به نام خدا

ساختمان داده‌ها و الگوریتم‌ها (۴۰۲۵۴)
دانشگاه صنعتی شریف
مدرس: دکتر مهدی صفرنژاد

تمرین دوم
انتشار: ۱۶ فروردین ۱۴۰۱

مرتب‌سازی و داده‌ساختارهای پایه

سؤال ۱. نشان دهید $k\ln k = \Theta\left(n\right)$ نتیجه می‌دهد که $k = \Theta\left(\frac{n}{\ln n}\right)$.

پاسخ:

$k\ln k = \Theta\left(n\right)$ نتیجه معادل است با:

$$n = \Theta\left(k\ln k\right)$$

حال با لوگاریتم گرفتن از دو طرف تساوی بالا داریم:

$$\ln n = \Theta\left(\ln\left(k\ln k\right)\right) = \Theta\left(\ln k + \ln\ln k\right) = \Theta\left(\ln k\right)$$

حال با دو عبارت فوق داریم:

$$\frac{n}{\ln n} = \frac{\Theta\left(k\ln k\right)}{\Theta\ln k} = \Theta\left(\frac{k\ln k}{\ln k}\right) = \Theta(k) \Longrightarrow k = \Theta\left(\frac{n}{\ln n}\right)$$

سؤال ۲. آرایه $A\left[۱...n\right]$ و عدد طبیعی k به عنوان ورودی داده شده‌اند. الگوریتمی ارائه دهید که در $O\left(n\right)$ مینیمم هر k عدد متوالی در آرایه را بیابد. (خروجی این الگوریتم باید $n-k+۱$ عدد باشد).

پاسخ:

برای حل این مسئله از داده‌ساختار Deque [1] استفاده میکنیم. به این شکل که هر بار اندیس k عنصر از آرایه را درون آن به گونه‌ای نگه می‌داریم که در آرایه اصلی عناصر مربوط به آن اندیس‌ها از جلو به عقب Deque ترتیبی صعودی داشته باشند. جزئیات اجرای الگوریتم به صورت زیر است:

۱. یک Deque به اسم Q میسازیم.
۲. ابتدا k عنصر اول آرایه را پردازش کرده درون Q قرار می‌دهیم. به این صورت که قبل از وارد کردن اندیس هر عنصر، اندیس تمام عناصر درون Q از عقب را که از عنصر فعلی بزرگتر هستند از همان سمت از Q خارج می‌کنیم. سپس اندیس عنصر فعلی را درون Q، push می‌کنیم.

---

[1] صف دوطرفه

۳.حال از عنصر kام تا انتهای آرایه را پردازش می‌کنیم. از این مرحله به بعد در هربار اجرا حلقه، عنصری از آرایه که اندیس آن در جلوی Q است، کوچک‌ترین عنصر k عنصر متوالی فعلی است. بعد از هر بار چاپ کردن این عنصر، اندیس‌هایی که از زیرآرایه kتایی فعلی بیرون‌اند را از Q خارج می‌کنیم.

۴.اندیس تمام عناصر درون Q از عقب را که از عنصر فعلی بزرگتر هستند از همان سمت از Q خارج می‌کنیم و اندیس عنصر فعلی را در

۵.در آخر نیز عنصر مینیمم مربوط به زیرآرایه آخر را چاپ می‌کنیم.

پیچیدگی زمانی این الگوریتم $O(n)$ است. زیرا هر عنصر را دقیقا یک بار وارد Q و حداکثر یک بار از آن خارج می‌کند و به عناصری که در هر مرحله قرار بر حذفشان نیست کاری ندارد.

سؤال ۳. برای استفاده از لیست پیوندی تک‌سویه، انتظار داریم در آن دور وجود نداشته باشد. اما ممکن است یک لیست پیوندی درست ایجاد نشده باشد و در آن دور وجود داشته‌باشد. می‌خواهیم پیش از شروع کار با یک لیست پیوندی، تشخیص دهیم که در آن دور وجود دارد یا خیر. الگوریتمی با پیچیدگی زمانی $O(n)$ و پیچیدگی حافظه $O(۱)$ ارائه دهید که این کار را انجام دهد.

