به نام خدا

ساختمان داده‌ها و الگوریتم‌ها (۴۰۲۵۴)

دانشگاه صنعتی شریف — تمرین اول

مدرس: دکتر مهدی صفرنژاد — انتشار: ۱۷ اسفند ۱۴۰۰

مقدمه و تحلیل الگوریتم‌ها

سؤالات را با دقت بخوانید و روی همه آن‌ها وقت بگذارید. تمرین‌های تئوری تحویل گرفته نمی‌شوند اما از آن‌ها سؤالات کوییز مشخص می‌شود. بنابراین روی سؤالات به خوبی فکر کنید و در کلاس‌های حل تمرین مربوطه شرکت کنید.

سؤال ۱. درستی یا نادرستی هر عبارت را با دلیل بیان کنید(همه توابع مثبت هستند).

همواره داریم:

آ.

$$f(n) + o(f(n)) = \theta(f(n))$$

ب.

$$f(n) = \theta(f(n/2))$$

پ.

$$f(n) = O(g(n)) \Rightarrow 2^{f(n)} = O(2^{g(n)})$$

ت.

$$\theta(f(n) + g(n)) = \max(f(n), g(n))$$

ث.

$$\nexists f(n), g(n); f(n) + g(n) \neq O(f(n)), f(n) + g(n) \neq O(g(n))$$

پاسخ:

۱. اگر فرض کنیم $g(n) = o(f(n)$ طبق تعریف داریم:

$$\forall c \exists n0; n > n0 \Rightarrow g(n) < cf(n) \Rightarrow \exists n0; n > n0 \Rightarrow g(n) < f(n) \Rightarrow g(n) + f(n) \leq 2f(n)$$

$$\Rightarrow f(n) + g(n) = O(f(n))$$

از طرف دیگر $g(n) > 0 \Rightarrow g(n) + f(n) \geq f(n) \Rightarrow g(n) + f(n) = \Omega(f(n))$

$$\Rightarrow f(n) + g(n) = O(f(n), f(n) + g(n) = \Omega(f(n)) \Rightarrow f(n) + g(n) = \theta(f(n))$$

۲. گزاره نادرست است، زیرا فرض کنید $f(n) = 2^n$ باشد آن گاه $f(\frac{n}{2}) = \sqrt{2}^n$ خواهد بود که بیان گر این خواهد بود که $f(n) \neq \theta(f(\frac{n}{2}))$

۳.گزاره نادرست است، زیرا فرض کنید $f(n) = n, g(n) = n/2 \Rightarrow f(n) = O(g(n))$ ولی $2^n \neq O(\sqrt{2}^n)$

۴.داریم:

$$g(n) > 0, f(n) > 0 \Rightarrow f(n) + g(n) \geq max\{f(n), g(n)\} \Rightarrow max\{f(n), g(n)\} = O(f(n) + g(n))$$

$$g(n) + f(n) \leq 2max\{f(n), g(n)\} \Rightarrow max\{f(n), g(n)\} = \Omega(f(n) + g(n))$$

$$\Rightarrow max\{f(n), g(n)\} = \theta(f(n) + g(n))$$

۵.گزاره نادرست است برای مثال فرض کنید: $f(n) = \left\{ \begin{array}{ll} 1, & n = 2k+1 \\ n, & n = 2k \end{array} \right\}$ و $g(n) = \left\{ \begin{array}{ll} n, & n = 2k+1 \\ 1, & n = 2k \end{array} \right\}$
باشد و $h(n) = f(n) + g(n) = n + 1$ واضح است که $h(n) \neq O(f(n)) \wedge h(n) \neq O(g(n))$ در واقع به ازای هر n۰ و c داریم: $\exists n > n0; h(n) > cf(n), h(n) > cg(n)$

سؤال ۲. روابط بازگشتی زیر را از روش دلخواه حل کنید و جواب را بر حسب θ به دست آورید

