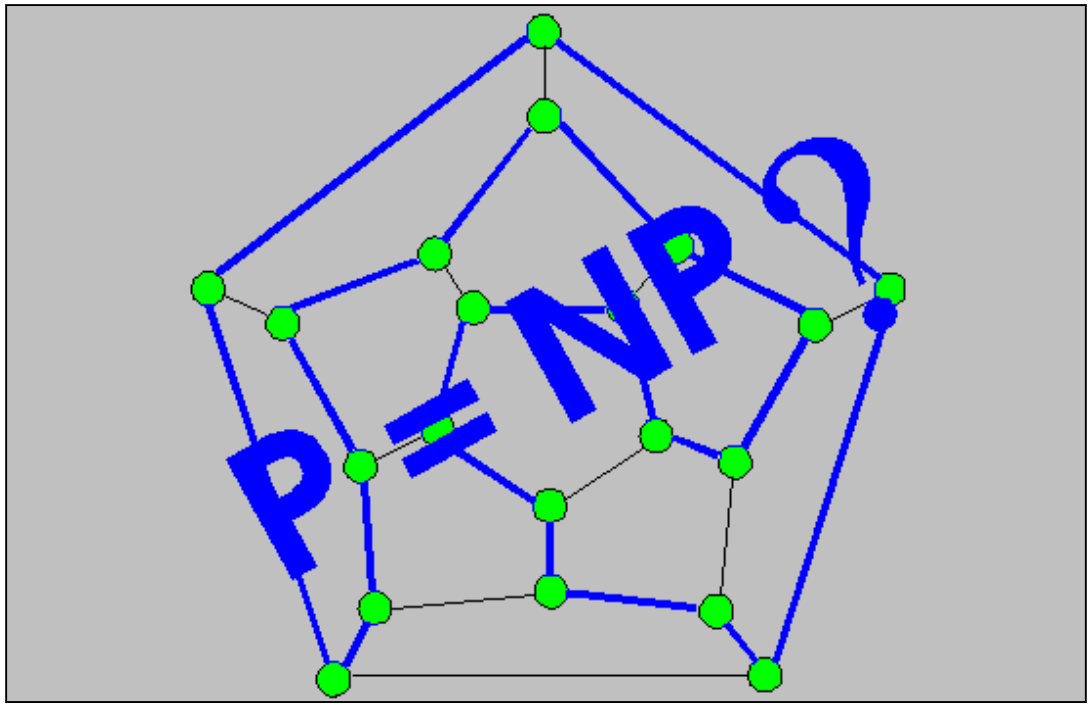
算法设计与分析

第1章　算法概述

计算机算法设计与分析



王晓东　编著

电子工业出版社

第1章 算法概述

学习要点:

- 理解算法的概念。
- 理解什么是程序，程序与算法的区别和内在联系。
- 掌握算法的计算复杂性概念。
- 掌握算法渐近复杂性的数学表述。
- 掌握用编程语言描述算法的方法。

算法(Algorithm)

- 算法（Algorithm）是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制。也就是说，**能够对一定规范的输入，在有限时间内获得所要求的输出**。如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用**不同的时间、空间**或效率来完成同样的任务。一个算法的优劣可以用**空间复杂度与时间复杂度**来衡量。

