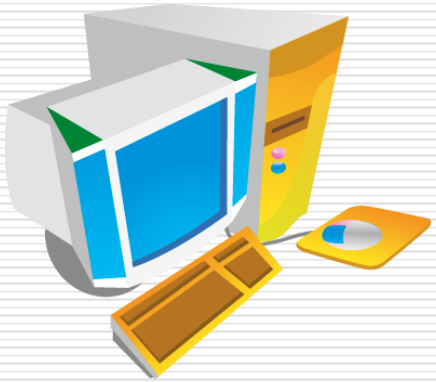
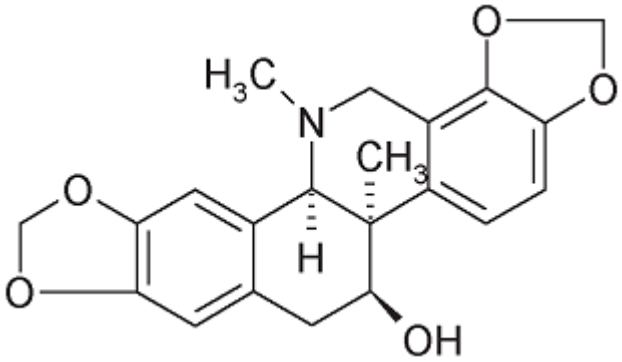
数据结构



人工智能学院

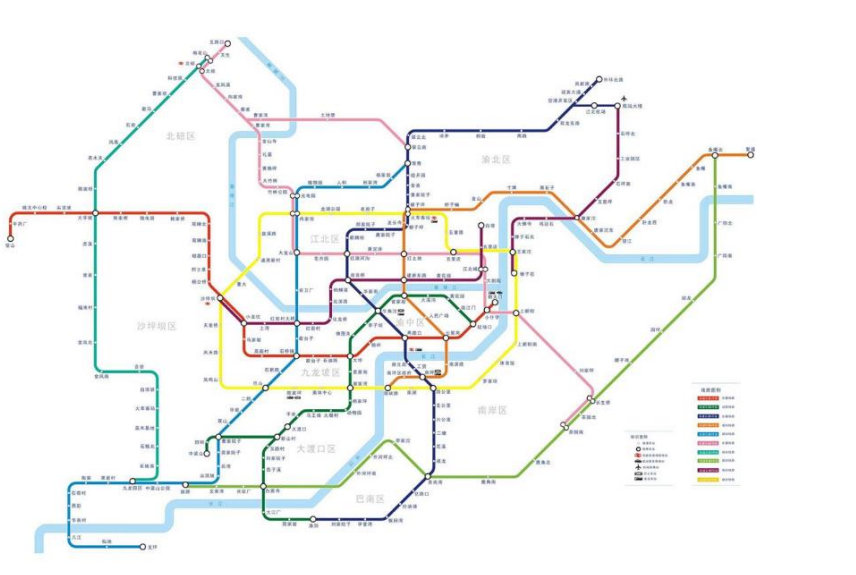
刘运

引言



引言

车载导航仪



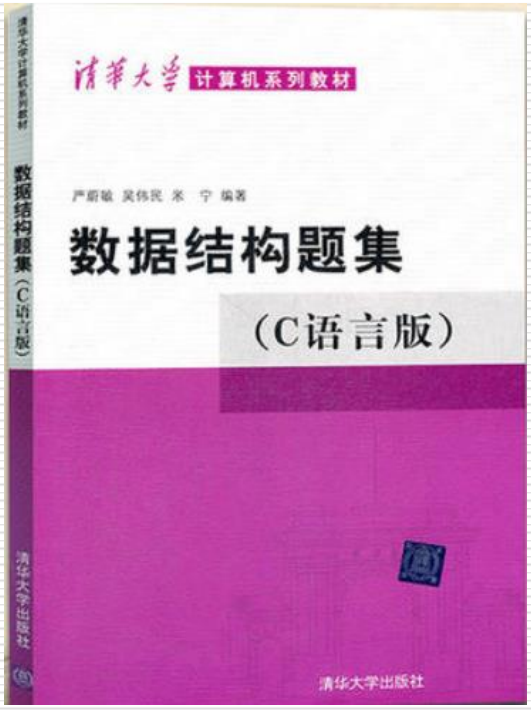
重庆市轻轨线路图

教材和题集

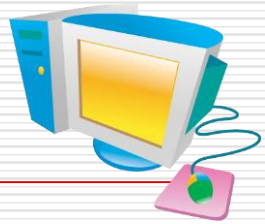
- 数据结构（C语言版）

  ISBN 978-7-302-14751-0

  严蔚敏，吴伟民，清华大学出版社



数据结构的课程特点

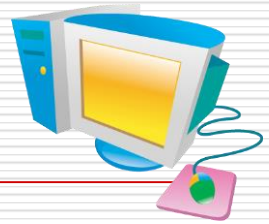


- 数据结构是计算机专业重要的专业基础课。
- 需要“程序设计语言”和“离散数学”课程基础。
- 实践性较强。

学时安排及考核方式

- 课堂讲授：48学时 (16次课)
- 上机实验：24学时 (8次课)
- 考试方式：闭卷考试
- 期末考试：60%
- 实验成绩： 30%
- 平时成绩： 10%

本课程的教学目标



- 掌握常用的数据的逻辑结构及存储方法，学会编写在常用的存储方式下数据的基本操作的算法。
- 学习分析问题所涉及数据对象的特征，操作的特征，选择合适的数据结构、存储结构及算法进行程序设计的方法。
- 了解算法时间、空间开销的分析方法。
- 通过基本算法和应用算法的学习，通过上机实践，提高编程能力，为进行软件开发打下良好的基础。

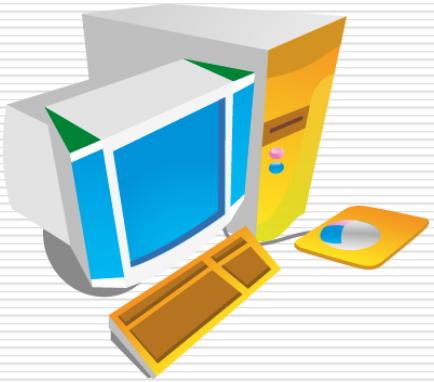
基本学习方法

勤学勤练

- 课前预习、上课认真听讲
- 课后多做习题
- 多上机编程（熟练掌握C或C++）

数据结构

第一章　绪论



主要内容

- 1.1 数据结构的研究内容
- 1.2 基本概念和术语
- 1.3 抽象数据类型的表示和实现
- 1.4 算法和算法分析
- 1.5 小结

学习要点

- 了解数据结构有关概念的含义，特别数据的逻辑结构，数据的存储结构之间的关系；
- 熟悉类C语言的书写规范，特别要注意值调用和引用调用的区别及出错处理方式；
- 了解计算算法时间复杂度的方法；

