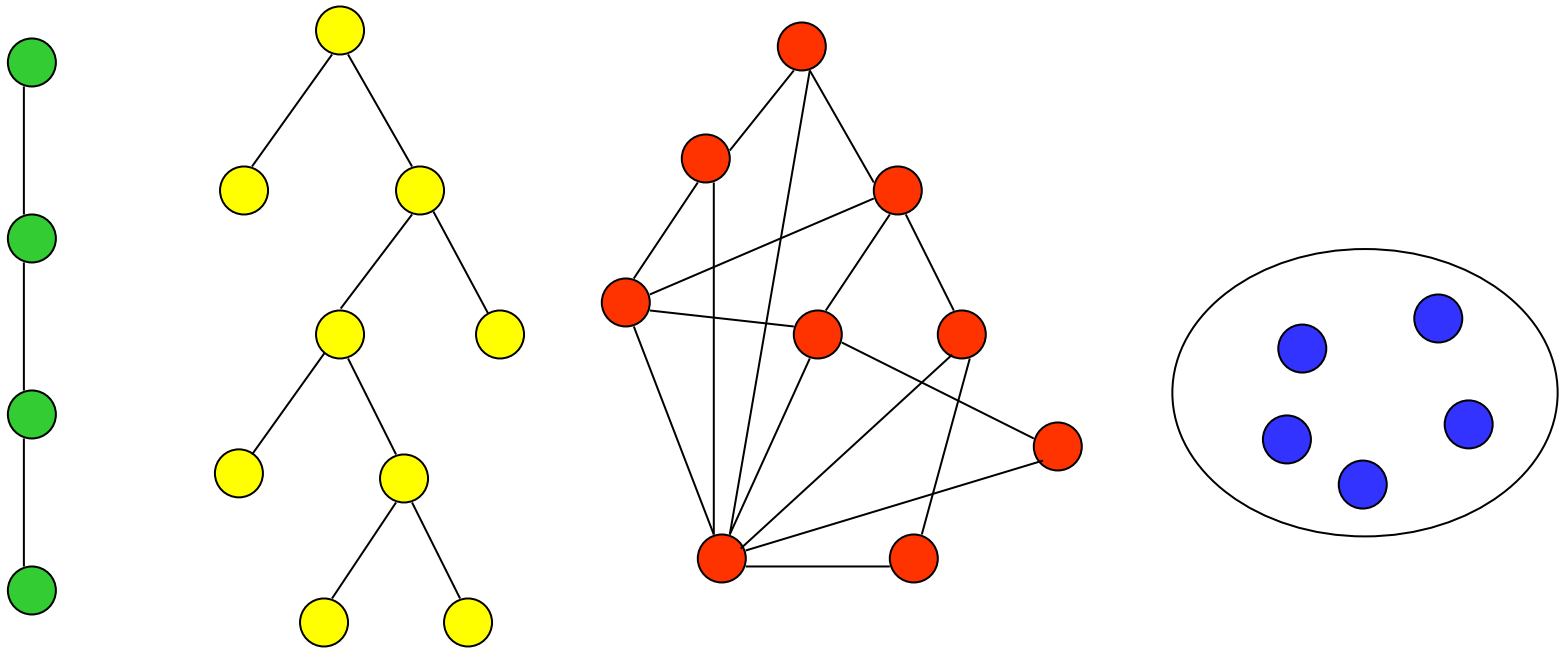
算法与数据结构



引 言

- （1）课程名称
  - 课程名称：算法与数据结构
  - 英文名称： **Algorithms and Data Structures**
- （2）课程性质
  - 《算法与数据结构》是计算机学科的技术基础和主干必修课。
- （3）基础知识要求
  - 课程《算法与数据结构》不需要特别的基础知识。
  - 已修本科数学分析(或高等数学)，离散数学和C语言程序设计，将有助于更好地理解课程内容。
- （4）课程教材
  - 《数据结构》(C语言版) （第3版）王晓东 电子工业出版社 2018

习题与实验

- (1)指导思想
  - 要求学生完成适量课后作业。
  - 课后作业需要学生综合运用教师在课堂上讲述的方法（包括思维方法），独立思考求解问题，以深化对课堂讲述内容的理解。
- (2)大作业和实验
  - 本课程是计算机科学与技术学科的专业基础课，教学难度较大。
  - 有的习题有较大难度和解题复杂度，这类习题中的各种算法的实现作为大作业和实验安排。

面向问题求解的实践教学模式

- 针对传统教学存在的问题，课程教学组经过调研、分析后决定开展课程实践教学改革，**构建面向问题求解的实践教学模式，以实际问题求解为主线索来组织和设计实践教学内容**，锻炼学生综合运用所学理论知识进行问题分析、设计和实现的能力，在实践中强化创新意识。
- 为了切实有效地推进《算法与数据结构》实践教学改革，我们专门开发了课程的网络教学系统（http://ds.fzu.edu.cn ），用于辅助课堂教学和实践教学。