آ.

$$T(n) = T(\frac{2n}{3}) + (\log n)^2$$

ب.

$$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + n$$

پ.

$$T(n) = 2T(\sqrt{n}) + \log n$$

ت.

$$T(n) = 9T(\frac{n}{27}) + \sqrt[3]{n}$$

ث.

$$T(n) = \frac{2}{n}(T(o) + T(1) + ... + T(n-1)) + c \quad , T(0) = 0$$

ج.

$$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n$$

چ.

$$T(n) = T(n-a) + T(a) + cn \quad c > 0, a \geq 1$$

پاسخ:

۱.طبق قضیه اصلی داریم:
$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = 1$ و $f(n) = (\log n)^2$ پس طبق حالت دوم قضیه اصلی
$T(n) = O((\log n)^{2+1}) = O((\log n)^3)$

۲.چون $\frac{n}{5}+\frac{7n}{10}<n$ حدس میزنیم T(n)=O(n) یعنی k وجود دارد به طوری که T(n) همواره کوچکتر از kn باشد. اکنون با استقرا آن را ثابت میکنیم. برای پایه استقرا به ازای مقادیر کوچک n مانند $n\leq 10$ واضح است و تنها کافی است k را به قدر کافی بزرگ انتخاب کنیم .

برای گام استقرا نیز فرض میکنیم که به ازای تمام مقادیر کوچکتر از n حکم برقرار است و سپس اثبات می‌کنیم که برای n نیز برقرار می‌باشد.

$$T(n)\leq T(\tfrac{n}{5})+T(\tfrac{7n}{10})+n$$
$$\leq k(\tfrac{n}{5})+k(\tfrac{7n}{10})+n$$
$$=k(\tfrac{9n}{10})+n$$
$$=n(\tfrac{9k}{10}+1)$$
$$\leq nk$$
$$\Rightarrow k\geq 10$$

پس اثبات شد به ازای $k\geq 10$ حکم برقرار است.

۳.از تغییر متغیر $m=\log n$ استفاده می‌کنیم. $T(2^m)=2T(2^{\frac{m}{2}})+m$ سپس قرار می‌دهیم: $S(m)=T(2^m)$ پس: $S(m)=2S(\frac{m}{2})+m$ طبق قضیه اصلی $S(m)=O(m\log m)$ و با جایگذاری m داریم: $T(n)=\log n\log(\log n)$

۴.طبق قضیه اصلی داریم:

$n^{\log_b a}=n^{\log_{27}9}=n^{\frac{2}{3}}$ و $f(n)=n^{\frac{1}{3}}$ پس طبق حالت اول قضیه اصلی

$$f(n)=O(n^{\log_{27}9-\epsilon})\Rightarrow T(n)=O(n^{\frac{2}{3}})$$

۵.با توجه به رابطه ی داده شده داریم: $nT(n)=2(T(0)+T(1)+...+T(n-2)+T(n-1))+cn$

$$(n-1)(T(n-1))=2(T(0)+T(1)+...+T(n-2)+c(n-1)$$

با تفاضل این دو رابطه از هم نتیجه می‌شود:

$$n(T(n))-(n-1)T(n-1)=2T(n-1)+c$$
$$nT(n)=(n+1)T(n+1)+c$$
$$\frac{T(n)}{n+1}=\frac{T(n-1)}{n}+\frac{c}{n(n+1)}$$
....
$$\frac{T(1)}{2}=\frac{T(0)}{1}+\frac{c}{1\times 2}$$
$$\frac{T(n)}{n+1}=\frac{c}{1\times 2}+\frac{c}{2\times 3}+...+\frac{c}{n(n+1)}=c\frac{n}{n+1}$$
$$T(n)=cn=O(n)$$

