University of Kashan

ساختمان داده‌ها و الگوریتمها

سید مهدی وحیدی پور

با تشکر از دکتر جواد سلیمی

مرجع

- عنوان مرجع:

Fundamental of data Structure in C++

- *Ellis Horowitz, Sartaj Sahni, Dinesh Mehta*
- ساختمان داده ها در ++C

حسین ابراهیم زاده قلزم

انتشارات سیمای دانش

- ساختمان داده ها به زبان C

امیر علیخانزاده

انتشارات باغانی

Web: https://faculty.kashanu.ac.ir/vahidipour/fa/page/ساختمان-داده‌ها

نحوه ارزیابی

- میان ترم۶ نمره، پایان ترم ۶ نمره، کلاسی ۸ نمره
  - (حل تمرین ۱.۵، پروژه ۲، تکالیف ۲، کوییز ۳)

قوانین کلاس:

- حداکثر تعداد مجاز غیبت ۳ جلسه
  - غیبت چهارم ۵ درصد نمره کل کسر
  - غیبت پنجم ۱۰ درصد نمره کل کسر
  - تعداد بالاتر حذف درس

دانشجو باید حداقل ۲۵ درصد نمره میانترم را کسب کند تا بتواند در امتحان پایانترم شرکت کند.

خصوصیات الگوریتم

برنامه

- ورودی: یک الگوریتم می تواند هیچ یا چندین کمیت ورودی داشته باشد
- خروجی: الگوریتم بایستی حداقل یک کمیت به عنوان خروجی داشته باشد.
- قطعیت (عدم ابهام): هر دستورالعمل باید واضح و بدون ابهام باشد.
- کارایی (انجام پذیر بودن): هر دستورالعمل باید به قدر کافی ساده و ابتدایی باشد به گونه ای که با استفاده از قلم و کاغذ بتوان آن را با دست نیز اجرا نمود.
- محدودیت (پایان پذیر بودن): برای تمام حالات ، الگوریتم باید پس از طی مراحل محدودی خاتمه یابد.

ارزیابی یک برنامه

- معیارهای مختلف
- آیا برنامه اهداف اصلی کاری را که می خواهیم، انجام می دهد؟
- آیا برنامه درست کار می کند؟
- آیا برنامه مستند سازی شده است تا نحوه استفاده و طرز کار با آن مشخص شود؟
- آیا برنامه برای ایجاد واحدهای منطقی ، به طور موثر از توابع استفاده می کند؟
- آیا کد گذاری خوانا است؟

ساختمان داده

- آیا برنامه از حافظه اصلی و کمکی به طور موثری استفاده می کند؟
- آیا زمان اجرای برنامه برای هدف شما قابل قبول است؟

ساختمان داده

- چگونه برنامه های خوب و بهینه بنویسیم
  - چگونه از حافظه سیستم به نحو مطلوب استفاده نماییم
  - چگونه زمان اجرای برنامه ها را پایین بیاوریم و سرعت اجرای آنها را بالا ببریم
- تعریف:
  - ساختمان داده ها درسی است که هدف نهایی آن حل مساله سرعت اجرا و حافظه مصرفی الگوریتم ها است

پیچیدگی یک برنامه

- پیچیدگی زمانی و پیچیدگی حافظه
  - پیچیدگی فضای یک برنامه مقدارحافظه مورد نیازبرای اجرای کامل یک برنامه است.
  - پیچیدگی زمان یک برنامه مقدار زمان کامپیوتر است که برای اجرای کامل برنامه لازم است.

پیچیدگی حافظه- مثال

```
Float sum ( float *a,  const int n)
{
    float s=0;
    For (int  i=0; i< n ; i++)
       S+=a[i];
    Return s;
}
```

مشخصه موردی: n تعداد عضوهایی که با هم جمع می شوند

```
void sort( float *a,  const int n)
{
    For (int  i=0; i< n ; i++)
       a[j] is the min. in a[i]…a[n-1];
    swap (a[j] , a[i]);
}
```

مشخصه موردی: n تعداد عضوهایی که باید مرتب شوند

حافظه مورد نیاز مستقل از n است.

$$S_{sum}() = 0$$

پیچیدگی حافظه

```
float rs ( float *a,  const int n)
{
    if (n<=0) return 0;
    else return ( rs( a,n-1)+a[n-1] )
}
```

مشخصه موردی: n تعداد عضوهایی که با هم جمع می شوند

عمق بازگشتی n+1

هر احضار تابع بازگشتی دست کم ۴ کلمه از حافظه

حافظه مقادیر a و n و مقدار برگشتی و ادرس برگشتی

$$S_{sum}() = 4(n+1)$$

پیچیدگی حافظه

نیازمندیهای فضای متغیر: فضای مورد نیاز که اندازه آن بستگی به نمونه I از مساله ای که حل می شود، دارد مانند حافظه مورد نیاز پشته بازگشتی و حافظه مورد نیاز برای متغیرهای ارجاعی

$$S(P) = c + S_p(I)$$

نیازمندیهای فضای کل ←

نیازمندیهای فضای ثابت: فضای مورد نیازی که به تعداد و اندازه ورودی و خروجی بستگی ندارد مانند حافظه مورد نیاز دستورها، ثابت ها، متغیرهای با طول ثابت و ...