1.1 数据结构的研究内容

- 早期的计算机主要用于数值计算
- 现在的计算机更多地是用于非数值数据处理（字符、表格、图像）
- 对非数值数据的处理：分析数据的逻辑特征→抽象出合适的数学模型→合理地存储到计算机→设计出算法→编写出程序

请看例1～3

例1 学生信息查询系统

表1-1 学生信息表

学号	姓名	性别	出生年月	……
20165101101	李　军	男	1997.12	……
20165101102	王颜霞	女	1998.06	……
20165101103	孙　涛	男	1998.03	……
20165101104	张晓宏	男	1998.01	……
……				

学生信息表的特点

- 每个学生的信息占据一行，所有学生的信息按学号顺序依次排列构成一张表格。
- 表中每个学生的信息依据学号的大小存在着一种前后关系，这就是我们所说的线性结构，线性结构的主要特点是数据元素之间的关系是“一对一”。
- 通常的操作
  - 插入某个学生的信息
  - 删除某个学生的信息
  - 更新某个学生的信息
  - 按条件查找某个学生的信息

例2 人机对弈

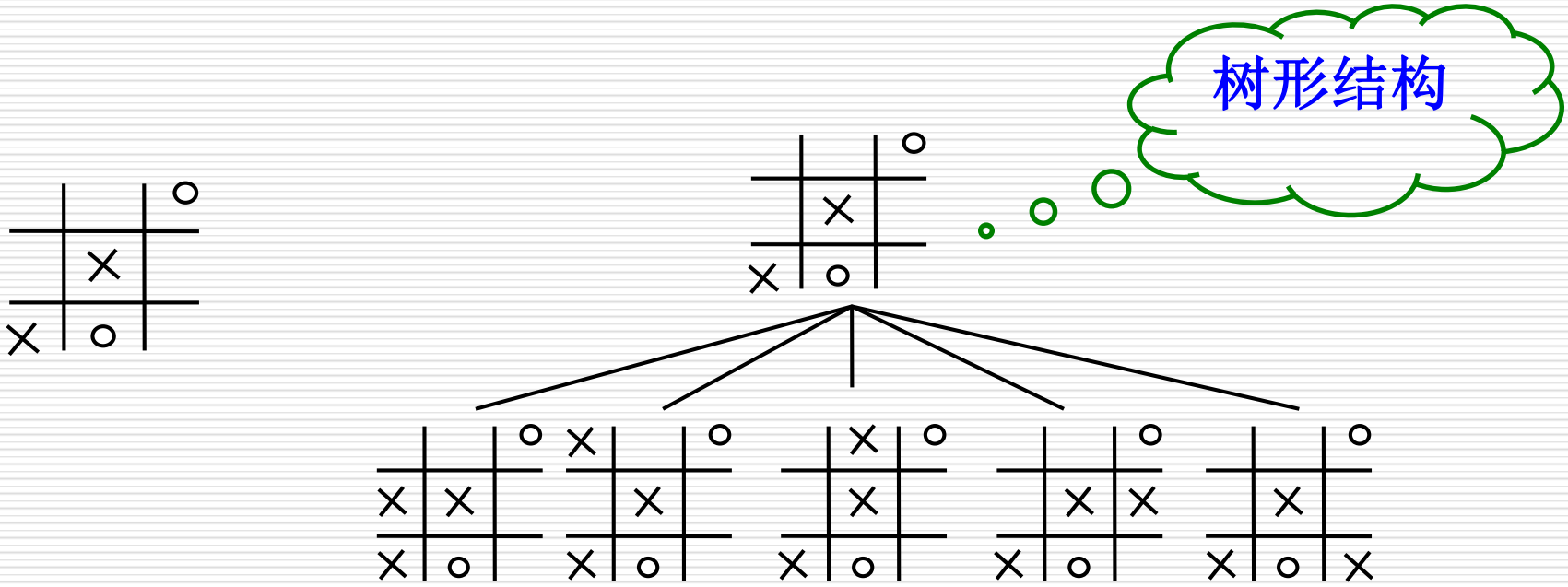
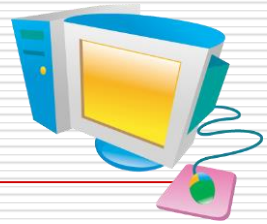