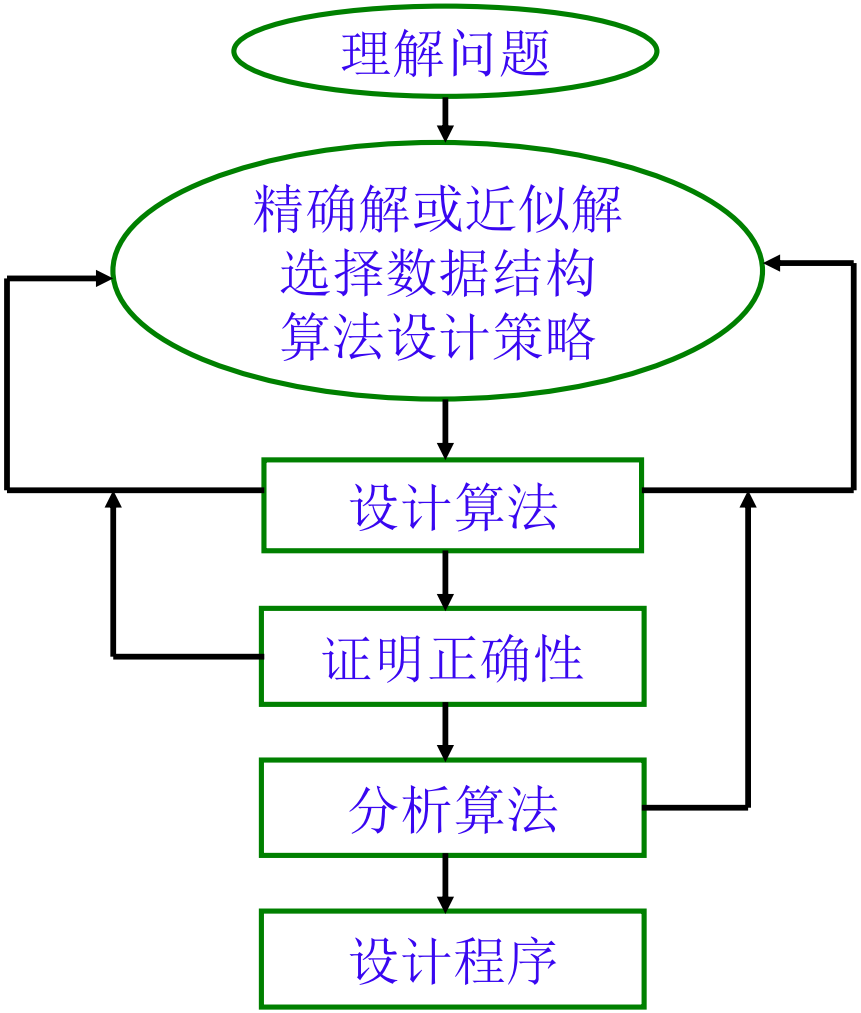
算法(Algorithm)

- 算法是指解决问题的一种方法或一个过程。
- 算法是若干指令的有穷序列，满足性质：
- (1)**输入**：有外部提供的量作为算法的输入。
- (2)**输出**：算法产生至少一个量作为输出。
- (3)**确定性**：组成算法的每条指令是清晰，无歧义的。
- (4)**有限性**：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

程序(Program)

- 程序是算法用某种程序设计语言的具体实现。
- **程序可以不满足算法的性质(4)**。
- 例如操作系统，是一个在无限循环中执行的程序，因而不是一个算法。
- 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

问题求解(Problem Solving)



算法复杂性分析

- 算法复杂性 = 算法所需要的计算机资源
- 算法的时间复杂性$T(n)$;
- 算法的空间复杂性$S(n)$。
- 其中n是问题的规模（输入大小）。

算法的时间复杂性

- （1）**最坏情况**下的时间复杂性
- $T_{max}(n) = \max\{ T(I) \mid size(I)=n \}$
- （2）**最好情况**下的时间复杂性
- $T_{min}(n) = \min\{ T(I) \mid size(I)=n \}$
- （3）**平均情况**下的时间复杂性
- $T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$
- 其中I是问题的规模为n的实例，$p(I)$是实 例I出现的概率。

算法渐近复杂性

- $T(n) \to \infty$ ， as $n \to \infty$;
- $(T(n) - t(n)) / T(n) \to 0$ ， as $n \to \infty$;
- $t(n)$是$T(n)$的渐近性态，为算法的渐近复杂性。
- 在数学上， $t(n)$是$T(n)$的渐近表达式，是$T(n)$略去低阶项留下的主项。它比$T(n)$ 简单。

渐近分析的记号

- 在下面的讨论中，对所有n，$f(n) \geq 0$，$g(n) \geq 0$。
- （1）**渐进上界记号O**
- $O(g(n))$ = { $f(n)$ | 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq f(n) \leq cg(n)$ }
- （2）**渐进下界记号Ω**
- $\Omega(g(n))$ = { $f(n)$ | 存在正常数c和n_0使得对所有$n \geq n_0$有：$0 \leq cg(n) \leq f(n)$ }

