به نام خدا

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

آزمون میان‌ترم
انتشار: ۱۱ آذر ۱۴۰۰

آزمون میان‌ترم

زمان این امتحان ۱۲۰ دقیقه است. امتحان از ۱۰۰ نمره است. توضیحات ابتدای سؤالات را به دقت بخوانید و پاسخ بخش‌های مختلف سؤال را بنویسید. ۵ دقیقه زمان آپلود در نظر گرفته‌شده‌است.

امیدواریم تا این‌جای کلاس با تفکر الگوریتمی آشنا شده باشید و با همین تفکر به سؤالات امتحان به خوبی پاسخ دهید.

سؤال ۱. [۲۰ نمره] به سوال‌های زیر پاسخ دهید. (ارتفاع درخت را برابر تعداد یال‌های طولانی‌ترین مسیر از ریشه‌ی درخت در نظر بگیرید. مثلا درختی با یک راس، ارتفاع برابر صفر دارد.)

آ. حداقل و حداکثر تعداد برگ‌های یک درخت دودویی کامل را به ارتفاع h را به دست آورید.

ب. حداقل و حداکثر تعداد رئوس یک درخت AVL به ارتفاع h را به دست آورید.

پ. درخت دودویی کاملی رسم کنید که نمایش پیشوندی آن به صورت $ABCDEFGHIJ$ باشد.

ت. برای درخت رسم‌شده در قسمت قبل، نمایش میان‌ترتیب را به دست آورید.

پاسخ:

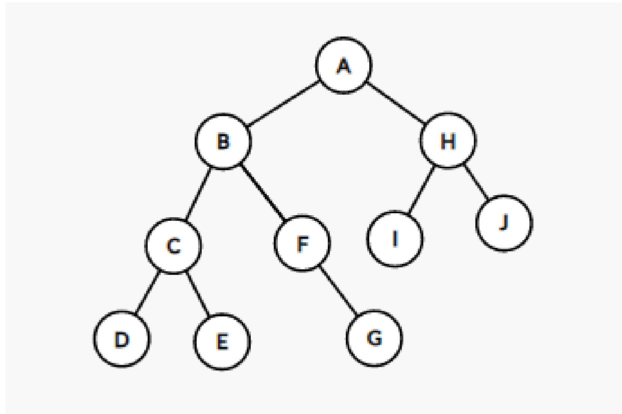
آ. مطابق تعریف درخت دودویی کامل، درختی‌ست که تمام طبقه‌های آن به جز احتمالا طبقه‌ی آخر پر شده‌اند. در نتیجه می‌توانیم بگوییم تمام رئوس تا طبقه‌ی $h-1$ حتما در درخت وجود دارند. تا اینجا درخت 2^{h-1} برگ دارد.

برای این که ارتفاع درخت برابر h بشود، لازم است که حداقل یک برگ در این طبقه قرار دهیم. دقت کنید که اضافه کردن این برگ، تعداد برگ‌ها را تغییر نمی‌دهد. (راس پدر از برگ بودن در آمده) پس حداقل تعداد برگ‌ها 2^{h-1} است. از طرفی می‌توانیم تمام برگ‌های طبقه‌ی h را اضافه کنیم که به 2^h برگ می‌رسیم.

ب. یک درخت AVL به ارتفاع h را می‌توان به ریشه و دو زیردرخت با ارتفاع $h-1$ یا $h-2$ شکاند.

پس برای حداقل رئوس می‌توانیم از رابطه‌ی $T(h) = T(h-1) + T(h-2) + 1$ با پایه‌های $T(0) = 1, T(1) = 2$ استفاده کنیم.

برای حداکثر رئوس، هر دو زیردرخت را با ارتفاع $h-1$ در نظر می‌گیریم پس $T(h) = 2T(h-1) + 1$ به کمک استقرا می‌توانیم نشان دهیم که $T(h) = 2^{h+1} - 1$.



پ. شکل زیر یک مثال از چنین درختی است.

