ساختمان داده‌ها و الگوریتم‌ها

نیم‌سال اول ۱۴۰۱ ـ ۱۴۰۰

مدرس: مسعود صدیقین

دانشکده‌ی مهندسی کامپیوتر

سؤالات آمادگی میان‌ترم

مسئله‌ی ۱. هم‌وزن

به شما یک سکه داده شده است و وزن آن را نمی‌دانید. همچنین در مقابل شما ۲۰ سکه چیده شده است که وزن آن‌ها را هم نمی‌دانید ولی می‌دانیم به ترتیب صعودیِ وزن‌شان چیده شده‌اند. تمامی سکه‌ها در مسئله به لحاظ ظاهری یکسان هستند. شما یک ترازوی کفه‌ای دقیق دارید که هر بار می‌توانید سکه‌ی خود را با یک سکه‌ی دیگر وزن کنید. هدف این است که با کمترین استفاده از ترازو، اگر سکه‌ای هم‌وزن سکه‌ی شما وجود دارد آن را پیدا کنید و اگر وجود ندارد، بگویید وجود ندارد. با حداکثر چند بار استفاده از ترازو می‌توانید پاسخ مسئله را بدهید؟

اگر به جای ۲۰ سکه n سکه داشتیم، آن‌وقت حداکثر چند بار استفاده از ترازو لازم است؟

مسئله‌ی ۲. هفتهٔ آخر شهریور

شبه‌کدِ الگوریتمِ ساده‌ی ضرب دو عدد nبیتی را ارائه کنید و تحلیل‌های زمانی در بهترین حالت و بدترین حالت را انجام دهید. فرض کنید دو عدد ورودی در آرایه‌های A و B هستند و خروجی در آرایه‌ی C ریخته می‌شود.

مسئله‌ی ۳. پالیندروم

یک پالیندروم، رشته‌ای است که از دوطرف یکسان خوانده می‌شود. اگر اجازه دهیم که یک پالیندروم تنها شامل یک حرف باشد، هر رشته‌ای را می‌توان به صورت دنباله‌ای از پالیندروم‌ها دید. ما می‌خواهیم با داشتن رشته‌ی s مقدار $MinPal(s)$ را محاسبه کنیم که برابر است با کمینه مقدار پالیندروم‌هایی که می‌توان با آن‌ها رشته s را ساخت. یعنی کمترین k ممکن که s را بتوان به صورت $w_1w_2...w_k$ نوشت به طوری که همه‌ی $w_1, w_2, ..., w_k$ پالیندروم باشند. الگوریتمی طراحی کنید که در زمان $O(n^3)$ که n طول رشته است، مقدار $MinPal(s)$ را محاسبه کند.

مسئله‌ی ۴. فتح قله

به شما یک آرایه تک بعدی از n عنصر مجزا داده شده است. به این شکل که ورودی های آن تا بیشترین عنصر به ترتیب در حال افزایش هستند، و پس از آن عناصر آن در حال کاهش هستند. یک الگوریتم برای محاسبه بیشترین عنصر که در زمان $O(log(n))$ اجرا می شود ارائه بدهید.

حل. به دو بخش آرایه را تقسیم می‌کنیم. اگر پاسخ در بخش چپ باشد به ادامه سوال در آن بخش می‌پردازیم و در غیر این صورت به راست می‌رویم. معیارمان برای انتخاب نیمه‌ها هم اختلاف عضو آخر نیمه چپ و عضو اول نیمه راست با عنصر مرکزی آرایه است. اگر ترتیب صعودی بود باید به نیمه راست رفت. اگر ترتیب نزولی بود باید به نیمه چپ رفت و اگر عنصر مرگزی بیشینه بود پاسخ همان است. ▷

مسئله‌ی ۵. سریع

به شما یک آرایه مرتب‌شده (صعودی اکید) A از n عدد صحیح مجزا داده می‌شود که می‌توانند مثبت، منفی یا صفر باشند. شما می‌خواهید تصمیم بگیرید که آیا اندیس i به گونه‌ای وجود دارد که $A[i] = i$ باشد یا خیر. برای حل این مسئله سریع‌ترین الگوریتم را طراحی کنید.

حل. ایده‌ای مشابه سوال ۴ دارد. فقط معیار انتخاب مسیر چپ یا راست در آن به بیشتر بودن یا کمتر بودن ایندکس از مقدارش بسته است که در حالتی که کمتر باشد به نیمه چپ و در غیر این صورت به نیمه راست می‌رویم. ▷

مسئله‌ی ۶. ضرب سریع (تقسیم و غلبه)

دو عدد $2n$ رقمی را به گونه‌ای در هم ضرب کنید که زمانی از اردر $O(log_2^3)$ داشته باشد.

