زیرآرایه‌ی اول از $A[q]$ کوچکتر هستند و تمام اعداد زیرآرایه‌ی دوم از $A[q]$ بیشتر هستند (در اصل عنصر $A[q]$ را به عنوان محور[۱] قرار دادیم). در خط ۴ عدد k را که طول زیرآرایه‌ی $A[p...q]$ می‌باشد را محاسبه می‌کنیم و در خط ۵ چک می‌شود که آیا $A[q]$ که kاُمین عنصر کوچک آرایه‌ی A می‌باشد، iاُمین عنصر کوچک A می‌باشد یا نه ($i = k$ است یا نه). اگر جواب این سوال مثبت باشد، همان مقدار $A[q]$ برگردانده می‌شود و در غیر این صورت تعیین می‌کنیم که iاُمین عضو کوچکتر در کدام یک از زیرآرایه‌های $A[p...q-۱]$ و $A[q+۱...r]$ قرار دارد و متناسب با آن به صورت بازگشتی تابع Randomized-Select را فراخوانی می‌کنیم.

[۱] pivot

۱.۳ بدترین حالت

فرض کنید بخواهیم عنصر مینیمم را پیدا کنیم یعنی تابع را به صورت $\text{Randomized-Select}(A, ۱, n, ۱)$ فراخوانی کنیم. در این صورت اگر تابع $\text{Randomized-Partition}$ در خط سوم، بزرگترین عدد را به عنوان محور انتخاب کنید، در این صورت مجبوریم در بین $n - ۱$ عدد دیگر (به جز عدد ماکزیمم) به دنبال عدد مینیمم باشیم، یعنی تابع $\text{Randomized-Select}(A, ۱, n - ۱, ۱)$ اجرا می‌شود. اگر هر دفعه بزرگترین عدد به عنوان محور انتخاب شود، چون تابع $\text{Randomized-Partition}$ در $\Theta(n)$ اجرا می‌شود، زمان اجرایی نهایی برای پیدا کردن مینیمم با این شرایط $\Theta(n^۲)$ خواهد بود. با این حال چون الگوریتم ارائه شده تصادفی است، به ازای هر ورودی داده شده، بدترین حالت با احتمال بسیار کمی اتفاق می‌افتد و زمان اجرای این الگوریتم به طور متوسط $\Theta(n)$ می‌باشد.

۲.۳ محاسبه‌ی زمان اجرای میانگین

فرض کنید زمان اجرای تابع Randomized-Select بر روی آرایه‌ی $A[p...r]$ با n عنصر که یک متغیر تصادفی است را با $T(n)$ نمایش دهیم. می‌خواهیم کرانی برای $E[T(n)]$ که متوسط زمان اجرای الگوریتم است، به دست آوریم.
ابتدا متغیر تصادفی شاخص X_k را تعریف می‌کنیم. اگر بعد از اجرای تابع $\text{Randomized-Partition}$ زیرآرایه‌ی $A[p...q]$ دارای دقیقا k عنصر باشد، X_k را برابر ۱، و در غیر این صورت آن را صفر در نظر می‌گیریم. چون در تابع $\text{Randomized-Partition}$ احتمال اینکه هر یک از اعضای آرایه به عنوان محور انتخاب شوند، برابر است، می‌توانیم نتیجه بگیریم که به ازای $۱ \leq k \leq n$ داریم:

$$E[X_k] = \frac{۱}{n}$$

حال با توجه به اینکه iاُمین عنصر کوچک نسبت به عنصر محور ($A[q]$) در کدام سمت باشد، تابع بر روی یکی از زیر آرایه‌های
$A[p...q-۱]$ و یا $A[q+۱...r]$ اجرا می‌شود. بنابراین می‌توانیم کرانی به صورت زیر برای $T(n)$ بنویسیم (عبارت $O(n)$ مربوط به اجرای تابع Randomized-Partition می‌باشد).

$$\begin{aligned} T(n) &\leq \textstyle\sum_{k=۱}^{n} X_k(T(\max(k-۱,\ n-k)) + O(n)) \\ &= \textstyle\sum_{k=۱}^{n} X_k T(\max(k-۱,\ n-k)) + O(n) \end{aligned}$$

پس خواهیم داشت:

$$\begin{aligned} E[T(n)] &\leq E[\sum_{k=۱}^{n} X_k T(\max(k-۱,\ n-k)) + O(n)] \\ &= \sum_{k=۱}^{n} E[X_k T(\max(k-۱,\ n-k))] + O(n) \\ &= \sum_{k=۱}^{n} E[X_k].E[T(\max(k-۱,\ n-k))] + O(n) \\ &= \sum_{k=۱}^{n} \frac{۱}{n} E[T(\max(k-۱,\ n-k))] + O(n) \end{aligned}$$

