پیچیدگی زمانی الگوریتم

طراح داکیومنت : علی ابراهیمی

در یک الگوریتم بهینه دو فاکتور مهم وجود دارد:

۱. حافظه مصرفی
۲. زمان مصرفی

یعنی الگوریتمی بهتر است که این دو فاکتور مهم را در دل خودش داشته باشد. در اغلب موارد، زمان اجرای الگوریتم مهم تر از فضای مصرفی آن است. در اینجا به این مورد مهم می پردازیم.

پیچیدگی زمانی و مرتبه اجرایی

غالباً پیچیدگی زمانی و مرتبه اجرایی الگوریتم را با T(n) نمایش می دهند (n تعداد ورودی های یک تابع است). در مثال زیر دستور [s=s+list[i] را دستور اصلی در نظر بگیرید.

```
float sum(float list[], int n)
{
    float s=0;
    int i;
    for(i=0;i<n;i++)
        s=s+list[i];
    return s;
}
```

تعداد مراحل برنامه فوق:

```
0
0
1
0
2n+2
```

```
n
1
0
---
3n+4
```

بطور کلی در الگوریتم به اینها هزینه می گوییم مثلاً در خط سوم مقدار صفر در متغیر s قرار گرفته بنابراین هزینه آن برابر با ۱ خواهد بود. در حقیقت اگر متغیری بدون مقدار تعریف شود هزینه آن صفر است.

برخی از نمادهای مهم و معروف در پیچیدگی اجرایی الگوریتم:

نام	نماد
ثابت	O(1)
خطی	O(n)
لگاریتمی	O(log n)
درجه دوم	O(n^2)
درجه سوم	O(n^3)
نمایی	O(2^n)
فاکتوریل	O(n!)

تابع ثابت

الگوریتم هایی را توصیف می کند که صرف نظر از اندازه ورودی، زمان یکسانی برای محاسبه صرف می کنند. به عنوان مثال، اگر یک تابع برای پردازش ده عنصر و 1 میلیون مورد به زمان یکسانی نیاز داشته باشد، می گوییم که دارای نرخ رشد ثابت یا O(1) است.

نمونه هایی از الگوریتم های زمان اجرا ثابت:

۱. زوج یا فرد بودن یک عدد را بررسی کنید.
۲. بررسی کنید که آیا یک آیتم در یک آرایه تهی است یا خیر.
۳. اولین عنصر را از یک لیست چاپ کنید.

پیاده سازی اولین مثال:

```
function isEvenOrOdd(n) {  return n % 2 ? 'Odd' : 'Even'; }
```

تابع خطی

زمانی گفته می شود که یک الگوریتم دارای پیچیدگی زمانی خطی است که زمان اجرا، به صورت خطی با طول ورودی افزایش می یابد. تابع خطی، پردازشی روی تمام مقادیر در داده های ورودی انجام می دهد.

نمونه هایی از الگوریتم های زمان خطی:

۱. مقدار max/min را در یک آرایه بدست آورید.
۲. یک عنصر معین را در یک مجموعه پیدا کنید.
۳. تمام مقادیر موجود در یک لیست را چاپ کنید.

پیاده سازی اولین مثال:

```
function findMax(array) {
    int max = array[0];
    for (int i = 1; i < array.length; i++) {
        if(max < array[i]) {
            max = array[i];
        }
    }
    return max;
}
```

تابع لگاریتمی

تابع لگاریتمی یکی از رایج ترین توابع مورد استفاده در زمینه تحلیل الگوریتم است. نحوه تقریب لگاریتم ها برای محاسبه لگاریتم دقیق برای هر عدد صحیح، باید حساب دیفرانسیل و انتگرال را اعمال کنیم، اما در زمینه تحلیل الگوریتم، ما لزوماً مقدار دقیق را نمی‌خواهیم، زیرا یک تقریب خوب برای هدف ما کافی است. برای انجام این کار، ما معمولاً از سقفی استفاده می کنیم که به کوچکترین عدد صحیح بزرگتر یا مساوی لگاریتم ترجمه می شود. معمولاً الگوریتم هایی که در هر تکرار مسئله را به نصف تقسیم می کنند با ترتیب لگاریتمی پیچیدگی زمانی مطابقت دارند. چنین مثالی می تواند الگوریتم محبوب جستجوی باینری باشد که در زیر نشان داده شده است.

```
def binary_search(numbers, item):
   first_index = 0
   last_index = len(numbers)-1

    while first_index <= last_index:
        mid_index = (first_index + last_index)//2
        if numbers[mid_index] == item:
           return True
        else:
            if item < numbers[mid_index]:
                last_index = mid_index - 1
            else:
                first_index = mid_index + 1
     return False
```