حل. الگوریتم کاراتسوبا ▷

مسئله‌ی ۷. دوطرفه

فرض کنید صف دوطرفه‌ای (با امکان درج و حذف از ابتدا و یا انتهای صف) داریم که به کمک سه پشته با اندازه‌ی مشخص پیاده‌سازی شده است؛ یک پشته شامل «سر»، یک پشته شامل «بدنه» و یک پشته نیز شامل «دم» داده‌ساختار است. برای افزودن (push) مقدار به این صف، مقدار جدید را در پشته‌ی «سر» یا «دم» ــ بسته به جهت درج ــ افزوده و برای برداشتن (pop) مقدار از این صف، مقدار را از پشته‌ی «سر» یا «دم» ــ بسته به جهت برداشتن ــ حذف می‌کنیم. لازم است الگوریتمی برای این داده‌ساختار طراحی شود تا هنگام خالی بودن پشته‌های دو سر طی عملیات pop بتواند از پشته‌ی میانی بهره گیرد که عملیات pop تا زمانی که داده‌ای درون این صف موجود است، با شکست مواجه نشود. فرض کنید که pop از یک پشته و بلافاصله push به یک پشته‌ی دیگر، به عنوان یک عملیات در نظر گرفته می‌شود.

الف) روش پیشنهادی خود برای چگونگی پیاده‌سازی pop را به طور دقیق، توضیح دهید، به گونه‌ای که زمان اجرای سرشکن این عمل، از زمان چندجمله‌ای سریع‌تر باشد.

ب) مرتبه‌ی زمانی الگوریتم بیان شده توسط خود در تابع pop را با روش تابع پتانسیل، محاسبه کنید.

حل. الف) الگوریتم را به این شکل طراحی می‌کنیم که تا وقتی پشته‌های دم و سر پر نشده‌اند، پشته‌ی مبانی خالی بماند. هر گاه قرار شود از پشته‌ای pop کنیم که شامل مقداری درون خود نیست (بدون کاستن از کلیت مسیله فرض می‌کنیم این پشته دم است) اگر پشته‌ی سر خالی باشد، به این معناست که مقداری درون صف وجود ندارد و در نتیجه منطقی است که عملیات با شکست مواجه شود. اگر پشته‌ی سر شامل مقداری باشد، فرض می‌کنیم که پشته‌ی سر شامل n مقدار است و n را نیز زوج فرض می‌کنیم (اگر فرد باشد، استدلال مشابه است، با این تفاوت که به عملگرهای کف و سقف نیاز می‌شود). در این حالت، ابتدا تمام اعداد را از پشته‌ی سر به پشته‌ی مبانی تک‌تک pop و push می‌کنیم، سپس $\frac{n}{2}$ عدد را از پشته‌ی مبانی تک‌تک pop کرده و به پشته‌ی دم push می‌کنیم و آن‌گاه $\frac{n}{2}$ عدد باقی‌مانده را نیز از پشته‌ی مبانی تک‌تک pop کرده، اما این بار به پشته‌ی سر push می‌کنیم. بررسی زمان این الگوریتم، در بخش (ب) آمده است.

ب) حالت pop عادی، واضح است. بدترین حالت، آن است که یک پشته خالی بوده (و می‌خواهیم از آن pop کنیم) و پشته‌ی دیگر که در تقابل با آن است، شامل اعداد زیادی باشد. در نتیجه، می‌توان تفاوت تعداد اعداد درون

دو پشته‌ی سر و دم را به عنوان تابع پتانسبل در نظر گرفت؛ البته، به دلیل هماهنگی با محاسبات، «دو برابر» تابع پتانسیل را در نظر می‌گیریم که هم‌چنان خواص تابع پتانسیل را داراست، در نتیجه، تابع پتانسیل برابر می‌شود با:

$$۲ \times ||head| - |tail||$$

قبل از انجام عملیات pop در این حالت، یکی از پشته‌ها خالی بوده و پشته‌ی دیگر شامل مثلا n عضو بوده است و پس از pop هر دو پشته دارای $\frac{n}{۲}$ عضو خواهند بود؛ در نتیجه، تفاوت مقدار دو پشته از n به صفر می‌رسد و در نتیجه تابع پتانسیل به اندازه‌ی $۲n$ کاهش خواهد داشت. با توجه به آن‌که کل عملیات انجام شده شامل n بار push و سپس دو بار push به تعداد $\frac{n}{۲}$ بار است، در کل $۲n$ عملیات رخ می‌دهد و در نتیجه هزینه‌ی کل عملیات به صورت سرشکن برابر است با:

$$۲n - ۲n = ۰ \in O(۱)$$

▷

مسئله‌ی ۸. قطعه‌کد عجیب

اگر $array$ آرایه‌ای شامل n عضو بوده و تمامی اعضای آن صفر باشند، در صورت فراخوانی n بار از تابع زیر، هزینه‌ی سرشکن هر بار اجرا را به کمک روش تابع پتانسیل به دست آورید.