- （3）非紧上界记号o
- $o(g(n))$ = { $f(n)$ | 对于任何正常数$c>0$，存在正数和$n_0>0$使得对所有$n \geq n_0$有：$0 \leq f(n) < cg(n)$ }
- 等价于 $f(n) / g(n) \to c$ ， as $n \to \infty$。
- （4）非紧下界记号ω
- $\omega(g(n))$ = { $f(n)$ | 对于任何正常数$c>0$，存在正数和$n_0>0$使得对所有$n \geq n_0$有：$0 \leq cg(n) < f(n)$ }
- 等价于 $f(n) / g(n) \to \infty$ ， as $n \to \infty$。
- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

- （5）紧渐近界记号Θ
- $\Theta(g(n)) = \{ f(n) \mid$ 存在正常数c_1, c_2和n_0使得对所有$n \geq n_0$有：$c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

- **定理1：** $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

渐近分析记号在等式和不等式中的意义

- $f(n)= \Theta(g(n))$的确切意义是：$f(n) \in \Theta(g(n))$。
- 一般情况下，等式和不等式中的渐近记号$\Theta(g(n))$表示$\Theta(g(n))$中的某个函数。
- 例如：$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ 表示
- $2n^2 +3n +1=2n^2 + f(n)$，其中$f(n)$ 是$\Theta(n)$中某个函数。
- 等式和不等式中渐近记号O,o, Ω和ω的意义是类似的。

渐近分析中函数比较

- $f(n)= O(g(n)) \approx a \leq b$;
- $f(n)= \Omega(g(n)) \approx a \geq b$;
- $f(n)= \Theta(g(n)) \approx a = b$;
- $f(n)= o(g(n)) \approx a < b$;
- $f(n)= \omega(g(n)) \approx a > b$.

渐近分析记号的若干性质

- （1）传递性：
- $f(n)=\Theta(g(n))$，$g(n)=\Theta(h(n)) \Rightarrow f(n)=\Theta(h(n))$；
- $f(n)=O(g(n))$，$g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$；
- $f(n)=\Omega(g(n))$，$g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$；
- $f(n)=o(g(n))$，$g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$；
- $f(n)=\omega(g(n))$，$g(n)=\omega(h(n)) \Rightarrow f(n)=\omega(h(n))$；

- （2）反身性：
- $f(n)= \Theta(f(n))$；
- $f(n)= O(f(n))$；
- $f(n)= \Omega(f(n))$.
- （3）对称性：
- $f(n)= \Theta(g(n)) \Leftrightarrow g(n)= \Theta(f(n))$.
- （4）互对称性：
- $f(n)= O(g(n)) \Leftrightarrow g(n)= \Omega(f(n))$ ；
- $f(n)= o(g(n)) \Leftrightarrow g(n)= \omega(f(n))$ ；

- **（5）算术运算：**
- $O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$；
- $O(f(n))+O(g(n)) = O(f(n)+g(n))$；
- $O(f(n))*O(g(n)) = O(f(n)*g(n))$；
- $O(cf(n)) = O(f(n))$；
- $g(n)= O(f(n)) \Rightarrow O(f(n))+O(g(n)) = O(f(n))$。

算法渐近复杂性分析中常用函数

- **（1）单调函数**
- 单调递增：$m \le n \Rightarrow f(m) \le f(n)$;
- 单调递减：$m \le n \Rightarrow f(m) \ge f(n)$;
- 严格单调递增：$m < n \Rightarrow f(m) < f(n)$;
- 严格单调递减：$m < n \Rightarrow f(m) > f(n)$.
- **（2）取整函数**
- $\lfloor x \rfloor$：不大于x的最大整数；
- $\lceil x \rceil$：不小于x的最小整数。

取整函数的若干性质

- $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$；
- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$;
- 对于$n \ge 0$，$a,b>0$，有：
- $\lceil \lceil n/a \rceil /b \rceil = \lceil n/ab \rceil$;
- $\lfloor \lfloor n/a \rfloor /b \rfloor = \lfloor n/ab \rfloor$;
- $\lceil a/b \rceil \le (a+(b-1))/b$;
- $\lfloor a/b \rfloor \ge (a-(b-1))/b$;
- $f(x)=\lfloor x \rfloor$, $g(x)=\lceil x \rceil$为单调递增函数。

