دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

11/18/2020

Homework 3

Sort & Ch9

ALGORITHM DESIGN

۱) نشان دهید زمان اجرای الگوریتم مرتب سازی سریع هنگامی که آرایه ورودی شامل اجزای متمایزی باشد که بصورت نزولی مرتب شده‌اند، برابر با $\Theta(\mathrm{n}^2)$ خواهد بود.

اگر ورودی ما اکیدا نزولی باشد، یعنی pivot انتخاب شده از همه‌ی عناصر دیگر اکیدا کمتر است. این به این معناست که باید $\Theta(\mathrm{n})$ صرف این کنیم که افراز را انجام دهیم. حالا باید همان مساله را برای دو بخش به اندازه‌های n-1 و ۰ تکرار کنیم. با صرف نظر از بخش خالی، این یعنی هر مرحله به اندازه $\Theta(\mathrm{n})$ و $T(n-1)$ باید زمان بگذاریم.

بنابراین پاسخ ما، پاسخ این رابطه بازگشتی است:

$$T(n) = T(n-1) + \Theta(\mathrm{n}) = T(n-1) + d\,n$$
$$\exists\, n0,\ \ c \neq 0, d \neq 0\ \ \forall n > n0 \quad T(n) \le cn^2\ \Rightarrow T(n) \le c(n-1)^2 + d\,n$$

$$\Rightarrow T(n) \le cn^2 - 2cn + 1 + dn \xrightarrow{(d-2c)<0,\quad n0>\frac{1}{2c-d}} T(n) \le cn^2$$

$$similarly\ we\ can\ prove\ T(n) \ge cn^2$$

$$\Rightarrow \boldsymbol{T(n) = \Theta(\mathrm{n}^2)}$$

۲) توضیح دهید که چگونه می‌توان n عدد صحیح در بازه‌ی 0 تا $\mathrm{n}^2 - 1$ را در زمان $\mathrm{O(n)}$ مرتب کرد.

برای حل این سوال از الگوریتم Radix sort استفاده میکنیم. میدانیم اگر n عدد d رقمی برای مرتب کردن داشته باشیم که هر رقم از این اعداد k حالت(مقدار) متفاوت بتواند داشته باشد، این الگوریتم مساله مرتب سازی را با $\Theta(\mathrm{d(n+k)})$ برایمان حل خواهد کرد.
برای این کار ابتدا همه اعداد را در مبنای n مینویسیم. این کار با توجه به فرض مساله مبنی بر عدد صحیح بودن اعداد ممکن است. میدانیم اگر عددی در مبنایی از عدد دیگری کمتر باشد، در مبناهای دیگر هم از آن کمتر است. همچنین بابت نمایش اعداد منفی هم نگرانی‌ای نداریم چون میدانیم اعداد همگی از صفر به بعد هستند.(← ادامه)

اعدادی که از $n^2 - 1$ کمتر باشند، در نمایش در مبنای n میتوانند حداکثر $\log_n(n^2-1)$ رقم داشته باشند. یعنی این اعداد در مبنای n حداکثر ۲ رقمی هستند. پس $d = 2$.
از طرفی هر عدد در مبنای n میتواند از ۰ تا n-1 مقدار بگیرد یعنی در مجموع n مقدار متفاوت دارد و این یعنی $k = n$.

$$\Rightarrow \Theta(d(n+k)) = \Theta(2(n+n)) = \Theta(4n) = \boldsymbol{\Theta(n)}$$

(همچنین میتوان گفت برای مرتب کردن هر رقم اگر از count sort استفاده کنیم واضح است که در زمان O(n) این کار انجام خواهد شد.)