面向问题求解的实践教学模式

面向问题求解的实践教学模式主要包括以下几个方面：

- **1 面向实际问题，精心设计实践教学内容**
- **2 作业自动评测**
- **3 解题报告交流**
- **4 优秀作业评选**

面向问题求解的实践教学模式

1 面向实际问题，精心设计实践教学内容

以问题求解为主线索，针对每个教学单元的重要知识点，精心设计若干有代表性的算法实验作业题，这些作业包括仅要求对数据结构和算法进行实现的验证型实验，以及要求运用所学理论知识设计求解典型或有趣实际问题的设计型实验和综合型实验。这些实验作业将以随机的方式分给学生，每位学生必须在规定的时间内独立完成。

面向问题求解的实践教学模式

2 作业自动评测

为了对大量的学生实验作业进行有效的评价，我们**开发作业自动评测系统，采取离线和在线作业自动评测相结合。**先用离线作业自动评测系统测试学生提交的作业，测出每次作业的成绩，并在网站上公布评测结果，**平时作业成绩将是期末成绩的重要组成部分。**成绩公布后，开放在线作业自动评测系统，作业成绩不理想的学生可再次改进自己的作业，并利用在线评测系统进行评测。

面向问题求解的实践教学模式

3 解题报告交流

我们把每道作业成绩前三名的学生定为优秀作业候选人，并要求他们准备解题报告。解题报告的内容主要包括：问题的思考过程、问题的数学化、数据结构和算法设计策略的选取、算法正确性的证明、算法复杂性的分析，算法的优化及改进、程序的设计与实现等。这些解题报告将在课程讨论课上交流。

面向问题求解的实践教学模式

4 优秀作业评选

解题报告交流后，我们将优秀作业候选人的解题报告和源程序都放到课程网站上，先由全班同学进行投票，然后由任课教师综合作业成绩、报告情况、解决问题算法的复杂性和创新性等几个方面来确定每道题的优秀作业，并对评上优秀作业的同学给予本次作业成绩双倍的奖励。

第1章 引论

1.1 问题求解

1.2《数据结构》研究什么

1.3《数据结构》课程的地位

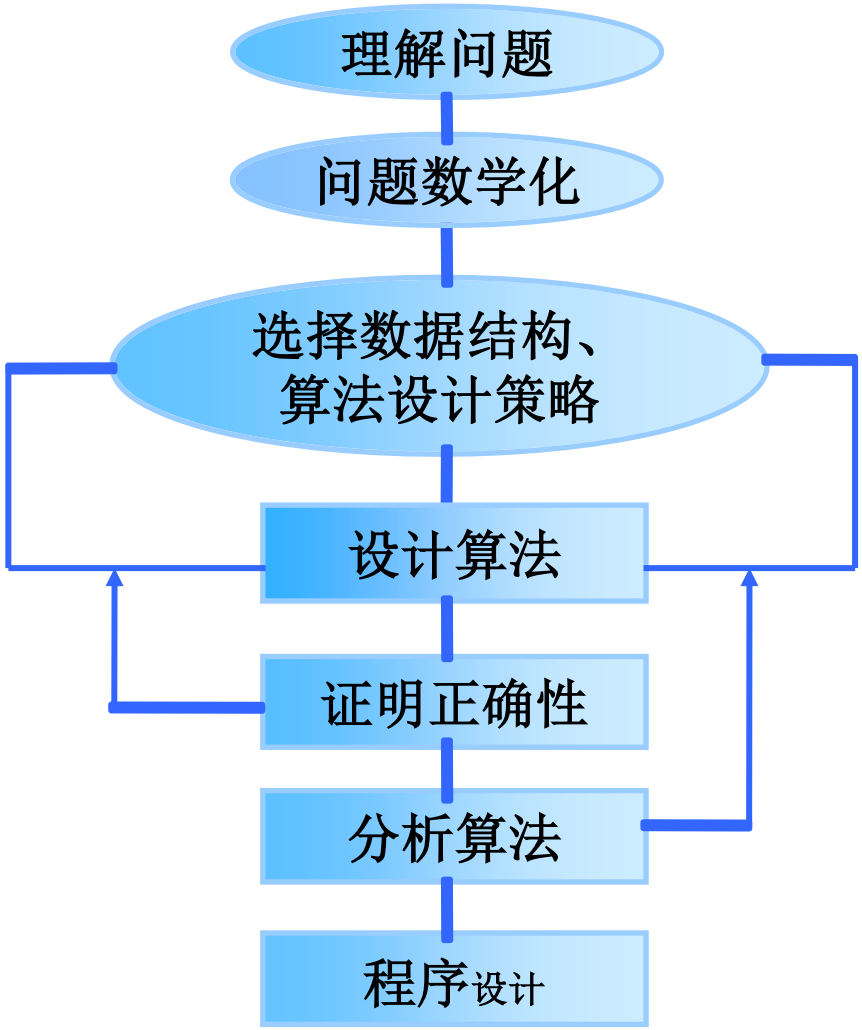
1.4 基本概念和术语

1.5 数据类型和抽象数据类型

1.6 算法和算法分析

1.7 用C语言描述数据结构与算法

1.1 问题求解(Problem Solving)



