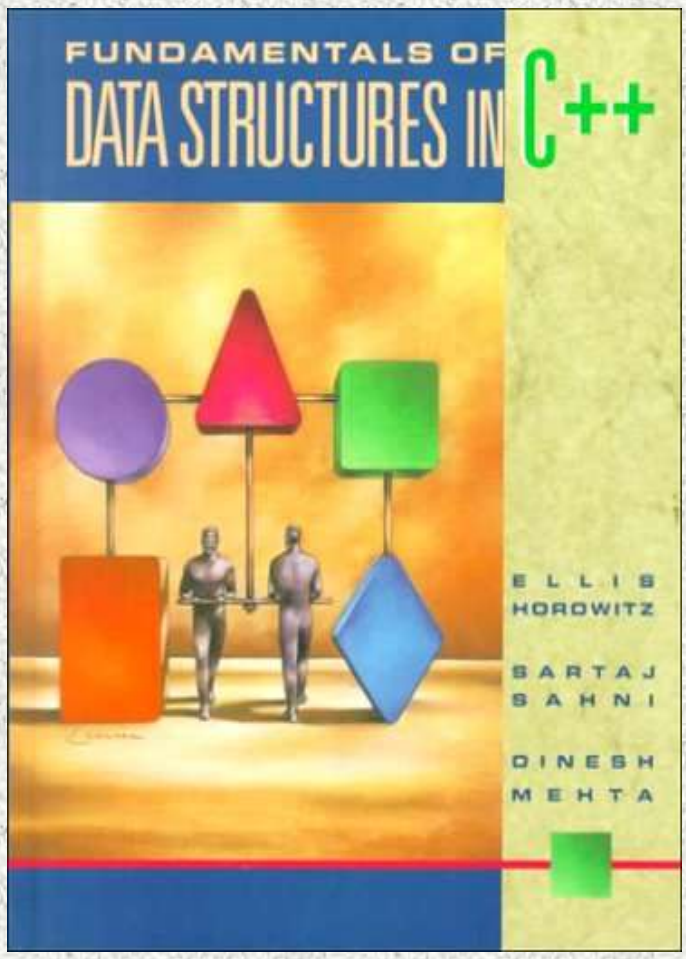
ساختمان داده ها در C++

Data Structure in C++

یکماز

دانشگاه جامع علمی کاربردی

منبع



- **عنوان منبع:**

  ساختمان داده ها در C++

- **مترجم:**

  حسین ابراهیم زاده ی قلزم

- **انتشارات:**

  سیمای دانش

- **منبع اصلی:**

  data Structure in C++

  Horowitz Ellis

بارم بندی

- پایان ترم: ۱۵ نمره
- تمرینات و فعالیت عملی: ۵ نمره
- حضور در کلاس: ۲ نمره

فصل اول : تحلیل الگوریتم ها

اهداف

- آشنایی با الگوریتم ها
- برنامه های بازگشتی
- محاسبه پیچیدگی زمانی و مرتبه اجرایی

۱–۱ مقدمه

روش های تحلیل سیستم و برنامه نویسی:

- پایین به بالا
- بالا به پایین

۲–۱ تعریف الگوریتم

مجموعه محدودی از دستورالعمل ها که
اگر دنبال شوند، موجب انجام فعالیت خاصی می گردد

۲-۱ شرایط الگوریتم

- **ورودی:** یک الگوریتم می تواند هیچ یا چندین کمیت ورودی داشته باشدکه از محیط خارج تامین می شود.
- **خروجی:** الگوریتم بایستی حداقل یک کمیت به عنوان خروجی داشته باشد.
- **قطعیت:** هر دستورالعمل باید واضح و بدون ابهام باشد.
- **محدودیت:**الگوریتم باید پس از طی مراحل محدودی خاتمه یابد.
- **کارایی:** هر دستورالعمل باید انجام پذیر باشد

۳-۱ ساختمان داده ها

ساختارهایی که جهت دریافت داده های خام به شکل مناسب توسط کامپیوتر و پیاده سازی و اجرای الگوریتم های مختلف روی آنها مورد استفاده قرار می گیرند