Algorithm 1: Insert

```
procedure INSERT(ELEMENT)
    n = n + 1
    temp = element
    for i = 0 to ceil(log2(n)) :
        if array[i] == 0 :
            array[i] = temp
            return
        else:
            temp += array[i]
            array[i] = 0
```

حل. تابع پتانسیل را برابر تعداد اعضای ناصفر آرایه در نظر می‌گیریم؛ اگر حلقه‌ی تابع m بار اجرا شود، در بدترین حالت در هر فراخوانی m درایه‌ی ناصفر، صفر شده و حداکثر یک درایه‌ی صفر، ناصفر می‌شود، در نتیجه به میزان حداکثر $m - ۱$ اختلاف در تابع پتانسیل خواهیم داشت و از طرفی، اجرای خود حلقه نیز $O(m)$ طول می‌کشد، در نتیجه هزینه‌ی سرشکن هر بار اجرا، $O(m) + O(۱ - m) = O(۱)$ است. ▷

مسئله‌ی ۹. نصفش رو پاک کن

داده‌ساختاری از اعداد طراحی کنید که اگر n عنصر در آن قرار داشته باشد، هر یک از دو عملیات «درج یک عدد

جدید» و «حذف $\frac{n}{2}$ عنصر بزرگ‌تر» را در $O(1)$ به صورت سرشکن انجام دهد. در فرآیند حل سوال، تحلیل‌های سرشکن را توسط روش «حساب‌داری» انجام دهید.

حل. می‌توانیم از یک آرایه به عنوان داده‌ساختار مدنظر سوال استفاده کنیم. می‌دانیم به کمک الگوریتم انتخاب می‌توان میانه‌ی آرایه را در زمان خطی، یافت؛ پس در زمان خطی، میانه را یافته و آرایه‌ی جدیدی تنها با اعداد کوچک‌تر از میانه می‌سازیم، در نتیجه هزینه‌ی هر عمل حذف، از زمان خطی است. بدیهتا، هر عملیات حذف مطابق صورت سوال، کم‌تر مساوی $2n$ عملیات خواهد بود، پس کافی است هنگام درج هر عنصر، ۵ سکه به همراه آن عنصر به داده‌ساختار بفرستیم که یک سکه صرف خود درج اولیه شده و ۴ سکه ذخیره شوند تا هرگاه نیاز به حذف $\frac{n}{2}$ عنصر بزرگ‌تر آرایه بود، بتوان هزینه‌ی لازم برای حذف این عناصر را از سکه‌های ذخیره شده در آن‌ها که $4 \times \frac{n}{2} = 2n$ هستند، تامین کرده (و هیچ‌گاه هزینه منفی نمی‌شود)؛ در نتیجه، عملا هزینه‌ی عملیات حذف، صفر می‌شود (و از $O(1)$ بوده) و هزینه‌ی درج نیز از $O(1)$ است.

$\triangleright$

مسئله‌ی ۱۰. پاپوش

با استفاده از پشته، داده ساختار صف را پیاده‌سازی کنید به طوری که هزینه عملیات اضافه کردن به صف در آن از $O(۱)$ باشد. هزینه سرشکن n عمل اضافه و حذف از صف را با استفاده از روش تابع پتانسیل تحلیل کنید. فرض کنید هزینه‌ها معادل تعداد درج‌ها و حذف‌ها است.

حل. می‌دانیم که پشته ها $FILO$ و صف ها FIFO هستند. حال فرض کنید که دو پشته داریم که پشته ی اول پر و پشته ی دوم خالی می‌باشد. حال به ترتیب اجزای پشته ی اول را خوانده و در پشته ی دوم می‌ریزیم، چون پشته ها FILO هستند، اجزایی که زودتر وارد پشته ی اول شده اند دیرتر وارد پشته ی دوم می‌شوند. چون پشته ی دوم نیز FILO است، اجزایی که دیرتر وارد شده اند زودتر خارج می‌شوند، پس هر عضوی که زودتر وارد پشته ی اول شده باشد، بعد از خالی کردن پشته ی اول به پشته ی دوم زودتر خارج می‌شود که مانند یک FIFO می‌باشد. حال با کمک این ایده به کمک دو پشته یک صف را طراحی می‌کنیم:

هنگام push به پشته ی اول push می‌کنیم. هنگام pop نیز از پشته ی دوم pop می‌کنیم، اما اگر پشته ی دوم خالی بود، ابتدا تمام اعضای پشته ی اول را وارد پشته ی دوم می‌کنیم. سپس از پشته ی دوم pop می‌کنیم. عملیات push چون عملیات push به پشته است از $O(۱)$ است. عملیات pop شامل حداکثر n عمل pop برای خالی کردن پشته ی اول n عمل push برای وارد شدن به پشته دوم و یک عملیات pop از پشته ی دوم می‌باشد. پس از $O(n)$ است.