返回章目录

1.2《数据结构》研究什么

数据结构的研究内容：

(1) 要对所加工的对象进行逻辑组织。

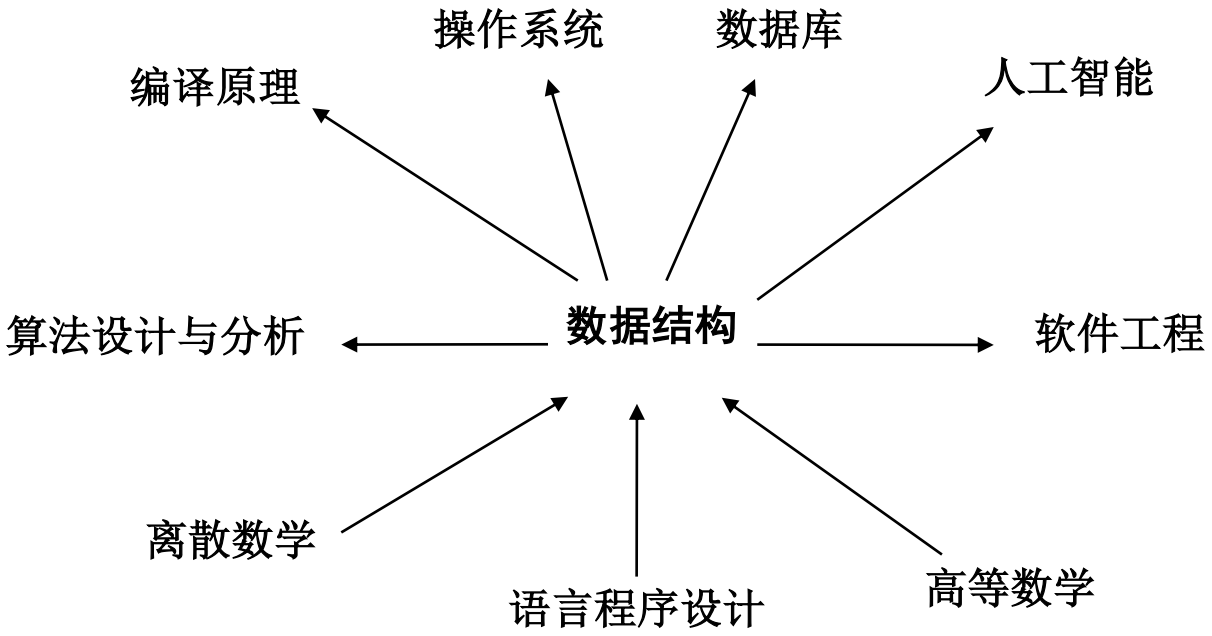
(2) 如何把加工对象存储到计算机中去？

(3) 数据运算。

返回章目录

1.3《数据结构》课程的地位

- 数据结构与其它课程关系图：



返回章目录

1.4 基本概念和术语

数据结构的相关名词：

- 数据（Data）
- 数据元素（Data Element）
- 数据对象（Data Object）
- 数据结构（Data Structure）

数据（Data）

- 定义：

  数据是所有能被输入计算机，且能被计算机程序加工处理的各种符号集合。

数据包含整型、实型、布尔型、图象、字符、声音等一切可以输入到计算机中的符号集合。

数据元素（Data Element）

- 定义：

数据元素是组成数据的基本单位，是计算机程序中加工处理的基本单位。例如：

数据项 ↓

学　号	姓　名	性　别	籍　贯	出生年月	住　址
101	赵虹玲	女	河北	1983.11	北京
…	…	…	…	…	…

← 数据元素

数据对象（Data Object）

- 定义：

  数据对象是性质相同的数据元素的集合，是数据的一个子集。

例如：

整数集合：N={0，±1，±2，…} 无限集

字符集合：C={ ´A´，B´，…， ´Z´} 有限集

数据结构（Data Structure）

数据结构是在整个计算机科学与技术领域中广泛使用的术语。它用来反映数据的内部构成，即数据由哪些成分数据构成，以什么方式构成，呈什么结构。数据结构是数据存在的形式。

简单地说，就是相互之间存在一种或多种特定关系的数据元素的集合。

- 数据结构定义： 是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作等等的学科。

常见的数据结构有：

线性结构、树形结构、图形结构和集合结构。

数据结构的三个方面

- 1、数据的逻辑结构
  - A. 线性结构
    - 线性表
    - 栈
    - 队列
    - 串
  - B. 非线性结构
    - 树形结构
    - 图形结构
    - 集合结构
- 2、数据的存储结构 (亦称物理结构)
  - A. 顺序存储
  - B. 链式存储
- 3、数据的运算：检索、排序、插入、删除、修改等。