۳-۱ ساختمان داده ها

۴-۱ الگوریتم بازگشتی

- مجموعه ای از دستورات مختلف که عملی منطقی را انجام میدهند میتوانند در کنار هم قرارگرفته و **تابع** را تشکیل دهند
- نام تابع و پارامترهای آن بعنوان **دستوری جدید** درنظر گرفته میشوند
- با تعیین مشخصات ورودی–خروجی یک تابع میتوانیم به تابع دسترسی پیدا کنیم از اینرو **لازم نیست بدانیم که تابع چگونه کارش را انجام میدهد**
- با داشتن چنین تصویری از تابع، ابتدا تابع فراخوانی میشود آنگاه اجرا شده و کنترل اجرا به مکان مربوطه در تابع فراخوانی کننده باز میگردد.
- توابع میتوانند قبل از اتمام کارشان خودشان را فراخوانی کنند **(توانمندی لازم برای الگوریتم های بازگشتی)**

۴-۱ الگوریتم بازگشتی

- تابع مجموعه ای از دستورات است که تحت یک نام در برنامه تعریف شده است.
- توابع میتوانند یکدیگر را فراخوانی نمایند.
- توابع می توانند خودشان را صدا بزنند (بازگشت مستقیم)
- همچنین توابع می توانند توابعی که تابع فراخواننده را صدا میزنند، فراخوانی نمایند (بازگشتی غیرمستقیم) .

۴–۱ مثال الگوریتم فاکتوریل

برنامه ۱۲–۱

```
#include<iostream>
using namespace std;
int factorial(int n);
int main()
{
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    cout << "Factorial of " << n << " = " << factorial(n);
     return 0;
}
int factorial(int n)
{
    if(n > 1)
        return n * factorial(n - 1);
    else
        return 1;
}
```

۴-۱ الگوریتم بازگشتی

- ویژگی های توابع بازگشتی
  - فراخوانی تابع توسط خود تابع (اغلب با کاهش مقدار آرگومان ها)
  - شرط اتمام فراخوانی
- استفاده از حافظه پشته(Stack)

۴-۱ میزان حافظه یا پیچیدگی فضای یک برنامه

- میزان حافظه یا پیچیدگی فضای یک برنامه مقدارحافظه مورد نیاز برای اجرای کامل یک برنامه است.

- میزان یا پیچیدگی زمان یک برنامه مقدار زمان کامپیوتر است که برای اجرای کامل برنامه لازم است.

۵-۱ تحلیل پیچیدگی الگوریتم

- بازدهی الگوریتم ها به عوامل زیادی بستگی دارد
- به طور کلی با افزایش اندازه ورودی زمان اجرا افزایش می یابد
- زمان اجرا متناسب با تعداد دفعاتی است که عملیات اصلی انجام می شود
- بنابر این بازدهی الگوریتم را با تعیین تعداد دفعات انجام عمل اصلی به عنوان تابعی از ورودی تحلیل می کنیم

Performance Analysis

Iterative function to sum a list of numbers

Statement	s/e	Frequency	Total steps
float sum(float list[ ], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for(i=0; i <n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
Total			2n+3

۵-۱علامت گذاری مجانبی(Asymptotic) (Θ،Ω،O)

انگیزه برای تعیین تعداد مراحل، قابلیت مقایسه پیچیدگی دو برنامه که یک تابع را اجرا می کنند و پیش بینی رشد و افزایش زمان اجرا با تغییر مشخصه های موردی می باشد.

$$f(n) = O(g(n)) \Leftrightarrow \exists c, n_0 > 0 \quad \forall n \geq n_0 \quad f(n) \leq cg(n)$$

f(n)=O(g(n)) (**f(n)** از مرتبه **g(n)**) است اگر و فقط اگر به ازای مقادیر ثابت n_0 و **c** برای تمامی مقادیر $n \geq$ **n0** مقدار **f(n)** کمتر یا مساوی **cg(n)** باشد

۴-۱علامت گذاری مجانبی(O)، Ω (Asymptotic)، Θ

- O(1) : زمان محاسبه ثابتی را نشان می دهد
- O(n) : یک تابع خطی نامیده می شود
- $O(n^2)$: تابع درجه دو نامیده می شود

قضیه :

اگر $f(n) = a_m n^m + \ldots + a_1 n + n_0$ باشد، بنابراین $f(n) = O(n^m)$ خواهد بود

$$O(n^n) > \ldots O(2^n) > \ldots O(n^3) > O(n^2)$$
$$> O(n) > O(\log(n)) > O(1)$$

رابطه بالا نشان میدهد که در صورتی که برای یک مساله راه حلی با پیچیدگی $O(n)$ داشته باشیم رشد آن بسیار کمتر از راه حلی است که پیچیدگی آن $O(n^2)$ است.