پاسخ:

برای پیدا کردن دور در لیست پیوندی از دو اشاره‌گر استفاده می‌کنیم. یک اشاره‌گر سریع و یک اشاره‌گر آهسته برای تعیین اینکه آیا یک دور در لیست وجود دارد یا خیر. اشاره‌گر آهسته هر بار یک node به جلو حرکت می کند، در حالی که اشاره‌گر سریع هر بار دو node به جلو حرکت می‌کند. اگر یک دور در لیست پیوندی وجود داشته باشد، اشاره‌گرهای سریع و آهسته در جایی به هم می‌رسند. اگر دوری در لیست وجود داشته باشد، در حداکثر یک بار پیمایش لیست توسط اشاره‌گر آهسته پیدا می‌شود. در غیر این صورت وقتی اشاره‌گر سریع به انتهای لیست می‌رسد به null میخورد و الگوریتم پایان می‌یابد.

node فعلی لیست پیوندی را با $head$ نشان می‌دهیم. شبه کد الگوریتم ذکر شده به صورت زیر است.

Algorithm 1 My algorithm

```
procedure HasCycle( head)
    if head is None then return flase
    slow ← head
    fast ← head.next
    while slow != fast do
        if fast is None or fast.next is None then return false
        slow ← slow.next
        fast ← fast.next.next
    return true
```

سؤال ۴. فرض کنید یک آرایه از اعداد طبیعی در اختیار داریم که هر عنصر حداکثر k خانه با جایگاه درستش در نسخه مرتب شده این آرایه دارد.

الف) نشان دهید الگوریتم مرتب‌سازی درجی این آرایه را در زمان $O(nk)$ مرتب می‌کند.

ب) نشان دهید مرتب کردن این آرایه با هر روشی به $\Omega(n \lg k)$ مقایسه نیاز دارد.

ج) الگوریتمی از مرتبه زمانی $\Theta(n \lg k)$ ارائه دهید که این آرایه را مرتب کند.

پاسخ:

الف) ابتدا با بررسی الگوریتم مرتب‌سازی درج نشان می‌دهیم که INSERTION-SORT(A) در زمان $O(n+I)$ اجرا می‌شود. که I تعداد نابجایی‌ها است. سپس تعداد نابجایی‌های ممکن را در آرایه‌ای می‌شماریم که در آن هر عنصر k خانه با موقعیت درست خود فاصله دارد و نشان می‌دهیم که حداکثر $O(nk)$ نابجایی وجود دارد.

لم: INSERTION-SORT(A) در زمان $O(n+I)$ اجرا می‌شود، که I تعداد نابجایی‌ها در A است.

اثبات: اجرای INSERTION-SORT را روی آرایه A در نظر بگیرید. در حلقه بیرونی $O(n)$ عملیات انجام می‌شود. هر تکرار حلقه داخلی دقیقاً یک نابجایی را اصلاح می‌کند. وقتی الگوریتم خاتمه می‌یابد، هیچ نابجایی باقی نمی‌ماند. بنابراین، باید I تکرار در حلقه داخلی وجود داشته باشد که منجر به زمان اجرای $O(I)$ شود. بنابراین زمان اجرای الگوریتم $O(n+I)$ است.

سپس تعداد نابجایی‌ها را در آرایه‌ای می‌شماریم که در آن هر عنصر k خانه با موقعیت درست خود فاصله دارد.

ما یک کران بالا برای تعداد نابجایی‌ها ارائه می‌دهیم. یک عنصر خاص $A[i]$ را در نظر بگیرید. حداکثر $۴k$ عنصر وجود دارد که می توان آنها را با $A[i]$ معکوس کرد. درواقع عناصر در محدوده $A[i-۲k \cdots i+۲k]$. بنابراین، i بخشی از حداکثر $۴k$ نابجایی است و از این رو حداکثر $۴nk$ وجود دارد.

پس نتیجه می‌گیریم که زمان اجرای مرتب‌سازی درجی در آرایه‌ای که در آن هر عنصر k خانه با موقعیت درست خود فاصله دارد، $O(nk)$ است.