数据结构（Data Structure）

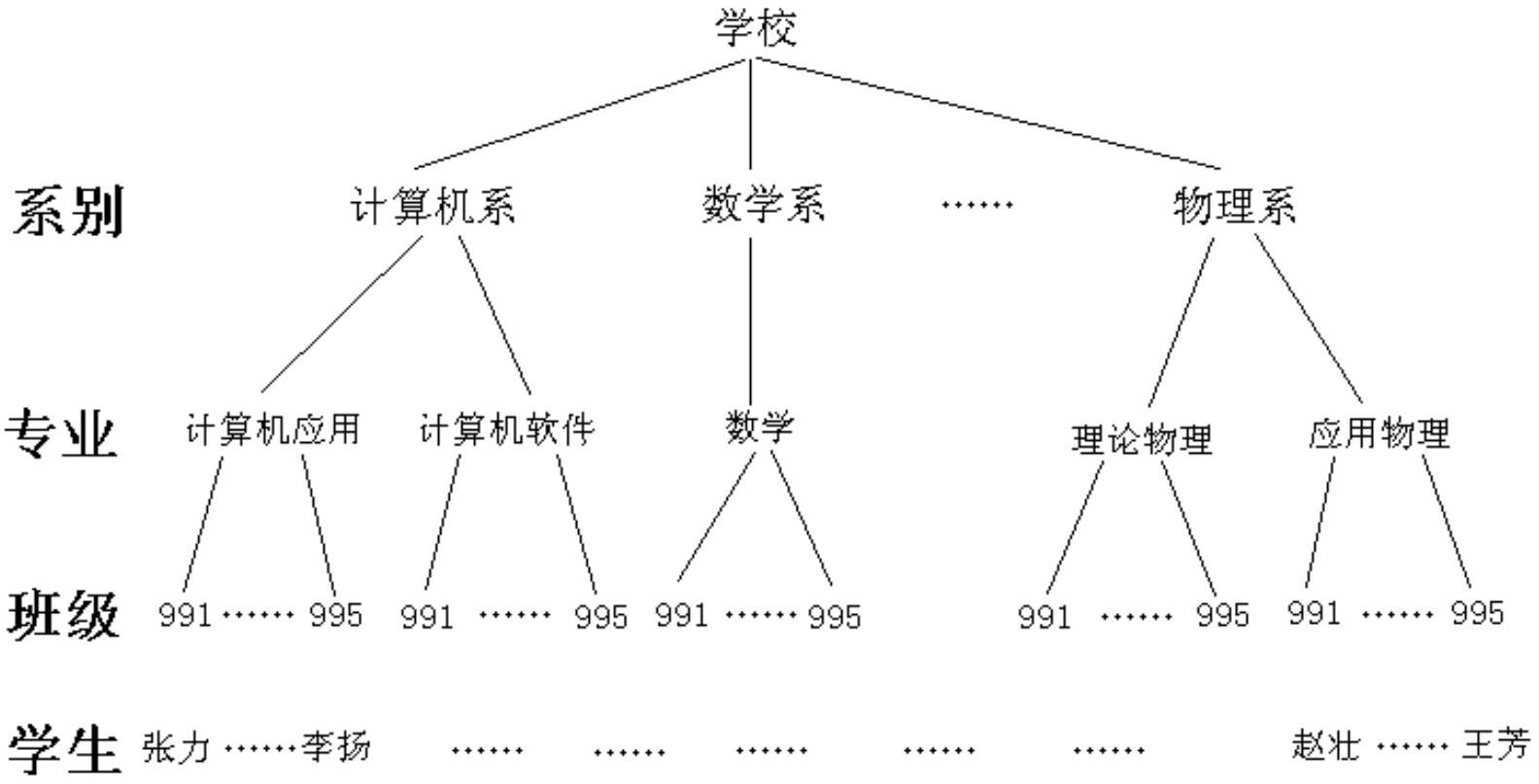
线性结构举例——学籍档案管理

假设一个学籍档案管理系统应包含如下表所示的学生信息。

学　号	姓　名	性　别	出生年月	……
99070101	李　军	男	80．12	……
99070102	王颜霞	女	81．2	……
99070103	孙　涛	男	80．9	……
99070104	单晓宏	男	81．3	……
……	……	……	……	……