- **（3）多项式函数**
- $p(n)= a_0+a_1n+a_2n^2+\ldots+a_dn^d$； $a_d>0$;
- $p(n) = \Theta(n^d)$;
- $f(n) = O(n^k) \Leftrightarrow f(n)$多项式有界;
- $f(n) = O(1) \Leftrightarrow f(n) \le c$;
- $k \ge d \Rightarrow p(n) = O(n^k)$;
- $k \le d \Rightarrow p(n) = \Omega(n^k)$;
- $k > d \Rightarrow p(n) = o(n^k)$;
- $k < d \Rightarrow p(n) = \omega(n^k)$.

- **（4）指数函数**
- 对于正整数m,n和实数$a>0$:
- $a^0=1$;
- $a^1=a$;
- $a^{-1}=1/a$;
- $(a^m)^n = a^{mn}$;
- $(a^m)^n = (a^n)^m$;
- $a^m a^n = a^{m+n}$;
- $a>1 \Rightarrow a^n$为单调递增函数;
- $a>1 \Rightarrow \lim_{n\to\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n)$

- **（5）对数函数**
- $\log n = \log_2 n$;
- $\lg n = \log_{10} n$;
- $\ln n = \log_e n$;
- $\log^k n = (\log n)^k$l;
- $\log \log n = \log(\log n)$;
- for a>0,b>0,c>0

$$a = b^{\log_b a}$$

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

- $|x| \le 1 \Rightarrow \quad \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \cdots.$
- for $x > -1$, $\frac{x}{1+x} \le \ln(1+x) \le x$
- for any $a > 0$, $\lim_{n\to\infty} \frac{\log^b n}{(2^a)^{\log n}} = \lim_{n\to\infty} \frac{\log^b n}{n^a} = 0$, $\Rightarrow \log^b n = o(n^a)$

- **（6）阶层函数**

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

$$n! = 1 \cdot 2 \cdot 3 \cdots n$$

- Stirling's approximation

$$n! = \sqrt{2\pi\, n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n} \qquad \frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$$