۴–۱ علامت گذاری مجانبی(O)، Ω (Asymptotic)، Θ

$$f(n)=\Omega(g(n)) \Leftrightarrow \exists c, n_0 > 0 \quad \forall n \geq n_0 \quad f(n) \geq cg(n)$$

تعریف:[امگا]: f(n)=Ω(g(n)) می باشد(خوانده می شود f(n) امگای g(n) اگر و فقط اگر به ازای مقادیر ثابت مثبت c و n0 ، f(n)≥cg(n) باشد(برای تمام مقادیر n به شرطی که n≥n0 باشد)

قضیه :

اگر $f(n)=a_m n^m+\ldots+a_1 n+a_0$ **باشد، و** $a_m > 0$

بنابراین $f(n)=\Omega(n^m)$ **خواهد بود**

۴–۱ علامت گذاری مجانبی(O، Ω، Θ)(Asymptotic)

$$f(n) = \Theta(g(n)) \Leftrightarrow \exists c_1, c_2, n_0 > 0 \quad \forall n \geq n_0 \quad c_1 g(n) \leq f(n) \leq c_2 g(n)$$

تعریف تتا[Theta] :f(n) = θ(g(n)) می باشد (خوانده می شود f(n) تتای g(n)) اگر و فقط اگر به ازای مقادیر ثابت c_1 و c_2 و n_0،

$c_1 g(n) <= f(n) <= c_2 g(n)$ وجود داشته باشد (برای تمام مقادیر $n >= n_0$)

نشانه گذاری تتا از دو نشانه گذاری ذکر شده " big oh " 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.

۴–۱ علامت گذاری مجانبی(O)، Ω، Θ)(Asymptotic)،

قضیه :

اگر $f(n) = a_m n^m + \ldots + a_1 n + a_0$ **باشد، و** $a_m > 0$

بنابراین $f(n) = \Theta(n^m)$ **خواهد بود**

۴-۱ مثال(پیچیدگی جمع ماتریس ها)

Statement	Asymptotic complexity

Void add (int a []MAX-SIZE...)
{
int i،j;
for(i=0;i<rows ; i++)
for(j=0 ; j<cols ; j++)
C[i][j]=a[i][j]+b[i][j];
}
	0
	0
	0
	Θ(rows)
	Θ(rows.cols)
	Θ(rows.cols)
	0

Total	Θ(rows.cols)

Performance Analysis

- Theorem 1.2**:**
  - **If $f(n) = a_m n^m + \ldots + a_1 n + a_0$, then $f(n) = O(n^m)$.**
- Theorem 1.3:
  - **If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Omega(n^m)$.**
- Theorem 1.4:
  - **If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Theta(n^m)$.**

- Examples
  - $f(n) = 3n+2$
    - $3n + 2 <= 4n$, for all $n >= 2$ ➔ $3n + 2 = O(n)$

      $3n + 2 >= 3n$, for all $n >= 1$ ➔ $3n + 2 = \Omega(n)$

      $3n <= 3n + 2 <= 4n$, for all $n >= 2$ ➔ $3n + 2 = \Theta(n)$
  - $f(n) = 10n^2+4n+2$
    - $10n^2+4n+2 <= 11n^2$, for all $n >= 5$ ➔ $10n^2+4n+2 = O(n^2)$