ت. در نمایش میان‌ترتیب ابتدا زیردرخت سمت چپ، سپس ریشه و سپس زیردرخت سمت راست را بیان می‌کنیم. پس نمایش میان‌ترتیب به صورت $DCEBFGAIHJ$ خواهد بود.

سؤال ۲. [۲۵ نمره] تابع مرتب‌سازی ادغامی برای مرتب کردن اعداد بازه‌ی $[l, r)$ لیست a را به صورت زیر در نظر بگیرید:

Algorithm 1 Merge Sort

```
procedure SORT(a, l, r)
    return if array is sorted in non-descending order.
    mid ← ⌊(l+r)/2⌋
    sort(a, l, mid)
    sort(a, mid, r)
    merge segments [l, mid) and [mid, r)
```

پس صدا زدن $sort(\{1, 2, 3\}, 0, 3)$ منجر به صدا شدن این تابع یک بار در کل و صدا زدن $sort(\{2, 1, 3\}, 0, 3)$ منجر به صدا زدن این تابع سه بار در کل برای بازه‌های $[0, 3), [0, 1), [1, 2)$ می‌شود.

به ازای هر n, k در ورودی الگوریتمی با پیچیدگی زمانی $O(n \log n)$ طراحی کنید که جایگشتی از اعداد 1 تا n خروجی بدهد که صدا زدن تابع $sort$ روی آن منجر به صدا زدن این تابع به تعداد k بار در کل بشود. (از حالاتی که خروجی معتبری وجود ندارد صرف‌نظر کنید.)

پاسخ: در ابتدا توجه کنید که در هر بار صدا زدن تابع، تابع $sort$ صفر و یا دو بار صدا زده می‌شود. پس با در نظر گرفتن صدا زدن اولیه‌ی تابع، برای مرتب‌سازی، لزوما تابع به تعداد فرد بار صدا زده می‌شود. پس کافی‌ست به ازای مقادیر فرد k دنباله‌ای پیدا کنیم.

دنباله‌ی مرتب از اعداد ۱ تا n را در نظر بگیرید. واضح است که تابع تنها یک بار برای این دنباله صدا زده می‌شود. دقت کنید که اگر اعداد $[1, \lceil \frac{n+1}{2} \rceil)$ را با همین ترتیب به انتهای دنباله منتقل کنیم، دیگر دنباله مرتب نخواهد بود. پس تابع یک بار برای بازه‌ی $[l, mid)$ و بار دیگر برای بازه‌ی $[mid, r)$ صدا زده می‌شود. در این تغییر تعداد صدا زدن تابع دو بار افزایش یافت. توجه کنید که با جابجا نکردن دو قسمت دنباله می‌توانستیم همان تعداد اولیه‌ی صدا زدن را نگه داریم. همچنین توجه کنید که چون در دنباله‌ی اول هر عدد کوچکتر از هر عدد در دنباله‌ی دوم است، اگر ترتیب اعداد در هر یک از این دنباله‌ها را تغییر دهیم، همچنان دنباله نامرتب خواهد بود.

با توجه به مواردی که گفته شد، تابعی بازگشتی طراحی می‌کنیم که تا جایی که نیاز است تعداد بار صدا شدن تابع را زیاد کند (هر بار به اندازه‌ی دو واحد).

Algorithm 2 Merge Unsort

procedure UNSORT(a, l, r, cnt)
    return cnt if array has length equal to one, i.e. $r - l = 1$
    $mid \leftarrow \lceil \frac{l+r}{2} \rceil$
    $cnt \leftarrow cnt + 2$
    swap sequences $[l, mid)$ and $[mid, r)$
    if $cnt < k$ **then**
        $cnt \leftarrow unsort(a, l, mid, cnt)$
    if $cnt < k$ **then**
        $cnt \leftarrow unsort(a, l, mid, cnt)$
    return cnt

توجه کنید که تابع $unsort$ تا زمانی که مقدار cnt یا تعداد بارهای صدا زدن تابع $unsort$ به k نرسیده، مشابه تابع $sort$ صدا زده می‌شود و به طور مشابه در هر بار صدا زده شدن، دو تا به تعداد اضافه می‌کند. پس اگر k معتبر باشد، cnt حتما می‌تواند از آن بیشتر شود.

