جزوه جلسه چهارم داده ساختارها و الگوریتم

۶ مهر ۱۴۰۰

فهرست مطالب

۱ مرتبه زمانی الگوریتم Insertion Sort

اگر مرتبه زمانی اجرای الگوریتم مرتب سازی درجی را با $T(n)$ نشان دهیم، میخواهیم ثابت کنیم: $T(n) = \Theta(n^2)$
در ریاضیات برای اثبات برابری a و b میتوانیم نشان دهیم: $a \geq b \quad and \quad b \geq a$
حال معادلا برای اثبات برابری $T(n) = \Theta(n)$ میتوانیم نشان دهیم:
$T(n) = O(n) \quad and \quad T(n) = \Omega(n)$

ابتدا کد مربوط به مرتب سازی درجی:

```
1. for i in ragne (1,n)
2.     while i>0 and A[i] < A[i-1]
3.         A[i], A[i-1] = A[i-1], A[i]
4.         i-=1
```

۱. $T(n) = O(n^2)$
باید نشان دهیم زمان اجرای الگوریتم کمتر یا مساوی n^2 میباشد. خطوط سه و چهار در زمان ثابت $\mathrm{O}(1)$ اجرا میشوند. حلقه خط دوم حداکثر n بار اجرا میشود و همچنین حلقه خط اول نیز حداکثر n بار پیمایش میشود. پس اردر زمانی اجرای برنامه $O(n^2)$ میباشد .

۲. $T(n) = \Omega(n^2)$
این بار نشان میدهیم زمان اجرای الگوریتم بیشتر یا مساوی n^2 است. پس بدترین حالت را در نظر میگیریم؛ یعنی با فرض اینکه میخواهیم آرایه به صورت صعودی مرتب شود، آرایه ای را در نظر میگیریم که به صورت نزولی مرتب شده است. $([n, n-1, ..., 3, 2, 1])$
حال محاسبات را به صورت دقیق انجام میدهیم. فرض کنیم خطوط ۳ و ۴ در c واحد زمانی اجرا میشوند. هر حلقه خط دوم، دقیقا i مرحله انجام میشود. پس هر مرحله از حلقه خط یک ic واحد زمانی طول میکشد. پس کل حلقه $\sum_{n=1}^{n-1} ic$ واحد زمانی به طول می انجامد.
$\sum_{n=1}^{n-1} ic = \binom{n}{2} c = \frac{n(n-1)}{2} c = \frac{n^2}{2} c - \frac{n}{2} c = \Omega(n^2)$

پس از دو بخش فوق نتیجه میشود: $T(n) = \Theta(n)$

۲ مرتب سازی درجی دودویی یا Binary Insertion Sort

در مرتب سازی درجی، جایگاه یک عضو را با swap های متوالی در یک زیر-آرایه مرتب پیدا میکنیم و در آن جایگاه قرار میدهیم. اما در مرتب سازی دودویی، ابتدا با جست‌وجوی

دودویی جایگاه عدد را پیدا میکنیم و با swap های متوالی آنرا در جایگاه خود قرار میدهیم. هنگامی که عمل مقایسه پیچیده شود(مانند مقایسه کردن اشیا مختلف) و هزینه زمانی آن دیگر $O(1)$ نباشد، مرتب سازی باینری بهتر است زیرا تعداد مقایسه ها برای پیدا کردن جایگاه کمتر میشود. به‌طور کلی:
اگر هر مقایسه از مرتبه $O(1)$ باشد:

Insertion Sort: $\Theta(n^2)$
Binary Insertion Sort: $\Theta(n^2)$

اگر هر مقایسه از مرتبه $O(k)$ باشد:

Insertion Sort: $\Theta(n^2k)$
Binary Insertion Sort: $\Theta(nLognk + n^2)$

$nlognk$ مربوط به مقایسه ها و n^2 مربوط به swap هاست.

