

数 据 结 构

Data Structures

周俊生

先修课程

- C程序设计
- C++程序设计
- 离散数学

考核方法

- 平时成绩： 20%
 - (出勤、作业、实验)
- 期中考试： 20%
- 期末考试： 60%

第一章 緒論

- 什么是数据结构
- 基本概念和术语
- 数据类型和抽象数据类型
- 算法和算法分析

数据结构的创始人——高德纳



Donald E. Knuth (

1938年出生，25岁毕业于加州理工学院数学系，博士毕业后留校任教，28岁任副教授。30岁时，加盟斯坦福大学计算机系，任教授。从31岁起，开始出版他的历史性经典巨著：

The Art of Computer Programming

他计划共写7卷，然而出版三卷之后，已震惊世界，使他获得计算机科学界的最高荣誉图灵奖，此时，他年仅36岁。

程序的概念

- 什么是程序?
 - 程序是用某种计算机能理解并执行的计算机语言描述解决问题的方法步骤。

② 程序的本质是什么?

- **数据表示:** 将数据存储在计算机中
- **数据处理:** 处理数据, 求解问题

数据结构问题起源于程序设计

数据结构的起源

- 计算机求解问题：
 问题 → 抽象出问题的模型 → 求模型的解

- 问题——数值问题、非数值问题

☞ 数值问题

- 程序处理的数据是纯粹的数值，数据之间的关系主要是数学方程或数学模型等。

☞ 非数值问题

- 处理多种复杂的具有一定结构关系的数据。
- 数据元素之间的相互关系一般无法用数学方程加以描述，而是要用线性表、树、图等**数据结构**来描述。

☞ 非数值计算问题：

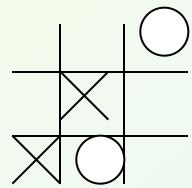
例1 电话号码查询问题：

设有一个电话号码薄，它记录了N个人的名字和其相应的电话号码，假定按如下形式安排：

$$(a_1, b_1) (a_2, b_2) \dots (a_n, b_n)$$

其中 $a_i, b_i (i=1, 2\dots n)$ 分别表示某人的名字和对应的电话号码，要求设计一个算法，当给定任何一个人的名字时，该算法能够打印出此人的电话号码，如果该电话簿中没有这个人，则该算法也能够报告没有这个人的标志。

例2：计算机和人对弈



例3 田径赛的时间安排问题：

设有六个比赛项目，规定每个选手至多可参加三个项目，有五人报名参加比赛（如下表所示）设计比赛日程表，使得在尽可能短的时间内完成比赛。

姓名	项目 1	项目 2	项目 3
丁一	跳高	跳远	100 米
马二	标 槍	铅 球	
张三	标 抢	100 米	200 米
李四	铅 球	200 米	跳 高
王五	跳 远	200 米	