图 1-1 井子棋对弈树

树形结构的特点

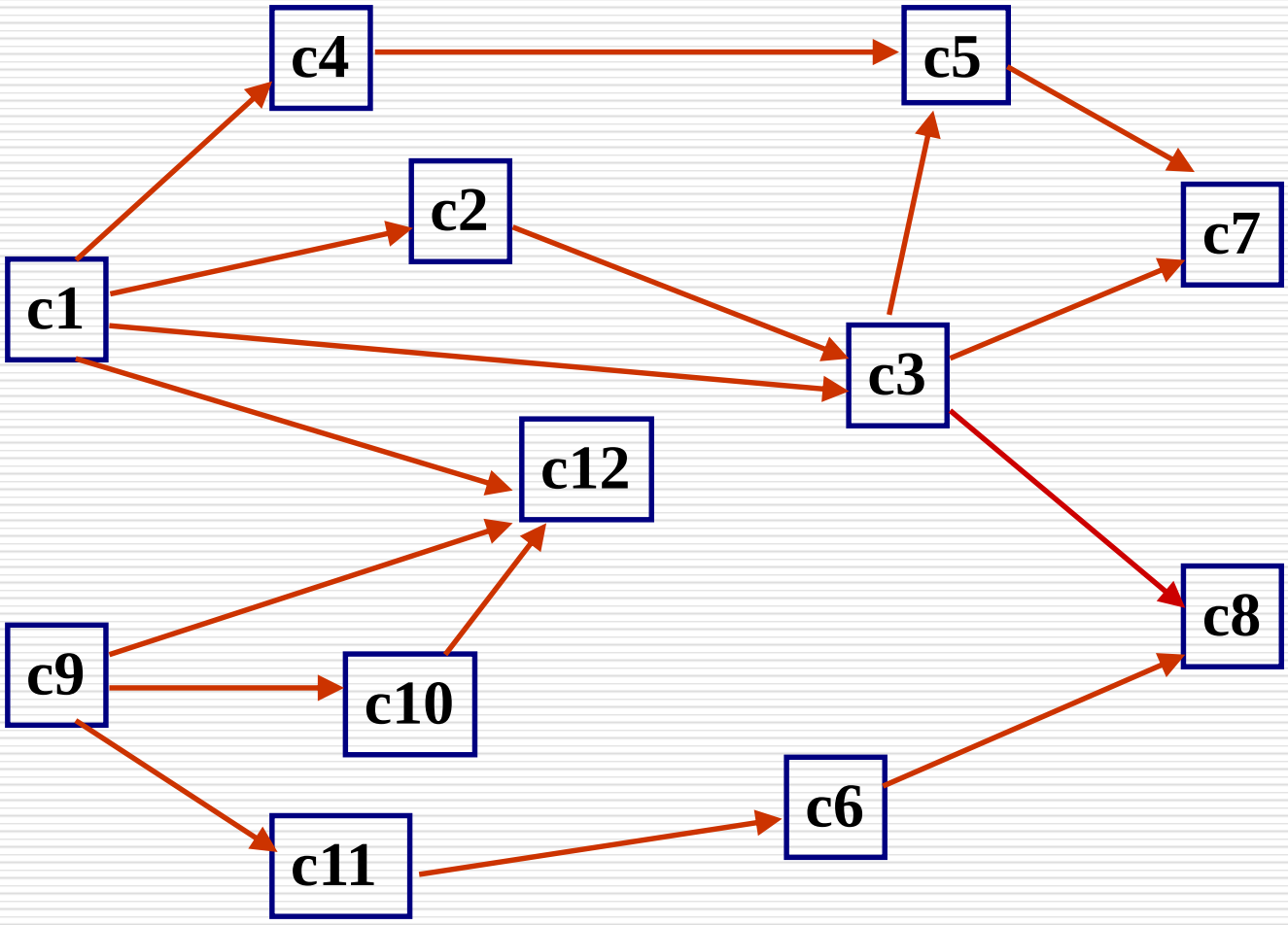


- 树形结构所处理的数据具有分支和层次关系，是一种非线性结构，数据元素之间的关系为“一对多”。
- 对它的操作有：建立树形结构、存储树、访问树中的每个结点等。

例3　计算机专业学生的必修课程

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
C 10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C9 ， C10 ， C1

课程先后关系的图描述



图结构的特点

- 关系比较复杂，用例1和例2的结构表达不出来，必须用图结构描述。数据元素之间的关系是“多对多”。
- 现实中，这类关系的数据非常多。如：网络规划、交通、通讯规划等。
- 图的操作有：建立图的存储结构、图的遍历、拓扑排序、求解最小生成树、关键路径、最短路径等。

结论

- 操作对象的关系复杂多样
- 操作不再是单纯的数值计算，更多的是非数值问题求解，需要对数据进行分析、组织及管理。

1.1数据结构的研究内容

- **Def1数据结构 (从学科角度定义的数据结构)**

  ——数据结构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的学科。

发展简史

(1) “数据结构”作为一门独立的课程在国外是1968年才开设.

唐欧克努特《计算机程序设计技巧》第一卷《基本算法》

(2) 从20世纪60年代末到70年代初，出现了大型程序，软件也相对独立，结构程序设计成为程序设计方法的主要内容，人们越来越重视数据结构，认为程序设计的实质是对确定问题选择一种好的结构，加上设计一种好的算法。

