جزوه جلسه یازدهم داده ساختارها و الگوریتم

۹ آبان ۱۴۰۰

فهرست مطالب

۱ مرتب سازی سریع یا Quick Sort

این شیوه مرتب سازی در سال ۱۹۶۲ معرفی شد و هم اکنون، در زبان های Java و C++ به عنوان الگوریتم مرتب سازی پیش فرض استفاده میشود.
حافظه مورد نیاز برای این مرتب سازی، به اندازه $O(1)$ برای ذخیره سازی آرایه اصلی و حافظه‌ای از مرتبه $logn$ برای اجرای الگوریتم به شیوه بازگشتی در stack میباشد.
این مرتب سازی اگر به شیوه درست پیاده سازی شود، به اندازه ۲ برابر سریع تر از مرتب سازی ادغامی (Merge Sort) است. (لازم به ذکر است مرتبه زمانی این الگوریتم برابر $O(nlogn)$ میباشد.)

۱.۱ مقایسه Quick Sort با Merge Sort

مرتب سازی سریع نیز مانند مرتب سازی ادغامی، بر اساس الگوریتم تقسیم و حل (Divide and Conquer) کار میکند. در مرتب سازی سریع، تمرکز اصلی روی مرحله تقسیم و در مرتب سازی ادغامی روی ادغام است. مرحله حل نیز در هر دو به دلیل کوچک شدن زیر مسئله ها، بسیار بدیهی و ساده است.
توجه شود چون میتوان Quick Sort را به صورت in-place پیاده سازی کرد، پس نیازی به مرحله ادغام در این متذ نیست و در هر مرحله از الگوریتم، آرایه مرتب میشود و نیازی به ادغام زیر ارایه ها نیست.

۱.۱.۱ مقایسه کوچک بین Quick Sort و Tim Sort!

مرتب سازی Tim سریعتر از مرتب سازی سریع است (ضریب کوچکتری از $nlogn$ میباشد) و همچنین برای آرایه هایی که تقریبا-مرتب هستند، استفاده از Tim Sort بسیار بهتر است؛ زیرا این متد مرتب سازی به initial state آرایه بستگی دارد.

۲.۱ مراحل مرتب سازی Quick Sort

۰. انتخاب عنصر لولا (pivot) به نام x از آرایه ورودی
انتخاب pivot میتواند خروجی یک الگوریتم خاص باشد یا یک عنصر تصادفی از آرایه (مانند عضو اول یا آخر یا وسط)

۱. تقسیم آرایه ورودی به سه قسمت L: اعضای کوچکتر از x، E: اعضایی که برابر با x هستند، G: عناصری که از x بزرگتر هستند.

۲. مرتب سازی L و G به صورت بازگشتی
مرحله تقسیم تا جایی پیش میرود که L و G حداکثر یک عضو داشته باشند که در این

حالت مسئله حل شده میشود.

۳. ادغام
همانگونه که ذکر شد، این مرحله نیاز نیست.
مرحله یک، با یک پیمایش ارایه انجام میشود و ارایه به سه زیر آرایه تقسیم میشود (این قسمت میتواند به صورت in-place نیز پیاده سازی شود و حافظه اضافی نگیرد)
اما نکته اصلی این الگوریتم، انتخاب عنصر لولا میباشد. بسته به اینکه این عنصر چگونه انتخاب مبشود، نسخه های مختاف Quick Sort معرفی میشود:

۳.۱ نسخه های مختلف مرتب سازی سریع

۱.۳.۱ مرتب سازی سریع پایه‌ای

عنصر اول آرایه به عنوان pivot درنظر گرفته میشود. در بدترین حالت (یک آرایه مرتب شده به صورت صعودی یا نزولی) زمان اجرا به $O(n^2)$ میرسد.
برای بهبود عملکرد این حالت، میتوان از درهم سازی آرایه در ابتدا و قبل از انتخاب pivot بهره برد. با انجام این کار، دیگر نمیتوان یک حالت را به عنوان بدترین حالت درنظر گرفت و اردر زمانی بدست آمده برای حالت میانگین یا Average Case میباشد.

۲.۳.۱ مرتب سازی سریع با انتخاب هوشمندانه لولا

در این نسخه، با یک الگوریتم خطی ($\Theta(n)$)، میانه عناصر در ارایه پیدا میشود و به عنوان لولا انتخاب میشود.
در بدترین حالت، مرتبه زمانی برابر میشود با:
$T(n) = \Theta(n) + 2T(n/2)$
که طبق Master Theory داریم:
$T(n) = \Theta(nlogn)$
اما نکته قابل توجه، ضریب بالا در این شیوه پیاده سازی است. ضریب $nlogn$ در این حالت ۲ برابر مرتب سازی ادغامی است و نتیجتا سرعت آن نیز نصف سرعت Merge Sort میشود.
الگوریتمی که برای پیدا کردن میانه استفاده میشود، میانه میانه ها یا Median of Medians نام دارد.