☞ 非数值计算问题：

例1 电话号码查询问题：

设有一个电话号码薄，它记录了N个人的名字和其相应的电话号码，假定按如下形式安排：

$$(a_1, b_1) (a_2, b_2) \dots (a_n, b_n)$$

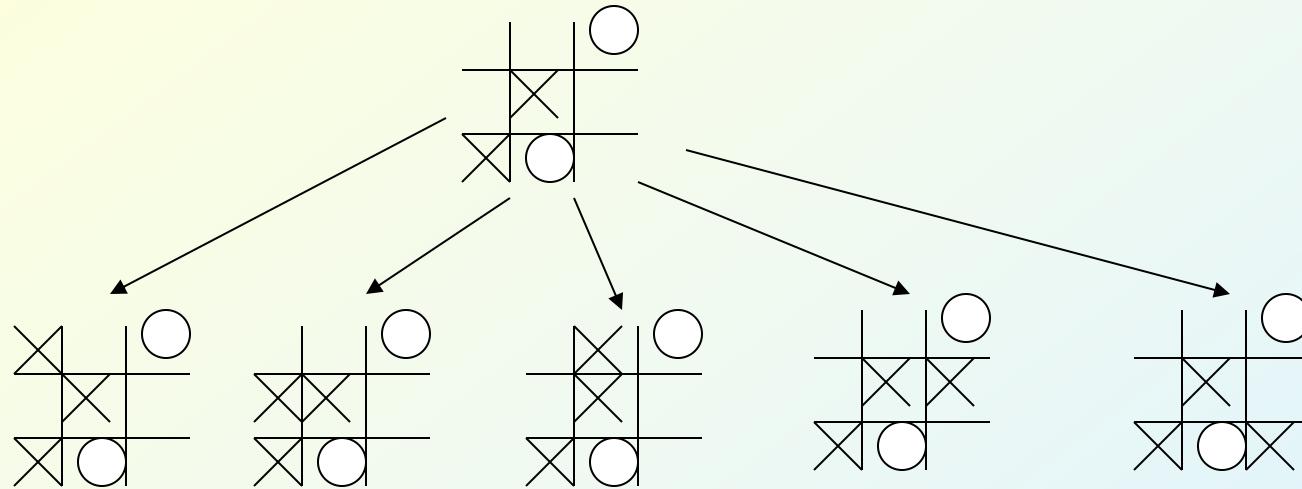
其中 a_i , b_i ($i=1, 2\dots n$) 分别表示某人的名字和对应的电话号码，要求设计一个算法，当给定任何一个人的名字时，该算法能够打印出此人的电话号码，如果该电话簿中根本就没有这个人，则该算法也能够报告没有这个人的标志。

- 算法的设计，取决于这张表的**结构及存储方式**。电话号码表的结构决定了查找（算法）的效率。

姓名	地址
李1	
李2	
.....	
张1	
张2	
.....	
王1	
王2	
.....	

模型：线性表结构

例2：计算机和人对弈



模型：树形结构

返回

例3 田径赛的时间安排问题：

设有六个比赛项目，规定每个选手至多可参加三个项目，有五人报名参加比赛（如下表所示）设计比赛日程表，使得在尽可能短的时间内完成比赛。

姓名	项目 1	项目 2	项目 3
丁一	跳高	跳远	100 米
马二	标 枪	铅 球	
张三	标 抢	100 米	200 米
李四	铅 球	200 米	跳 高
王五	跳 远	200 米	