      $10n^2+4n+2 >= n^2$, for all $n >= 1$ ➔ $10n^2+4n+2 = \Omega(n^2)$

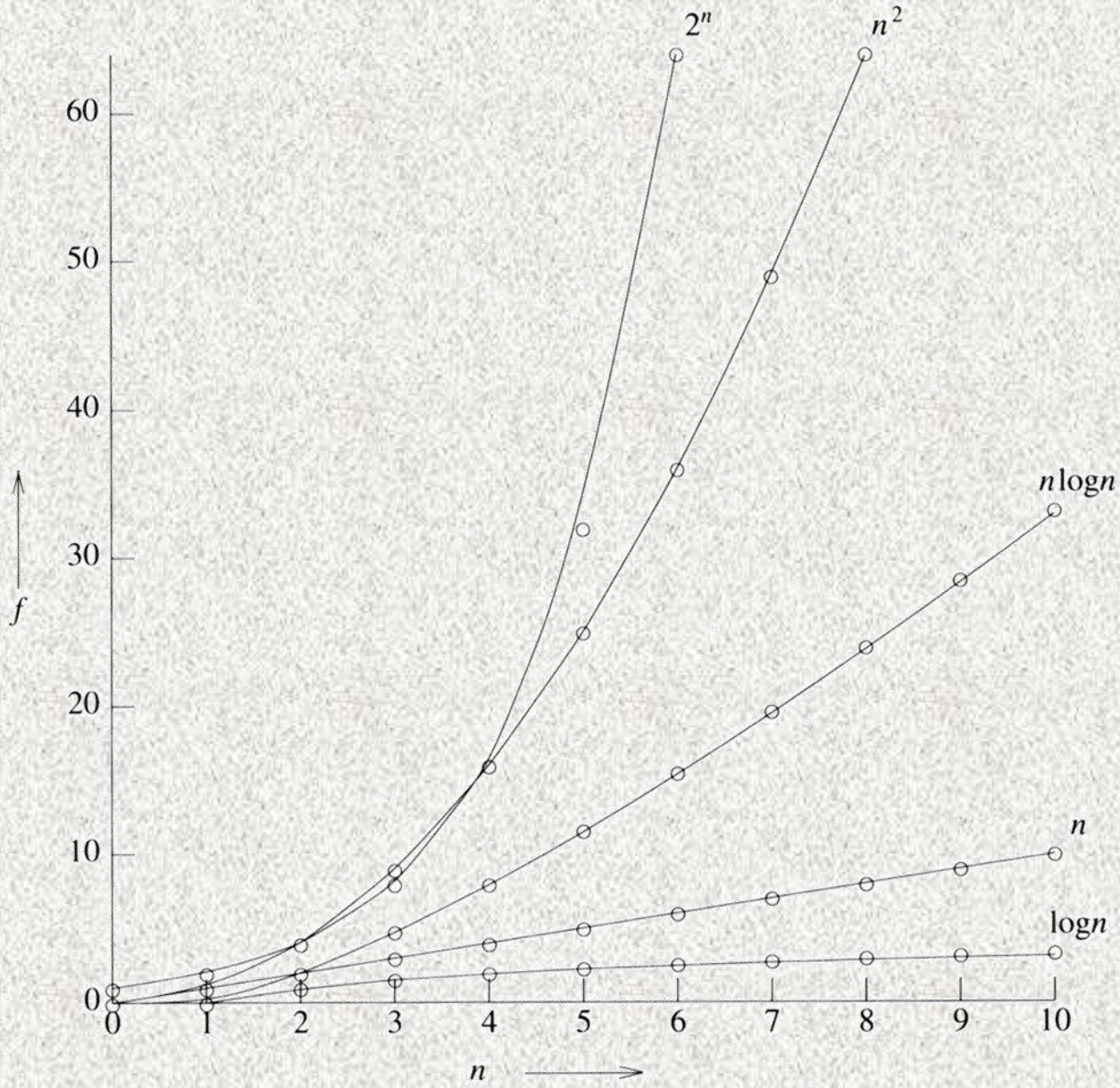
      $n^2 <= 10n^2+4n+2 <= 11n^2$, for all $n >= 5$ ➔ $10n^2+4n+2 = \Theta(n^2)$
  - $100n+6=O(n)$ /* $100n+6 \leq 101n$ for $n \geq 10$ */
  - $10n^2+4n+2=O(n^2)$ /* $10n^2+4n+2 \leq 11n^2$ for $n \geq 5$ */
  - $6*2^n+n^2=O(2^n)$ /* $6*2^n+n^2 \leq 7*2^n$ for $n \geq 4$ */

نمایش میزان رشد توابع با توجه به تغییر مشخصه ورودی

Instance characteristic n							
Time	Name	1	2	4	8	16	32
1	Constant	1	1	1	1	1	1
$\log n$	Logarithmic	0	1	2	3	4	5
n	Linear	1	2	4	8	16	32
$n \log n$	Log linear	0	2	8	24	64	160
n^2	Quadratic	1	4	16	64	256	1024
n^3	Cubic	1	8	64	512	4096	32768
2^n	Exponential	2	4	16	256	65536	4294967296
$n!$	Factorial	1	2	24	40326	20922789888000	26313×10^{33}

Figure 1.7 Function values

Performance Analysis



1.9 Times on a 1 billion instruction per second computer

	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