(3) 目前在我国，“数据结构”不仅是计算机专业的教学计划中的核心课程之一，也是其它非计算机专业的主要选修课程之一。

地位

“数据结构”是介于数学、计算机硬件和计算机软件三者之间的一门核心课程，在计算机科学中是一门专业基础课。

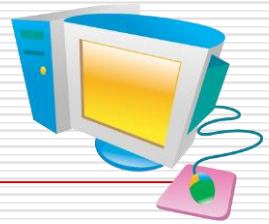
数据结构课程研究的主要内容

1. 研究数据元素之间的客观联系（逻辑结构）

2. 研究具有某种逻辑关系的数据在计算机存储器内的存储方式（存储结构）

3. 研究如何在数据的各种结构（逻辑的和物理的）的基础上对数据实施一系列有效的基本操作（算法）

1.2 基本概念和术语



1. 数据（Data）：客观事物的符号表示，是指所有能输入到计算机中并被计算机程序处理的符号的总称。

例如：数学计算中的整数和实数，文本编辑中的字符串，以及图形、图像、声音、动画等。

2. 数据元素（Data Element）：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。

例如一名学生的记录，对弈树中的一个棋盘格局，图中的一个顶点等。

人类的数据元素？

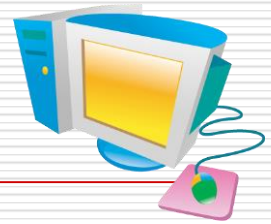
1.2 基本概念和术语

3. 数据项（Data Item）:是组成数据元素的、有独立含义的、不可分割的最小单位。

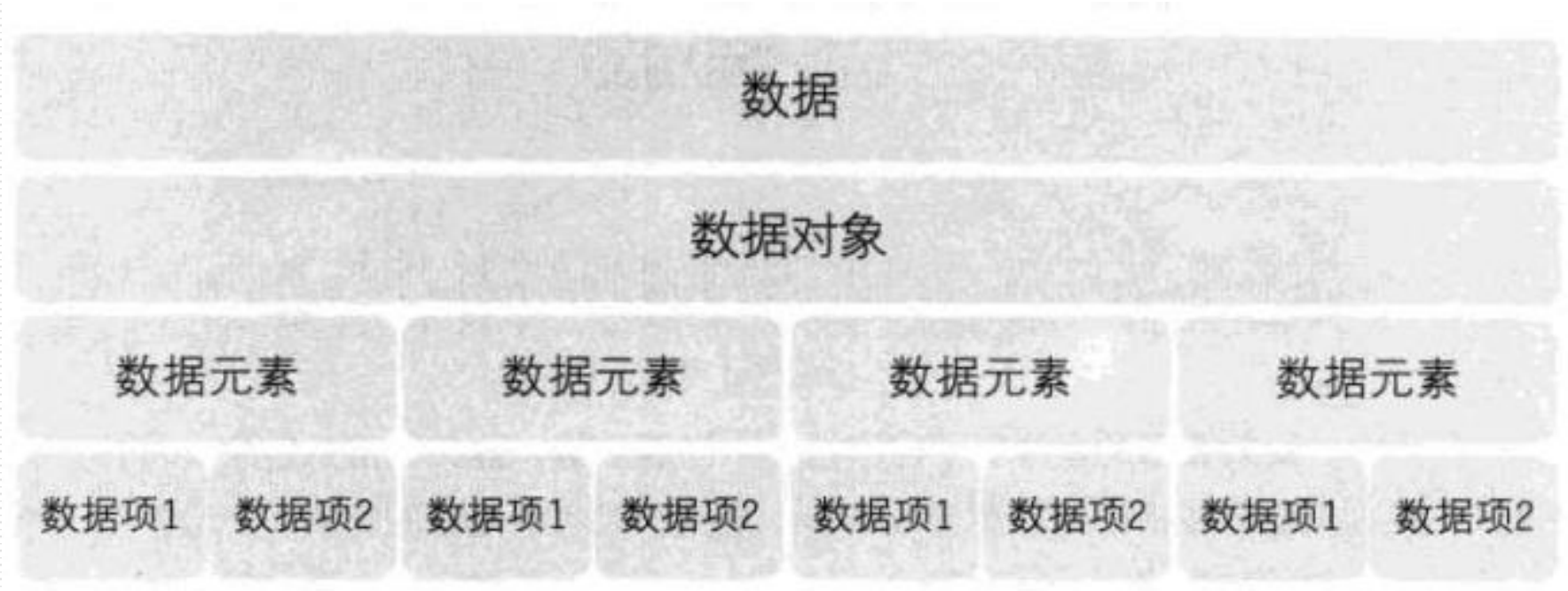
例如：学生信息表中的学号、姓名、性别等。

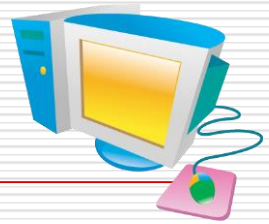
4. 数据对象（Data Object）:是性质相同的数据元素的集合，是数据的一个子集。

例如：整数数据对象是集合N={0，1，-1，2，-2，……}



基本概念之间的关系


数据
数据对象
数据元素
数据元素
数据元素
数据元素
数据项1
数据项2
数据项1
数据项2
数据项1
数据项2
数据项1
数据项2



5. Def2数据结构 (作为研究对象的数据结构)

——是相互之间存在一种或多种特定关系的数据元素的集合。数据元素之间的关系称为结构。

6. Def3数据结构 (形式定义)

——数据结构是一个二元组

$$Data\_Structure = (D, S) \qquad (1\text{-}1)$$

其中，D是数据元素的有限集，S是关系的有限集。

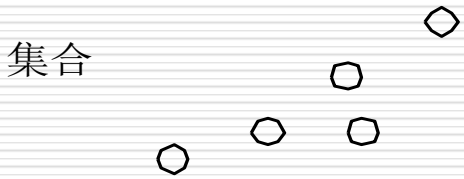
7. 逻辑结构： 逻辑关系是指元素之间的关联方式或称“邻接关系”，数据元素之间的逻辑关系的整体称为逻辑结构。数据的逻辑结构就是数据的组织形式。

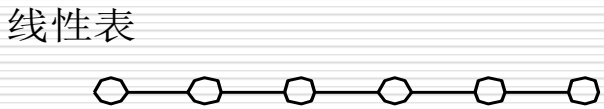
线性结构： 数据元素之间存在一个对一个的关系，按逻辑关系依次排列成一条锁链。

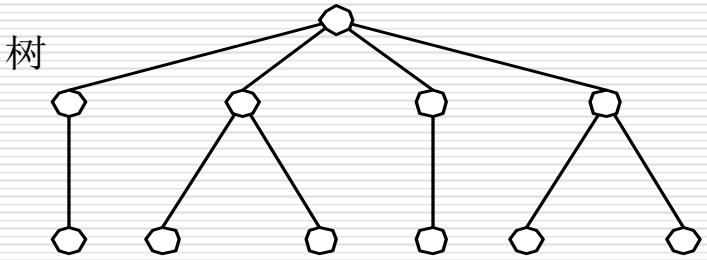
非线性结构：

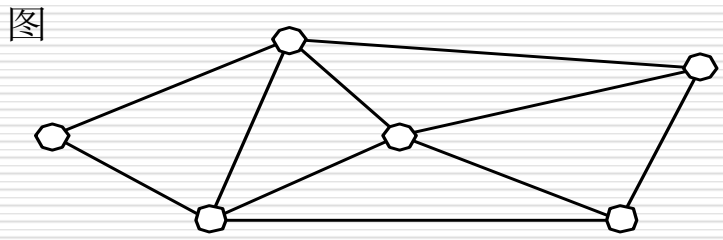
- 集合：除了“同属于一个集合”外别无其它关系。
- 树形结构：一对多，具有分枝、层次特性
- 图状结构：多对多，最复杂。其中各个数据元素按逻辑关系互相缠绕，任何两个都可邻接。