----田径赛的时间安排问题解法

(1) 设用如下六个不同的代号代表不同的项目：

跳高	跳远	标枪	铅球	100米	200米
A	B	C	D	E	F

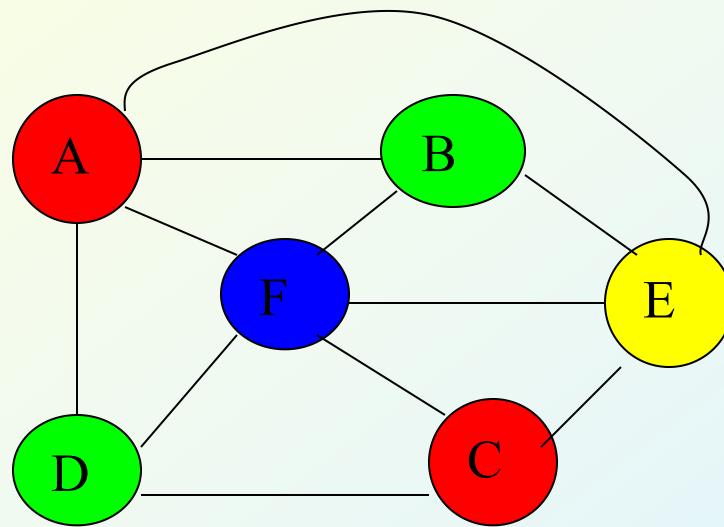
(2) 用顶点代表比赛项目

不能同时进行比赛的项目之间连上一条边。

(3) 某选手比赛的项目必定有边相连（不能同时比赛）。

姓名	项目1	项目2	项目3
丁一	A	B	E
马二	C	D	
张三	C	E	F
李四	D	F	A
王五	B	F	

只需
安排四
个单位
时间进
行比赛



模型：图结构

比赛时间	比赛项目
1	A, C
2	B, D
3	E
4	F



求解非数值计算的问题：

主要考虑的是设计出合适的数据结构及相应的算法。

即：首先要考虑对相关的各种信息如何表示、组织和存储？

因此，可以认为：数据结构是研究非数值问题中计算机的操作对象以及它们之间的关系和操作的课程。

1. 2 基本概念和术语

几个概念：

- **数据(Data)**: 是对信息的一种符号表示。在计算机科学中是指所有能输入到计算机中并能被计算机程序处理的符号的总称。
- **数据元素(Data Element)**: 是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。
 - 一个数据元素可由若干个**数据项**组成。数据项是数据的不可分割的最小单位。
- **数据对象(Data Object)**: 是性质相同的数据元素的集合。是数据的一个子集。

数据、数据元素、数据项之间的关系

- 包含关系：数据是由数据元素组成，数据元素是由数据项组成。
- **数据元素**是讨论数据结构时涉及的最小数据单位，其中的数据项一般不予考虑。

数据结构：相互之间存在一定**关系**的数据元素的集合。



数据结构的三个方面的含义：



逻辑结构---

- 数据元素间抽象化的相互关系（简称为数据结构）；
- 与数据的存储无关，独立于计算机，它是从具体问题抽象出来的数据模型。



存储结构（物理结构）----

- 数据元素及其关系在计算机存储器中的存储方式；
- 是逻辑结构用计算机语言的实现，它依赖于计算机语言。



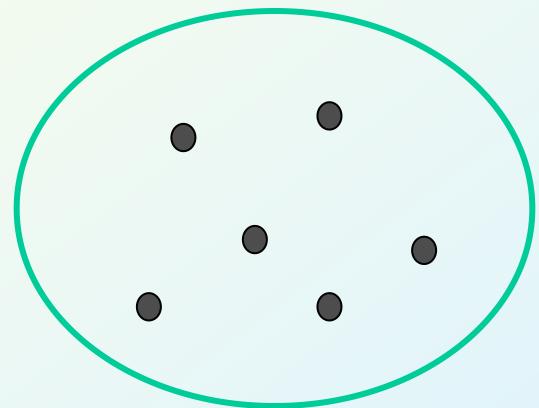
运算（算法）

【开本】数据结构的三个方面的含义之：

【开本】逻辑结构---

数据结构从逻辑上分为四类：

(1) 集合：数据元素之间就是
“属于同一个集合”；



数据结构的三个方面的含义之：

逻辑结构---

数据结构从逻辑上分为四类：

- (1) 集合：数据元素之间就是“属于同一个集合”；
- (2) 线性结构：数据元素之间存在着一对一的线性关系；

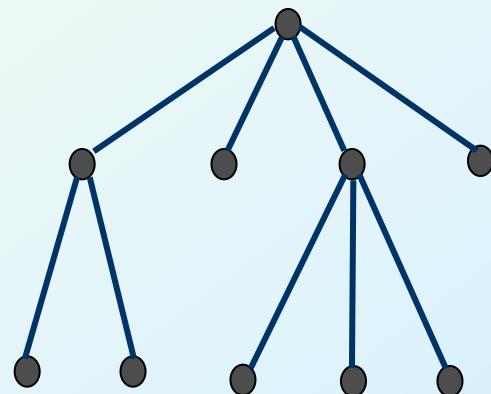


数据结构的三个方面的含义之：

逻辑结构---

数据结构从逻辑上分为四类：

- (1) 集合：数据元素之间就是“属于同一个集合”；
- (2) 线性结构：数据元素之间存在着一对一的线性关系；
- (3) 树结构：数据元素之间存在着一对多的层次关系；

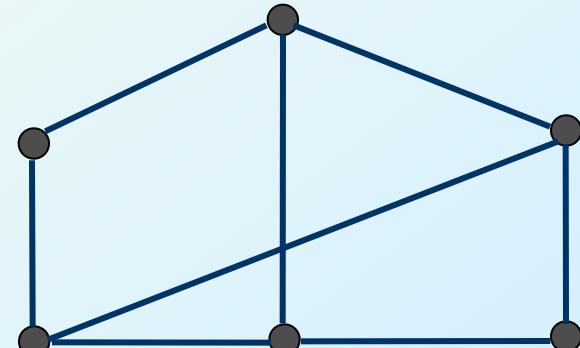


数据结构的三个方面的含义之：

逻辑结构---

数据结构从逻辑上分为四类：

- (1) 集合：数据元素之间就是“属于同一个集合”；
- (2) 线性结构：数据元素之间存在着一对一的线性关系；
- (3) 树结构：数据元素之间存在着一对多的层次关系；
- (4) 图结构：数据元素之间存在着多对多的任意关系。