برای محاسبه هزینه سرشکن: $\phi = ۲len(S_A)$

$$c_i = c_i + \phi_i - \phi_{i-۱}$$

چون در این سوال هزینه ی push و pop یکسان است پس داریم :

$$c_{ipop} = c_{ipush}$$

برای Enqueue فقط یک push در استک A داریم که c_i در آن یک است که به علاوه ی دوبرابر تفاوت فاصله ی استک A قبل و بعد از push کردن می‌شود.

$$Enqueue : c_i = ۱ + ۲(len(S_A) - len(S_{A-۱})) = ۱ + ۲ = ۳ = O(۱)$$

برای Dequeue ابتدا حالتی را در نظر می‌گیریم که استک B خالی نباشد. در این صورت تنها کافی است یکی از B برداریم.

$$Dequeue : iflen(S_B) > ۰ci = ۱ + ۲(len(S_A)len(S_{A-۱})) = ۱ + ۰ = ۱ = O(۱)$$

در حالتی که استک B خالی است باید تمام استک A را پاپ کنیم و بعد به اندازه ی طول A در استک B بریزیم و در آخر یکی pop می‌کنیم.

$$c_i = ۲len(S_A) + ۱$$

$$\phi_i = ۰$$

$$\phi_{i-۱} = ۲len(S_A)$$

$$c_i = ۱ = O(۱)$$

بنابراین هزینه سرشکن برای این پیاده سازی برای Enqueue برابر ۳ است و برای Dequeue برابر ۱ است.

▷

مسئله‌ی ۱۱. لیست

در لیست L اعمال زیر تعریف شده اند:

$Insert(L,x)$: عنصر x را به انتهای L اضافه می‌کند.

$Delete(L)$: عنصر انتهایی L را حذف می‌کند.

$multi-Delete(L,k)$: k عنصر انتهایی L را حذف می‌کند.

اگر n تا از اعمال فوق به ترتیب صحیح و دلخواه روی لیست L که در ابتدا تهی است اعمال شود، هزینه هر عمل به صورت سرشکن چقدر خواهد بود؟

حل. دستورات Insert و Delete و multi-Delete به ترتیب $O(۱)$، $O(۱)$ و $O(k)$ هستند. در نظر بگیرید هر شیئی که در لیست Insert می‌شود، حداکثر یک بار می‌تواند Delete شود. پس تعداد هاDelete حداکثر برابر تعداد Insert ها است و بدیهی است که حداکثر تعداد Insert برابر با n است. بنابراین تعداد عملیات Delete هم حداکثر برابر با n است. اگر $T(n)$ را هزینه ی n عملیات در نظر بگیریم؛ داریم : : $T(n) = O(n)$: $T(n) \leqslant ۲n$
$\frac{T(n)}{n} = O(۱)$

پس هزینه ی n عملیات به صورت سرشکن $O(۱)$ می‌شود.

▷

مسئله‌ی ۱۲. آرایه

فرض کنید که A یک آرایه به طول n از اعداد است. برای هر اندیس i اولین خانه سمت راستش که مقداری بزرگتر مساوی A_i دارد را در پیچیدگی زمانی $O(n)$ بیابید.

حل. مرحله اول ــ اولین عنصر لیست را در استک پوش کنید.

مرحله دوم ــ عنصر بعدی را next بنامید . next را با عنصر بالای استک مقایسه کنید . مادامی که next بزرگتر بود، عنصر بالای استک را pop کنید .

مرحله سوم ــ next را در استک پوش کنید.

مرحله چهارم ــ اگر که عنصر بعدی در آرایه موجود بود، به مرحله دوم بروید.

تحلیل الگوریتم: هر عنصر یکبار در استک push و ماکزیمم یک بار pop می‌شود . هر مقایسه ای هم که انجام می‌شود یا متناظر است با یک عملیات pop یا این که باعث می‌شود که حلقه داخل الگوریتم پایان بپذیرد و عنصر بعدی داخل استک push شود . پس در کل الگوریتم ما از $O(n)$ است .

▷

مسئله‌ی ۱۳. برعکس

راهی غیر بازگشتی با زمان $\theta(n)$ و حافظه‌ی اضافی $\theta(n)$ ارائه کنید که یک لیست پیوندی یک طرفه با n عنصر را معکوس می‌کند.

حل.

REVERSE(L):

$b = L.head$

$a = b.next$

while $a \neq Null$ **do**

$tmp = a.next$

$a.next = b$

$b = a$

$a = tmp$

end while

$L.head = b$

$\triangleright$

مسئله‌ی ۱۴. پیوندی عجیب

توضیح دهید چگونه می‌توان یک لیست پیوندی دو طرفه را با استفاده از فقط یک فیلد اشاره‌گر np به جای دو فیلد $next$ و $prev$ در هر عنصر پیاده‌سازی کرد. نشان دهید چگونه می‌توان عملیات Search و Insert و Delete را روی چنین لیستی پیاده‌سازی کرد. هم‌چنین نشان دهید چونه می‌توان چنین لیستی را در زمان $O(۱)$ معکوس کرد.