$$n! = o(n^n)$$

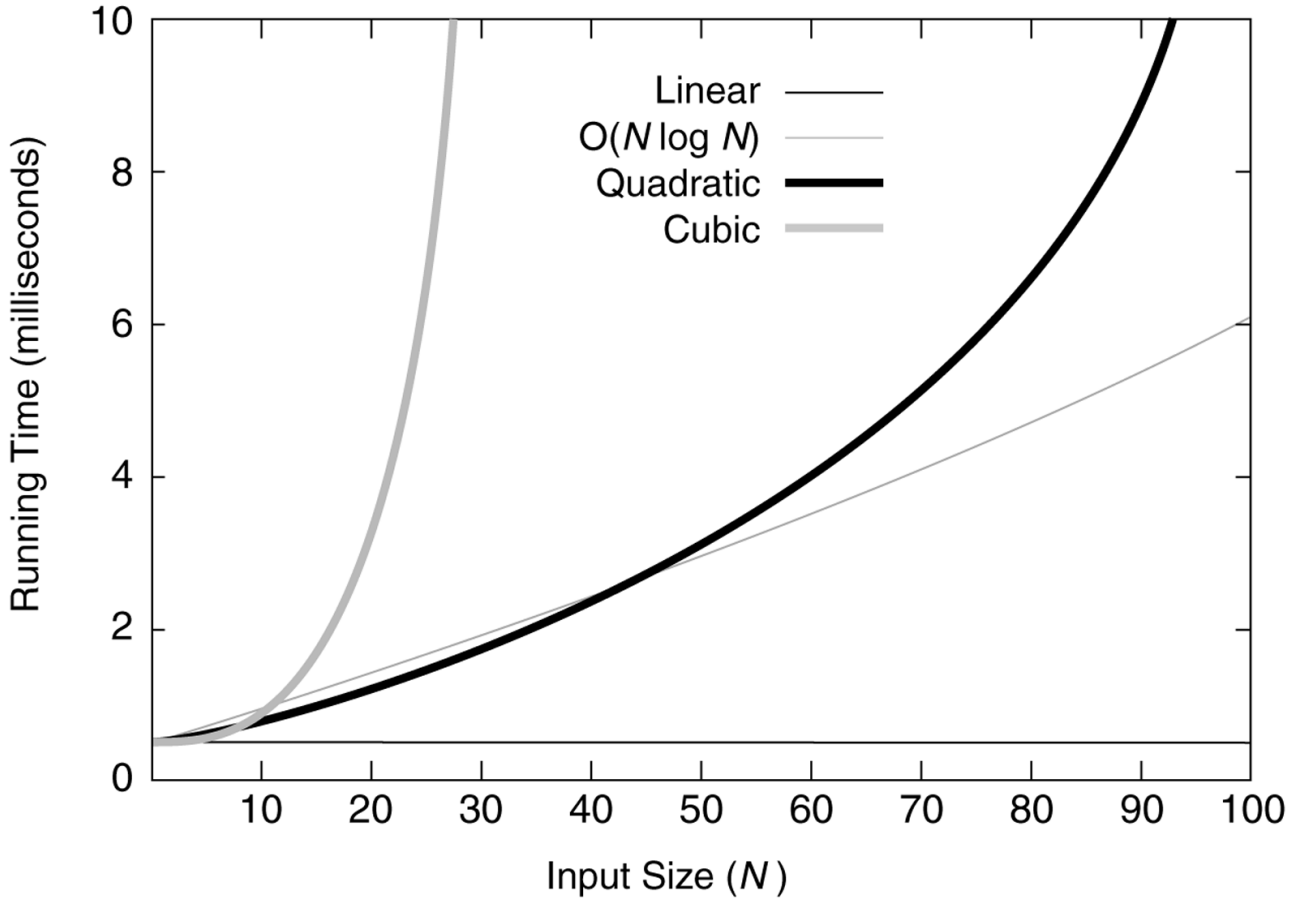
$$n! = \omega(2^n)$$

$$\log(n!) = \Theta(n \log n)$$

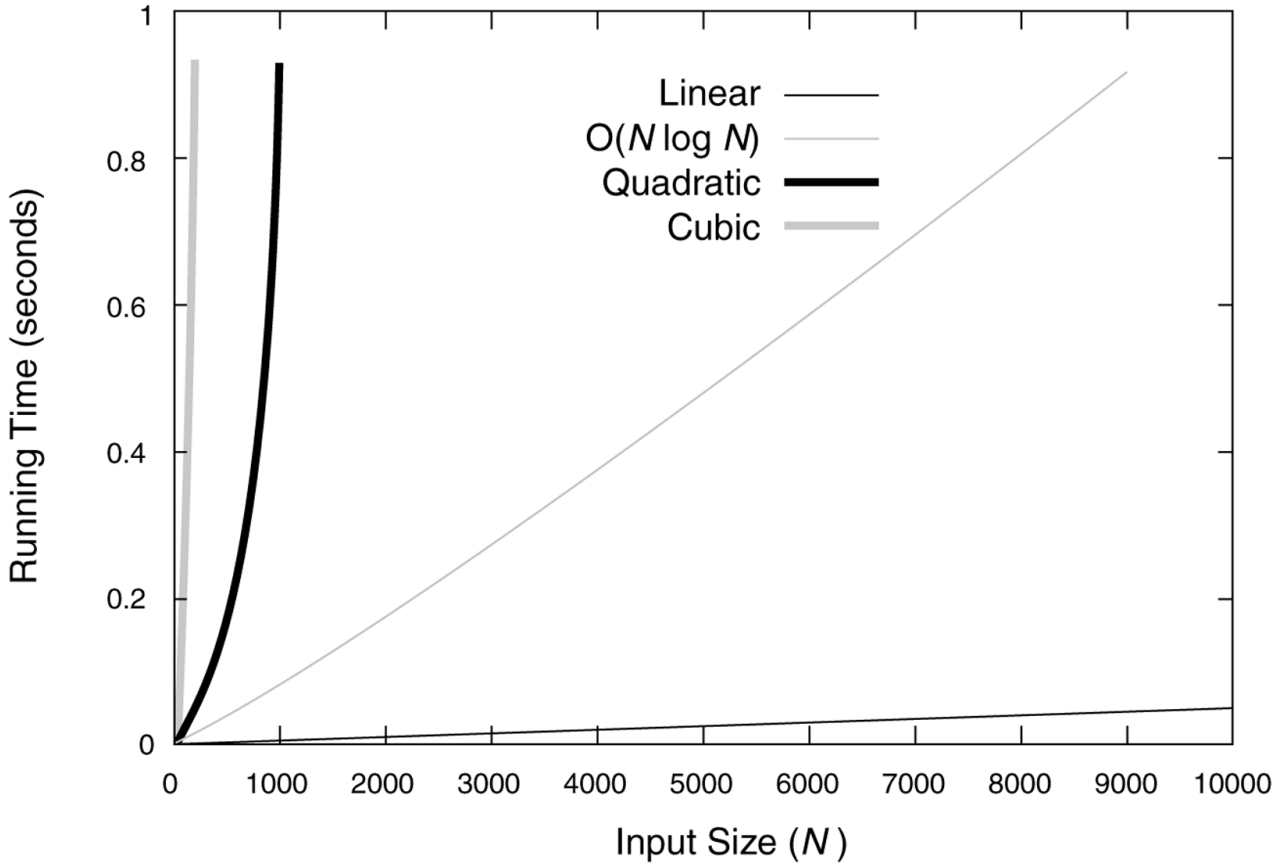
算法分析中常见的复杂性函数

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

小规模数据


Running Time (milliseconds)
Input Size (N)
Linear
O(N log N)
Quadratic
Cubic
0
2
4
6
8
10
10
20
30
40
50
60
70
80
90
100

中等规模数据


Linear
O(N log N)
Quadratic
Cubic
Running Time (seconds)
Input Size (N)
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

用c++描述算法

CATEGORY	EXAMPLES	ASSOCIATIVITY
Operations on References	`. []`	Left to right
Unary	`++ -- ! - (type)`	Right to left
Multiplicative	`* / %`	Left to right
Additive	`+ -`	Left to right
Shift (bitwise)	`<< >>`	Left to right
Relational	`< <= > >= instanceof`	Left to right
Equality	`== !=`	Left to right
Boolean (or bitwise) AND	`&`	Left to right
Boolean (or bitwise) XOR	`^`	Left to right
Boolean (or bitwise) OR	`|`	Left to right
Logical AND	`&&`	Left to right
Logical OR	`||`	Left to right
Conditional	`?:`	Right to left
Assignment	`= *= /= %= += -=`	Right to left

- **（1）选择语句：**
- **（1.1）if 语句：**

```