数据结构的三个方面的含义之：



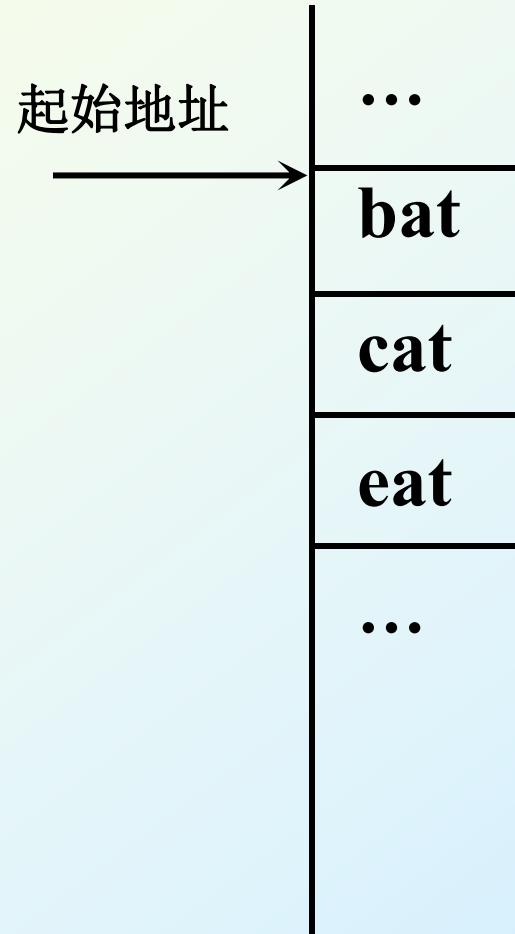
存储结构

- 存储结构两方面的内容：
 - (1) 数据元素自身值的表示（数据域）
 - (2) 该结点与其它结点关系的域
- 四种基本的存储方法：
 - (1) 顺序存储方法
 - (2) 链接存储方法
 - (3) 索引存储方法
 - (4) 散列（哈希）存储方法

例：(bat, cat, eat)

•通常有两种存储结构：

1. **顺序存储结构**：用一组连续的存储单元依次存储数据元素，数据元素之间的逻辑关系由元素的存储位置来表示。

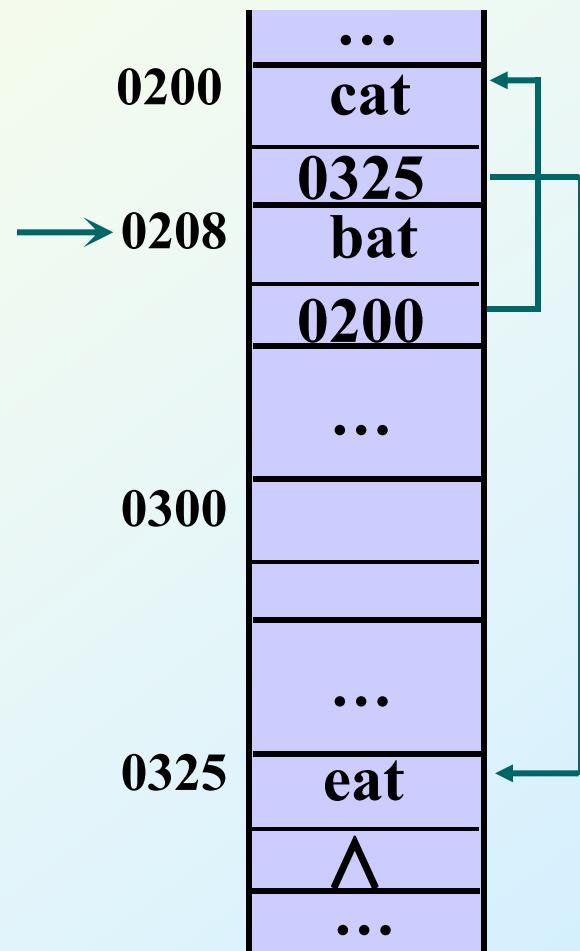


例： (bat, cat, eat)