۶.اگر درخت مربوط به T(n) را رسم کنیم، بلندترین شاخه دارای ارتفاع $\log_{\frac{3}{2}}n$ و کوتاه ترین شاخه دارای ارتفاع $\log_3 n$ است. در نتیجه:

$$T(n)=O(n\log_{\frac{3}{2}}n)=O(n\log n)$$
$$T(n)=\Omega(n\log_3 n)=\Omega(n\log n)$$
$$\Rightarrow T(n)=\theta(n\log n)$$

۷.برا این سوال پس داریم که : T(n) = T(n-a) + T(a) + cn حالا ورودی را n - a فرض می‌کنیم و دوباره می‌نویسیم T(n-a) = T((n-a)-a) + T(a) + c(n-a) و این بار هم a را ورودی فرض می‌کنیم و دوباره می‌نویسیم T(a) = T(a-a) + T(a) + ca

حالا با ترکیب این دوتا داریم :

T(n) = (T((n-a)-a) + T(a) + c(n-a))+ (T(a-a) + T(a) + ca) + cn

= T(n-۲a) + T(a) + c(n-a) + T(۰) + T(a) + ca + cn

= T(n-۲a) + ۲T(a) + T(۰) + c((n-a) + a + n)

حال با استقرا به دست می اید که تابع از $O(n^2)$ است.

سؤال ۳. زمان اجرای الگوریتم های زیر را در بدترین حالت بصورت θ به دست آورید.

آ.

```
j=0
i=0
while i<n :
    i+=j
    j+=1
```

ب.

```
i=3
if n==2 :
    return True
if n==1 :
    return False
if n%2==0:
    return False
while i ∧ 2 <= n:
    if n%i == 0:
        return False
    else:
        i+=2
return True
```

پ.

```
i=n
while i >= 1 :
    j=i
```

```
    while j <= n:
        j = 2 * j
    i=i//2
```

ت.
```
c=0
for i in range(1,n+1):
    for j in range(1,n):
        c += 1
```

ث.
```
i=1
while i <= n :
    j=1
    while j ∧ 2 <= i:
        j += 1
    i += 1
```

پاسخ:

۱. پاسخ: $\theta(\sqrt{n})$

i در هر مرحله جمع اعداد ۱ تا j است. اگر j تا k مرحله پیش رود (برنامه k مرحله اجرا شود)

$1 + 2 + ... + k = \frac{k(k+1}{2}$

و حلقه زمانی پایان می‌یابد که: $\frac{k(k+1)}{2} > n$

$\Rightarrow k = \theta(\sqrt{n})$

۲. پاسخ: $\theta(\sqrt{n})$

داخل حلقه while به اندازه $\sqrt{n}$ بار اجرا می‌شود.

۳. پاسخ: $\theta((\log n)^2)$

در ابتدا که i=n داخل while دوم 0 بار اجرا می‌شود. دفعه بعدی که $i = \frac{n}{2}$ داخل while دوم یک بار اجرا میشود و به همین ترتیب دفعه آخر که i=۱ داخل while دوم $\log n$ بار اجرا می‌شود. پس در کل داخل while دوم به اندازه $0 + 1 + 2 + ... + \log n = \frac{\log n(\log n+1)}{2}$ بار اجرا شده پس از $\theta((\log n)^2)$ است.

۴. پاسخ: $\theta(n^2)$

j ، i کاملا مستقل از هم هر کدام n مقدار میگیرند پس داخل for دوم $n(n-1)$ بار اجرا می‌شود.

