ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان 99]

جلسه : مفهوم حافظه و زمان

نگارنده: علی شفیعی

در جلسه گذشته مطالبی درباره ماشین‌های محاسباتی آموختیم و همچنین تعریف قابل قبولی برای الگوریتم ارائه کردیم.

تحلیل الگوریتم‌ها

در پیاده سازی الگوریتم‌ها با محدودیت زمانی و حافظه مواجه هستیم ،به همین دلیل نیاز به تحلیل الگوریتم‌ها و طراحی الگوریتم‌های بهینه تر داریم.
در ادامه مسئله‌ای را حل و سعی میکنیم درباره زمان اجرای آن صحبت کنیم.

سوال

1000 عدد به عنوان ورودی به ما داده شده است .100 عدد بزرگ تر را به عنوان خروجی چاپ کنید.
ابتدا روش حل را توضیح داده و الگوریتم آن را می‌بینیم سپس به محاسبه تعداد دستورهای الگوریتم میپردازیم و در آخر سعی می‌کنیم درباره ارتباط تعداد دستورهای اجرایی با زمان اجرا مطالبی را بیان کنیم.

توضیح کلی الگوریتم

هر بار روی آرایه شامل اعداد پیمایش می‌کنیم و در پیمایش i ام ، i امین عدد بزرگ را به جایگاه i ام می‌بریم. این پیمایش را 100بار تکرار می‌کنیم و در نهایت 100 عضو اول آرایه را در خروجی چاپ می‌کنیم.

الگوریتم :

```
FINDTOP100(A)
1  Input A[1000]
2  for i = 1 to 100
3        m=i
4        for j = i + 1 to 1000
5              if A[m] < A[j]
6                    m=j
7        swap(A[m], A[i])
8  print A[1:100]
```

محاسبه تعداد دستورات :

برای محاسبه تعداد دستورات اجرا شده در الگوریتم بالا ، خط به خط جلو می‌رویم .
در خط اول نیاز به گرفتن 1000 عدد از ورودی هستیم بنابراین1000عملیات از جنس input انجام می‌شود.
در خط دوم حلقه 100بار اجرا می‌شود (به طور دقیق تر101 بار)، بنابراین 100 عملیات از جنس for داریم.
خط سوم نیز به همان تعداد اجرای حلقه یعنی100بار اجرا می‌شود ، بنابراین 100 عملیات از جنس مقداردهی داریم.
در خط چهارم حلقه به ازای مقادیر مختلف i تعداد اجرای متفاوتی دارد و در کل به تعداد: $50 * (900 + 1000) \simeq 900 + ... + 998 + 999$ بار اجرا می‌شود.
خط بعدی نیز به همین تعداد اجرا می‌شود.
در خط ششم اما قضیه متفاوت است و بسته به ورودی مسئله، تعداد دفعات اجرای این خط متفاوت است و در حالت ماکزیمم به تعداد $50 * (900 + 1000) \simeq 900 + ... + 998 + 999$ و در حالت مینیمم 0 بار اجرا می‌شود.
خط بعدی نیز100بار اجرا می‌شود زیرا در هر مرحله از اجرای حلقه بیرونی یک بار به این دستور می‌رسیم.
خط آخر نیز 100 بار اجرا می‌شود و 100 عدد بزرگ تر را چاپ می‌کند.

تحلیل زمانی :

در تحلیل زمانی الگوریتم‌ها باید به سربارهایی که زبان برنامه‌نویسی برای ما بوجود می‌آورد توجه کنیم ، برای مثلا ورودی گرفتن در زبان جاوا با استفاده از Scanner زمانی مصرف می‌کند ، گرفتن و آزادسازی حافظه نیز زمانی مصرف می‌کند.
در جاوا garbage collector متغیرهایی که در حافظه فضا اشغال کرده‌اند ولی ارجاعی به آن‌ها در برنامه وجود ندارد را شناسایی و فضای اشغالی توسط آن‌ها را آزاد می‌کند.