四类基本逻辑结构示意

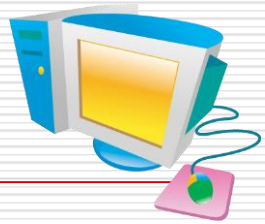








关于逻辑结构，以下几点需特别注意：



①逻辑结构与数据元素本身的形式、内容无关；

②逻辑结构与数据元素的相对位置无关；

③逻辑结构与所含数据元素的个数无关。

故一些表面上很不相同的数据可以有相同的逻辑结构。因此逻辑结构是数据组织的某种“本质性”的东西。

7. 存储结构（物理结构）：数据结构（逻辑结构）在计算机中的表示（又称映像）称为数据的物理结构，又称存储结构。它包括数据的表示和关系的表示。

8. 结点:数据元素在计算机中的映像（位串），对应于各数据项的子位串称为数据域。

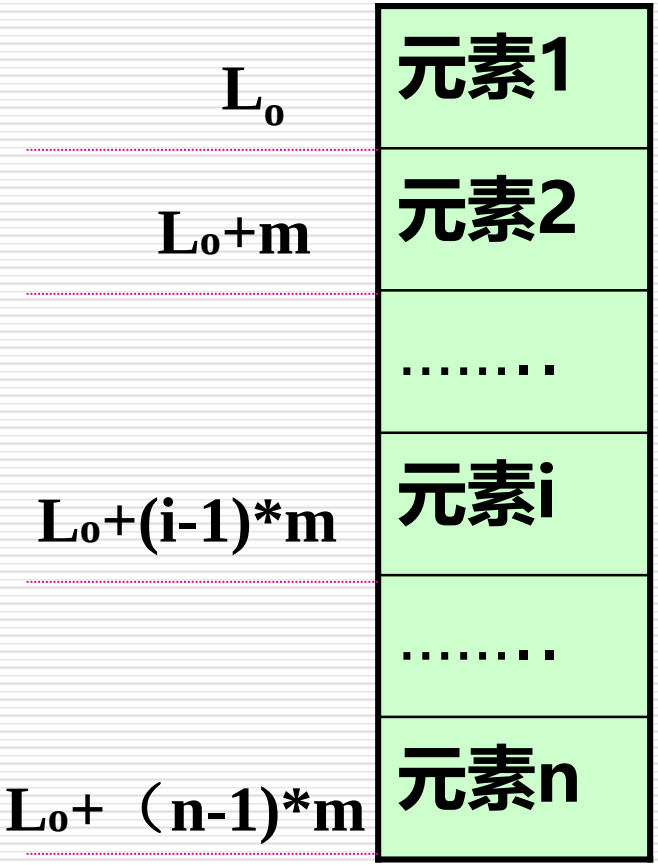
存储结构分为：

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

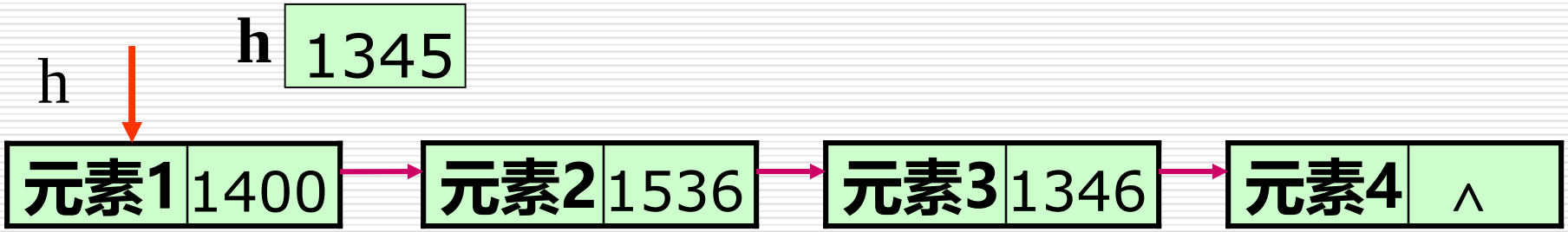
链式存储结构——借助指示元素存储地址的指针表示数据元素间的逻辑关系

存储地址　存储内容

顺序存储



Loc(元素i)=Lo+（i-1)*m



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

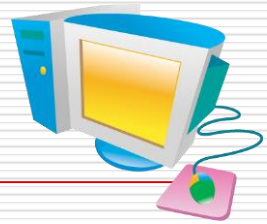
链式存储

9. 数据类型（Data Type）：是一组性质相同的值的集合和定义在这个集合上的一组操作的总称。

{ 原子类型：其值不可以分解。
结构类型：可分解，其值由若干成分按某种结构组成，其成分可以是结构的，也可以是非结构的。

例 C语言中，提供int, char, float, double等基本数据类型，数组、结构体、共用体等构造数据类型，还有指针、空(void)类型等。用户也可用typedef自己定义数据类型。

```
typedef   struct
{     int   num;
      char  name[20];
      float   score;
}STUDENT;
STUDENT    stu1,stu2, *p;
```



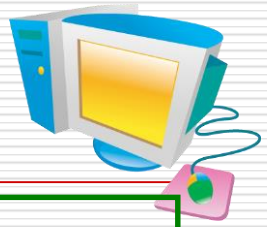
10. 抽象数据类型（Abstract Data Type, ADT）：
是指一个数学模型及定义在该模型上的一组操作。

11. 抽象数据类型的表示: 可用以下三元组表示

（D，S，P）

其中，D是数据对象，S是D上的关系集，P是对D的基本操作集。

采用以下格式定义抽象数据类型：



ADT　抽象数据类型名{

　　数据对象：〈数据对象的定义〉

　　数据关系：〈数据关系的定义〉

　　基本操作：〈基本操作的定义〉

}ADT　抽象数据类型名

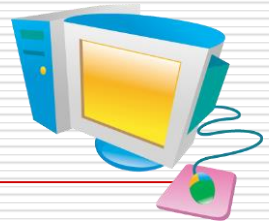
其中数据对象和数据关系的定义伪码描述，而基本操作的定义格式为：

基本操作名（参数表）

　　初始条件：<初始条件描述>

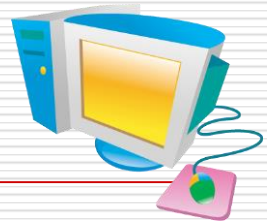
　　操作结果：<操作结果描述>

1.3 抽象数据类型的表示与实现



1 预定义常量和类型

```
//函数结果状态代码
#define   TRUE    1
#define   FALSE   0
#define   OK    1
#define    ERROR    0
#define    INFEASIBLE   -1
#define   OVERFLOW   -2
//Status  是函数的类型，其值是函数结果状态代码
typedef   int   Status;
```



2 数据的存储结构用C的类型定义（typedef）描述。

typedef 类型名 新类型名;

例 **typedef int Status;**

数据元素类型约定为**ElemType,** 由用户在使用该数据类型时自行定义。

类C的数据类型

(1) 数组

1） 数组类型变量（数组变量）由一组类型相同的数据元素组成.

2）数组的类型定义和变量定义

typedef　数组元素类型名　数组类型名[常量表达式];

数组类型名　　数组变量名;

例 某班40个学生的数学成绩，可以用有40个数组分量的整型数组变量存储。

```
Typedef int SCoreType[40];
SCoreType class;
```

数组类型名

数组变量

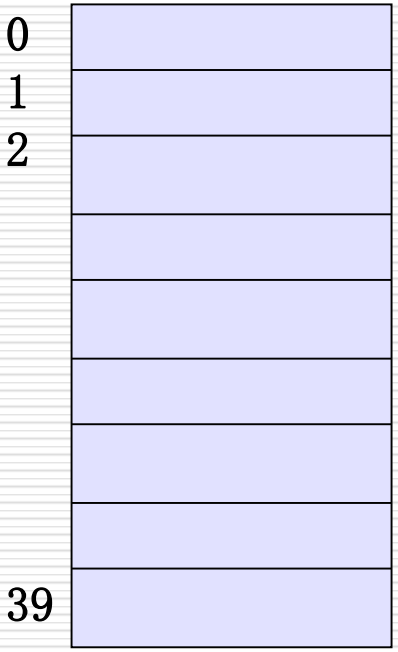
3）数组在内存中的存储示意图

4）数组分量(数组元素）的引用

数组变量[下标]

例：...

```
for ( i=0; i<=39; ++i)
    class[i]= 0;
```



数组变量

数组变量

数组元素下标

(2) 结构体

1）结构类型变量（结构变量）由一组类型可以不同的数据元素组成

2）结构的类型定义和变量定义

typedef　结构定义　结构类型名；

结构类型名　结构变量名；

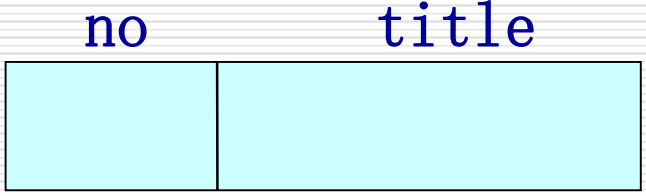
例 一本书可以用有2个数据成员（数据域）结构变量存储。

```
Typedef struct {
        int no;
        char title[40];
}BookType;
BookType book1;
```

