ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان 99]

جلسه ۱۴: بهترین مرتبه زمانی مرتب‌سازی

نگارنده: فاطمه سادات موسوی

در جلسه‌ی گذشته درباره‌ی مرتبه‌ی زمانی انواع روش‌های مرتب‌سازی صحبت کردیم:

۱ـ insertion sort: در بدترین حالت از مرتبه‌ی زمانی $O(n^2)$ و در بهترین حالت از مرتبه‌ی زمانی $O(n)$ است.

۲ـ bubble sort: در بدترین و بهترین حالت از مرتبه‌ی زمانی $O(n^2)$ است.

۳ـ selection sort: در بدترین و بهترین حالت از مرتبه‌ی زمانی $O(n^2)$ است.

۴ـ merge sort: در بدترین و بهترین حالت از مرتبه‌ی زمانی $O(n \log n)$ است.

۵ـ quick sort: در بدترین حالت از مرتبه‌ی زمانی $O(n^2)$ و در حالت متوسط از مرتبه‌ی زمانی $O(n \log n)$ است.

۶ـ radix sort , count sort:هردو در مرتبه زمانی خطی نسبت به ورودی قابل اجرا هستند .

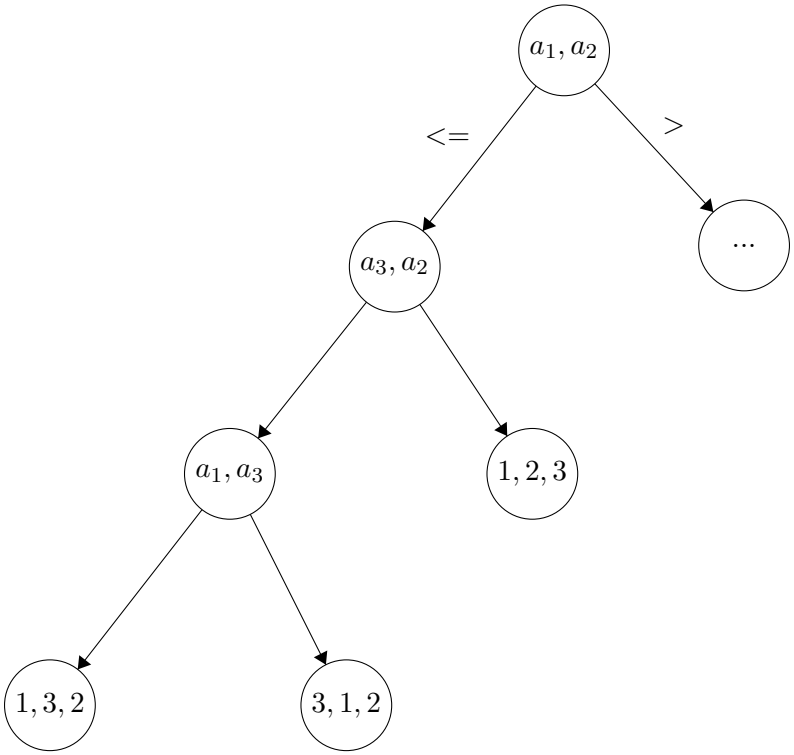
در این جلسه قصد داریم بهترین مرتبه زمانی ممکن برای مرتبسازی را بررسی کنیم.

۱ درخت تصمیم (Decision Tree) و محاسبه بهترین مرتبه زمانی

برای بررسی بهترین مرتبه زمانی از ایده‌ای به نام درخت تصمیم استفاده می‌کنیم. در هر نود این درخت یک عملیات مقایسه انجام می‌شود که در آن دو عضو از ورودی $a_1a_2...a_n$ با یکدیگر مقایسه می‌شوند.

برای مثال فرض کنید برای $n = 3$ ورودی به فرم $a_1a_2a_3$ داریم. در ابتدا عناصر a_1 و a_2 مقایسه می‌شوند. در صورتی که a_1 کوچکتر یا مساوی a_2 باشد وارد شاخه سمت چپ و در غیر این صورت وارد شاخه‌ی سمت راست می‌شویم.

فرض کنید وارد شاخه سمت چپ شده‌ایم. یعنی $a_1 \leq a_2$. حال a_3 و a_2 را مقایسه می‌کنیم. اگر $a_2 < a_3$ عناصر به صورت $a_1a_2a_3$ مرتب‌سازی می‌شوند . در غیر این صورت باید یک مقایسه هم بین عنصر a_1 و a_3 انجام دهیم. در صورتی که $a_1 \leq a_3$ عناصر به شکل $a_1a_3a_2$و در غیر این صورت عناصر به شکل $a_3a_1a_2$ از کوچک به بزرگ مرتب می‌شوند.