۳) در الگوریتم SELECT، ورودی ها در ابتدا به گروه‌های ۵ تایی تقسیم می‌شوند. آیا درصورتی که ورودی‌ها به دسته‌های ۷ تایی تقسیم شوند الگوریتم همچنان در زمان خطی کار می‌کند؟

بله، چون با دسته ۷تایی کردن هم همچنان میدانیم میانه‌ی میانه در $\frac{1}{2}$ دسته‌های ۷ تایی، از حداقل ۴ تا از عناصر آن کمتر است. پس از بین $\left\lceil \frac{n}{7} \right\rceil$ تا دسته، به اندازه نصف آنها یعنی در $\left\lceil \frac{n}{14} \right\rceil$ دسته ما حداقل ۴ عنصر کوچکتر داریم. پس میتوان گفت از حداقل $\frac{4n}{14}$ اعضا بزرگتر است.(چون حداقل داریم میتوانیم براکت سقف را برداریم) برای *بقیه اعضا* نمیتوانیم از راه بازگشتی کمک بگیریم. اگر در این تعداد ثابت کنیم همچنان T(n) یک رابطه خطیست، حکم را ثابت کرده‌ایم.

$$T(n) \le T\left(\frac{n}{7}\right) + T\left(\frac{10n}{14}\right) + O(n)$$

$$T(n) < cn \quad for \quad n < n0 \quad \Rightarrow for \;\; n > n0\colon \; T(n) \le T\left(\frac{n}{7}\right) + T\left(\frac{10n}{14}\right) + O(n)$$

$$\le cn\left(\frac{1}{7} + \frac{10}{14}\right) + O(n) \le \frac{12c}{14}n + O(n)$$

اگر $O(n)$ را برابر با d.n فرض کنیم، پس میخواهیم رابطه $\frac{12c}{14}n + O(n) = \frac{12c}{14}n + dn \le n$ برقرار شود. پس اگر شرط زیر را داشته باشیم، پاسخ خطی خواهد بود.

$$d \le \frac{2c}{14} \Rightarrow d \le \frac{c}{7}$$

۴) فرض کنید می‌خواهیم n عدد صحیح را که شامل log n عدد متمایز است مرتب کنیم. الگوریتمی با پیچیدگی زمانی $O(n \log \log n)$ برای مرتب کردن مقایسه‌ای این اعداد طراحی کنید.

با استفاده از یک درخت قرمز-سیاه (rb) که هر راس آن شامل دو متغیر مقدار (value) و تعداد (count) و مبنای مقایسه‌ی آن، متغیر مقدار است، اعداد را مرتب می‌کنیم. از آنجایی که عناصر تکراری را به این درخت اضافه نمیکنیم، پس اندازه درخت rb حداکثر به اندازه log n است.
به این صورت عمل میکنیم که با شروع از اولین عدد آرایه(فرض کنیم مقدار آن برابر با x باشد)، راسی را با مقدار x در درخت جست‌وجو می‌کنیم. (جستجو در یک درخت rb به اندازه log n، $O(\log \log n)$ زمان میبرد.)

- اگر راسی با این مقدار وجود داشت، متغیر تعداد آن را یک واحد افزایش داده و به سراغ عدد بعدی آرایه می‌رویم. (این تغییر به اندازه پیچیدگی زمانی عدد ثابت زمان میبرد.)
- اگر راسی با این مقدار وجود نداشت، راس با مقدار x و تعداد ۱ را ساخته و به درخت اضافه می‌کنیم. (اضافه کردن در یک درخت rb به اندازه log n، $O(\log \log n)$ زمان میبرد.)

همین روند را تا آخرین عدد آرایه تکرار می‌کنیم.
درنهایت با پیمایش میان‌وندی (in order) درخت، اعداد را در آراسته‌ی اصلی جایگزین می‌کنیم.

در روش فوق، چون تعداد اعداد متمایز آرایه‌ی اولیه، log n و مقدار راس‌های درخت متمایز است، تعداد راس‌های درخت از log n تجاوز نمی‌کند و چون جست‌وجو و اضافه کردن راس جدید در یک درخت قرمز-سیاه n راسی، به اندازه‌ی $O(\log n)$ زمان می‌برد، بنابراین در این درخت، برای هر عدد به زمانی از $O(\log \log n)$ و درنتیجه برای n عدد به $O(n \log \log n)$ زمان نیاز است.
همچنین در پیمایش میان‌وندی درخت و قرار دادن هر مقدار به تعداد ذخیره شده در آرایه، هر راس یک بار دیده شده و به اندازه‌ی تعداد ذخیره شده در آن، در آرایه‌ی اصلی کپی می‌شود. بنابراین این قسمت هم به اندازه‌ی $O(n)$ زمان می‌برد.
پس پیچیدگی زمانی الگوریتم، برابر با $O(n \log \log n) + O(n) = O(n \log \log n)$ است.