پیچیدگی زمانی

$$T(P) = c + T_p(I)$$

زمان اجرای برنامه

نیازمندیهای زمان برنامه

زمان کامپایل

زمان کامپایل مشابه اجزای فضای ثابت است زیرا به خصیصه های نمونه بستگی ندارد.

$$T_p(n) = c_a ADD(n) + c_s SUB(n) + c_m MUL(n) + c_d DIV(n) + \ldots$$

Ca, Cs, Cm, Cd زمان لازم برای جمع، تفریق، ضرب و تقسیم
ADD, SUB, MUL, DIV توابعی که تعداد آنها را مشخص می کنند

پیچیدگی زمانی

بسیاری عوامل در زمان اجرا دخیل هستند ⇐ تخمینی از زمان اجرا

تنها تعداد مراحل داخل برنامه را می شماریم

یک مرحله برنامه ، قسمت با معنی برنامه است که زمان اجرای آن مستقل از خصیصه های نمونه باشد

دستورات اجرایی، دستورات تعیین نوع، توضیحات، شرط if، حلقه

تعداد مراحل

- توضیحات comments، تعاریف زیر برنامه و توابع، { }، begin، end
  - تعداد مراحل اجرایی صفر

```
Procedure f( ...) ;        0
void f( ...);              0
```

- دستورهای تعیین نوع
  - تعداد مراحل اجرایی صفر مگر آنکه برای آنها مقدار دهی اولیه صورت گیرد در اینصورت یک

```
int   x;                   0
int   x=3;                 1
float a,b=5;               1
```

- دستور اجرایی
  - به ازای هر بار اجرا دارای گام ۱

```
y=x*y+z;                   1
write (y)                  1
return  p;                 1
```


تعداد مراحل

- دستور شرطی if: عبارت شرط ۱ گام و گام کل دستور وابسته به درست و غلط بودن شرط

```
If (x<y)          1
   S=S+1;         1
Else
   t=t+1;         1
   r=r+1          1
```

۱+۱ شرط درست → ۲
۱+۲ شرط نا درست → ۳

- عداد گام در حلقه: حلقه به تعداد "تکرار +۱" گام و جملات تکرار شونده داخل حلقه به تعداد "تکرار" گام اختیار می کنند

```
For (i=2; i<n; i++)        n-1
   s=s+1                   n-2
                        +
                           2n-3
```


تعداد مراحل

- تعداد گام در حلقه های تو در تو
  - از بیرونی ترین حلقه شروع کرده و تعداد تکرار هر حلقه را برای تمام حلقه ها و دستورات تکرار شونده پایین آن در "تعداد تکرار+۱" را برای خود حلقه در نظر می گیریم

```
procedure add( var a,b,c: matrix; m,n: integer);     0
var i,j : integer;                                   0
begin                                                0
for i:=1 to m do                                     m+1
   for j:=1 to n do                                  m(n+1)
      c[i,j]:= a[i,j]+ b[i,j];                       mn
end                                                  0
                                                  +
                                                     2mn+2m+1
```

اگر m>n باشد بهتر است جای دو دستور for را در برنامه عوض کنیم تا شمار مراحل 2mn+2n+1 شود

حال میتوانیم از شمارش برای مقایسه دو کد استفاده کنیم

مقایسه دو code: روش ساده برای جمع کردن

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

```
0
0
1
n+1
n
1
0
+
2n+3
```


حال میتوانیم از شمارش برای مقایسه دو کد استفاده کنیم

مقایسه دو code: روش بازگشتی برای جمع

```
float rs ( float *a,  const int n)
{
    if (n<=0) return 0;
    else return ( rs( a,n-1)+a[n-1] )
}
```

n<=0	n>0
0	0
0	0
2	1
0	1+t(n-1)
0	0
+	
2	2+t(n-1)

$$t(n)=\begin{cases}2 & \text{if } n<=0\\ 2+t(n-1) & \text{if } n>0\end{cases}$$

$$\begin{aligned}t(n)&= 2+t(n-1)\\ &=2+2+t(n-2)\\ &=2+2+2+t(n-3)\\ &=2n+t(0)\\ t(n)&=2n+2\end{aligned}$$