۵. پاسخ: $\theta(n^{\frac{3}{2}})$

داخل while دوم در مرحله اول که i=۱ ۱ بار اجرا می‌شود در مرحله دوم به اندازه $\lfloor\sqrt{2}\rfloor$ بار اجرا می‌شود. تا در مرحله n ام به اندازه $\lfloor\sqrt{n}\rfloor$ بار اجرا می‌شود. پس در کل به اندازه

$\lfloor\sqrt{1}\rfloor + \lfloor\sqrt{2}\rfloor + ... + \lfloor\sqrt{n}\rfloor$ بار اجرا می‌شود.
$\sum_{i=1}^{n} i < \int_0^n \sqrt{x} = \frac{2}{3}n^{\frac{3}{2}}$ و $\sum_{i=1}^{n} i > \int_0^{n-1} \sqrt{x} = \frac{2}{3}(n-1)^{\frac{3}{2}}$ پس مرتبه اجرای الگوریتم از $\theta(n^{\frac{3}{2}})$ است.

سؤال ۴. می‌خواهیم در رشته ای از حروف کوچک انگلیسی، بلندترین زیررشته‌ای که در آن هیچ دو حرفی تکرار نشده است را پیدا کنیم. شبه کدی بنویسید که این کار را در $O(n^2)$ انجام دهد. سپس شبه کدی بنویسید که این کار را در $O(n)$ انجام دهد.

پاسخ:

فرض کنید که رشته به صورت آرایه ای از اعداد ۰ تا ۲۵ به شما داده شده است کد اول در $O(n^2)$ انجام می شود و کد دوم در O(n) انجام می‌پذیرد.

```
#first code
def longestUniqueSubsttr(str):
    n = len(str)
    res = 0
    for i in range(n):
        visited = [0] * 26
        for j in range(i, n):
            if visited[str[j]] :
                break
            else:
                res = max(res, j - i + 1)
                visited[str[j]] = True
        visited[str[i]] = False
    return res
    #O(n^2)
```

```
#second code
def longestUniqueSubsttr(str):
    result=0
    i=0
    j=0
    last_idx=[-1]*26
    while i<len(str):
       if j<=last_idx[str[i]]+1:
           j=last_idx[str[j]]+1
       if result<=i-j+1:
           result=i-j+1
       last_idx[str[i]] = i
       i+=1
    return result
```

```
#O(n)
```

سؤال ۵. آیا می‌توان گفت که ثابت c وجود دارد به صورتی که $\lfloor \log n \rfloor ! = O(n^c)$

اکنون برای $\lfloor \log\log n \rfloor !$ چطور؟

پاسخ: برای حل این مسئله فرض می‌کنیم که

$$\exists n0, k; k > 0, n0 \in n$$

و بعد از آن داریم که

$$n > n0 \Rightarrow \lfloor \log n \rfloor ! \leq kn^c$$

حالا برای حل این سوال فرض می‌کنیم که $t \in n, n = 2^t$ پس با جایگذاری برای ما بدست می‌آید که

$$\lfloor \log 2^t \rfloor ! \leq k(2^t)^c \Rightarrow t! \leq k(2^c)^t$$

حالا اینجا، یک تابع فاکتوریلی داریم و یک تابع نمایی که از آنجایی که رشد تابع فاکتوریلی همواره بیشتر از تابع نمایی‌ست، و ما هر چقدر هم اعداد بزرگ به تابع نمایی دهیم، هیچ ثابت ایc در این حالت برای ما وجود ندارد

در قسمت دوم سوال، باز هم فرض می‌کنیم که

$$\exists n0, k; k > 0, n0 \in n$$

سپس گوییم:

$$n > n0 \Rightarrow \lfloor \log\log n \rfloor ! \leq kn^c$$

برای حل این سوال فرض می‌کنیم که

$$t \in n, n = 2^{2^t}$$

که این فرض فقط برای راحتی کار است پس با جایگذاری بدست می‌آید که

$$t! \leq k(2^{2^t})^c$$

پس چیزی که مشخص است این است که در صورتی که k =۱ و c = ۱ در نظر بگیریم، عبارت سمت راست خیلی بیشتر از عبارت سمت چپ است و در نتیجه با استفاده از جمع دنباله‌های هندسی، بدست می‌آید در نهایت که قسمت دوم سوال درست است.