راهنمایی: فرض کنید اشاره‌گرها اعداد صحیح k بیتی هستند و $x.np = x.prev\ XOR\ x.next$ و فرض کنید $Null$ (در فیلد $next$ عنصر آخر و $prev$ عنصر اول) با ۰ نشان داده می‌شود.

حل.

SEARCH(L،k):

$p = Null$

$x = L.head$

while $x \neq Null$ **do**

$tmp = x$

$x = p\ XOR\ x.np$

$p = tmp$

end while

$return\ x$

INSERT(L،x):

$x.np = L.head$

$L.head.np = L.head.np\ XOR\ x$

$L.head = x$

DELETE(L):

$L.head.np.np = L.head.np.np\ XOR\ L.head$

$L.head = L.head.np$

REVERSE(L):

$tmp = L.head$

$L.head = L.tail$

$L.tail = tmp$

$\triangleright$

مسئله‌ی ۱۵. زیر درخت

نشان دهید در یک درخت دودویی با n برگ (که $n \geqslant ۲$)، زیردرختی وجود دارد که اگر برگ‌های آن را m بنامیم آن گاه داریم:

$$\frac{n}{۳} \leqslant m \leqslant \frac{۲n}{۳}$$

حل. مشاهده: می‌دانیم برای هر راس، یکی از زیردرخت‌هایش حداقل نصف تعداد برگ‌های زیر درخت شامل خود

آن راس برگ دارد.
بنابراین از ریشه‌ی درخت دودویی شروع می‌کنیم و در هر مرحله زیردرختی را انتخاب می‌کنیم که تعداد برگ بیشتری دارد. فرض می‌کنیم تعداد برگ این زیر درخت برابر m است. طبق مشاهده‌ای که گفته شد، $m \geqslant \frac{n}{2} > \frac{n}{3}$ و شرط اول برقرار است. اگر شرط دوم، یعنی $m \leqslant \frac{2n}{3}$ برقرار باشد که زیردرخت مورد نظرمان را یافته‌ایم. در غیر این صورت $m > \frac{2n}{3}$ است. پس مجددا زیردرختی از این راس را انتخاب می‌کنیم که تعداد برگ بیشتری دارد و تعداد برگ آن را m می‌نامیم. باز هم طبق مشاهده $m > \frac{n}{3}$ است؛ زیرا:

$$m > number\ of\ leaves\ in\ previous\ subtree\ \div 2 > \frac{2n}{3} \div 2 = \frac{n}{3}$$

پس شرط اول برقرار است. اگر شرط دوم برقرار بود که زیردرخت مورد نظرمان را یافته‌ایم. در غیر این صورت انقدر این مرحله را تکرار می‌کنیم تا به زیردرختی با ویژگی مورد نظر برسیم. اگر فرض کنیم هرگز به چنین زیردرختی نمی‌رسیم، روش بالا به ما می‌گوید که تعداد برگ‌های درخت اصلی بی‌نهایت است که با فرض سوال در تناقض است. پس حکم ثابت شد.

$\triangleright$

مسئله‌ی ۱۶. درخت لگاریتمی

فرض کنید T یک درخت دودویی کامل با n گره و به ارتفاع $\log n$ است. می‌خواهیم مسیر ساده‌ای بین یک رأس u به یک رأس v بیابیم. می‌دانیم که هر گره از این درخت به گره‌های فرزند و گره‌ی پدر خود دسترسی دارد. این کار را در چه مرتبه‌ی زمانی می‌توان انجام داد؟

حل. از u شروع می‌کنیم و به سمت گره‌های پدر آن حرکت می‌کنیم و رئوسی که در حین این پیمایش از آن می‌گذریم را seen می‌کنیم (در آرایه‌ی $seen$ درایه‌ی مربوط به آن راس را $true$ می‌کنیم). این پیمایش در ریشه‌ی درخت متوقف می‌شود. این پیمایش به $O(\log n)$ زمان نیاز دارد.

به طور مشابه از راس v شروع می‌کنیم و به سمت رئوس پدر آن حرکت می‌کنیم. اولین رأسی که seen شده باشد، اولین جد مشترک u و v است. این پیمایش نیز به $O(\log n)$ زمان نیاز دارد. اگر این رأس را x بنامیم، مسیر u به x به v مسیر بین u و v خواهد بود (اگر u جد v باشد یا بالعکس، x برابر u یا v خواهد بود). این کار در مجموع در زمان $O(\log n)$ انجام خواهد شد. $\triangleright$

مسئله‌ی ۱۷. درخت‌سازی

پیمایش پیش‌ترتیب و میان‌ترتیب یک درخت دودویی داده شده است. الگوریتمی پیدا کنید که درخت مربوط به آن را ایجاد کند.