از طرف دیگر، cnt تنها زمانی افزایش پیدا می‌کند که اکیدا از k کوچکتر باشد، پس حداکثر برابر $k+1$ خواهد شد. از طرفی چون هر دوی این اعداد فرد هستند، پس نهایتا پاسخ تابع $unsort(a, 1, n, 1)$ به ازای $a = [1, n]$ برابر k خواهد بود و اگر پس از اجرای این تابع، تابع $sort$ را روی دنباله‌ی a صدا کنیم تعداد صدا زدن دقیقا برابر k خواهد بود.

سؤال ۳. [۱۵ نمره] با استفاده از دو پشته، یک صف طراحی کنید و نشان دهید که هزینه‌ی حذف و اضافه‌ی یک عنصر به طور سرشکن از $O(1)$ است.

پاسخ:

پشته‌ی اول را IN و پشته‌ی دوم را OUT می‌نامیم. برای پیاده‌سازی $push$ کافی‌ست، مقدار را به پشته‌ی IN اضافه کنیم.

توجه کنید که اگر عملیات دیگری روی پشته‌ی IN انجام ندهیم، مقدار سر پشته، مقداری‌ست که در آخرین مرحله باید حذف شود و مقدار ته IN اولین مقداری‌ست که باید حذف شود.

برای پیاده‌سازی pop مقدار سر پشته‌ی OUT را حذف می‌کنیم. در صورتی که در پشته‌ی OUT مقداری وجود نداشت، مقادیر IN را تک تک از IN حذف و وارد OUT می‌کنیم. دقت کنید که در هر بار تغییر پشته کاملا خالی می‌شود و در نتیجه خاصیت پشته‌ی IN برای اعداد موجود در آن حفظ می‌شود. همچنین در پشته‌ی OUT تنها در زمانی که خالی‌ست، کل پشته‌ی IN اضافه می‌شود که باعث می‌شود مشابه پشته‌ی IN مقادیر در آن به ترتیب ورود باشند، با این تفاوت که در OUT آخرین مقدار در ته پشته قرار می‌گیرد و اولین مقدار در سر پشته قرار می‌گیرد. پس مقداری که از OUT خارج می‌شود، اولین مقداری‌ست که وارد شده. پس پیاده‌سازی معرفی‌شده، خاصیت صف را دارد.

توجه کنید که هر عدد در هر مرحله حداکثر دو بار وارد پشته و حداکثر دوبار از پشته خارج شده است. پس هزینه‌ی هر عملیات به طور سرشکن ثابت و از $O(1)$ است.

سؤال ۴. [۲۵ نمره] می‌دانیم حاصل حل رابطه‌ی بازگشتی $T(0) = 1, T(n) = 1 + \sum_{i=0}^{n-1} T(i)$ برابر $T(n) = 2^n$ است. فردی برای پیدا کردن پیچیدگی این رابطه استدلال زیر را به کار برده:

به استقرا روی n ثابت می‌کنیم $T(n) = \theta(5^n)$. به ازای $n = 0$ به عنوان پایه، $T(0)$ برابر $5^0 = 1$ است.

به ازای هر $n > 0$ اگر $0 \leq m < n$ و داشته باشیم $T(m) = \theta(5^m)$ ثابت می‌کنیم $T(n) = \theta(5^n)$.

$$\begin{aligned}
T(n) = 1 + \sum_{i=0}^{n-1} T(i) &\Longrightarrow \theta(T(n)) = \theta(1 + \sum_{i=0}^{n-1} T(i)) \\
&\Longrightarrow \theta(T(n)) = \theta(1 + \sum_{i=0}^{n-1} \theta(T(i))) \\
&\Longrightarrow \theta(T(n)) = \theta(1 + \sum_{i=0}^{n-1} 5^i) \\
&\Longrightarrow \theta(T(n)) = \theta(1 + \frac{5^n - 1}{4}) \\
&\Longrightarrow \theta(T(n)) = \theta(5^n) \\
&\Longrightarrow T(n) = \theta(5^n)
\end{aligned} \tag{۱}$$

اما با توجه به آن‌چه پیشتر گفتیم، این استدلال نمی‌تواند صحیح باشد، چون $2^n \neq \theta(5^n)$. ایراد این استدلال را بیان

کنید.