ب) می‌دانیم که مرتب سازی آرایه با اندازه n به $\Omega(n \lg n)$ مقایسه نیاز دارد. اگر $k > \frac{n}{۲}$ آنگاه $n \lg n = \Omega(n \lg k)$ و حکم اثبات می‌شود. حال فرض کنید $k \leq \frac{n}{۲}$. می‌خواهیم یک کران پایین برای تعداد برگ‌ها درخت تصمیم‌گیری برای یک الگوریتم که آرایه‌ای با شرایط ذکر شده در مسئله را مرتب می‌کند ارائه دهیم. بنابراین ما یک کران پایین برای تعداد جایگشت‌های ممکن که این شرط را برآورده می‌کنند ارائه می دهیم.

ابتدا، آرایه با اندازه n را به $\lfloor \frac{n}{k} \rfloor$ بلوک به اندازه k ــ احتمالا بلوک آخر کمتر ــ تقسیم می‌کنیم. برای هر بلوک $k!$ جایگشت وجود دارد که در مجموع حداقل $(k!)^{\lfloor \frac{n}{k} \rfloor}$ جایگشت از کل آرایه را می‌دهد. هیچ کدام از این جایگشت‌ها هیچ عنصری را بیشتر از k خانه جابجا نمی‌کند. بنابراین درخت تصمیم‌گیری حداقل $(k!)^{\lfloor \frac{n}{k} \rfloor}$ برگ دارد. از آنجایی که درخت تصمیم‌گیری یک درخت باینری است، ارتفاع آن برابر است با:

$$\geq \lg\left((k!)^{\lfloor \frac{n}{k} \rfloor}\right) \geq \left\lfloor \frac{n}{k} \right\rfloor \lg(k!) \geq \left\lfloor \frac{n}{k} \right\rfloor (c_۱ k \lg k)$$

$$\geq c_۱ (n-k) \lg k \geq \frac{c_۱ n \lg k}{۲} \geq \Omega(n \lg k)$$

ج) یک نمونه الگوریتم مطلوب با استفاده از هرم کمینه به صورت زیر است:

۱. یک هرم کمینه با اندازه $k+1$ از $k+1$ عنصر اول آرایه می‌سازیم. این کار در زمان $O(k)$ انجام می‌گیرد.

۲. در هر مرحله عنصر کمینه را از هرم میگریم و در آرایه نتیجه قرار می‌دهیم و عنصر بعدی در آرایه اولیه را وارد هرم کمینه می‌کنیم. این کار در هر مرحله در $O(k)$ اجرا می‌شود.

پیچیدگی زمانی این الگوریتم $O(k+(n-k)\lg k)$ است. درستی آن نیز واضح است. از آنجایی که هر عنصر حداکثر k خانه با جای درستش فاصله دارد پس در هر مرحله، عنصر کمینه هرم، عنصر کمینه کل آرایه باقی‌مانده نیز هست.

سؤال ۵. می‌خواهیم یک آرایه پویا [۲] با استفاده از آرایه‌های عادی پیاده‌سازی کنیم.

برای این کار ابتدا یک آرایه با اندازه ۱ در نظر می‌گیریم که چیزی در آن نیست. بعد برای هر عملیات درج اگر آرایه فعلی جا داشت عنصر جدید را به انتهای آن اضافه می‌کنیم و اگر نداشت یک آرایه جدید با اندازه ۲ برابر آرایه قبلی می‌سازیم. تمام عناصر آرایه قبلی را به این آرایه منتقل می‌کنیم و عنصر جدید را به انتهای آن اضافه می‌کنیم. هزینه انتقال n عضو از آرایه‌ای به آرایه دیگر $O(n)$ است.

الف) فرض کنید برای عملیات حذف، عنصر آخر را از آرایه فعلی حذف می‌کنیم و بعد اگر تعداد عناصر باقیمانده کمتر مساوی نصف اندازه آرایه فعلی بود، یک آرایه با اندازه نصف آرایه فعلی می‌سازیم و همه عناصر باقیمانده را به آن منتقل می‌کنیم. نشان دهید هزینه سرشکن اعمال $O(1)$ نخواهد بود.