•通常有两种存储结构：

1. **顺序存储结构**：用一组连续的存储单元依次存储数据元素，数据元素之间的逻辑关系由元素的存储位置来表示。

2. **链接存储结构**：用一组任意的存储单元存储数据元素，数据元素之间的逻辑关系用指针来表示。



逻辑结构和存储结构之间的关系

- 数据的逻辑结构属于用户视图，是面向问题的，反映了数据内部的构成方式；
- 数据的存储结构属于具体实现的视图，是面向计算机的。
- 一种数据的逻辑结构可以用多种存储结构来存储，而采用不同的存储结构，其数据处理的效率往往是不同的。

➤ 数据（逻辑）结构的形式定义：

由某一数据对象及该对象中所有数据成员之间的关系组成。记为：

$$\text{Data_Structure} = (D, R)$$

其中， D 是某一数据对象， R 是该对象中所有数据成员之间的关系的有限集合。

✳ 例1 树形结构 $T = (D, R)$

其中， $D = \{a, b, c, d, e, f, g\}$

$$R = \{(a, b), (a, c), (a, d), (b, e), (c, f), (c, g)\}$$

程序设计的实质是对实际问题选择一个好的数据结构，加之设计一个好的算法。而好的算法在很大程度上取决于描述实际问题的数据结构。

本课程讨论：

- 在解决问题时可能遇到的典型的逻辑结构（数据结构）
- 逻辑结构的存储映象（存储实现）
- 数据结构的相关操作及其实现。

1.3 数据类型和抽象数据类型

数据类型

- 定义：一组值的集合，以及定义于这个值集上的一组操作的总称。

C语言中的数据类型

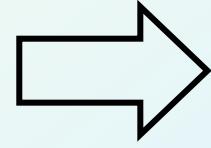
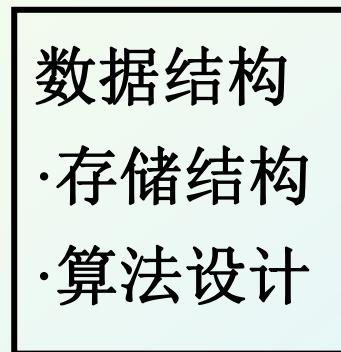
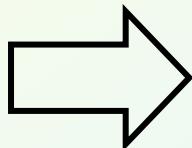
基本数据类型、指针类型、数组类型、结构体类型、共用体类型、枚举类型

抽象数据类型 (ADTs: Abstract Data Types)

- 一个**数据结构**以及定义在该结构上的一组**操作**的总称。
- 数据类型和ADT的区别在于：
 - 数据类型指的是高级程序设计语言支持的**ADT**
 - 而**ADT**指的是自定义的数据类型。
- 本课程将要学习的线性表、栈、队列、树、图等结构就是一些不同的ADT。

- **ADT**包括**定义**和**实现**两个方面，其中**定义**是独立于**实现**的。
 - 定义仅给出一个**ADT**的逻辑特征，不必考虑如何在计算机中实现；
 - **ADT**的实现者依据这些**定义**来完成该**ADT**的各种操作的具体实现

ADT的不同视图



(a) 使用视图
ADT的定义

(b) 设计视图
ADT的设计

(c) 实现视图
ADT的实现

抽象数据类型的定义：

ADT: 抽象数据类型名

数据对象： 数据对象的定义

数据关系： 数据逻辑关系的定义

基本操作： 基本操作的定义

抽象数据类型可以用三元组表示：

(D , S , P)

ADT的定义形式

ADT 抽象数据类型名

Data

数据元素之间逻辑关系的定义

Operation

操作1

初始条件：执行此操作前数据所必须的状态

输入：执行此操作所需要的输入

功能：该操作将完成的功能

输出：执行该操作后产生的输出

操作结果：执行该操作后数据的状态

操作2

.....