结构类型名

结构变量名

3）结构变量在内存中的存储示意图

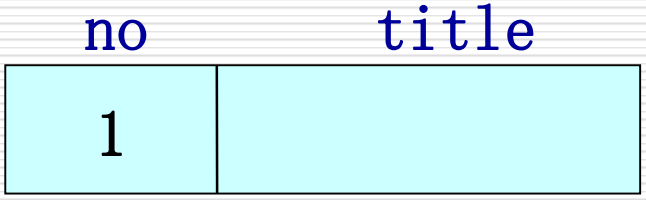


4)结构变量的引用

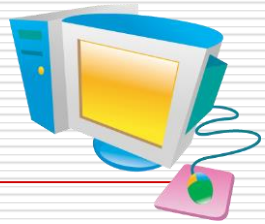
结构变量名.成员名(结构变量名.域名）

例 ：

```
...
book1.no=1;
scanf(“%s”,book1.title);
...
```



（3）指针



1）指针类型变量（指针变量）用于存储变量地址（或称指向该变量）

2）指针的类型定义和变量定义

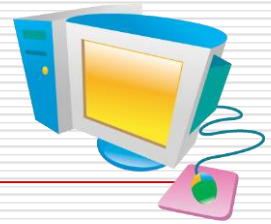
（只介绍本课程用到的指向结构变量指针类型）

typedef　结构定义　*指针的类型名；

类型定义

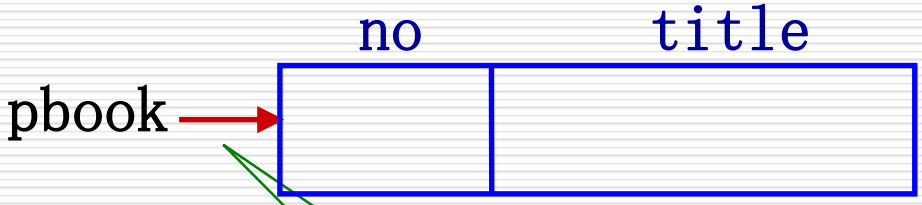
指针的类型名　　指针变量名；

变量定义



例
```
typedef struct {
        int  no;
        char title;
}*BookPtType;
BookPtType  pbook;
```

指向结构类型变量的指针类型



pbook是指针变量，存放结构类型变量的地址，通常用箭头表示pbook中存放的是它所指向的结构变量的地址。

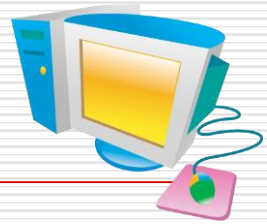
指向结构类型变量的指针变量

3)指针所指变量的引用

指针变量->结构变量成员名（指针变量->结构变量的域名）

例
```
typedef struct {
        int  no;
        char title;
}*BookPtType;
BookPtType  pbook;
pbook->no=1;
scanf("%s",pbook->title);
```

3 本课程中算法描述的约定



1）本课程中的算法都用以下形式的函数描述：

函数类型　函数名（函数参数表）{

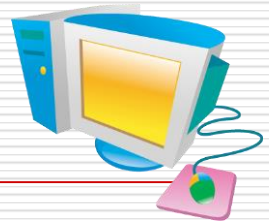
//算法说明

　　语句序列

}//函数名

例 求两个整数m，n中的最大数max的算法。

```
int  Maxnum (int m,int n) {
//该算法返回两个整数m，n中的最大数
if (m >= n) max = m;
else max= n;
return max;
}
```



2) 除了函数的参数需要说明类型外，算法中使用的辅助变量可以不作变量说明，必要时对其作用给予注释。一般而言，a、b、c、d、e 等用作数据元素名，i、j、k、l、m、n等用作整型变量名，p、q、r 等用作指针变量名;

3）若以操作成功或失败为函数的返回值时，函数的类型定义为**Status**;

4）为了便于算法描述，使算法易于理解，类C中函数的形参，除了允许值参外，**增添了C++语言的引用参数**。在函数的形参表中，以&打头的参数即为引用参数;

引用参数是实参的别名，所谓别名就是同一变量的另外一个名字。

例：

```
#include<stdio.h>
void fa(int a)
{
 a++;
 printf("in fa: a=%d\n",a);
 }

void fb(int &a)
{
 a++;
 printf("in fb: a=%d\n",a);
}
```

```
void main()
{
 int n=1;
 printf("in main,before fa: n=%d\n",n);
 fa(n);
 printf("in main,after fa: n=%d\n",n);
 fb(n);
 printf("in main,after fb: n=%d\n",n);
 getchar();
 }
```

输出结果：

```
in main,before fa: n=1
in fa: a=2
in main,after fa: n=1
in fb: a=2
in main,after fb: n=2
```

对数据有修改作用的操作（函数）要用引用参数作形参，而不能用值参作形参。

4 内存的动态分配与释放

使用**new**和**delete**动态分配和释放空间：

分配空间　指针变量=new数据类型

释放空间　delete指针变量

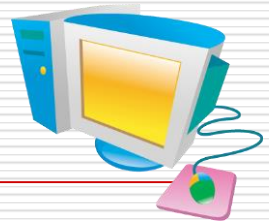
5 赋值语句

简单赋值　　变量名=表达式;
串联赋值　　变量名1=变量名2= … 变量名k=表达式;
成组赋值（变量名1，…变量名k）=（表达式1，…，表达式k）;
结构名=结构名;
结构名=（值1，…，值k）;
变量名[ ]=表达式;
变量名[起始下标. .终止下标]=变量名[起始下标. .终止下标];
交换赋值　　变量名← →变量名;
条件赋值　　变量名=条件表达式？表达式T;表达式F;

6 选择语句

条件语句1 if(表达式)语句；
条件语句2 if（表达式）语句；
else 语句；
开关语句 switch（表达式）

```
{
    case值1：语 句序列1；break；
    …
   case值n：语句序列n；break；
   default：语句序列n+1；
}
```



7 循环语句

for 语句　for （赋初值表达式序列；条件；修改表达式序列）语名；

while 语句　while(条件)语名；

do-while 语句 do{

语句序列；

}while (条件);

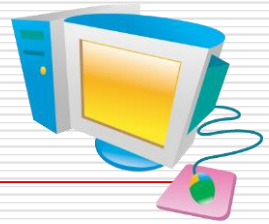
8 结束语句

函数结束语句　return 表达式；

return;

case 结束语句　break;

异常结束语句　exit(异常代码)



9 输入和输出语句有

输入语句　cin>>变量1>>…>>变量n;

输出语句　cout<<表达式1<<…<<表达式n;