ب) حالا فرض کنید به جای اینکه وقتی تعداد عناصر باقیمانده به نصف اندازه آرایه رسید آرایه را نصف کنیم، وقتی این تعداد به $\frac{1}{4}$ رسید این کار را انجام دهیم. نشان دهید در این صورت هزینه سرشکن برای هر دنباله از اعمال درج و حذف $O(1)$ خواهد بود.

پاسخ:

الف) حالتی را فرض کنید که آرایه پر باشد. حال به صورت پی‌درپی یک عنصر را درج و حذف کنید. هزینه سرشکن این اعمال $O(n)$ خواهد بود.

ب) در واقع ما باید ثابت کنیم تعداد عملیات لازم برای دو برابر کردن و نصف کردن آرایه، از مرتبه عملیات درج و حذف است. فرض کنید سایز آرایه n و تعداد عناصر $\frac{1}{4}$ شده باشد. در این صورت سایز آرایه به $\frac{n}{2}$ کاهش میابد. پس تا اینجا حداقل $\frac{n}{4}$ عنصر حذف شده است و هزینه نصف کردن آرایه هم همین قدر است. برای این که فرآیند دو برابر شدن انجام بگیرد باید $\frac{n}{4}$ عنصر درج شود و هزینه دو برابر کردن آرایه هم $\frac{n}{2}$ است. برای این که مجددا فرآیند نصف شدن انجام بگیرد، باید حداقل $\frac{n}{8}$ عنصر حذف شده است و هزینه نصف کردن آرایه هم همین قدر است. این شرایط برای هر دنباله‌ای از عملیات درج و حذف برقرار است. پس حکم ثابت می‌شود.

[۲] آرایه‌ای با قابلیت تغییر اندازه

سؤال ۶. فرض کنید k لیست مرتب داریم که هر کدام دارای n عضو هستند. هدف این است که این لیست‌ها را به یک لیست مرتب ادغام کنیم.

الف) اگر ابتدا لیست اول و دوم را ادغام کنیم و سپس نتیجه حاصل را لیست سوم ادغام کنیم و الی آخر، آنگاه پیچیدگی زمان اجرای این الگوریتم برحسب k و n را بدست بیاورید.

ب) با استفاده از روش تقسیم و غلبه یک الگوریتم بهتر ارائه دهید و رابطه بازگشتی آن را نوشته و حل کنید.

پاسخ:

الف) می‌دانیم ادغام دو لیست مرتب m و nعضوی با روشی مشابه روش مورداستفاده در مرتب‌سازی ادغامی، از $\Theta\left(m+n\right)$ است. پس می‌نویسیم:

$$n+n=2n$$

$$2n+n=3n$$

$$3n+n=4n$$

$$\vdots$$

$$\left(k-1\right)n+n=kn$$

با توجه به اینکه این مراحل یکی پس از دیگری اجرا می‌شوند، درنهایت این فرآیند ادغام از مجموع Θهای تمام مراحل است یعنی $\Theta\left(\left(\frac{k(k+1)}{2}-1\right)n\right)$ یا درنهایت همان $\Theta\left(k^2 n\right)$.

ب) با استفاده از روش تقسیم و غلبه، مجموعه kتایی موردنظر را در $\Theta\left(kn\right)$ به دو مجموعه $\frac{k}{2}$تایی تقسیم می‌کنیم و ابتدا این دو مجموعه را مرتب کرده، سپس دو آرایه $\frac{k}{2}$تایی را در $\Theta\left(kn\right)$ با هم ادغام می‌کنیم. رابطه بازگشتی به‌صورت زیر است:

$$T\left(k\right)=2T\left(\frac{k}{2}\right)+2kn$$

با استفاده از حالت دوم $Master\ Theorem$ به دست می‌آوریم که این الگوریتم از $\Theta\left(nk\log k\right)$ است.

سؤال ۷. الف) داده‌ساختار پشته را به گونه‌ای پیاده‌سازی کنید که هر کدام از اعمال PUSH، POP، GETMIN و GETMAX را در هر لحظه در $O(1)$ انجام دهد. تابع GETMIN عضو کمینه پشته و تابع GETMAX عضو بیشینه پشته را برمی‌گرداند.

