ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان ۹۹]

جلسه : ششم

نگارنده: غزل فراهانی

در جلسه گذشته مفاهیم $o, O, \theta, \omega, \Omega$ را معرفی کردیم، از مرتبه‌های زمانی صحبت کردیم و برخی توابع را به لحاظ مرتبه زمانی آن‌ها باهم مقایسه و بررسی کردیم. در این جلسه می‌خواهیم مطالب بررسی شده در جلسه قبل را در عمل و برخی الگوریتم‌ها ببینیم.

۱ جلسه ششم

می‌خواهیم برخی از الگوریتم‌های search را بررسی و با هم مقایسه کنیم. الگوریتم زیر یک آرایه از اعداد و یک عدد ورودی میگیرد و اگر عدد ورودی در آرایه وجود داشت $True$ و در غیر اینصورت $False$ را برمی‌گرداند.

```
FIND(numbers, value)
1  for i = 0 to length[numbers]
2        if numbers[i] == value
3              return TRUE
4  return FALSE
```

در شبه کد بالا، عبارت i = 0 یک بار، شرط i < length[numbers] ، n + 1 بار و i++، n بار اجرا می‌شود. همچنین عبارت numbers[i] == value ، n بار و در نهایت Return True یا Return False یک بار اجرا می‌شوند. در نتیجه الگوریتم بالا در بدترین حالت از $O(n)$: ۳n + ۳ و در بهترین حالت از $O(1)$: ۵ است.
همچنین برای search میتوانیم ابتدا تکرر هر عدد را حساب کنیم. الگوریتم آن به این صورت است:

```
CALCULATE-OCCURRENCE(numbers, occurrence)
1  for i = 0 to length[occurrence]
2  // i = 0 : 1, i < length[occurrence] : m + 1, i++ : m
3        occurrence[i] = 0 // 2m
4  for i = 0 to length[occurrence]
5  // i = 0 : 1, i < length[occurrence] : n + 1, i++ : n
6        occurrence[(int)numbers[i]]++ // 4n
```

در این بخش عبارت occurrence[i] = 0 را اگر به طور دقیق حساب کنیم 2m بار طول خواهد کشید m) بار خواندن از آرایه و m بار عملگر تساوی) که m برابر طول آرایه occurrence یا به عبارتی بیشترین عدد است و با همین استدلال عبارت occurrence[(int)numbers[i]]++، 4n بار طول خواهد کشید(cast کردن + دوبار خواندن از آرایه + یک واحد افزایش). در نتیجه مرتبه زمانی در بهترین و بدترین حالت برابر با $O(m + n)$ خواهد بود.

SEARCH$(numbers, occurrence, value)$

1 CALCULATE-OCCURENCE$(numbers, occurrence)$
2 **if** FIND$(numbers, value)$
3 $\quad print("Yes, occurrence : ", occurrence[value])$
4 **else** $print("No")$

حال اگر بخواهیم به جای یک عدد، تعداد q عدد را جست و جو کنیم، اندازه ورودی برابر با n+q می‌شود. همچنین اگر بخواهیم پارامتر بزرگ‌ترین عدد را هم وارد محاسبات مرتبه زمانی کنیم، مرتبه زمانی تابع calculate-occurrence برابر $O(n + ۲^M)$ خواهد بود که M برابر با تعداد بیت‌های بزرگترین عدد ورودی است.(زیرا بزرگ‌ترین عدد را می‌توان با log maxNumber بیت نمایش داد).

حال می‌توانیم تابع find را به نحوی دیگر پیاده‌سازی کرد که این عمل را در $O(۱)$ انجام دهیم:

FIND$(occurrence, value)$

1 **return** $(occurrence[(int)value]! = 0)$

مشابه استدلال بخش‌های قبلی این تابع در $O(۱)$ قابل انجام است(return کردن، cast کردن، خواندن آرایه، عملگر $= !$) مرتبه الگوریتم find اولیه از $O(nq)$ و الگوریتم دوم از $O(n + q + ۲^M)$ است. تفاوت این الگوریتم با الگوریتم قبلی در میزان حافظه‌ایست که مصرف میکنیم. پس الگوریتم‌های مختلف برای حل یک مساله در بعضی موارد بهتر و در بعضی موارد بدتر عمل می‌کنند و بسته به نوع مسئله ممکن است هر کدام کاراتر باشند.

همچنین می‌توانیم مرتبه‌های زمانی را به روش دیگری نیز حساب کنیم به این ترتیب که مرتبه زمانی را برحسب تعدادی instruction حساب کنیم که بر بقیه instruction ها غالب است. برای مثال در الگوریتم find اول، می‌توان تنها عبارت $i <$ length[numbers] را محاسبه کرد زیرا تعداد تکرر این قسمت از باقی قسمت‌ها بیشتر یا مساوی بوده و در محاسبه مرتبه زمانی باقی قسمت‌ها را پوشش می‌دهد. در نتیجه با پیدا کردن instructionهای کلیدی که حتما تعداد تکرار آن‌ها از بقیه بزرگ‌تر است، می‌توان راحت‌تر مرتبه زمانی را محاسبه کرد.