数据结构（Data Structure）

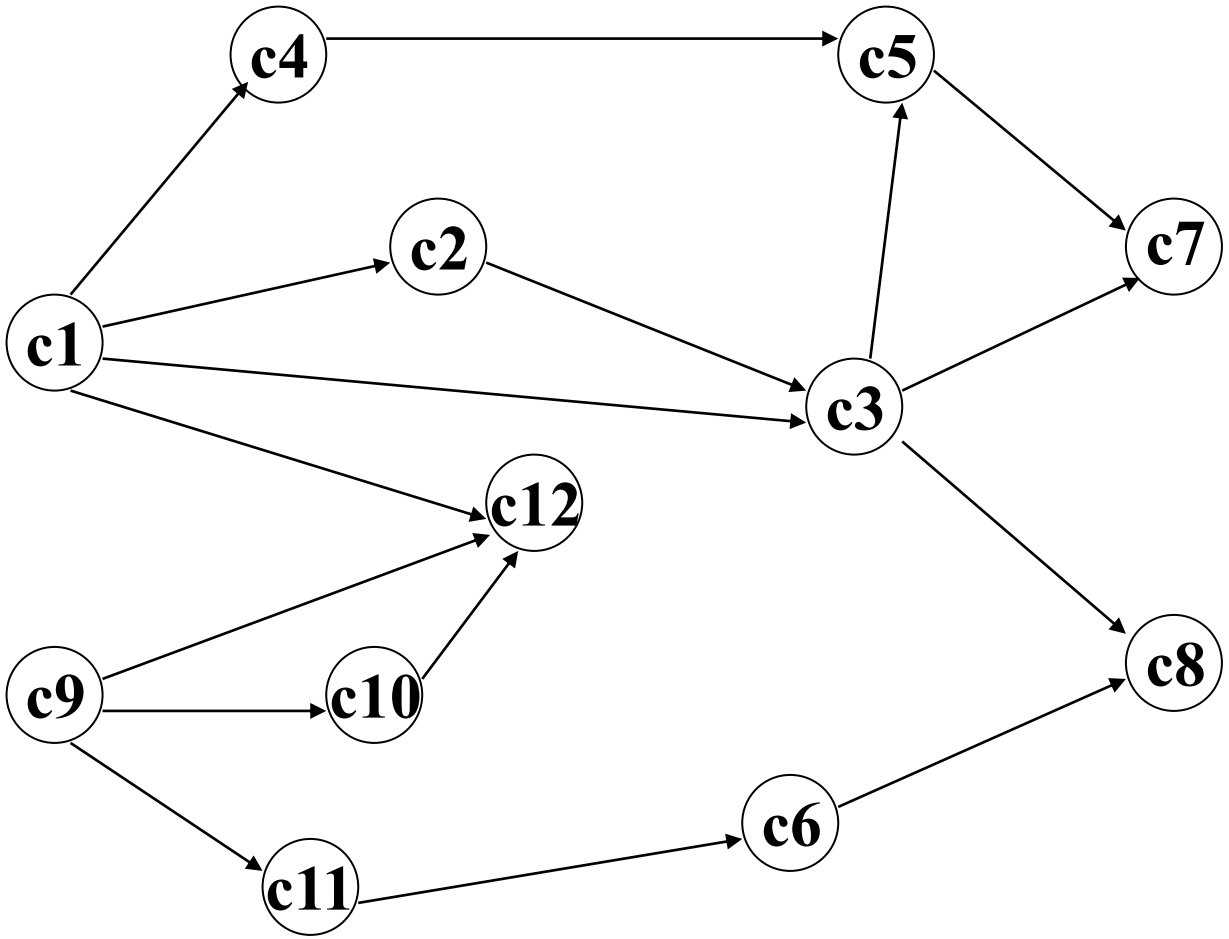
树型结构举例——全校学生档案管理的组织方式



课程编号	课程名称	需要的先导课程编号
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1，C2
C4	汇编语言	C1
C5	算法分析与设计	C3，C4
C6	计算机组成原理	C11
C7	编译原理	C5，C3
C8	操作系统	C3，C6
C9	高等数学	无
C10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C9，C10，C1

