ساختمان داده‌ها و الگوریتم‌ها

نیم‌سال دوم ۹۷ــ۹۸

دانشکده‌ی مهندسی کامپیوتر

پاسخنامه ی آزمون تئوری اول ۲۰ اسفند ۹۷

سوالات کوتاه پاسخ

حل.

الف)

ــ نادرست، باید از الگوریتمی استفاده کنیم که پایدار باشد.

ــ درست

ــ نادرست، درجا بودن به معنای استفاده $O(1)$ از حافظه می‌باشد و ربطی به پایدار بودن که ویژگی مورد نظر سوال است ندارد.

ب)

ــ جواب درست Insertion Sort می‌باشد. زیرا در این حالات مرتبه زمانی آن خطی می‌شود و بهترین گزینه است. مرتبه زمانی: $O(5n)$

ج)

در هر دو بخش هزینه سرشکن هر عمل از $O(1)$ است.

د) تابع $f1$ از مرتبه زمانی $O(N^2)$ و تابع $f2$ از مرتبه زمانی $O(NlogN)$ می‌باشد.

$\triangleright$

مسئله‌ی ۱. تحلیل زمانی

حل.

الف)

از روش تغییر متغیر استفاده می‌کنیم؛ قرار می‌دهیم $n = 2^m$. خواهیم داشت:

$$T(2^m) = T(2^{\frac{m}{2}}) + 2^m$$

تابع $T(2^m)$ را به $F(m)$ تغییر نام می‌دهیم؛ خواهیم داشت:

$$F(m) = F(\frac{m}{2}) + 2^m$$

که با استفاده از قضیه‌ی اصلی به راحتی مشخص می‌شود که این تابع از $\theta(2^m)$ و به بیان دیگر $\theta(n)$ است.

ب)

از ظاهر این تابع این‌طور بر می‌آید که احتمالا عدد ثابت ۱۸ تاثیری در مرتبه‌ی زمانی نداشته

باشد و مرتبه زمانی تابع شبیه تابع merge sort باشد. پس سعی می‌کنیم با استقرا ثابت کنیم که این تابع از $\theta(n\log n)$ است؛ برای این کار باید به صورت جداگانه ثابت کنیم که از $O(n\log n)$ و $\Omega(n\log n)$ است:

اثبات O: به صورت استقرایی فرض می‌کنیم برای kهای کوچکتر از n، $T(n) \leqslant cn\log n$ است؛ حال داریم:

$$T(n) = ۲T(\frac{n}{۲} + ۱۸) + n$$

$$\Rightarrow T(n) \leqslant ۲(c(\frac{n}{۲} + ۱۸)\log(\frac{n}{۲} + ۱۸)) + n$$

$$\Rightarrow T(n) \leqslant cn\log(\frac{n}{۲} + ۱۸)) + ۱۸\log(\frac{n}{۲} + ۱۸)) + n$$

برای nهای به قدر کافی بزرگ می‌توانیم فرض کنیم که $\frac{۳n}{۴} \leqslant ۱۸ + \frac{n}{۲}$ ؛ بنابراین خواهیم داشت:

$$T(n) \leqslant cn\log(\frac{۳n}{۴}) + ۱۸\log(\frac{۳n}{۴}) + n$$

$$\Rightarrow T(n) \leqslant cn\log n - cn\log\frac{۴}{۳} + ۱۸\log(\frac{۳n}{۴}) + n$$

باید c را طوری پیدا کنیم که $cn\log\frac{۴}{۳} \geqslant ۱۸\log(\frac{۳n}{۴}) + n$ شود و آن‌گاه حکم اثبات می‌شود؛ با کمی دقت می‌توان دریافت که اگر c را $\frac{۲}{\log\frac{۴}{۳}}$ را قرار دهیم سمت چپ برابر با $۲n$ می‌شود که برای nهای به قدر کافی بزرگ از سمت راست بزرگ‌تر خواهد بود.

اثبات Ω: به ازای تمام nها، مقادیر این تابع از تابع زیر بزرگتر یا مساوی است:

$$T(n) = ۲T(\frac{n}{۲}) + n \quad (n > ۱۰۰)$$

$$T(n) = ۱ \quad (n \leqslant ۱۰۰)$$

که هزینه‌ی این تابع مانند merge sort، $\theta(n\log n)$ است؛ پس تابع داده‌شده در صورت سوال از $\Omega(n\log n)$ است.

$\triangleright$

مسئله‌ی ۲. نقطه ها

حل.

پیش‌پردازش

نقاط را بر حسب مختصات x یا yشان مرتب می‌کنیم. در اینجا مختصات x را در نظر می‌گیریم.