if (expression) statement;
else statement;
```

- **（1.2）？语句：**
-

```
exp1?exp2:exp3
y= x>9 ? 100:200;  等价于：
if (x>9) y=100;
else y=200;
```

（1.3) switch语句：

```
switch (expression) {
    case 1:
       statement sequence;
       break;
    case 2:
       statement sequence;
       break;
     …
    default:
       statement sequence;
 }
```

（2）迭代语句：

- **（2.1) for 循环：**
- for (init;condition;inc) statement;
- **（2.2) while 循环：**
- while (condition) statement;
- **（2.3) do-while 循环：**
- do{
- statement;
- } while (condition);

（3）跳转语句：

- **（3.1) return语句：**
- return expression;
- **（3.2) goto语句：**
- goto label;
- …
- label:

（4）函数：

```
return-type function name(para-list)
{
     body of the function
 }
```

- 例：

```
int max(int x,int y)
{
   return x>y?x:y;
}
```

（5）模板template：

```
template <class Type>
Type max(Type x,Type y)
{
   return x>y?x:y;
}

int i=max(1,2)；
double x=max(1.0,2.0)；
```

（6）动态存储分配：

- **（6.1）运算符new：**
- 运算符new用于动态存储分配。
- new返回一个指向所分配空间的指针。
- 例：int *x；y=new int；*y=10；
- 也可将上述各语句作适当合并如下：
- int *y=new int；*y=10；
- 或 int *y=new int(10)；
- 或 int *y；y=new int(10)；