- 课程先后关系的图形描形式：

计算机专业必修课程开设先后关系



数据结构有逻辑上的数据结构和物理上的数据结构之分。

- 数据的逻辑结构—只抽象反映数据元素的逻辑关系
- 数据的存储（物理）结构—数据的逻辑结构在计算机存储器中的实现

数据的存储结构分为：

顺序存储结构——借助元素在存储器中的相对位置来表示数据元素间的逻辑关系 Click Here

链式存储结构——借助指示元素存储地址的指针表示数据元素间的逻辑关系 Click Here

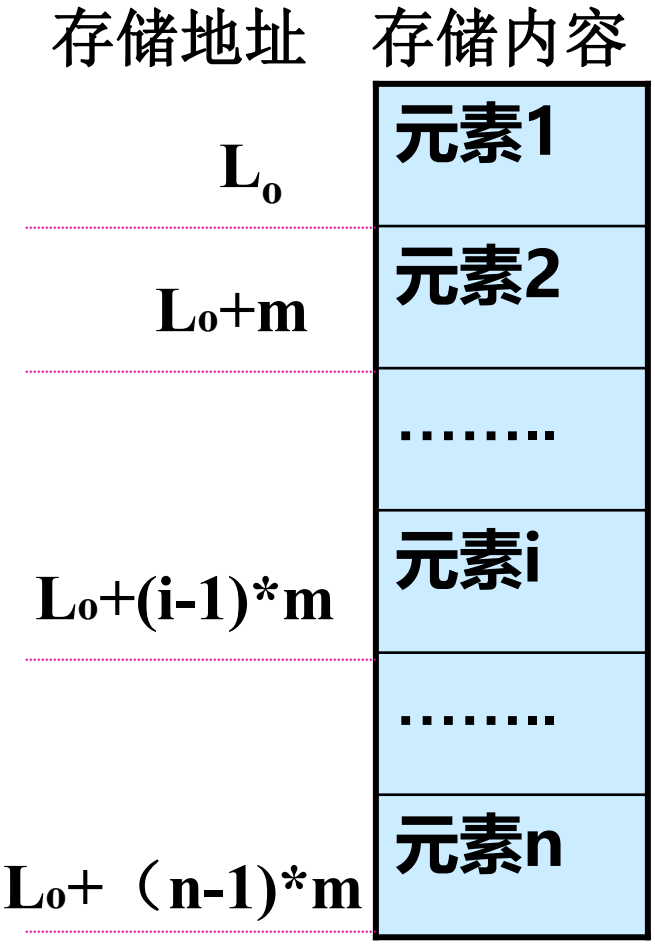
数据的逻辑结构与存储结构密切相关

算法设计 ⟶ 逻辑结构

算法实现 ⟶ 存储结构

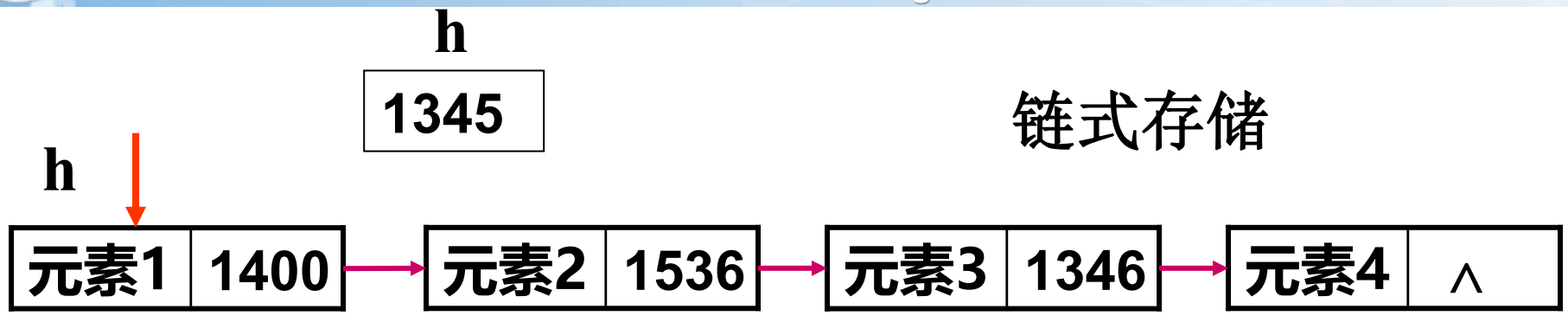
返回章目录

顺序存储



Loc(元素i)=Lo+（i-1)*m

Back



存储地址	存储内容	指针
1345	元素1	1400
1346	元素4	∧
……	……	……
1400	元素2	1536
……	……	……
1536	元素3	1346