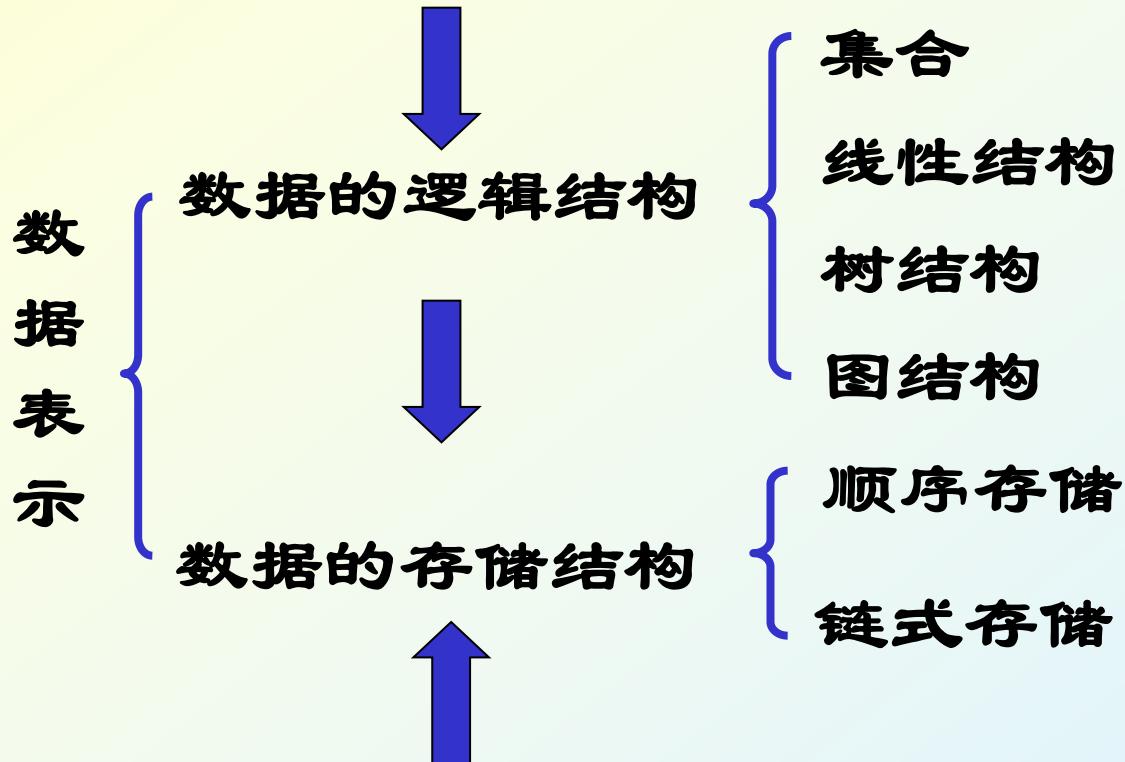
.....
操作n

.....

endADT

数据结构的基本概念（小结）

非数值问题



数据的操作：插入、删除、修改、检索、排序等

1.4 算法和算法分析

- 定义: 是对特定问题求解步骤的一种描述，是指令的有限序列。
- 特性:
 - 1、 输入: 有零个或多个输入
 - 2、 输出: 有一个或多个输出
 - 3、 有穷性: 要求序列中的指令是有限的; 每条指令的执行包含有限的工作量; 整个指令序列的执行在有限的时间内结束。
 - 4、 确定性: 算法中的每一个步骤都必须是确定的，而不应当含糊、模棱两可。
 - 5、 可行性: 算法中的每一个步骤都应当能被有效的执行，并得到确定的结果。

思考： 算法与程序有何区别？

算法的描述: c, c++等语言

算法的描述方法

- 自然语言
- 流程图
- 伪代码
- 程序设计语言

例：用辗转相除法求两个自然数 m 和 n 的最大公约数，
并假设 $m \geq n$

```
int CommonFactor(int m, int n)
{
    r=m mod n;
    while(r!=0)
    {
        m=n;
        n=r;
        r=m mod n;
    }
    return n;
}
```