تعدحال میتوانیم از شمارش برای مقایسه دو کد استفاده کنیم

مقایسه دو code

روش اول: $2n+3$

روش دوم: $2n+2$

هر دو روش خطی هستند یا به اصطلاح از $O(n)$

تعداد مراحل هر دستور

نوع دستور	تعداد مراحل اجرا
توضیحات	صفر
دستورهای تعیین نوع	صفر
عبارتها و دستورهای جایگزین	اکثر عبارتها ۱ به استثناء عبارتهای حاوی احضار توابع
دستورهای تکرار	تنها برای قسمت کنترلی این دستورات
دستور switch	هزینه جایگزینی + هزینه شرط + هزینه تمام شرطهای قبل از آن
دستور if else	متناظر با شرط و عبارتهای درون if & else
احضار توابع	یک در نظر گرفته می شود مگر هنگامی که احضار حاوی پارامترهای انتقال از طریق مقدار باشد اگر تابع به صورت بازگشتی باشد متغیرهای محلی را نیز باید در نظر گرفت
دستورهای مدیریت حافظه	شمار مراحل مانند احضار تابع محاسبه می شود
دستورهای تابع	صفر
دستورهای پرشی	به استثنای return یک است این دستور نیز یک است مگر آنکه مقدار بازگردانده شده تابعی از مشخصه موردی باشد

تعداد مراحل؟ تنها برای قسمت کنترلی این دستورات

```
for ( i=0;i<n; i++)
    for (j=i;j<n;j++)
      count++;
```

n+1
n-i+1

(n+1)+(n)+(n-1)+...+2=n+(n-1)+...+1=n(n+1)/2
اصطلاحا از درجه دوم است.
O(n^2)

علامت گذاری مجانبی O، Ω، Θ

- انگیزه ما برای تعیین شمار مراحل توانایی مقایسه پیچیدگی زمانی دو برنامه است که یک عمل را انجام می دهند و نیز پیش بینی رشد زمان اجرا با تغییر مشخصه موردی است.

علامت گذاری مجانبی O، Ω، Θ

- تعریف O (Big "oh"):
  - f(n)=O(g(n)) اگر و فقط اگر ثابتهای مثبتی مانند C و N وجود داشته باشند به طوری که به ازای تمامی مقادیر n و n>=N، f(n)<=cg(n) باشد.
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) حداکثر (کوچکتر یا مساوی) g(n) خواهد بود.
  - وقتی می نویسیم O(1) منظور این است که زمان اجرا ثابت است، O(n) یعنی زمان اجرا خطی است، $O(n^2)$ یعنی زمان اجرا از درجه دوم و ...
  - مثال
- $f(n) = 3n+2$
  - $3n + 2 <= 4n$, for all $n >= 2$, $\therefore 3n + 2 = O(n)$
- $f(n) = 10n^2+4n+2$
  - $10n^2+4n+2 <= 11n^2$, for all $n >= 5$, $\therefore 10n^2+4n+2 = O(n^2)$

علامت گذاری مجانبی O، Ω، Θ

- ✓ تعریف o:
  - f(n)=o(g(n)) اگر و فقط اگر برای هر ثابت حقیقی مثبتی C یک عدد N وجود داشته باشند به طوری که به ازای تمامی مقادیر n و n>=N، f(n)<cg(n) باشد.
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) کوچکتر از g(n) خواهد بود.

علامت گذاری مجانبی O، Ω، Θ

- تعریف امگا Ω:
  - $f(n)= \Omega(g(n))$ می باشد اگر و فقط اگر به ازای مقادیر ثابت مثبت c و n_0 ، برای تمام مقادیر n به شرطی که $n \geq n_0$ باشد داشته باشیم $f(n) \geq cg(n)$
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) حداقل (بزرگتر یا مساوی) g(n) خواهد بود.
  - مثال
- $f(n) = 3n+2$
  - $3n + 2 >= 3n$, for all $n >= 1$, $\therefore 3n + 2 = \Omega(n)$
- $f(n) = 10n^2+4n+2$
  - $10n^2+4n+2 >= n^2$, for all $n >= 1$, $\therefore 10n^2+4n+2 = \Omega(n^2)$

علامت گذاری مجانبی O، Ω، Θ

- تعریف تعریف امگای کوچک ω:
  - برای تابع پیچیدگی g(n)، ω(g(n)) شامل مجموعه ای از توابع پیچیدگی f(n) می باشد که برای آنها برای هر ثابت حقیقی مثبتی C یک عدد صحیح مثبت n0 وجود دارد به قسمی که برای تمام مقادیر n که n≥n0 باشد داشته باشیم f(n)>cg(n)
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) بزرگتر از g(n) خواهد بود.