Back

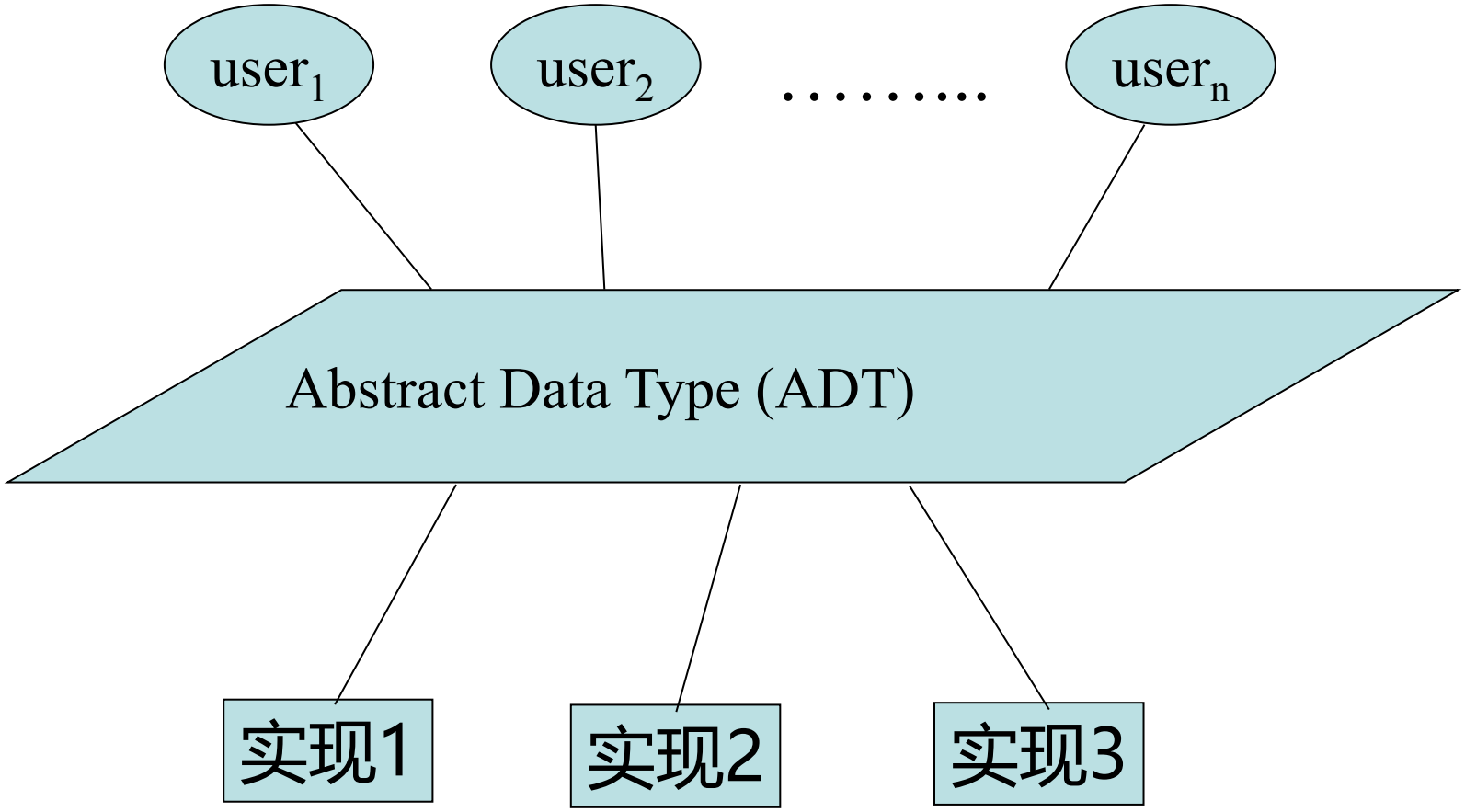
1.5数据类型和抽象数据类型

数据类型：

- 具有相同数据结构的数据属同一类。同一类数据的全体称为一个数据类型。
- 在高级程序设计语言中，数据类型用来说明数据在数据分类中的归属。它是数据的一种属性。
- 简单数据类型对应于简单的数据结构。
- 复杂的数据结构允许成分数据本身具有复杂的数据结构，因此，构造数据类型允许复合嵌套。
- 数据类型包括对成分数据的运算。

抽象数据类型

- 基本数据类型概念的引伸和发展。
- 数据模型连同定义在该模型上并作为数据模型构件的一组运算。
- 抽象数据类型明确地把数据模型与该模型上的运算紧密地联系起来。


user1
user2
.....
usern
Abstract Data Type (ADT)
实现1
实现2
实现3

从使用的角度看，一个抽象数据类型隐藏了所有使用者不关心的实现细节。使程序模块的实现与使用分离，从而能够独立地考虑模块的外部界面、内部算法和数据结构的实现。

在许多程序设计语言中预定义的类型，例如整数类型、浮点类型、指针类型等，都可以看作是简单的抽象数据类型。

从原则上讲，数据结构课程中讨论的各种表、栈、队列、二叉树和字典等，也可以像整数、浮点数等一样定义为程序设计语言预定义的抽象数据类型。

例：抽象数据类型Circle

```
ADT circle is
    data
    float  r ;
operations
    void constructor( )
          处理：  构造一个圆
    float  area ( )
       { return(3.14*r*r);  }
   float  circumference( )
       { return(2*3.14*r);  }
end ADT circle;
```

```
typedef struct cirnode * Circle;
typedef struct cirnode
{
   float r;  //圆的半径
}Cirnode;
Circle CirInit()
{    Circle C=malloc(sizeof(C));
     C->r=5.0;
     return C;  }
float area (Circle c)
{    return (3.14*c->r*c->r);  }
float circumference(Circle c)
{    return (2*3.14*c->r);  }
```

返回章目录

1.6 算法(Algorithm)与算法分析

- 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。

算法与程序(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。

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）渐近上界记号O

  $O(g(n)) = \{ f(n) \mid$ 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）渐近下界记号Ω

  $\Omega(g(n)) = \{ f(n) \mid$ 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

$o(g(n)) = \{ f(n) \mid$ 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 $f(n) / g(n) \to 0$ ， as $n \to \infty$。

- （4）非紧下界记号ω

$\omega(g(n)) = \{ f(n) \mid$ 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

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

- 规则$O(f(n))+O(g(n)) = O(\max\{f(n),g(n)\})$ 的证明：
- 对于任意$f_1(n) \in O(f(n))$，存在正常数c_1和自然数n_1，使得对所有$n \geq n_1$，有$f_1(n) \leq c_1 f(n)$。
- 类似地，对于任意$g_1(n) \in O(g(n))$，存在正常数c_2和自然数n_2，使得对所有$n \geq n_2$，有$g_1(n) \leq c_2 g(n)$。
- 令$c_3=\max\{c_1, c_2\}$，$n_3 =\max\{n_1, n_2\}$，$h(n)= \max\{f(n),g(n)\}$。
- 则对所有的 $n \geq n_3$，有
- $f_1(n) + g_1(n) \leq c_1 f(n) + c_2 g(n)$

  $\leq c_3 f(n) + c_3 g(n) = c_3(f(n) + g(n))$

  $\leq c_3 2 \max\{f(n),g(n)\}$

  $= 2c_3 h(n) = O(\max\{f(n),g(n)\})$.