ب) داده ساختار صف را طوری پیاده‌سازی کنید که هر یک از اعمال ENQUEUE، DEQUEUE، GETMIN و GET-MAX را به طور سرشکن در $O(1)$ انجام دهد. تابع GETMIN عضو کمینه صف و تابع GETMAX عضو بیشینه صف را برمی‌گرداند.

پاسخ:

الف) دو متغیر min و max را تعریف می‌کنیم که به‌ترتیب مینیمم و ماکسیمم اعضای پشته را در خود نگه می‌دارند. چالش زمانی است که این مقدار از پشته حذف می‌شود. برای رفع این چالش توابع PUSH و POP را دستکاری می‌کنیم.

عملیات PUSH مقدار x در پشته را به‌صورت زیر پیاده‌سازی می‌کنیم. دقت کنید هزینه تمام عملیات‌های زیر ثابت و درنتیجه از $\Theta\left(1\right)$ است.

۱. اگر پشته خالی باشد، x را PUSH می‌کنیم و مقادیر min و max را برابر x قرار می‌دهیم.

۲. اگر x از min کوچک‌تر یا از max بزرگ‌تر نباشد آن را مانند پشته معمولی PUSH می‌کنیم.

۳. اگر x از min کوچک‌تر باشد، به‌جای x، عدد $2x - min$ را در پشته PUSH می‌کنیم و مقدار min را برابر x قرار می‌دهیم.

۴. اگر x از max بزرگ‌تر باشد، به‌جای x، عدد $2x - max$ را در پشته PUSH می‌کنیم و مقدار max را برابر x قرار می‌دهیم.

دقت کنید با این روش POP ما درحقیقت چالش حذف min و max را حل کردیم. در زمان‌هایی که درنتیجهٔ PUSH مقدار متغیره min یا max تغییر پیدا می‌کند، عددی که درنهایت PUSH می‌شود از min کوچک‌تر یا از max بزرگ‌تر است. این در زمان POP این مقدار، یک نشانه است تا مقدار min یا max را به‌روزرسانی کنیم. حال دقت کنید که از آن‌جایی که مقدار x را به‌عنوان متغیر min داریم و مقدار $2x - min$ را نیز در پشته داریم، می‌توانیم به‌سادگی مقدار قبلی متغیر min را محاسبه و جایگزین کنیم، مشابه این شرایط برای max نیز برقرار است.

فرض کنید آخرین مقدار واردشده در پشته برابر x است. با توجه به توصیحات داده‌شده، عملیات POP را به‌صورت زیر پیاده‌سازی می‌کنیم. دقت کنید هزینه تمام عملیات‌های زیر ثابت و درنتیجه از $\Theta\left(1\right)$ است.

۱. اگر x از min کوچک‌تر یا از max بزرگ‌تر نباشد، مقدار این دو متغیر را تغییر نمی‌دهیم و x را برمی‌گردانیم.

۲. اگر x از min کوچک‌تر باشد، مقدار min را برمی‌گردانیم و min را برابر $2min - x$ قرار می‌دهیم.

۳. اگر x از max بزرگ‌تر باشد، مقدار max را برمی‌گردانیم و max را برابر $2max - x$ قرار می‌دهیم.

ب) از بخش الف استفاده می‌کنیم.

ابتدا با کمک دو پشته یک صف پیاده‌سازی می‌کنیم. برای ENQUEUE عضو موردنظر را در پشته اول PUSH می‌کنیم. برای DEQUEUE ابتدا پشته دوم را نگاه می‌کنیم. اگر پشته دوم خالی بود تمام اعضای پشته اول را POP و در پشته دوم PUSH می‌کنیم. در هر صورت آخرین عضو پشته دوم را POP می‌کنیم و آن را به‌عنوان حاصل DEQUEUE خروجی می‌دهیم. دقت کنید هرکدام از اعضای این صف تنها یک‌بار از پشته اول POP می‌شوند و در پشته دوم PUSH می‌شوند درنتیجه هزینه سرشکن عملیات DEQUEUE نیز مانند ENQUEUE از $\Theta\left(1\right)$ است.