$preorder(T)$: ۶, ۲, ۴, ۳, ۵, ۸, ۷, ۱

$inorder(T)$: ۴, ۲, ۶, ۸, ۵, ۷, ۳, ۱

حل.

به کمک پیمایش پیش‌ترتیب، ریشه را پیدا کرده و در پیمایش میان‌ترتیب این رأس را پیدا کرده و دو زیر درخت چپ و راست را مشخص می‌کنیم. به همین ترتیب این مراحل را برای زیردرخت چپ و راست انجام می‌دهیم تا جایی که زیردرخت هیچ رأسی نداشته باشد.

▷

مسئله‌ی ۱۸. بی تو هرگز

فرض کنید $G = (V, E)$ یک گراف ساده همبند باشد. فرض کنید $T_s = (V, F)$ درختی در G باشد که با DFS و شروع از راس s بدست بیاید. ثابت کنید s بیش از ۱ فرزند در T_s دارد اگر و تنها اگر حذف کردن s از G این گراف را ناهمبند کند.

مسئله‌ی ۱۹. د.د.ج

آ. فرض کنید یک د.د.ج با درج متوالی مقادیر متفاوت در درخت ساخته شده باشد. نشان دهید تعداد گره‌هایی که برای جست‌وجوی یک مقدار در درخت دیده می‌شوندیکی بیش از تعداد گره‌های دیده‌شده در زمان درج آن مقدار در درخت است.

ب. آیا عمل حذف در د.د.ج یک عمل جابجایی‌پذیر است؟ یعنی درصورتی که ابتدا عنصر x و سپس y را حذف کنیم معادل است با اینکه ابتدا y و سپس x را حذف کنیم؟ این گزاره را اثبات یا رد کنید.

مسئله‌ی ۲۰. پیمایش برعکس

درخت عبارت T را در نظر بگیرید. آیا ممکن است که پیمایش پیش‌ترتیب، راس ها را در جهت مخالف پیمایش پس‌ترتیب طی کند؟ اگر آری، مثالی ارائه دهید. در غیر این صورت علت ممکن نبودن آن را شرح دهید.

مسئله‌ی ۲۱. تبدیل عبارت ۲

نمایش میان‌ترتیب عبارت زیر داده شده است. نمایش پیش‌ترتیب این عبارت را بدست آورید.

$$(((A \times (B + (C \div D))) \times E) - F)$$

حل.

$$- \div \div A + B \div CDEF$$

▷

مسئله‌ی ۲۲. تبدیل عبارت ۳

نمایش پیش‌ترتیب عبارت زیر داده شده است. نمایش پس‌ترتیب این عبارت را بدست آورید.

$$\times + a - bc \div -de - f + gh$$

حل.

$$ab + c - de - *fg - h+$$

$\triangleright$

مسئله‌ی ۲۳. تعداد تعویض ها

درخت دودویی با n راس داده می شود. حداقل تعداد تعویض های لازم برای تبدیل درخت مذکور به یک درخت دودویی جستجو را بیابید.

حل. از این ایده استفاده می کنیم که پیمایش $inorder$ در د.د.ج عناصر را به صورت صعودی مرتب می‌کند. پس پیمایش $inorder$ را بر روی درخت دودویی اولیه انجام می‌دهیم و آن را در یک آرایه ذخیره می‌کنیم. حال این آرایه را به صورت صعودی مرتب می‌کنیم، تنها نیاز است حداقل تعداد $swap$ ها برای مرتب کردن یک آرایه را محاسبه کنیم که پیچیدگی زمانی $O(nlog\, n)$ دارد. کد زیر کمترین مقدار $swap$ ها را برای مرتب کردن یک آرایه محاسبه می‌کند:

```
def minSwaps(arr):
    n = len(arr)
    arrpos = [*enumerate(arr)]
    arrpos.sort(key = lambda it: it[۱])
    vis = {k: False for k in range(n)}
    ans = ۰
    for i in range(n):
        if vis[i] or arrpos[i][۰] == i:
            continue
        cycle_size = ۰
        j = i
        while not vis[j]:
            vis[j] = True
            j = arrpos[j][۰]
            cycle_size+ = ۱
        if cycle_size > ۰:
            ans + = (cycle_size − ۱)
    return ans
```

$\triangleright$

مسئله‌ی ۲۴. ادراج!

تعداد راه های درج کردن اعداد ۱ تا n در یک د.د.ج با حداکثر ارتفاع ممکن چقدر است؟ برای $n =$ ۵ این مقدار را بیابید.