پاسخ: به تعریف تابع θ دقت کنید.

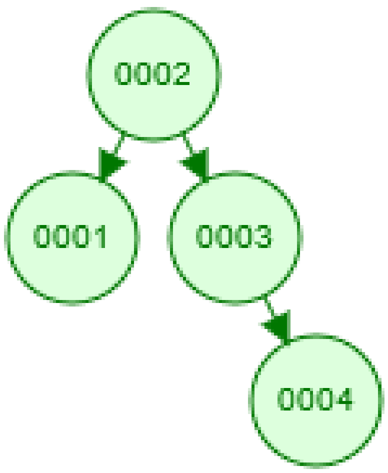
$$\theta(g(n)) = \{f(n) | \exists c_1, c_2 > 0, n_0 : \forall n \geq n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

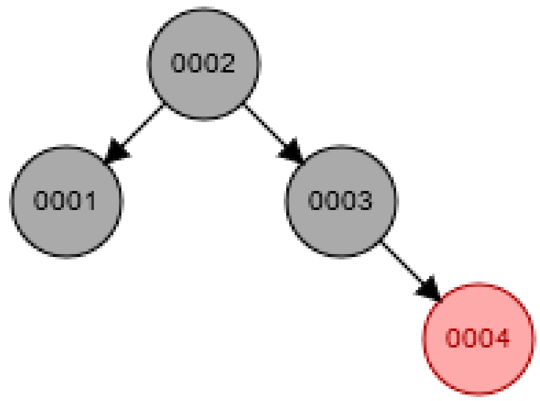
در فرض استقرا آمده که به ازای $0 \leq m < n$ داریم $T(m) = \theta(5^m)$، هرچند گام استقرا صحیح است، اما با توجه به تعریف ما نمی‌توانیم تابع θ را تنها به ازای مقادیر کوچک‌تر از n بیان کنیم. (هرچند می‌توانیم این تابع‌را تنها برای مقادیر بزرگ‌تر از n استفاده کنیم.) در واقع تابع θ دامنه‌ی به اندازه‌ی بی‌نهایت از تابع را در نظر می‌گیرد، پس نمی‌توان از آن تنها با توجه به تعداد اعضای محدودی از دامنه استفاده کرد.

پس اساسا فرض استقرا غلط است و نه در گام و نه در پایه، اثبات نشده.

۶

سؤال ۵. [۱۵ نمره] به دو درخت Red/Black و AVL زیر، ابتدا مقدار 5 را اضافه کرده، و سپس مقدار 2 را از درخت حاصل حذف کنید. (شکل درخت پس از انجام هر عملیات را رسم کنید و علت تغییر ایجادشده را بیان کنید.)





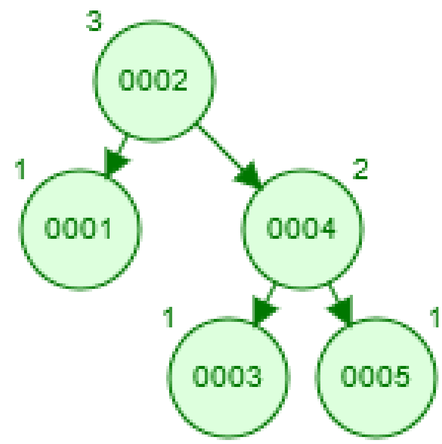
پاسخ:

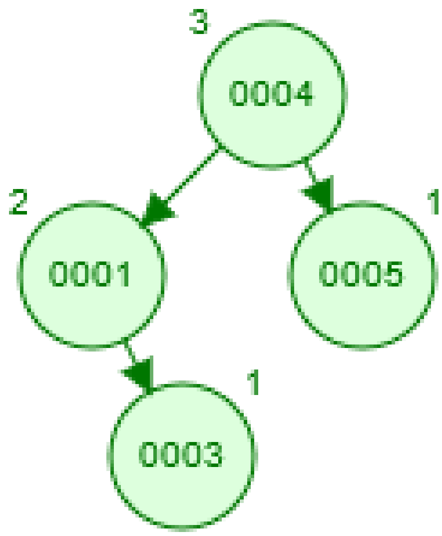
برای درخت AVL مقدار 5 در ابتدا به عنوان فرزند سمت راست 4 قرار می‌گیرد. در نتیجه‌ی این کار ارتفاع دو زیردرخت 3 برابر 2 و 0 می‌شود، به همین دلیل یک عملیات چرخش به چپ انجام می‌دهیم که در نتیجه‌ی آن، راس 3 فرزند راس 4 می‌شود.

در ادامه برای حذف 2 ابتدا بزرگترین راس زیردرخت سمت چپ یعنی 1 را به جای 2 قرار می‌دهیم. در نتیجه‌ی این کار زیردرخت سمت راست ریشه ارتفاع 2 و زیردرخت سمت راست ارتفاع 0 خواهد داشت. پس نیاز به عملیات چرخش به چپ خواهیم داشت. در این چرخش راس 4 ریشه می‌شود و زیردرخت سمت چپ آن به سمت راست راس 1 منتقل می‌شود.

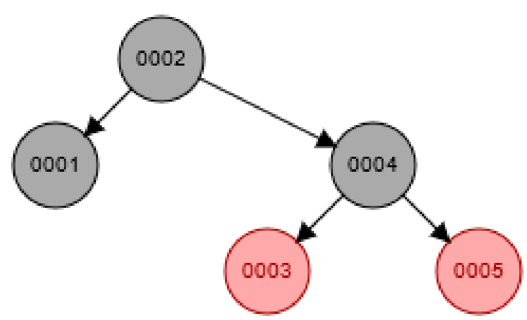
برای درخت Red/Black مقدار 5 در ابتدا سمت راست 4 به رنگ قرمز قرار می‌گیرد. چون هم این راس و هم پدرش قرمز هستند و رنگ عموی 5 یعنی هیچ‌مقدار سیاه است، می‌توانیم با یک چرخش چپ ایراد را برطرف کنیم. پس راس 3 را فرزند چپ 4 می‌کنیم و زیردرخت سمت چپ 4 را در سمت راست آن قرار می‌دهیم. در ادامه رنگ 3 و 4 را به ترتیب قرمز و سیاه می‌کنیم.

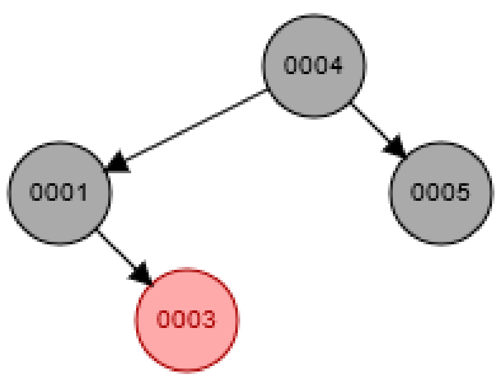
در ادامه برای حذف مقدار 2 باز هم بزرگ‌ترین راس زیردرخت سمت چپ یعنی 1 را جایگزین 2 می‌کنیم و راس فرزند آن یعنی هیچ‌مقدار را سیاه می‌کنیم. این راس دو بار سیاه شده است، و برادرزاده‌ی قرمز دارد. پس با یک چرخش به چپ





می‌توانیم مشکل را حل کنیم. در نتیجه 4 ریشه می‌شود، زیردرخت سمت چپ آن به سمت راست 1 منتقل می‌شود، و رنگ 5 به سیاه تغییر پیدا می‌کند.





موفق باشید