حال تمام اعضای صف ما در یکی از دو پشته وجود دارند. برای به‌دست‌آوردن ماکسیمم و مینیمم آن‌ها کافی‌ست ماکسیمم و مینیمم هر پشته را در $\Theta\left(1\right)$ به دست بیاوریم و آن‌ها را با هم مقایسه کنیم. بدیهی‌ست که این دو عملیات نیز از $\Theta\left(1\right)$ می‌باشند.

سؤال ۸. یک پشته را به‌روش بازگشتی و بدون استفاده از داده‌ساختار دیگری مرتب کنید.

پاسخ:

ابتدا به روش بازگشتی پشته را خالی می‌کنیم و سپس در هر مرحله، کوچک‌ترین درایه را به پشته اضافه می‌کنیم. در این الگوریتم، امکان مقایسهٔ دوبه‌دوی تمام اعضای پشته وجود دارد، پس از $\Theta\left(n^{2}\right)$ زمان و $\Theta\left(n\right)$ حافظه است.

```
def sort(stack):
   if stack:
     item = stack.pop()
     sort(stack)
     insert(stack, item)


def insert(stack, item):
   if stack:
     top = stack[-1]
     if item < top:
       stack.pop()
       insert(stack, item)
       stack.append(top)
     else:
       stack.append(item)
   else:
     stack.append(item)
```

سؤال ۹.   یک آرایه با n عضو داریم که اعضای آن بیشتر از k با هم اختلاف ندارند. الگوریتمی از $\Theta\left(k+n\right)$ برای مرتب‌سازی این آرایه ارائه دهید.

پاسخ:

ابتدا ماکسیمم و مینیمم اعضای آرایه و سپس اختلاف این دو عدد را به دست می‌آوریم، سپس آرایه‌ای با تعداد اعضای این اختلاف می‌سازیم و مقدار اولیه اعضای آن را برابر صفر قرار می‌دهیم. به این شکل برای هر مقدار ممکن برای اعضای آرایه مانند a یک عضو در خانه $a-min$م آرایه دوم داریم. حال تمام اعضای آرایه اول را بررسی می‌کنیم و به‌ازای هر مقدار a، به مقدار عضو $a-min$م آرایه دوم یکی اضافه می‌کنیم. به این شکل تعداد دفعات تکرار تمام مقادیر آرایه اول را به دست می‌آوریم.

حال با توجه به آرایه دوم، آرایه مرتب‌شده را می‌نویسیم. به این صورت که به‌صورت صعودی تمام درایه‌های این آرایه را بررسی می‌کنیم و به‌ازای درایه iم که مقدار آن n است، به آرایه مرتب‌شده، n عضو با مقدار $i+min$ اضافه می‌کنیم. با توجه به اینکه این حلقه به‌صورت صعودی است، آرایه خروجی ما نیز صعودی می‌باشد.

سؤال ۱۰.   یک لیست پیوندی داریم. برای مرتب‌سازی این لیست از مرتب‌سازی سریع استفاده می‌کنید یا از مرتب‌سازی ادغامی؟ دلیل انتخاب خود را توضیح دهید.

پاسخ:

مرتب‌سازی ادغامی درمقایسه‌با مرتب‌سازی سریع برای لیست‌های پیوندی انتخاب بهتری است.

لیست‌های پیوندی قابلیت $random\ access$ را ندارند، یعنی اینکه برای به‌دست‌آوردن عضو iم آن‌ها باید از اولین عضو شروع کرد و $i-1$ بار عضو بعدی آن‌ها را بررسی کرد. در مرتب‌سازی سریع ما همواره نیاز داریم که مقدار عضو $pivot$ را با مقادیر دیگر اعضای آرایه مانند عضو iم مقایسه کنیم و این مقایسه‌ها در لیست‌های پیوندی هزینه زمانی زیادی را در پی دارند. از طرف دیگر

در مرتب‌سازی ادغامی ما تنها هنگامی نیاز به $random\ access$ داریم که می‌خواهیم آرایه موردنظر را به آرایه‌های کوچک‌تر تقسیم کنیم و این به این معنی است که تعداد $random\ access$ها تنها از $\Theta\left(\log n\right)$ است.

موفق باشید