سوال

دانستن این موارد چه کمکی در تحلیل زمانی الگوریتم به ما می‌کند؟
گاربج کالکتر جاوا در طول برنامه چندین بار صدا زده می‌شود اما هنگامی که بخش قابل توجه‌ای از حافظه در اختیار برنامه اشغال شده باشد، این فرایند به تعداد بیشتری اتفاق می‌افتد و بنابراین زمان قابل توجهی استفاده می‌شود، بنابراین یکی از مواردی که باید در تحلیل و بررسی زمان اجرای الگوریتم‌ها به آن توجه کنیم گاربج کالکتر است . علاوه براین مورد باید به زمانی که برنامه صرف دریافت ورودی از کاربر می‌کند هم توجه کنیم و در صورت نیاز از روش‌های دیگری برای ورودی گرفتن مانند BufferedInputStream استفاده کنیم.

نکاتی درباره محاسبه زمان اجرا الگوریتم :

پس از بدست آوردن تعداد عملیات اجرا شده در هر خط الگوریتم ، اولین ایده ای که ممکن است برای بدست آوردن زمان اجرای الگوریتم به ذهن برسد این است که تعداد عملیات اجرا شده این خط‌ها را با هم جمع بزنیم و نتیجه را به عنوان زمان اجرا در نظر بگیریم ، اما این روش صحیح نیست زیرا همانطور که پیش‌تر گفته شد ، زمان اجرای هر کدام از اینستراکشن‌ها با هم متفاوت است و بعضی عملیات مانند xor ، زمان کم‌تری از ضرب مصرف می‌کنند.
اگر بخواهیم به اندازه کافی سریع بودن الگوریتم را تحلیل کنیم ، می‌توانیم عملگری که بیشتر از همه زمان می‌برد را در نظر گرفته و طبق آن تحلیل کنیم.

نحوه برخورد با ضرایب در تابع زمان اجرای الگوریتم :

حال فرض کنید تابعی برای زمان اجرای الگوریتم مورد نظر برحسب اندازه ورودی پیدا کرده‌ایم ، نحوه برخورد با ضریب‌های موجود در این تابع باید چگونه باشد؟کدام ضرایب را می‌توان نادیده گرفت و کدام ضرایب نقش اساسی ایفا می‌کنند؟
ضرایبی را می‌توان نادیده گرفت که در مقابل ورودی ناچیز باشند یا به تعبیر دیگر ،تابعی از ورودی نباشند.
دلیل گزاره فوق این است که اگر تابع زمانی دو الگوریتم تنها در ضرایبی که در مقابل ورودی ناچیز هستند با هم تفاوت داشته باشند آنگاه می‌توان با قوی‌تر کردن پردازنده کامپیوتر، این ضعف را جبران کرد و کارایی مورد نظر را از برنامه انتظار داشت.
اما عملیات فوق برای زمانی که تابع زمانی دو الگوریتم در مواردی غیر از ضرایب با هم تفاوت دارند امکان پذیر نیست ، زیرا مثلا فرض کنید تابع زمان اجرای یک الگوریتم برابر n^2 و تابع زمان اجرای الگوریتم دیگر برابر n باشد که n اندازه ورودی مسئله است.
حال اگر $n = 10$ باشد، باید پردازنده کامپیوتری که الگوریتم کندتر را اجرا می‌کند را 10 برابر سریع تر کنیم ، که چندان دور از دسترس نیست ، اما اگر $n = 1000$ باشد باید پردازنده کامپیوتری که الگوریتم کندتر را اجرا می‌کند را 1000برابر سریع تر کنیم ، که ممکن است چنین چیزی در عمل ممکن نباشد. بنابراین در این حالت نمی‌توان با تغییر سخت افزار از عیب‌های الگوریتم کندتر چشم پوشی کنیم و به وضوح دو الگوریتم از نظر کارایی با هم متفاوت‌اند.
در جلسه آینده با تحلیل مرتبه زمانی آشنا خواهیم شد و ابزارهای خوبی برای تحلیل الگوریتم‌ها یاد خواهیم گرفت.