زمان اجرا: $O(nlgn)$

تقسیم
در هر مرحله نقاط را به دو زیرمجموعه‌ی A و B تقسیم می‌کنیم. مجموعه‌ی A نقاط با مختصات x کمتر و مجموعه‌ی B نقاط با مختصات x بیشتر است.
زمان اجرا: $O(1)$

حل
برای هر مجموعه، نقاط روشن را به صورت بازگشتی به دست می‌آوریم.
زمان اجرا: $T(\frac{n}{2})$

ادغام
به عنوان مرحله‌ی آخر می‌خواهیم نقاط روشن زیرمجموعه‌ی A را با B ادغام کنیم. نقاط روشن مجموعه‌ی B همگی نقاط روشن مجموعه‌ی $A \cup B$ نیز هستند (چرا؟). بدین ترتیب، اولین نقطه‌ی روشن مجموعه‌ی B، یعنی نقطه با کمترین مختصات x بین نقاط این مجموعه را انتخاب می‌کنیم. این نقطه را b^* می‌نامیم.
نقاط روشن مجموعه‌ی A را در نظر بگیرید. مختصات y این نقاط به صورت نزولی قرار گرفته است (چرا؟). از انتهای مجموعه‌ی نقاط روشن A شروع می‌کنیم. تا جایی که مختصات y این نقاط از مختصات y نقطه‌ی b^* کمتر باشد پیش می‌رویم و نقاط را حذف می‌کنیم. نقاطی که در نهایت باقی می‌مانند، نقاط روشن مجموعه‌ی $A \cup B$ خواهند بود.
زمان اجرا. $O(n)$

در نهایت زمان اجرا به صورت زیر بدست می‌آید:

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

با استفاده از قضیه‌ی اصلی می‌توان بدست آورد که

$$T(n) \in O(nlgn)$$

$\triangleright$

مسئله‌ی ۳. دکتر شریفی و تعدد امتحانات

حل.

ایده‌ی این سوال مشابه سوال مطرح‌شده در تمرین تئوری است که یک استک در نظر می‌گیریم و هر پرانتز باز را در آن push می‌کنیم و به ازای هر پرانتز بسته، از بالای استک pop می‌کنیم که عنصر حذف‌شده درواقع پرانتز باز نظیر آن پرانتز بسته است.

دقت کنید که اگر در لحظه‌ای پرانتز بسته دیدیم ولی استک خالی بود یعنی آن پرانتز بسته نظیر ندارد و آنگاه هیچ زیررشته‌ی معتبری شامل آن وجود نخواهد داشت!

راه‌حل سوال به این صورت است که هربار که پرانتز باز وارد استک می‌کنیم، اندیسش را هم

کنارش نگه می‌داریم؛ حال هروقت که پرانتز بسته دیدیم و از استک pop کردیم، زیررشته‌ی با شروع از بعد از عنصر جدیداً بالای استک (بعد از pop) تا اندیس فعلی یک زیررشته‌ی معتبر است (چون همه‌چیز دارد به خوبی پیش میرود و پرانتز بسته‌ها نظیر خود را پیدا میکنند). در ابتدا هم یک اندیس صفر وارد استک می‌کنیم (اشاره‌گر به قبل از شروع رشته) برای مثال در رشته‌ی ()()()، هر بار که pop میشود، عنصر اندیس صفر در بالای استک است و به ما میگوید که از اندیس ۱ تا اندیس فعلی یک زیررشته‌ی معتبر داریم که در نهایت به این ختم میشود که کل رشته یک زیررشته‌ی معتبر است. اگر به مرحله‌ای رسیدیم که استک خالی شد، همان اندیس فعلی را به استک اضافه میکنیم یعنی یک پرانتز بسته دیدیم که نظیر ندارد و بقیه‌ی کار باید از اینجا به بعد رشته انجام شود.

▷

مسئله‌ی ۴. مریخ

حل. این مساله به سادگی با یک ساختار صف حل می‌شود. راه حل غیر بهینه: عمل جذب و فارغ التحصیلی دانشجو را با عملیات درج در انتها و حذف از ابتدای صف پیاده‌سازی کنید. برای تابع افزودن سواد، کافی است عدد x را به همه‌ی اعداد داخل صف اضافه کنید. با این کار، دو تابع اول زمان ثابت و تابع سوم زمان حداکثر خطی خواهد داشت.

راه حل بهینه: می‌توان علاوه بر صف یک متغیر y در نظر گرفت که معادل یک سطح سواد پایه برای همه‌ی دانشجویان داخل صف است. مقدار اولیه‌ی y را برابر صفر می‌گذاریم. تا وقتی از تابع افزایش سواد استفاده نکنیم، سطح سواد هر دانشجو معادل همان عددی است که به ازای او در صف درج شده است. اما وقتی تابع $Add(x)$ فراخوانی می‌شود، به جای آن که به تک تک عناصر داخل صف مقدار x را اضافه کنیم، به متغیر y این مقدار را اضافه می‌کنیم. بدین ترتیب، هنگام فارغ التحصیلی هر دانشجو، عدد ابتدای صف را خارج و آن را با y جمع می‌زنیم و به عنوان خروجی برمی‌گردانیم.

اما باید توجه کرد که اگر یک دانشجو با سطح سواد x هنگامی جذب شود که متغیر y مقداری غیر صفر دارد، آنگاه می‌بایست به جای آن که خود x را به انتهای صف اضافه کنیم، $x - y$ را به صف اضافه کنیم.

▷