حل. ریشه یا عدد ۱ میتواند باشد یا عدد n پس دو حالت ممکن است که پس از قرار دادن یکی از این دو عدد $n -$ ۱ عدد باقی می ماند که همان $T(n-1)$ است. پس داریم: $T(n) = 2T(n-1)$ می دانیم که: $T(1) = 1$ پس نهایتا تعداد روش های درج کردن اعداد ۱ تا n در درخت دودویی جستجو به این صورت است: $۲^{n-۱}$ حال برای $n =$ ۵ پاسخ ما برابر ۱۶ خواهد بود. ▷

مسئله‌ی ۲۵. پیش به پس

الف) پیمایش پیش ترتیب دو د.د.ج به ترتیب به صورت زیر هستند:

۳۴, ۲۴, ۲۹, ۷۴, ۹۴
۱۳, ۸, ۱۰, ۹, ۱۸, ۱۶, ۱۴, ۱۷

پیمایش پس ترتیب این دو د.د.ج چگونه خواهد بود؟

ب) در حالت کلی پیچیدگی زمانی تبدیل پیمایش پیش ترتیب یک د.د.ج به پس ترتیب آن با n عنصر چقدر خواهد بود؟(بدون بدست آوردن خود درخت)

حل. می دانیم در پیمایش پیش ترتیب اولین عنصر ریشه درخت است. با توجه به پیمایش پیش ترتیب موجود هر عنصر را با عنصر ریشه مقایسه می کنیم تا جایی که عناصر از ریشه کمتر باشند. مقدار $index$ را ذخیره می کنیم. حال عناصر را از عنصر $index$ تا عنصر ۱ چاپ می‌کنیم. سپس از عنصر اخر پیمایش پیش ترتیب داده شده شروع کرده و تا عنصر $index +$ ۱ چاپ می‌کنیم و نهایتا عنصر اول که ریشه است را چاپ می‌کنیم. واضح است این روش از $O(n)$ می باشد. با توجه به این روش، پیمایش پس ترتیب درخت های دودویی جستجو ما به ترتیب به صورت زیر می شوند:

۲۹, ۲۴, ۹۴, ۷۴, ۳۴
۹, ۱۰, ۸, ۱۷, ۱۴, ۱۶, ۱۸, ۱۳

▷

مسئله‌ی ۲۶. پیش به خود

اگر پیمایش پیش‌ترتیب یک د.د.ج به شما داده شود، کارآمدترین الگوریتم از نظر پیچیدگی زمانی برای پیدا کردن این د.د.ج را ارائه دهید.

حل. کارآمد ترین الگوریتم از $O(n)$ می‌باشد که به صورت زیر است:
اگر پیمایش پیش ترتیب یا پس ترتیب را داشته باشیم و بخواهیم که درخت اصلی را بدست بیاوریم راه حل عملا

یکسان است. در این سوال فرض کردیم پیمایش پیش ترتیب را داریم حال ما با استفاده از دو متغیر min و max بازه تعریف می‌کنیم که در ابتدا این مقادیر را منفی بی نهایت و مثبت بی نهایت قرار می‌دهیم سپس از ابتدای پیمایش پیش ترتیب حرکت می‌کنیم و بررسی می‌کنیم که آیا در بازه مورد نظر ما موجود هست یا خیر. بدیهی است که اولین عنصر در بازه هست و در واقع ریشه درخت ما می‌باشد. هربار بررسی می‌کنیم که عنصر بعدی در پیمایش پیش ترتیب اگر در بازه $(min, root)$ باشد در زیر درخت سمت چپ و اگر در بازه $(root, max)$ باشد در زیر درخت راست ریشه قرار میگیرد که با ادامه این کار در $O(n)$ انجام می‌شود و درخت دودویی جستجوی ما بدست می‌آید. ◁

مسئله‌ی ۲۷. شمارش

فرض کنید آرایه ای به طول n به شما داده می شود. حال الگوریتمی از $O(nlog\,n)$ طراحی کنید که تعداد عناصری که از همه ی عناصر قبل خود بیشتر و حداقل از k عنصر سمت راست خود بیشتر باشد را محاسبه کند.

حل. ابتدا آرایه را از اول تا اخر پیمایش می کنیم و همه عناصر را در درخت AVL درج می‌کنیم. حال با استفاده از درخت AVL آرایه ای به نام $counterOfSmallerElements[]$ ایجاد می کنیم که برای هر عنصر تعداد عناصر کوچکتر از آن در سمت چپش را ذخیره کند.این عمل با استفاده از درخت AVL در پیچیدگی زمانی $O(nlog\,n)$ امکان پذیر است. حال یک متغیر به نام $counter$ که تعداد عناصر با شرایط خواسته سوال را در آن ذخیره کنیم. سپس آرایه ی اولیه را پیمایش می‌کنیم و برای هر عنصر مشاهده می کنیم که آیا ماکسیمم عناصر طی شده تا به اون لحظه است یا خیر و در صورت ماکسیمم بودن آرایه ی $counterOfSmallerElements[]$ مربوط به عنصر را چک می‌کنیم که آیا از k بزرگتر است و در صورت بزرگتر بودن متغیر شمارنده ی خود را یکی زیاد می‌کنیم. پس از اتمام این پیمایش که از $O(n)$ متغیر شمارنده ی ما تعداد عناصر مطلوب را نشان می دهد و در کل الگوریتم ما دارای پیچیدگی زمانی $O(nlog\,n)$ است. ◁