توجه: در علوم کامپیوتر، به دلیل ماهیت کامپیوترهایی که برای ذخیره باینری طراحی شده اند، معمولاً از لگاریتم در پایه 2 استفاده می کنیم. این می تواند کمی گیج کننده باشد زیرا در ریاضیات پایه "پیش فرض" 10 است و اکثر ماشین حساب ها پایه 10 را فرض می کنند. در تجزیه و تحلیل الگوریتم، زمانی که پایه حذف می شود، معمولاً تابع log را با پایه ای برابر با 2 تجزیه می کنیم.

تابع درجه دوم

تابع دیگری که در تحلیل پیچیدگی رایج است تابع درجه دوم است که مجذور ورودی n را به خود اختصاص می دهد. در تجزیه و تحلیل الگوریتم، توابع درجه دوم برای توصیف پیچیدگی حلقه های تو در تو، یعنی

دنباله ای از n عملیات که n بار انجام می شود، استفاده می شود.

چند نمونه:

۱. بررسی کنید که آیا یک مجموعه مقادیر تکراری دارد یا خیر.
۲. مرتب کردن یک آرایه با استفاده از مرتب‌سازی حبابی، مرتب‌سازی درجی یا مرتب‌سازی انتخابی.
۳. همه جفت های مرتب شده ممکن را در یک آرایه پیدا کنید.

پیاده سازی مثال اول:

```
Example with n^2-time complexity
n = [1, 2, 3, 4, 5]

for i in n:
    for j in n:
        if i != j and n[i] == n[j]:
            print(f'{n[i]} is a duplicate')
```

تابع نمایی

زمان اجرای نمایی (مبنای 2) به این معنی است که محاسبات انجام شده توسط یک الگوریتم هر بار که ورودی رشد می کند دو برابر می شود. **چند نمونه:**

۱. مجموعه توانی (یافتن تمام زیرمجموعه های یک مجموعه).
۲. فیبوناچی
۳. مساله فروشنده دوره گرد با استفاده از برنامه نویسی پویا.

تابع فاکتوریل

فاکتوریل ضرب تمام اعداد صحیح مثبت کمتر از خودش است. برای مثال:

```
5! = 5*4*3*2*1 = 120
```

همانطور که حدس زدید، احتمالا می خواهید از الگوریتم هایی که این زمان اجرا را دارند دوری کنید!

مثال ها:

۱. جایگشت های یک رشته

۲. حل مشکل فروشنده دوره گرد با جستجوی brute-force

در مثال اول:

```
getPermutations('ab') // ab, ba...// n = 2, T(n) = 2;
getPermutations('abc') // abc, acb, bac, bca, cab, cba...// n = 3, T(n) = 6;
getPermutations('abcd') // abcd, abdc, acbd, acdb, adbc, adcb, bacd...// n = 4, T(n) =
getPermutations('abcde') // abcde, abced, abdce, abdec, abecd, acbde...// n = 5, T(n)
```

به سوال زیر پاسخ دهید.

- شبه کدی بنویسید که یک تابع بررسی کند آیا در یک آرایه از اعداد صحیح با اندازه n، دو عدد با مجموع k وجود دارد یا خیر. ابتدا الگوریتم را با پیچیدگی زمانی **O(n^2)** بنویسید و سپس سعی کنید الگوریتم دیگری ارائه دهید که همان کار را با پیچیدگی **O(n)** انجام دهد. این دو الگوریتم را از نظر *حافظه مصرفی* مقایسه کنید.