علامت گذاری مجانبی O، Ω، Θ

- تعریف تتا [Theta]
  - f(n) = θ(g(n)) می باشد اگر و فقط اگر به ازای مقادیر ثابت c1 و c2 و n0، برای تمام مقادیر n>=n0 داشته باشیم c1g(n)<= f(n) <= c2g(n).
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) برابر از g(n) خواهد بود.
  - مثال
    - $f(n) = 3n+2$
      - $3n <= 3n + 2 <= 4n$, for all $n >= 2$, $\therefore 3n + 2 = \Theta(n)$
    - $f(n) = 10n^2+4n+2$
      - $n^2 <= 10n^2+4n+2 <= 11n^2$, for all $n >= 5$, $\therefore 10n^2+4n+2 = \Theta(n^2)$

علامت گذاری مجانبی O، Ω، Θ

- نشانه گذاری تتا از دو نشانه گذاری ذکر شده O و امگا دقیق تر می باشد . f(n) = Θ(g(n)) می باشد اگر و فقط اگر g(n) هم به عنوان کرانه بالا و هم به عنوان کرانه پایین در f(n) باشد.

علامت گذاری مجانبی O، Ω، Θ

- قضیه

$f(n) = a_m n^m + \ldots + a_1 n + n_0 \Rightarrow f(n) = O(n^m)$

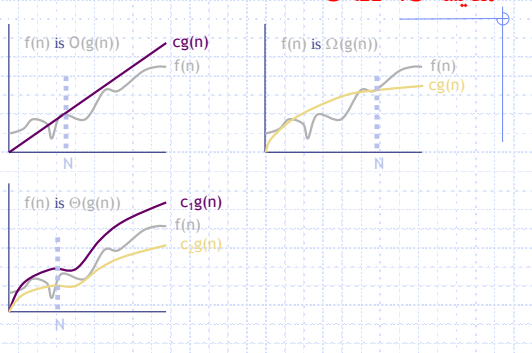
$f(n) = a_m n^m + \ldots + a_1 n + a_0 \Rightarrow f(n) = \Omega(n^m)$
$a_m > 0$

$f(n) = a_m n^m + \ldots + a_1 n + a_0 \Rightarrow f(n) = \Theta(n^m)$
$a_m > 0$

نمونه هایی از توابع رشد

تابع رشد
O(1)
$O(\log_2 N)$
O(N)
$O(N \log_2 N)$
$O(N^2)$
$O(2^N)$
O(N!)
$O(N^k)$

مقایسه O، Ω، Θ

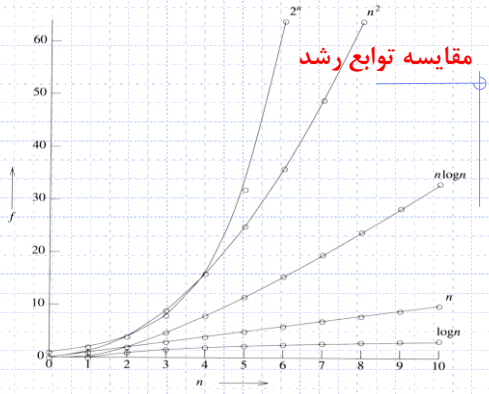


مقایسه توابع رشد

	Time for $f(n)$ instructions on a 10^9 instr/sec computer						
n	$f(n)=n$	$f(n)=\log_2 n$	$f(n)=n^2$	$f(n)=n^3$	$f(n)=n^4$	$f(n)=n^{10}$	$f(n)=2^n$
10	.01μs	.03μs	.1μs	1μs	10μs	10sec	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84hr	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1sec
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121.36d	18.3min
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1yr	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171yr	$4*10^{13}$yr
1,000	1.00μs	9.96μs	1ms	1sec	16.67min	$3.17*10^{13}$yr	$32*10^{283}$yr
10,000	10.00μs	130.03μs	100ms	16.67min	115.7d	$3.17*10^{23}$yr	
100,000	100.00μs	1.66ms	10sec	11.57d	3171yr	$3.17*10^{33}$yr	
1,000,000	1.00ms	19.92ms	16.67min	31.71yr	$3.17*10^7$yr	$3.17*10^{43}$yr	

μs = microsecond = 10^{-6} seconds
ms = millisecond = 10^{-3} seconds
sec = seconds
min = minutes
hr = hours
d = days
yr = years

مقایسه توابع رشد



تمرین

- پیچیدگی زمانی برنامه های زیر را به دست آورید

برنامه های ۱-۹، ۱-۱۰، ۱-۱۱، ۱-۲۲

تا اینجا

روابط بازگشتی را بعدا خواهم گفت