10 基本函数有

求最大值　max (表达式1，…，表达式n)

求最小值　min(表达式1，…，表达式n)

求绝对值　abs(表达式)

1.4 算法和算法分析

1. 算法（Algorithm)的定义及特性

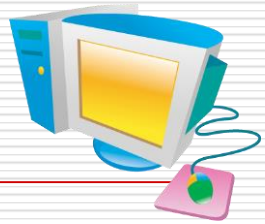
算法是对特定问题求解步骤的一种描述,它是指令的有限序列,其中每一条指令表示一个或多个操作。

例　求两个正整数 m，n 中的最大数MAX的算法。

（1）若 m > n 则 max=m

（2）若 m <= n 则 max=n

描述算法的方法有很多：自然语言、流程图、伪码语言、计算机语言等，用计算机语言表达的算法就是程序。



算法（Algorithm)的特性

(1) 有穷性：有限步骤之内正常结束。

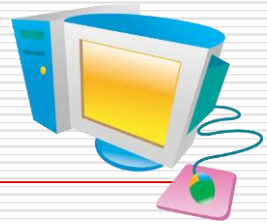
(2) 确定性： 算法中的每一个步骤必须有确定含义， 无二义性。

(3) 可行性： 原则上能精确进行， 操作可通过已实现的基本运算执行有限次而完成。

(4) 输入：有多个或0个输入。

(5) 输出： 至少有一个或多个输出。

2. 评价算法优劣的基本标准



- 正确性

(1) 所设计的程序没有语法错误；

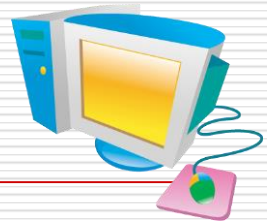
(2) 所设计的程序对于几组输入数据能够得出满足要求的结果；

(3) 所设计的程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得到满足要求的结果。

(4) 程序对于一切合法的输入数据都能产生满足要求的结果。

- 可读性
- 健壮性
- 时间效率高和存储量低（高效性）

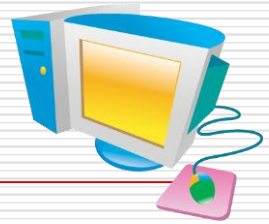
算法的效率的度量



- 算法效率：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

  事后统计

  事前分析估计



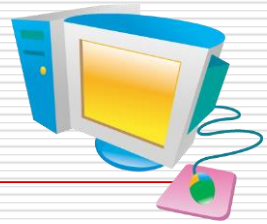
事后统计：利用计算机内的计时功能，不同算法的程序可以用一组或多组相同的统计数据区分

缺点：

①必须先运行依据算法编制的程序

②所得时间统计量依赖于硬件、软件等环境因素，掩盖算法本身的优劣

③算法的测试数据设计困难，并且程序的运行时间往往与测试数据的规模有很大关系



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

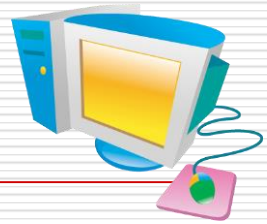
③程序语言

④编译程序产生机器代码质量

⑤机器执行指令速度

同一个算法用不同的语言、不同的编译程序、在不同的计算机上运行，效率均不同，————使用绝对时间单位衡量算法效率不合适

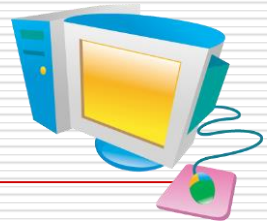
3. 算法的时间复杂度



1） 问题规模和语句频度

问题规模：是算法求解问题输入量的多少，是问题大小的本质表示，一般用整数n表示。一般来说n越大算法的执行时间越长。

一个算法的执行时间大致上等于其所有语句执行时间的总和， 对于语句的执行时间是指该条语句的执行次数和执行一次所需时间的乘积。



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];
```

语句频度（Frequency Count): 重复执行的次数：n*n;

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

即：矩阵加法的运算量和问题的规模n的平方是同一个量级

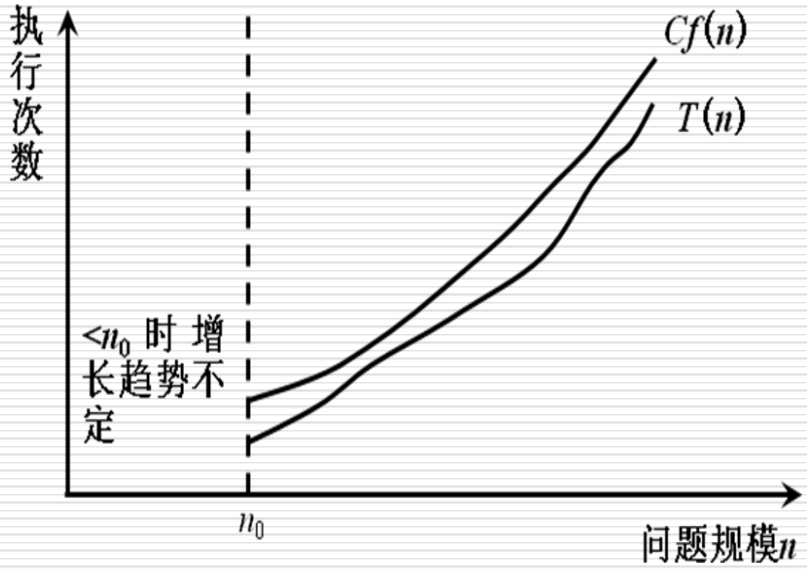
•算法中基本语句重复执行的次数是问题规模n的某个函数f(n),算法的时间量度记作：

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

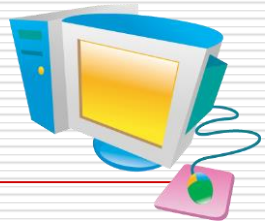
表示随着n的增大，算法执行的时间的增长率和f(n)的增长率相同，称渐近时间复杂度，简称时间复杂度

数学符号“O”的定义为：

若$T(n)$和$f(n)$是定义在正整数集合上的两个函数，则$T(n)=O(f(n))$表示存在正的常数C和n_0，使得当$n \geqslant n_0$时都满足$0 \leqslant T(n) \leqslant Cf(n)$。



分析算法时间复杂度的基本方法



- 找出语句频度最大的那条语句作为基本语句
- 计算基本语句的频度得到问题规模n的某个函数$f(n)$
- 取其数量级用符号“O”表示

```
x = 0;  y = 0;
```

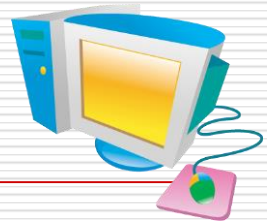
```
for ( int k = 0; k < n; k ++ )
    x ++;
```