算法渐近复杂性分析中常用函数

- 取整函数
  - ⌊ *x* ⌋：不大于*x*的最大整数；
  - ⌈ *x* ⌉：不小于*x*的最小整数。

- **Stirling's approximation**

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

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

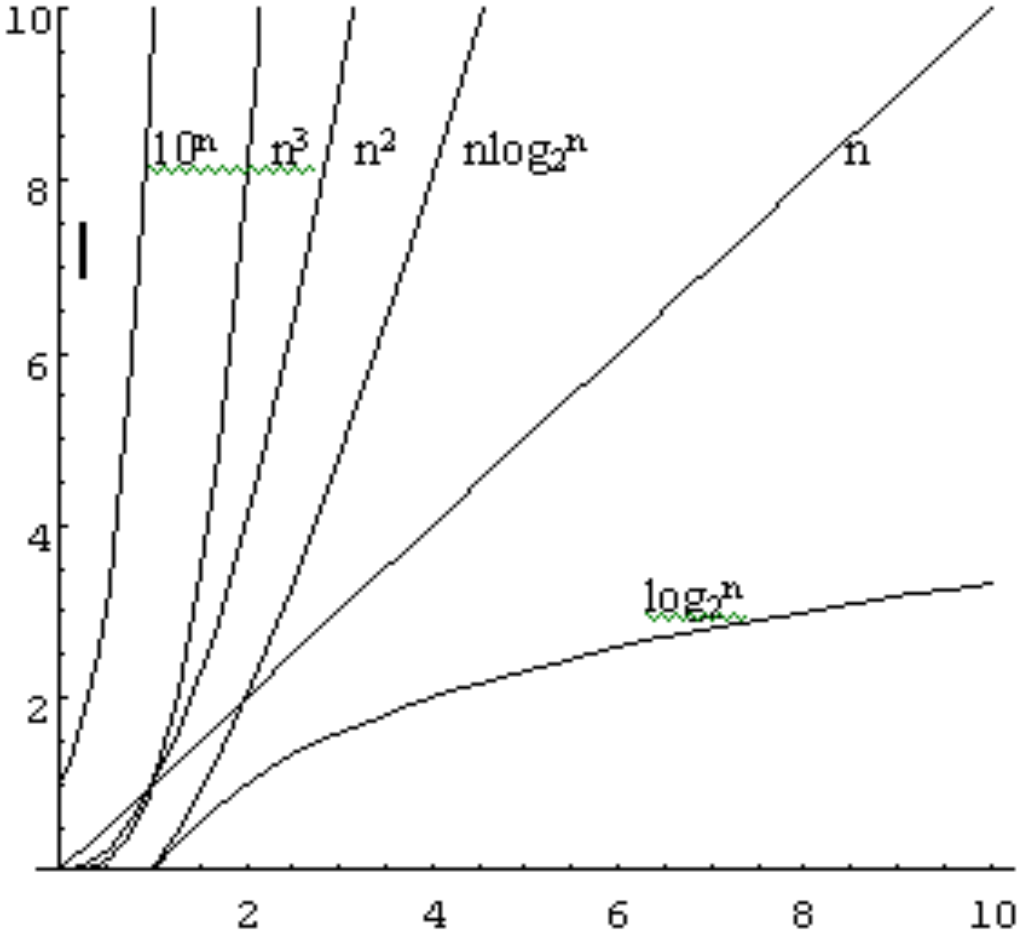


图 1.3 $f(n)$函数曲线变化速度的比较

- 常用的时间复杂度频率表：

$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	5096	65536
5	32	160	1024	32768	2147483648

算法分析方法

- 例：顺序搜索算法

```
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。

例1：两个n阶矩阵相乘

算法语句	对应的语句频度

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

对应的语句频度
n
n^2
n^2
n^3
n^3

总执行次数：$T_n=2n^3+2n^2+n$

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

例2：插入排序算法

```
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 of (ti-1)
             }
        a[j+1]=key;                   //    c7        n-1
       }
}
```

$$T(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4\sum_{i=1}^{n-1}t_i+c_5\sum_{i=1}^{n-1}(t_i-1)+c_6\sum_{i=1}^{n-1}(t_i-1)+c_7(n-1)$$

- 在最好情况下，$t_i=1$, for $1 \le i < n$;

$$T_{\min}(n)=c_1n+c_2(n-1)+c_3(n-1)+c_4(n-1)+c_7(n-1)$$

$$=(c_1+c_2+c_3+c_4+c_7)n-(c_2+c_3+c_4+c_7)=O(n)$$

- 在最坏情况下，$t_i \le i+1$, for $1 \le i < n$;

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

$$T_{\max}(n) \le c_1n+c_2(n-1)+c_3(n-1)+$$

$$c_4\left(\frac{n(n+1)}{2}-1\right)+c_5\left(\frac{n(n-1)}{2}\right)+c_6\left(\frac{n(n-1)}{2}\right)+c_7(n-1)$$

$$=\frac{c_4+c_5+c_6}{2}n^2+\left(c_1+c_2+c_3+\frac{c_4-c_5-c_6}{2}+c_7\right)n-(c_2+c_3+c_4+c_7)$$

$$=O(n^2)$$

- 对于输入数据a[i]=n-i,i=0,1,…,n-1，算法insertion_sort 达到其最坏情形。因此，

$$T_{\max}(n) \geq \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7 \right) n - (c_2 + c_3 + c_4 + c_7)$$

$$= \Omega(n^2)$$

- 由此可见，$T_{\max}(n) = \Theta(n^2)$

递归算法复杂性分析

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

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

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

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。

返回章目录

1.7 用c语言描述数据结构与算法

返回章目录

THE END