（6.2）一维数组：

- 为了在运行时创建一个大小可动态变化的一维浮点数组x，可先将x声明为一个float类型的指针。然后用new为数组动态地分配存储空间。
- 例：
- float *x=new float[n];
- 创建一个大小为n的一维浮点数组。运算符new分配n个浮点数所需的空间，并返回指向第一个浮点数的指针。
- 然后可用x[0]，x[1]，…，x[n-1]来访问每个数组元素。

（6.3）运算符delete：

- 当动态分配的存储空间已不再需要时应及时释放所占用的空间。
- 用运算符delete来释放由new分配的空间。
- 例：
- delete y；
- delete [ ]x；
- 分别释放分配给*y的空间和分配给一维数组x的空间。

（6.4）动态二维数组：

- 创建类型为Type的动态工作数组，这个数组有rows行和cols列。

```
template <class Type>
void Make2DArray(Type** &x,int rows, int cols)
{
      x=new Type*[rows];
      for (int i=0;i<rows;i++)
          x[i]=new Type[cols];
}
```

- 当不再需要一个动态分配的二维数组时，可按以下步骤释放它所占用的空间。首先释放在for循环中为每一行所分配的空间。然后释放为行指针分配的空间。

```
template <class Type>
void Delete2DArray(Type** &x,int rows)
{
      for (int i=0;i<rows;i++)
         delete []x[i];
     delete []x;
     x=0;
}
```

- 释放空间后将x置为0，以防继续访问已被释放的空间。

算法分析方法

- 例：顺序搜索算法

```
template<class Type>
int seqSearch(Type *a, int n, Type k)
{
    for(int i=0;i<n;i++)
            if (a[i]==k) return i;
    return -1;
}
```

- （1）$T_{max}(n) = \max\{ T(I) \mid size(I)=n \} = O(n)$
- （2）$T_{min}(n) = \min\{ T(I) \mid size(I)=n \} = O(1)$
- （3）在平均情况下，假设：
- (a) 搜索成功的概率为p ($0 \le p \le 1$)；
- (b) 在数组的每个位置i ($0 \le i < n$)搜索成功的概率相同，均为p/n。

$$T_{avg}(n) = \sum_{size(I)=n} p(I)T(I)$$

$$= \left(1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + 3 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n} \right) + n \cdot (1 - p)$$

$$= \frac{p}{n} \sum_{i=1}^{n} i + n(1 - p) = \frac{p(n+1)}{2} + n(1 - p)$$

算法分析的基本法则

- **非递归算法：**
- （1）for / while 循环
- 循环体内计算时间*循环次数；
- （2）嵌套循环
- 循环体内计算时间*所有循环次数；
- （3）顺序语句
- 各语句计算时间相加；
- （4）if-else语句
- if语句计算时间和else语句计算时间的较大者。

```
template<class Type>
void insertion_sort(Type *a, int n)
{
    Type key;                              //   cost       times
    for (int i = 1; i < n; i++){           //    c1         n
        key=a[i];                          //    c2         n-1
        int j=i-1;                         //    c3         n-1
        while( j>=0 && a[j]>key ){         //    c4         sum of ti
            a[j+1]=a[j];                   //    c5         sum of (ti-1)
            j--;                           //    c6         sum og (ti-1)
            }
        a[j+1]=key;                        //    c7         n-1
       }
}
```

最优算法

- 问题的计算时间下界为$\Omega(f(n))$，则计算时间复杂性为$O(f(n))$的算法是最优算法。
- 例如，排序问题的计算时间下界为$\Omega(n\log n)$，计算时间复杂性为$O(n\log n)$的排序算法是最优算法。
- 堆排序算法是最优算法。

递归算法复杂性分析

```
int factorial(int n)
{
    if (n == 0) return 1;
    return n*factorial(n-1);
 }
```

$$T(n)=\begin{cases}0 & n=0\\ T(n-1)+1 & n>0\end{cases}$$

$$T(n)=n$$

NP完全理论

- 非确定性计算模型下的易验证问题类

哈密顿回路问题HAM-CYCLE

给定无向图G=（V，E），判定其是否含有一条哈密顿回路。