سؤال ۶. آرایه A شامل n عدد داده شده است، در این آرایه، همه عناصر به غیر از ۵ عنصر در جای درست خود در حالت مرتب شده هستند. در واقع اگر A را مرتب کنیم، تنها در ۵ خانه با A قبل از مرتب شدن، تفاوت دارد و سایر عناصر در جای قبلی خود قرار می‌گیرند. الگوریتمی ارائه دهید که این آرایه را مرتب می‌کند. پیچیدگی زمانی الگوریتم خود را تا حد ممکن بهینه کنید پس از هر بار بررسی!

پاسخ: برای حل این سوال از insertion sort استفاده می‌کنیم با این تفائت که این عمل را ۵ بار تکرار می‌کنیم. ارایه ذکر شده در صورت سوال یک آرایه است که حداکثر دارای ۵ insertion است زیرا تنها در ۵ خانه با آرایه مرتب شده تفاوت دارد. حال همانطور

که می‌دانیم در مرتب سازی درجی در درج در هر مرحله یک عنصر را در نظر می‌گیریم و invertion های اول از آن را درست می کنیم و آن را در موقعیت درست قرار می دهیم. بنابر این در این سوال که میدانیم حداکثر ۵(n-۱) نابجایی وجود دارد کافی است این عمل را ۵ بار تکرار کنیم. سپس نابجایی هایی که یک طرف آن ها این ۵ عنصر قرار دارند را تحلیل زمانی از O(n) است زیرا همان طور که می دانیم تحلیل زمانی مرتب سازی درجی از تعداد+O(n نابجایی) است. هر عنصر حداکثر می تواند با n-۱ عنصر دیگر نابجایی ایجاد کند. پس تعداد نابجایی ها حداکثر ۵(n-۱) می‌باشد. از طرفی حد پایین این الگوریتم O(n) است زیرا برای پیدا کردن نابجایی ها باید یک دور کامل عناصر را پیمایش کنیم.

سؤال ۷. آرایه n عنصری A را در نظر بگیرید که تمام عناصر آن اعدادی طبیعی و متمایز و به ترتیب صعودی هستند. الگوریتمی بهتر از جستجوی خطی ارائه دهید که تشخیص دهد آیا اندیسی مانند i وجود دارد که $A[i] = i$ باشد یا خیر؟

پاسخ: بهترین الگوریتم موجود برای این سوال، باینری سرچ هستش که حتما سر کلاس با اُردر و نحوه اجرای آن آشنا شده‌اید. در این سوال به این شکل پیش می‌رویم که عنصر وسط آرایه را چک می‌کنیم که آیا مقدار آن برابر است با i/۲ یا خیر. اگر برابر بود، عنصر وسط نصفه دوم را چک می‌کنیم و اگر برابر نبود، عنصر وسط نصفه اول و به همین شکل جواب را بدست می‌آوریم.

سؤال ۸. داده ساختاری طراحی کنید که بتواند اعمال Push و Pop و FindMin (یافتن و برگرداندن کوچکترین عنصر) را در O(۱) انجام دهد. سپس با فرض اینکه می‌دانیم نمی‌توان یک آرایه را در حالت کلی در بهتر از $O(n \log n)$ مرتب کرد، ثابت کنید اگر این داده ساختار بخواهد عمل DeleteMin را هم پشتیبانی کند، نمی‌تواند آن را هم در مرتبه O(۱) انجام دهد.

پاسخ:

دو پشته در نظر بگیرید. در پشته اول به صورت معمولی از push و pop استفاده می‌کنیم. در پشته دوم، عمل push را تنها در صورتی انجام می‌دهیم که عنصر در حال درج، از عنصر بالای پشته کوچک‌تر باشد. همچنین هنگام صدا زده شدن دستور pop اگر عنصر بالای پشته اول، برابر با عنصر بالای پشته دوم (که همواره مینیمم فعلی اعداد موجود می‌باشد.) بود، در پشته دوم هم pop انجام می‌دهیم و در غیر این صورت فقط عدد بالای پشته اول pop می‌شود. در رابطه با حذف عنصر در مرتبه زمانی ثابت، به این موضوع توجه کنید که در صورت امکان این امر، می‌توان با درج همه عناصر یک آرایه در این داده‌ساختار و اجرای DeleteMin تا زمانی که داده‌ساختار خالی شود، آرایه را در زمان خطی مرتب کرد که این کار غیرممکن می‌باشد.

