石家莊鐵道大學
SHIJIAZHUANG TIEDAO UNIVERSITY

在线开放课程

绪论

算法和算法分析

主讲 ： 刘辉

目录

一、算法的概念和特性

- 算法的概念
  - **一个有穷的指令集**，这些指令为解决某一特定任务规定了一个运算序列
- 算法的描述
  - 自然语言
  - 流程图
  - 程序设计语言
  - 伪语言

一、算法的概念和特性

- 算法的特性
  - **输入**　有0个或多个输入
  - **输出**　有一个或多个输出(处理结果)
  - **确定性**　每步定义都是确切、无歧义的
  - **有穷性**　算法应在执行有穷步后结束
  - **可行性**　算法中的所有操作都可以通过已经实现的基本运算执行有限次来实现

一、算法的概念和特性

- 算法的评价
  - 正确性
  - 可读性
  - 健壮性
  - 高效性

二、算法分析

算法效率的度量

- **时间复杂度**：用依据该算法编制的程序在计算机上执行**所消耗的时间**来度量，即**时间复杂度**。
- **算法的时间复杂度**越低则说明算法的效率越高

事后统计

事前分析估计

二、算法分析

- 事后统计
  - 利用计算机内的计时功能，不同算法的程序可以用一组或多组相同的统计数据区分
  - 一个高级语言程序在计算机上运行所消耗的时间取决于：

①依据的算法选用何种策略
②问题的规模
③程序语言
④编译程序产生机器代码质量
⑤机器执行指令速度

二、算法分析

事前分析估计

- 一个**算法的执行时间**大致等于其**所有语句执行时间的总和**，**一个语句的执行时间**则为该条语句的**重复执行次数**和**执行一次所需时间**的**乘积**。

所谓的**算法分析**并非精确统计算法实际执行所需时间，而是**针对算法中语句执行次数作出估计**。

二、算法分析

事前分析估计

- 一条语句的重复执行次数称为**语句的频度**。
- 设每条语句执行一次所需的时间均为**单位时间**，一个算法的执行时间就是该算法中所有**语句的频度之和**。

```
for(i=1;i<n;i++ )          //频度为n
  for(j=1;j<n;j++)         //频度为n*n
 {   c[i][j]=0;            //频度为n*n
     for(k=1;k<=n; k++)    //频度为n^2*n
        c[i][j]=c[i][j]+a[j][k]*b[k][j];  //频度为n^3
  }
```

算法执行时间：

$T(n)=2n^3+2n^2+n$

二、算法分析

算法的时间复杂度

- **算法的规模**：求解问题的输入量，用n表示；
- 决定算法规模n复杂性的函数，用f(n)表示；f(n)一般是**算法中最大的语句频度**，如循环或递归中最深层语句频度。
- **时间复杂度**：是算法的执行时间，也是n的函数，用T(n)表示；
- 当问题**规模** $n \to \infty$，T(n)的**数量级（称为算法的渐进时间复杂度）**的增长率与f(n)的增长率相同，因此有：

$$T(n)=O(f(n))$$

二、算法分析

算法的时间复杂度

- n * n阶矩阵加法：

```
for( i = 0; i < n; i++)
    for( j = 0; j < n; j++)
        c[i][j] = a[i][j] + b[i][j];
```

最深的语句频度是：
$f(n)=n*n;$
$T(n) = O(n^2)$

- 两个数据交换

```
t=x;   x=y; y=t;
```

最深的语句频度是：
$f(n)=1;$ $T(n) = O(1)$

二、算法分析

算法的时间复杂度

- 变量自增：

```
x=y=0;
for(i=1;i<=n;i++)   x++;
for(i=1;i<=n;i++)
    for(i=1;i<=n;i++)  y++;
```

最深的语句频度是：

$f(n)=n*n;$

$T(n)=O(n^2)$

二、算法分析

算法的时间复杂度是由：嵌套最深层语句的频度决定的

算法的时间复杂度

```
void exam ( float x[ ][ ], int m, int n ) {
    float sum [ ];
    for ( int i = 0; i < m; i++ ) {  //x中各行
        sum[i] = 0.0;                    //数据累加
        for ( int j = 0; j < n; j++ )
                sum[i] += x[i][j];  //关键操作
    }
    for ( i = 0; i < m; i++ )  //打印各行数据和
     cout << i << " : " <<sum [i] << endl; //关键操作
}
```

渐进时间复杂度为 $O(max(m*n, m))$

二、算法分析

算法的时间复杂度—例1

```
for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
    {  c[i][j]=0;
       for(k=1;k<=n;k++)
          c[i][j]=c[i][j]+a[i][k]*b[k][j];
    }
```

$$T(n)=O(n^3)$$

$$T(n)=\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}1=\sum_{i=1}^{n}\sum_{j=1}^{n}n=\sum_{i=1}^{n}n^2=n^3=o(n^3)$$

二、算法分析

算法的时间复杂度—例2

```
for( i=1; i<=n; i++)
    for (j=1; j<=i; j++)
        for (k=1; k<=j; k++)
            x=x+1;
```

$$T(n)=O(n^3)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}\frac{i(i+1)}{2}$$

$$=\frac{1}{2}\left(\sum_{i=1}^{n}i^2+\sum_{i=1}^{n}i\right)=\frac{1}{2}\left(\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right)$$

$$=\frac{n(n+1)(n+2)}{6}$$

二、算法分析

- 算法的时间复杂度—例3

```
i=1;          ①
while(i<=n)
    i=i*2;    ②
```

$$T(n)=O(\log_2 n)$$

$$2^{f(n)} \le n$$

即f(n)≤$\log_2 n$，取最大值f(n)=$\log_2 n$

二、算法分析

- 算法的时间复杂度—例4
  - 顺序查找，在数组a[i]中查找值等于e的元素，返回其所在位置

```
for (i=0;i< n;i++)
    if (a[i]==e) return i+1;
        return 0;
```

$$T(n)=O(n)$$

最好情况：1次
最坏情况：n
平均时间复杂度为:O(n)

二、算法分析

◆ 算法的时间复杂度

常数阶	对数阶	线性阶	线性对数阶	平方阶	立方阶	…	K次方阶	指数阶
$O(1)$	$O(\log_2 n)$	$O(n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n^3)$		$O(n^k)$	$O(2^n)$

复杂度低 → 复杂度高

指数阶算法的关系为：

$$O(2^n)<O(n!)<O(n^n)$$

当n取得很大时，指数阶算法和多项式时间算法在所需时间上非常悬殊

二、算法分析

空间复杂度

- 空间复杂度：算法所需**存储空间的度量**，记作：$S(n)=O(f(n))$ （其中n为问题的规模）

- 算法要占据的空间
  - 算法本身要占据的空间，输入/输出，指令，常数，变量等
  - 算法要使用的辅助空间
- 若输入数据所占空间和算法无关，则不考虑输入本身所占空间，否则应同时考虑。

小结

- 掌握算法的概念和特性
- 掌握算法分析的两个主要方面（时间复杂度、空间复杂度）
- 掌握算法的时间复杂度分析

谢谢！