

به نام خدا

الگوریتم Timsort

مهدی حقوردی



فهرست مطالب

مقدمه و معرفی

تاریخچه

چرا تیمسورت؟

مقدمه و معرفی

- در دنیای علوم کامپیوتر، مرتب‌سازی یک عملیات اساسی با کاربردهای بی‌شمار است.
- در میان انبوهی از الگوریتم‌های مرتب‌سازی، یکی از الگوریتم‌ها به دلیل کارایی، تطبیق‌پذیری و طراحی زیبا متمایز شده است: الگوریتم تیم‌سورت¹.
- این الگوریتم که توسط تیم پیترز² برای زبان برنامه نویسی پایتون³ توسعه یافته است، به سنگ بنای پیاده‌سازی مرتب‌سازی در زبان‌ها و محیط‌های مختلف برنامه‌نویسی تبدیل شده است.
- ترکیب منحصر به فرد مرتب‌سازی ادغامی⁴ و مرتب‌سازی درجی⁵ به همراه بهینه‌سازی‌های مخصوص روی هر الگوریتم و بهینه‌سازی‌های تطبیقی، تیم‌سورت را به یکی از پیچیده‌ترین و کاربردی‌ترین الگوریتم‌های مرتب‌سازی موجود تبدیل کرده است.

¹ Timsort

² Tim Peters

³ Python programming language

⁴ Merge sort

⁵ Insertion sort

تاریخچه

- الگوریتم تیم‌سورت، در سال ۲۰۰۲ توسعه یافت.

- تیم پیترز این الگوریتم را اینگونه توصیف می‌کند:

“A non-recursive adaptive stable natural mergesort / binary insertion sort hybrid algorithm”

- این الگوریتم از Python 2.3 تا حدود بیست سال، الگوریتم استاندارد مرتب‌سازی در پایتون بود و از نسخه‌ی 3.11.1 به دلیل تغییراتی که در سیاست‌های ادغام آن بوجود آمد، الگوریتمی به اسم Powersort بر پایه‌ی تیم‌سورت، جایگزین آن شد.

- الگوریتم تیم‌سورت در 7 Java SE، Android، GNU Octave، V8، Swift و Rust پیاده‌سازی شده است.

- چرا non-recursive؟

چون طبق گفته‌ی تیم پیترز: «به طور خلاصه، روتین اصلی یک بار از سمت چپ تا راست، آرایه را طی، Runها² را شناسایی و هوشمندانه آنها را با هم ادغام می‌کند.»

- چرا adaptive؟

چون این الگوریتم با توجه به طول و ترتیب‌های از قبل موجود در آرایه، و همچنین بر اساس اندازه‌ی Runهای پیدا شده، تصمیماتی می‌گیرد تا از الگوریتم بهتری برای آن موقعیت استفاده کند.

- چرا stable؟

چون این الگوریتم، ترتیب عناصر یکسان در آرایه‌ی اولیه را حفظ می‌کند. برای مثال اگر لیستی از این اسامی داشته باشیم: [peach, straw, apple, spork] و آنرا بخواهیم بر اساس حرف اول کلمات مرتب کنیم، چنین چیزی می‌گیریم: [apple, peach, straw, spork] اگر دقت کنید در لیست اولیه، straw قبل از spork آمده بود و در لیست مرتب شده هم همین ترتیب حفظ شد. به این نگهداری ترتیب پایداری الگوریتم مرتب‌سازی می‌گویند.

² در ادامه مفهوم Run توضیح داده می‌شود.

- چرا hybrid?

چون این الگوریتم از ترکیب دو الگوریتم merge sort و binary insertion sort برای مرتب سازی استفاده می کند.

چرا تیم سورت؟

چرا تیم سورت؟

- پیچیدگی زمانی الگوریتم تیم سورت با الگوریتم های Merge sort، Quick sort و Heap sort برابری می کند و برابر $O(n \lg n)$ است.
- اما این تحلیل کلی یک سری جزئیات راجع به پیچیدگی زمانی الگوریتم را پنهان می کند که آن پیچیدگی یک constant factor اثرگذار در میزان پیچیدگی الگوریتم است. $(c_f \cdot n \lg n)$
- برای مثال در الگوریتم Quick sort انتخاب مقدار left، right و pivot تاثیرگذار است و در n های کوچک سرعت را پایین می آورد.
- در الگوریتم Merge sort هم ما فضایی به اندازه $n + m$ برای ادغام کردن آرایه ها آن هم به صورت بازگشتی و تعداد زیاد نیاز دارد. همچنین این الگوریتم یک الگوریتم بازگشتی است و درخت بازگشتی و یک system stack برای اجرا نیاز دارد.
- بخاطر جابجایی هایی در الگوریتم Heap sort انجام می شود، Locality of Reference در آن نقض شده و پیشبینی های پردازنده برای کش کردن داده ها را تضعیف می کند.

چرا تیم سورت؟

پس اگر بتوانیم این constant factor را کاهش دهیم
می توانیم سرعت بیشتری از $O(n \lg n)$ بگیریم.

مرتب سازی درجی دودویی

- پیچیدگی زمانی insertion sort برابر با $O(n^2)$ است و constant factor آن بسیار بسیار پایین است چون اولاً inplace عمل می‌کند (پس نیازی به فضای اضافه ندارد) و ثانياً فقط بین عناصر آرایه پیمایش انجام می‌دهد (پس Locality of Reference هم در آن بسیار خوب است و پردازنده می‌تواند داده‌ها را کش کند).
- در تحلیل‌های انجام شده روی الگوریتم‌ها، این الگوریتم روی تعداد ورودی ۶۴ و پایین‌تر از الگوریتم‌های دیگر مرتب سازی سریع‌تر عمل می‌کند.
- الگوریتم binary insertion sort بجای جستجوی خطی در آرایه (با پیچیدگی $O(n)$) در آن جستجوی دودویی انجام داده و در زمان لوگاریتمی ($O(\lg n)$) مکان صحیح آیتم را پیدا می‌کند (علت استفاده از این الگوریتم در ادامه روشن خواهد شد).

مرتب سازی درجی دودویی

- با تعویض نوع جستجوی این الگوریتم میزان پیچیدگی آن (حالت مورد انتظار و در بدترین حالت) تغییری نکرده و همان $O(n^2)$ باقی می ماند؛ اما در CPython مقایسه ها (بخاطر ماهیت dynamic typed بودن زبان) نسبت به جابجا کردن آبجکت ها بسیار وحشتناک کندتر هستند.
- جابجا کردن آبجکت ها صرفا کپی کردن ۸ بایت pointer است اما مقایسه ها میتوانند بسیار کند باشند (چون ممکن است چند متد در سطح پایتون را صدا بزنند) و حتی در حالات ساده ممکن است بین ۳ یا ۴ تصمیم گرفته بشود:
 ۱. تایپ عمل وند چپ چیست؟
 ۲. تایپ عمل وند راست چیست؟
 ۳. آیا باید آنها را به یک تایپ مشخص تبدیل کرد؟
 ۴. چه کدی برای مقایسه این دو موجود هست؟ ...

مرتب سازی درجی دودویی

- پس یک مقایسه ساده باعث تعداد بسیار زیادی C-level pointer dereference، عملیات‌های شرطی و صدا زده شدن توابع می‌شود.
- پس اگر ما تعداد مقایسه‌ها کمتر کنیم میتوانیم سرعت مرتب سازی را بیشتر کنیم (که با استفاده از binary insertion sort ما تعداد مقایسه‌ها را کم می‌کنیم).