۵) فرض کنید که $L = (x_1, x_2,...,x_n)$ فهرستی از n عدد حقیقی باشد. ترتیب اعداد مشخص نیست. می‌گوییم عدد x در فهرست L «غالب» است اگر اکیدا بیشتر از $1/3$ اعداد در فهرست L مساوی x باشند. این (لینک) الگوریتم قطعی مبتنی بر مقایسه زیر را برای پیدا کردن اعداد «غالب» در نظر بگیرید:

نکته: الگوریتم ارائه شده در لینک، اگر لیست دارای حداقل یک عدد اغلب باشد، یکی از آن اعداد اغلب را، و اگر نه -۱ باز میگرداند و قادر به یافتن بیش از ۱ عدد اغلب نیست.

الف) درستی الگوریتم و کد آن را ثابت کنید. (می‌توانید از loop invariantها کمک بگیرید)

اگر این الگوریتم درست باشد، یعنی پس از اجرای حلقه‌ی اول، باید یکی از دو حالت زیر پیش بیاید:

۱- در این لیست عنصر اغلبی نداشته ایم، پس هیچ یک از دو عنصری که انتخاب کرده ایم اغلب نیستند.
۲- در این لیست حداقل یک عنصر اغلب داشته ایم، و حداقل یکی از دو عنصری که انتخاب کرده ایم اغلب است.

در حالت اول متغیرهای first و second هر مقداری هم که داشته باشند، در حلقه دوم مقادیر آنها به تعدادی بیش از ۳/۱ اعداد نیست و در هر صورت تابع -۱ برمیگرداند.

پس با حل کردن حالت اول، فرض میکنیم حتما حداقل یک عنصر اغلب داریم. ثابت میکنیم ممکن نیست هیچ کدام از متغیرهای first و second بعد از اتمام حلقه اول، اغلب نباشند.

در حلقه اول میبینیم که دو عنصر first و second با کمک شمارنده‌هایشان عناصر با ماکسیمم تعداد تا آن لحظه را در خود نگه میدارند. پس اگر عنصری باشد که اغلب باشد؛ اگر این عنصر یکی از first و second باشد که پس ما حداقل یک اغلب پدا کرده ایم و اگر نه، امکان ندارد در بین دو عنصری که حداقل یکی از آنها از آن عنصر اغلب بیشتر یا مساوی تکرار شده است، عنصر اغلبی وجود نداشته باشد. به این ترتیب در صورت وجود تعدادی عنصر اغلب، ما حتما یکی از آنها را پیدا خواهیم کرد.

ب) ثابت کنید که این الگوریتم از $\theta(n)$ مقایسه انجام می‌دهد.

تابع appearsNBy3 فقط یک بار اجرا میشود و در هر اجرایش هم ۲ حلقه n تایی و تعدادی محاسبات با پیچیدگی عدد ثابت انجام میدهد. در حلقه اول تعدادی if/else داریم. چون این if/elseها همگی else هم دارد و در همه بخش ها دقیقا یک مقایسه وجود دارد، در حلقه اول دقیقا n مقایسه خواهیم داشت. در انتها هم دو مقایسه برای بررسی اغلب بودن دو عنصری که انتخاب کردیم داریم. این یعنی n+۲ مقایسه در کل داشته ایم. بدیهیست که این تعداد مقایسه از $\theta(n)$ است.

- مهلت ارسال تمرین ساعت ۲۳:۵۵ روز **جمعه ۷** آذرماه می‌باشد.
- سوالات خود را می‌توانید از طریق ایمیل از تدریسیاران بپرسید.
  - parsa.abdollahi.pa@gmail.com
  - faridi.mina.1@gmail.com
- ارائه پاسخ تمرین به دو روش ممکن است:
  1. تایپ داخل همین فایل و ارائه فایل Pdf
  2. چاپ تمرین و پاسخ دهی به صورت دستنویس خوانا
- ارائه تمرین به روش اول شامل ۱۰٪ نمره امتیازی می‌گردد.
- فایل پاسخ تمرین را تنها با قالب **HW3-9531747.pdf** در مودل بارگزاری کنید.
- فایل زیپ ارسال **نکنید**.