算法设计的要求：

1、正确性

2、健壮性（Robustness）

3、可读性 要求算法易于理解，便于分析

4、可修改可扩展性

5、高效率

- 执行算法所耗费的存储空间（空间复杂性）

- 执行算法所耗费的时间（时间复杂性）

► 我们重点考虑时间复杂度。

算法的时间复杂度分析

算法的执行时间 = 每条语句执行时间之和

每条语句执行次数之和

单位时间

执行次数 × 执行一次的时间

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

指令系统、编译的代码质量

算法分析：

我们将算法求解问题的输入量称为**问题的规模**，用一个整数 n 来表示，一般地说，时间复杂度是问题规模的函数 — $T(n)$

例：求两个方阵的乘积 $C = A * B$

```
#define n 自然数
void MatrixMLT(float A[n][n],float B[n][n],float C[n][n])
{
    int i, j, k;
    for(i=0;i<n;i++)                                // n+1
        for(j=0;j<n;j++)                            // n(n+1)
    {
        C[i][j]=0;                                  // n*n
        for( k=0;k<n;k++)                         // n*n*(n+1)
            C[i][j]=A[i][k]*B[k][j]                // n*n*n
    }
}
```

$$T(n) = 2n^3 + 3n^2 + 2n + 1$$

- ▶ 一个算法的时间复杂度 $T(n)$ 是问题规模n的函数。
- ▶ 当问题的规模n 趋于无穷大时，把时间复杂度 $T(n)$ 的数量级（阶）称为算法的渐进时间复杂度。

严格的数学定义为：若 $T(n)$ 和 $f(n)$ 是定义在正整数集合上的两个函数，当存在两个正的整数 c 和 n_0 时，使得对所有的 $n \geq n_0$