مسئله‌ی ۲۸. تبدیل

الگوریتمی از مرتبه ی $O(n)$ ارائه دهید که یک درخت دودویی جستجو نامتوازن را به یک درخت دودویی جستجو متوازن از نظر ارتفاع تبدیل کند.(منظور از درخت دودویی جستجو متوازن از نظر ارتفاع درخت دودویی جستجویی است که ارتفاع زیر درخت های چپ و راست آن برای هر راس حداکثر یک واحد اختلاف داشته باشند)

حل. برای این کار کافی است یک پیمایش میان ترتیب برروی د.د.ج اولیه داشته باشیم و ما این عناصر را در آرایه ای ذخیره می کنیم. می دانیم که این پیمایش عناصر د.د.ج را به صورت مرتب شده به ما می دهد پس آرایه ما مرتب شده خواهد بود. حال برای اینکه درخت دودویی جستجو مطلوب را ایجاد کنیم کافی است ریشه ی درخت جدید را عنصر وسط بگیریم که در این صورت عناصر کوچکتر از عنصر ریشه ما در زیر درخت چپ قرار می‌گیرند و عناصر بزرگتر از ریشه در زیر درخت راست. ما می‌توانیم این عمل را به صورت بازگشتی انجام دهیم که در آخر درخت مورد نظر ما ایجاد می‌شود که این عمل با پیچیدگی زمانی $O(n)$ انجام می‌شود. ◁

مسئله‌ی ۲۹. درج و حذف کن

یک درخت AVL درنظر بگیرید که عناصر زیر به ترتیب در آن درج شده است.

۲۲, ۲۷, ۳۰, ۹, ۱۴, ۲۴, ۲۸, ۴, ۱۸, ۱۵

الف) این درخت را پس از درج شدن عناصر ۹، ۴ و ۱۵ نشان دهید.

ب) حال فرض کنید که عناصر ۴، ۳۰ و ۲۴ را از این درخت حذف کنیم. در هر مرحله درخت حاصل را رسم کنید و مراحل را توضیح دهید.

حل. از این لینک استفاده کنید. ▷

مسئله‌ی ۳۰. بررسی کن

به خانواده‌ای از درخت‌ها متوازن می‌گوییم هرگاه ارتفاع هر درخت عضو این مجموعه $O(\log(n))$ باشد. درستی یا نادرستی موارد زیر را بررسی کنید (برای درست، اثبات و برای نادرست مثال نقض).

- هر گره‌ای یا برگ است یا دو فرزند دارد
- میانگین عمق گره‌ها $O(logn)$ است.
- یک عدد ثابت c وجود دارد به گونه‌ای که برای هر گره، اختلاف ارتفاع فرزندانش حداکثر c باشد

مسئله‌ی ۳۱. تمرین هرم

برای هر آرایه داده شده، آن را به صورت یک درخت دودویی کامل (پر کردن از چپ به راست) رسم کنید و سپس تعیین کنید که درخت بدست آمده، یک هرم بیشینه یا یک هرم کمینه یا هیچکدام است. اگر درخت به شکل هرم نبود، با تعویض کردن مکان گره‌های مجاور در درخت، آن را به یک هرم کمینه تبدیل کنید و مراحل تبدیل درخت به هرم کمینه را نشان دهید.

$a.$ [۴, ۱۲, ۸, ۲۱, ۱۴, ۹, ۱۷]
$b.$ [۷۰۱, ۲۵۳, ۲۴, ۲۲۹, ۱۷, ۲۲]
$c.$ [۲, ۹, ۱۳, ۸, ۰, ۲]
$d.$ [۱, ۳, ۶, ۵, ۴, ۹, ۷]

مسئله‌ی ۳۲. بچرخونشون

دو درخت دودویی جست و جو $T_۱$ و $T_۲$ را درنظر بگیرید که شامل کلید های برابر و غیرتکراری هستند.

نشان دهید با انجام تعداد محدودی عملیات‌های Rotation می‌توان این دو درخت را به هم تبدیل کرد.

مسئله‌ی ۳۳. هرم اتفاقی

آرایه $[A, B, C, D, E, F, G, H, I, J, K, L]$ نشان‌دهنده یک هرم کمینه دوتایی با ۱۲ گره است که هر گره نشانگر یک عدد صحیح است. تعیین کنید که کدام گره(ها) دارای ویژگی‌های زیر می‌توانند باشند.

الف) کوچکترین عدد صحیح

ب) سومین عدد صحیح کوچک

ج) بزرگترین عدد صحیح

د) دومین عدد صحیح بزرگ