تعداد برگ‌های این درخت برابر با تعداد کل جایگشت های n عنصر یعنی $n!$ است. برای بررسی بدترین حالت باید عمق درخت تصمیم را محاسبه کنیم. در سطح اول درخت یک مقایسه، در سطح بعدی دو مقایسه، سپس چهار مقایسه و به همین ترتیب تعداد مقایسه‌هایی که تا عمق H داریم برابر با $2^{H+1} - 1$ می‌شود.

$$2^{H+1} - 1 = n! \;\; \rightarrow \;\; H \approx \log n! \;\; \rightarrow \;\; O(n \log n)$$

پس مرتبه زمانی این الگوریتم مانند merge sort برابر با $O(n \log n)$ است و الگویتم‌های مرتب‌سازی نمی‌توانند مرتبه زمانی بهتر از $O(n \log n)$ داشته باشند.

ممکن است این سوال ایجاد شود که با وجود اینکه بهترین مرتبه زمانی برای مرتب‌سازی را $O(n \log n)$ به دست آوردیم، چطور ممکن است الگوریتم‌هایی مانند count sort , radix sort با مرتبه زمانی $O(n)$ انجام شوند؟
پاسخ به این سوال این است که در حالت کلی این دو الگوریتم نیز مرتبه زمانی مشابهی خواهند داشت و در واقع حالت خاصی از اجرای آن ها مرتبه زمانی $O(n)$ دارد. در مرتب‌سازی به روش count sort مرتب سازی با مرتبه $O(n \log M)$انجام می‌شود و اگر عدد ماکسیمم را constant در نظر بگیریم به مرتبه زمانی $O(n)$ می‌رسیم. در این الگوریتم نسبت به اندازه ورودی باید $O(n^2)$ تا کار انجام شود. در radix sort نیز به صورت مشابه مبنا را constant فرض کردیم و دلیل خطی شدن الگوریتم نیز همین بوده‌است.

۲ overview مطالب جلسه آینده

صف اولویت :در جلسه آینده به معرفی داده ساختار صف اولویت (priority Queue) می‌پردازیم. عمل $push$ و pop را برای آن تعریف می‌کنیم. در عمل $push$ یک عنصر وارد صف شده و در عمل pop عنصر مینیمم حذف می‌شود. برای مرتب‌سازی با استفاده از صف اولویت همه‌ی عناصر را یکبار در آن $push$ کرده و سپس pop می‌کنیم.
برای پیاده‌سازی صف اولویت داده ساختاری به نام heap را معرفی خواهیم کرد که عمل $push$ و pop را با مرتبه زمانی $O(logn)$ انجام می‌دهد. پس در نهایت الگوریتمی با مرتبه زمانی $O(n \log M)$ برای مرتب سازی به دست می‌آید.

درخت : سپس داده ساختار درخت را معرفی می‌کنیم. در این داده ساختار حذف و درج عناصر در مرتبه زمانی $O(logn)$ انجام می‌شود و می‌تواند عناصر را به ترتیب از کوچک به بزرگ در مرتبه زمانی $O(n)$ به ما بدهد. اما به هرحال باید در ابتدا درخت را بسازیم پس در نهایت همان مرتبه زمانی $O(nlogn)$ خواهد داشت.

درخت و صف اولویت داده ساختارهای dynamic هستند. یعنی می‌توان به آن‌ها داده جدید اضافه کرد.

تابع هش: یک مقدار را می‌گیرد و یک اندیس بر می‌گرداند.

یک مسأله: فرض کنید n نقطه در صفحه داریم. می‌خواهیم یک چند ضلعی بیابیم که از تمام این نقاط عبور کند. الگوریتمی وجود دارد که این کار را در مرتبه زمانی $O(nlogn)$ انجام می‌دهد. با استفاده از الگوریتم sort می‌توان نشان داد این کار را با مرتبه زمانی کمتر از $O(nlogn)$ نمیتوان انجام داد. (در واقع اگر میشد این کار را با مرتبه زمانی کمتر از $O(nlogn)$ انجام داد، مرتبه زمانی sort هم می‌توانست کمتر از $O(nlogn)$ باشد.)