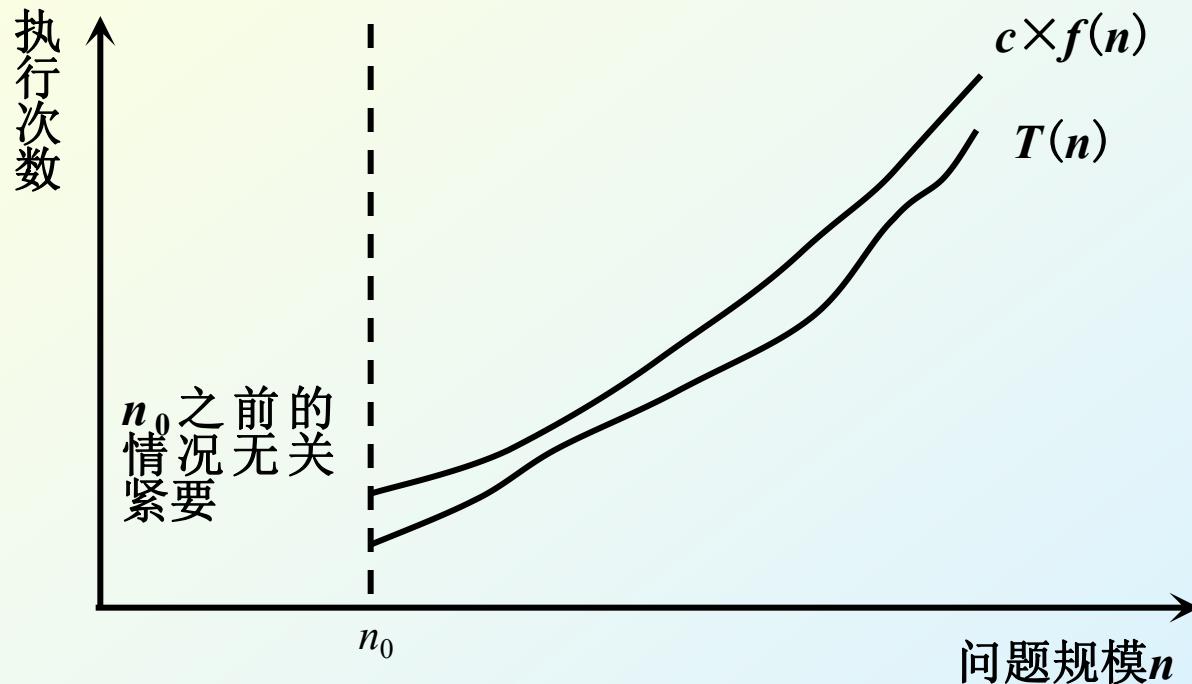
都有 $T(n) \leq c \times f(n)$

成立，则 $T(n) = O(f(n))$

这时称 $T(n)$ 的时间复杂度为 $f(n)$ 数量级。

算法分析——大O符号

定义 若存在两个正的常数 c 和 n_0 ，对于任意 $n \geq n_0$ ，都有 $T(n) \leq c \times f(n)$ ，则称 $T(n) = O(f(n))$



❖ 当问题规模充分大时在渐近意义下的阶

算法分析——大O符号

定理：若 $T(n)=a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$ 是一个 m 次多项式，则 $T(n)=O(n^m)$ 。

说明：在计算算法时间复杂度时，可以忽略所有低次幂和最高次幂的系数。

例：求两个方阵的乘积 $C = A * B$

```
#define n 自然数
void MatrixMLT(float A[n][n],float B[n][n],float C[n][n])
{
    int i, j, k;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)          // n+1
    {
        C[i][j]=0;              // n*n
        for( k=0;k<n;k++)       // n*n*(n+1)
            C[i][j]=A[i][k]*B[k][j] // n*n*n
    }
}
```

基本语句

$$T(n) = 2n^3 + 3n^2 + 2n + 1 = O(n^3)$$

算法分析

问题规模： 输入量的多少。

基本语句： 是执行次数与整个算法的执行次数成正比的原语句。

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

问题规模： n
基本语句： x++

例 1:

temp = i;

i = j;

j = temp;

- ▶ 如果算法的执行时间是一个与问题规模n无关的常数，则算法的时间复杂度为常数阶，记作 $T(n)=O(1)$ 。

例 2：

x=0;y=0;

for (k=1;k<=n;k++)

x++;

for (i=1;i<=n;i++)

for (j=1;j<=n;j++)

y++; //n*n

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

例 3：

x=1;

for (i=1;i<=n;i++)

for (j=1;j<=i;j++)

for (k=1;k<=j;k++)

x++;

$$T(n) = O(n^3 / 6 + \dots) = O(n^3)$$

问题：

i=n, sum=0;

while (i>1){

i=i/5;

sum += i;

}

$$T(n) = O(\log_5 n)$$

- 而有些算法的时间复杂度不仅与问题的规模有关，而且与它所处理的数据集的状态有关。

例4：在数组A[n]中查找值为k的元素。

i = n-1;

while ((i>=0) && A[i]!=k))

i--;

return i;

- 通常根据数据集中可能出现的最坏情况估计出算法的**最坏时间复杂度**，以及**平均时间复杂度**。

例5：起泡排序算法

```
void bubble_sort( int a[], int n )
{
    for( int i=n-1, change=TRUE; i>1 &&change; i-- )
    {
        change=FALSE;
        for( j=0; j<i; j++ )
            if( a[j]>a[j+1] ){
                a[j] ↔ a[j+1]; change=TRUE;
            }
    }
}
```

1.4 算法及算法分析

最好情况、最坏情况、平均情况

结论：如果问题规模相同，时间代价与输入数据有关，则需要分析最坏情况、平均情况、最好情况。

- ✓ **最好情况：**出现概率较大时分析
- ✓ **最差情况：**实时系统
- ✓ **平均情况：**已知输入数据是如何分布的，
通常假设等概率分布

常见的时间复杂度，按数量级递增排序：

常数阶 $O(1)$

对数阶 $O(\log_2 n)$

线性阶 $O(n)$

线性对数阶 $O(n \log_2 n)$

平方阶 $O(n^2)$

立方阶 $O(n^3)$

.....

K次方阶 $O(n^k)$

指数阶 $O(2^n)$

本章小结——知识结构图