۳ رویکرد حل مسئله تقسیم و حل یا Divide and Conquer

در این رویکرد حل مسئله، مسئله اصلی به زیر مسئله های کوچکتر تقسیم شده و پس از حل کردن زیر مسئله ها، آنها را باهم ادغام میکنیم. به طور کلی این رویکرد از سه بخش تشکیل شده است:
۱. تقسیم (Divide)
۲. حل (Solve)
۳. ادغام (Merge)
هنگامی که مسئله را با این رویکرد حل میکنیم، مرتبه زمانی آن به صورت زیر محاسبه میشود:
$T(n)$ = زمان تقسیم + زمان حل + زمان ادغام

۴ مرتب سازی ادغامی یا Merge Sort

با استفاده از رویکرد تقسیم و حل میتوان مرتب سازی ادغامی را معرفی کرد. سورس کد مرتب سازی اذغامی به زبان های مختلف در لینک زیر موجود است.
https://www.geeksforgeeks.org/merge-sort/
با توجه به کد، میتوان مرتبه زمانی را برای الگوریتم فوق به صورت زیر تعریف کرد:
۱. زمان تقسیم: $O(1)$
۲. زمان حل: $2T(n/2)$
۳. زمان ادغام: $O(n)$

$$T(n) = O(1) + 2T(n/2) + O(n) = 2T(n/2) + O(n)$$

حال برای پیدا کردن یک رابطه صریح برای $T(n)$ از درخت بازگشتی استفاده میکنیم و

فرض میکنیم $O(n) = cn$
رابطه فوق برای $T(n)$ از دو بخش cn و $2T(n/2)$ تشکیل شده است. پس ریشه درخت cn است و دو خوشه متصل به آن، هریک $T(n/2)$ هستند. به صورت بازگشتی میتوان این مراحل را برای $T(n/2)$، $T(n/4)$ و... انجام داد.
$T(n) = 2T(n/2) + cn = 2(2T(n/4) + cn/2) + cn = ...$
پس ارتفاع درخت برابر $logn$ میباشد و که مجموع برگ ها در هر مرحله برابر cn میباشد.
پس داریم: $T(n) = cn * logn = \Theta(nlogn)$

۵ حل مسئله به روش درخت بازگشتی

در حالت کلی فرض کنیم رابطه مربوط به $T(n)$ به صورت زیر تعریف شده است:
$T(n) = aT(n/b) + f(n)$ f(n) تابعی از n میباشد :
ریشه از مرتبه $f(n)$ میباشد و برگ های متصل به آن به تعداد aتا برگ از مرتبه $f(n/b)$ است. به طور کلی هر برگ از مرتبه $f(n/b^k)$ به تعداد a فرزند از مرتبه $f(n/b^{k+1})$ دارد.
طبق توضیحات فوق واضح است که درخت بازگشتی ذکر شده دارای $L = n^{log_b a}$ برگ و ارتفاع $log_b n$ میباشد.
به کمک قضیه اصلی(Master Theorem) یک درخت بازگشتی در سه حالت قابل حل است.

قضیه اصلی یا Master Theorem

قضیه اصلی با توجه به سه حالت ممکن برای وضعیت مجموع برگ های با یک ارتفاع میتواند رابطه صریح برای $T(n)$ ارائه دهد.
۱. $f(n) = O(L^{1-\epsilon})$: حالتی که در آن، برگ های با بیشترین ارتفاع غالب هستند.
در این حالت داریم: $T(n) = \Theta(L)$
۲. $f(n) = \Theta(L(logn)^k)$: حالتی که در آن، مجموع برگ ها در هر ارتفاعی برابر است.
در این حالت داریم: $T(n) = \Theta(L(logn)^{k+1})$
۳. $f(n) = \Omega(L^{1+\epsilon})$: حالتی که در آن، ریشه غالب است
در این حالت داریم: $T(n) = \Theta(f(n))$
نکته این که قصیه اصلی برای حالتی غیر این سه حالت، جوابی ارائه نمیدهد.

۶ تحلیل زمانی الگوریتم ها برای nهای بزرگ

در تحلیل زمانی الگوریتم های مختلف، علاوه بر مرتبه بزرگی آنها (Θ, Ω, O) باید به ضرایب موجود در رابطه مربوط به $T(n)$ و همچنین رابطه صریح آن نیز توجه کرد. زیرا

برای مقادیری از n رفتار توابع $T(n)$ مربوط به الگوریتم های مختلف تغییر میکند.

مثال

۱. مرتب سازی درجی در زبان C++ در زمان $0.01n^2\mu s$ انجام میشود.
۲. مرتب سازی درجی در زبان Pytho در زمان $0.2n^2\mu s$ انجام میشود.
۳. مرتب سازی ادغامی در زبان Python در زمان $2.2n^2\mu s$ انجام میشود.
در مقایسه الگوریتم های اول و سوم، برای $n \geq 400$ مرتب سازی ادغامی پایتون عملکرد بهتری دارد. همچنین برای $n \geq 67$ استفاده از مرتب سازی ادغامی پایتون بهتر از مرتب سازی درجی در همان زبان است.