سؤال ۹. داده ساختاری طراحی کنید که اعمال زیر را به صورت بهینه انجام دهد:

آ. یک عدد را به انتهای لیست اضافه کند.

ب. یک عدد را از انتهای لیست کم کند.

پ. k عنصر انتهایی لیست را قرینه کند که k برای داده ساختار همیشه یک عدد ثابت است (در عمل نتیجه مثل این است که k عنصر را به ترتیب بخواند و روی هم بریزد، سپس آن‌ها را وارونه کند و سپس به لیست برگرداند.)

ت. عناصر را به ترتیبی که در لیست قرار دارند چاپ کند.

پاسخ:

۱. یک لیست پیوندی دوطرفه (به صورتی که هر node در لیست، قبلی و بعدی خود را نگه می‌دارد) و یک پشته در نظر می‌گیریم. برای درج یک عدد به لینکدلیست، عدد را به آخرین node اضافه می‌کنیم و اولین node را حذف کرده و مقدارش را در پشته push می‌کنیم.

۲. برای حذف عدد، node نهایی در لیست را حذف می‌کنیم، سپس از پشته pop می‌کنیم و عدد خروجی را در ابتدای لینکدلیست قرار می‌دهیم.

۳. برای وارونه کردن جایگاه k عنصر نهایی لیست، کافی است تا مقادیر next و previous مربوط به هر node را با هم‌دیگر جابه‌جا کنیم تا لیست پیوندی‌مان وارونه شود.

۴. همچنین برای چاپ عناصر به ترتیب، می‌توان با تعریف یک پشته کمکی، هر عنصر از پشته اول را ابتدا pop کرده و سپس در پشته کمکی push کنیم تا پشته‌ای با ترتیب عکس داشته باشیم؛ سپس همین روند را دوباره تکرار کرده و اعداد را به پشته اول برگردانیم، با این تفاوت که این بار هر عددی که از پشته کمکی pop می‌شود، قبل از برگشتن به پشته اول، چاپ نیز خواهد شد. سپس با شروع از ابتدای لیست‌پیوندی و پیمایش آن می‌توان عناصر را به ترتیبی که در لیست هستند در خروجی نوشت.

سؤال ۱۰. رشته ای به طول n از پرانتز باز و بسته داده شده است و می‌خواهیم بدانیم آیا رشته پرانتزها معتبر است یا نه؟

مثال۱:

ورودی:((())())

خروجی:بله

مثال۲:

ورودی:(((())())

خروجی:خیر

حالا الگوریتمی ارائه دهید که در بدترین حالت، بیشتر از ۵۰n عملیات انجام ندهد و خروجی را اعلام کند.

منظور از هر عملیات نیز، هر یک از عملیات‌های جمع و تفریق و ضرب و تقسیم و یا تخصیص است.

درستی الگوریتم خود را اثبات کنید.

پاسخ: رشته را با یک صف مدل‌سازی می‌کنیم با دو شرط که

۱. در زمان خالی کردن صف، عضوی برای برداشتن وجود داشته باشد.

۲. زمانی که به انتهای پرانتزها رسیدیم، عضوی در داخل صف موجود نباشد.

```
def isValid(p):
  for i in range(n):
      if p[i]=='(':
          size+=1
        else:
```

```
          if size==o:
             return False
          size-=1
  if size ==0:
       return True
   return False
```

موفق باشید