۳.۳.۱ مراحل الگوریتم میانه میانه ها

این الگوریتم به صورت کلی پیاده سازی میشود و میتواند برای هر k، kامین عضو آرایه را برگرداند.

۱. آرایه ورودی(A) را به ستون های حداکثر ۵ عضوی افراز میکنیم. (زمان: $\Theta(1)$)

۲. هر ستون ۵تایی را مرتب میکنیم (زمان: $\Theta(n)$)

۳. میانه سطر وسط ستون هارا با همین الگوریتم پیدا میکنیم و آنرا x مینامیم (زمان: $T(n/5)$)
عنصر x پیدا شده حتما از سی درصد عناصر بزرگتر و از سی درصد عناصر کوچکتر است.

۴. عناصر A را به سه مجموعه L، E ، G تقسیم میکنیم (زمان: $\Theta(n)$)

۵. بر اساس مقدار k، در یکی از سه مجموعه به صورت بازگشتی به دنبال عنصر مورد نظر میگردیم (زمان: $T(7n/10)$)
بدیهی است در هر مرحله از مسیر بازگشتی، k مقدار جدیدی میگیرد؛ چون اندازه زیر ارایه ها تغییر میکند.
همچنین ضریب ۷/۱۰ در مرتبه زمانی به این دلیل است که حداکثر ۷۰ درصد اعضا میتوانند در یک مجموعه که در آن جست و جو را انجام میدهیم قرار داشته باشند.

همانگونه که ذکر شد، حداقل سی درصد و حداکثر هفتاد درصد اعضا در L و G قرار دارند. با محاسبه دقیق میتوان به این نتیجه رسید که حداکثر تفاضل تعداد اعضای L و G برابر ۶ میباشد.

۴.۳.۱ زمان اجرای الگوریتم

$$T(n) = \Theta(n) + T(n/5) + T(7n/10)$$

از طرفی داریم:

$$T(7n/10) + T(n/5) = T(9n/10) < T(n)$$

بنابر قضیه اصلی، به دلیل کوچک شده برگ ها نسبت به ریشه، مرتب زمانی کل، برابر مرتبه زمانی ریشه میشود.

$$T(n) = \Theta(n)$$

درنظر داشته باشیم که ضریب n در این حالت بزرگ است.

۵.۳.۱ مرتب سازی سریع تصادفی

قبل از توضیح این بخش، کمی درباره خود الگوریتم های تصادفی صحبت میکنیم

۶.۳.۱ الگوریتم های تصادفی

الگوریتم هایی که یک عدد تصادفی r بین ۱ تا R تولید کرده و بر اساس این مقدار، الگوریتم اجرا میشود. برای انتخاب تعداد بیشتری عدد تصادفی، باید عدد R را بسیار بزرگ در نظر گرفت.
ترجیح بر این است که از اعداد و الگوریتم ها تصادفی زیاد استفاده نکنیم.
بنابراین اجراهای مختلف الگوریتم روی ورودی های یکسان میتواند متفاوت باشد. این تفاوت یا در تعداد مراحل اجرا (کند یا سریع بودن الگوریتم) یا در خروجی (خروجی های صحیح یا بد و یا گاها اشتباه) نمود پیدا میکند. (باید احتمال تولید خروجی اشتباه در الگوریتم تصادفی بسیار کم باشد).
الگوریتم های تصادفی به سه دسته تقسیم میشوند:

۱. الگوریتم های مونت کارلو یا احتمالا درست: درستی خروجی در این الگوریتم ها احتمالی است اما همیشه سریع هستند؛ به دیگر بیان، زمان اجرا کم است ولی خروجی نامعلوم است.
مانند الگوریتم بررسی اول بودن یک عدد، بررسی درست بودن ضرب ماتریس ها، شبیه سازی فرایند های فیزیکی، 3D Rendering

۲. الگوریتم های لاس وگاس یا احتمالا سریع: برخلاف الگوریتم های مونت کارلو، خروجی الگوریتم همیشه درست است ولی سریع بودن آن احتمالی است و معلوم نیست.
مانند مرتب سازی سریع تصادفی

۳. الگوریتم های اتلانتیک یا احتمالا درست و احتمالا صحیح: در این الگوریتم ها درستی و سرعت اجرای الگوریتم هردو احتمالی و نامعلوم هستند