در سیگمای بالا تعدادی از جملات دو بار تکرار شده‌اند (با توجه به اینکه n فرد است یا زوج). پس می‌توانیم عبارت بالا را به صورت زیر بازنویسی کنیم.

$$E[T(n)] \leq \frac{۲}{n} \sum_{k=\lfloor \frac{n}{۲} \rfloor}^{n-۱} E[T(k)] + O(n)$$

با توجه به تعریف ثابت می‌کنیم که $E[T(n)] = O(n)$. پس فرض می‌کنیم که به ازای مقادیر کوچکتر از n رابطه‌ی ذکر شده برقرار باشد یعنی عددی ثابت و مثبت مانند c وجود دارد که $E[T(k)] \leq ck$. همچنین برای عبارت $O(n)$ عدد ثابت مثبتی مانند a در نظر می‌گیریم. پس از عبارت بالا خواهیم داشت:

$$\begin{aligned} E[T(n)] &\leq \frac{۲}{n} \sum_{k=\lfloor \frac{n}{۲} \rfloor}^{n-۱} ck + an \\ &= \frac{۲c}{n}(\sum_{k=۱}^{n-۱} k - \sum_{k=۱}^{\lfloor \frac{n}{۲} \rfloor -۱} k) + an \\ &\leq \frac{۲c}{n}(\frac{(n-۱)n}{۲} - \frac{(\frac{n}{۲}-۲)(\frac{n}{۲}-۱)}{۲}) + an \\ &= c(\frac{۳n}{۴} + \frac{۱}{۲} - \frac{۲}{n}) + an \\ &\leq \frac{۳cn}{۴} + \frac{c}{۲} + an \\ &= cn - (\frac{cn}{۴} - \frac{c}{۲} - an) \end{aligned}$$

اگر نشان دهیم که عبارت بالا کمتر یا مساوی cn است، اثبات کامل می‌شود. برای این منظور کافی است که عبارت داخل پرانتز، بزرگتر یا مساوی صفر شود. با اعمال این شرط و شرط اضافی $c > ۴a$، کرانی مانند $n \geq t$ به دست می‌آید که t عدد ثابتی برحسب a و c است. اگر به ازای $n < t$، $T(n)$ را $O(۱)$ در نظر بگیریم، مشکل حل می‌شود و اثبات کامل می‌شود. پس توانستیم اثبات کنیم که $E[T(n)] = O(n)$.

۴ الگوریتم

در این بخش الگوریتم قطعی را ارائه می‌دهیم که زمان اجرای آن برای پیدا کردن آماره‌ی iاُم در بدترین حالت $O(n)$ می‌باشد.

۱.۴ نحوه‌ی عملکرد الگوریتم

ایده‌ی الگوریتم همانند Randomized-Select می‌باشد با این تفاوت که در سعی می‌کنیم بهترین محور را برای تقسیم آرایه انتخاب کنیم (به جای انتخاب تصادفی). در این الگوریتم از تابع که در از آن استفاده کردیم، بهره می‌بریم با این تفاوت که در اینجا تابع عنصر محور را به عنوان ورودی دریافت می‌کند و سپس تقسیم‌بندی را انجام می‌دهد.

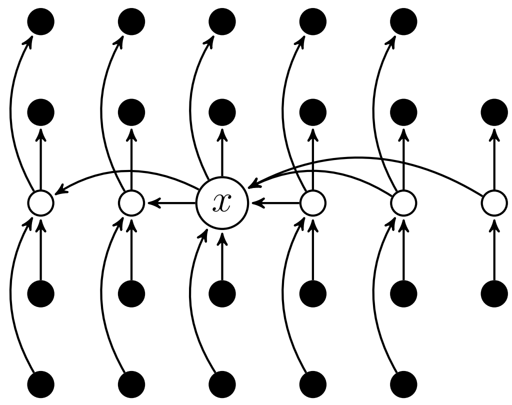
الگوریتم شامل ۵ مرحله‌ی زیر می‌باشد:

- تقسیم n عدد به دسته‌های ۵ تایی ($\lfloor \frac{n}{۵} \rfloor$ دسته‌ی ۵ تایی و حداکثر یک دسته با $n \mod ۵$ عنصر).
- پیدا کردن میانه‌ی هر یک از دسته‌ها (برای این کار می‌توانیم از الگوریتم مرتب‌سازی درجی استفاده کنیم).
- با استفاده از خود تابع میانه‌ی میانه‌هایی که در مرحله‌ی ۲ پیدا کرده‌ایم را به دست می‌آوریم و آن را x می‌نامیم.
- حال از تابع که عنصر محور را به عنوان ورودی دریافت می‌کند، استفاده می‌کنیم. با فراخوانی تابع با ورودی x، آرایه به دو قسمت تقسیم می‌شود، اعداد سمت چپ x در آرایه که همگی کوچکتر از x هستند و اعداد سمت راست x که مقادیری بزرگتر از x دارند. فرض کنید x بعد از فراخوانی تابع در جایگاه kاُم آرایه باشد (یعنی x عنصر کوچک kاُم باشد).
- اگر $i = k$ بود، آنگاه x به عنوان جواب برگردانده می‌شود در غیر این صورت اگر $i < k$، باید عنصر کوچک iاُم را در زیرآرایه‌ی سمت چپ x پیدا کنیم و اگر $i > k$ باید عنصر کوچک $i - k$اُم را در زیرآرایه‌ی سمت راست پیدا کنیم.

۲.۴ زمان اجرا

برای تحلیل زمان اجرا کران پایینی روی تعداد عناصری که از x بزرگتر هستند، به دست می‌آوریم. شکل زیر دسته‌بندی اعداد و میانه‌ها را مشخص کرده است.
در مرحله‌ی دو $\lceil \frac{n}{۵} \rceil$ میانه به دست آوردیم و در مرحله‌ی ۳، عنصر x را میانه‌ی این $\lceil \frac{n}{۵} \rceil$ عدد معرفی کردیم، بنابراین حداقل بیش از نصف میانه‌ها، مقداری بزرگتر یا مساوی x دارند و در نتیجه در هر کدام از دسته‌هایی که این میانه‌های بزرگتر از x قرار دارند،حداقل ۳ مقدار بزرگتر از x یافت می‌شود (به جز دسته‌ای که خود x در آن قرار دارد و دسته‌ای که شامل $n \mod ۵$ عدد می‌باشد). پس تعداد اعداد بزرگتر از x حداقل برابر است با:

$$۳(\lceil\frac{۱}{۲}\lceil\frac{n}{۵}\rceil\rceil - ۲) \geq \frac{۳n}{۱۰} - ۶$$



این شکل دسته‌های اعداد را مشخص می‌کند که هر دسته‌ی ۵ تایی در یک ستون مشخص شده است و مقدار میانه‌ی هر دسته با دایره‌ی سفید نشان داده شده است. فلش‌ها از اعداد بزرگتر به اعداد کوچکتر رسم شده‌اند و عدد x که میانه‌ی میانه‌ها می‌باشد در شکل مشخص شده است.

به طور مشابه تعداد عناصر کوچکتر از x حداقل برابر $\frac{۳n}{۱۰} - ۶$ می‌باشد. پس در بدترین حالت، تابع بر روی آرایه‌ای به طول حداکثر $n - (\frac{۳n}{۱۰} - ۶) = \frac{۷n}{۱۰} + ۶$ اجرا می‌شود. حال می‌خواهیم رابطه‌ی بازگشتی زمان اجرای $T(n)$ مربوط به تابع را محاسبه کنیم.
در مراحل ۱ و ۲ و ۴ زمان $O(n)$ استفاده شده است (در مرحله‌ی ۲ چون اندازه‌ی هر یک از دسته‌ها ثابت و برابر ۵ است، الگوریتم مرتب‌سازی درجی برای هر یک از دسته‌ها در $O(۱)$ اجرا می‌شود و در کل برای همه‌ی دسته‌ها $O(n)$ خواهد بود). مرحله‌ی ۳ که به طور بازگشتی اجرا می‌شود زمان $T(\lceil\frac{n}{۵}\rceil)$ را مصرف می‌کند و در مرحله‌ی ۵ حداکثر زمان $T(\frac{۷n}{۱۰} + ۶)$ مصرف خواهد شد. بنابراین برای $T(n)$ خواهیم داشت:

$$T(n) \leq T(\frac{n}{۵}) + T(\frac{۷n}{۱۰} + ۶) + O(n)$$

با استفاده از تعریف و فرض وجود ثابت مثبتی مانند c که $T(k) \leq ck$، می‌توان ثابت کرد که رابطه‌ی بازگشتی بالا از $O(n)$ می‌باشد. بنابراین توانستیم الگوریتمی قطعی برای پیدا کردن iاُمین عضو کوچک مجموعه‌ای از اعداد ارائه دهیم که بدترین زمان اجرای آن $O(n)$ می‌باشد.