```
for ( int i = 0; i < n; i++ )
   for ( int j = 0; j < n; j++ )
     y ++;
```

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

$f(n)=n^2$

时间复杂度是由嵌套最深层语句的频度决定的



```
void exam ( float x[ ][ ], int m, int n ) {
    float sum [ ];
    for ( int i = 0; i < m; i++ ) {
        sum[i] = 0.0;
        for ( int j = 0; j < n; j++ )
            sum[i] += x[i][j];
    }
    for ( i = 0; i < m; i++ )
        cout << i << " : " <<sum [i] << endl;
}
```

T(n) = O(m*n)

f(n)=m*n

例1：N×N矩阵相乘

```
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];
    }
```

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

算法中的基本操作语句为c[i][j]=c[i][j]+a[i][k]*b[k][j];

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

例2：

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

定理1.1

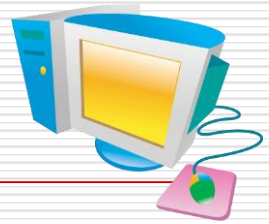
若$f(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$是$m$次多项式，则$T(n)=O(n^m)$。

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

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

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

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



例3：分析以下程序段的时间复杂度

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

$2^{f(n)} > n$ 即$f(n)>\log_2 n$，

取最小整数值$f(n)=[\log_2 n]+1$

所以该程序段的时间复杂度$T(n)=O(\log_2 n)$

有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同

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;
```

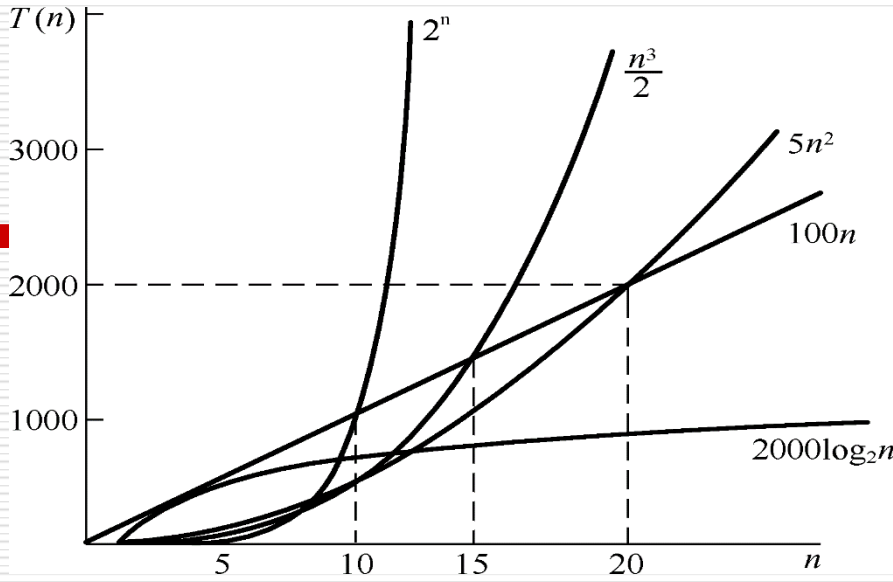
- 最好情况：1次
- 最坏情况：n
- 平均时间复杂度为:O(n)

表1 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	32 768	2147 483 648

一般地讲，前三种可实现，后三种虽理论上是可实现的，实际上只有当n限制在很小的范围时才有意义，当n较大时，不可能实现

当n取得很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊



时间复杂度T(n)按数量级递增顺序为：

复杂度低 → 复杂度高

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
$O(1)$	$O(\log_2 n)$	$O(n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n^3)$		$O(n^k)$	$O(2^n)$

渐进空间复杂度

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

空间复杂度：算法所需辅助存储空间的度量，记作：

$$S(n)=O(f(n))$$

其中n为问题的规模(或大小)

例5：将一维数组a中的*n*个数逆序存放到原数组中。

S(n) = O(1)
原地工作

【算法1】

```
for(i=0;i<n/2;i++)
{   t=a[i];
    a[i]=a[n-i-1];
    a[n-i-1]=t;
}
```

S(n) = O(n)

【算法2】

```
for(i=0;i<n;i++)
    b[i]=a[n-i-1];
for(i=0;i<n;i++)
    a[i]=b[i];
```

1.5 小结

1. 数据、数据元素、数据项、数据结构等基本概念

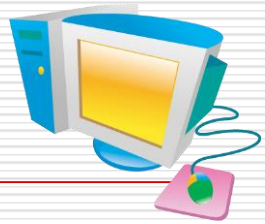
2、对数据结构的两个层次的理解

- 逻辑结构
- 存储结构

3、抽象数据类型的表示方法

4、算法、算法的时间复杂度及其分析的简易方法

作业选讲



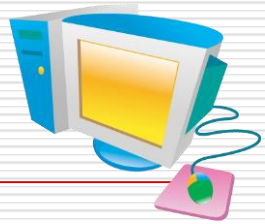
1. 分析下列各算法的时间复杂度。

（1）x=90; y=100;

```
while(y>0)
  if(x>100)
    {x=x-10; y--;}
  else x++;
```

答案：O(1)

作业选讲



（2）for (i=0; i<n; i++)

for (j=0; j<m; j++)

a[i][j]=0;

答案：O(m*n)

作业选讲

（3）s=0;

```
for( i=0; i<n; i++)
  for(j=0; j<n; j++)
      s+=B[i][j];
  sum=s;
```

答案：$O(n^2)$

作业选讲

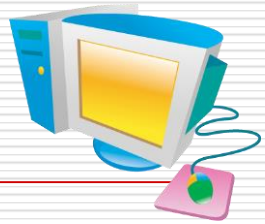
（4）i=1;

```
while(i<=n)
    i=i*3;
```

答案：$O(\log_3 n)$

解释：语句i=i*3;的执行次数为 $\lfloor \log_3 n \rfloor$。

作业选讲



（5）x=0;

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

答案：$O(n^2)$

解释：语句x++;的执行次数为n-1+n-2+……+1=n(n-1)/2。

作业选讲

（6）x=n; //n>1

y=0;

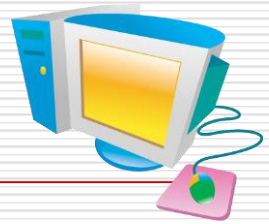
while(x≥(y+1)* (y+1))

y++;

答案：$O(\sqrt{n})$

解释：语句y++;的执行次数为 $\lfloor\sqrt{n}\rfloor$。

作业选讲



2. 以下算法的时间复杂度为（ D ）。

```
void fun(int n){
   int i=1;
   while(i<=n)
   i=i*2;
}
```

A. O(n)　B. $O(n^2)$　C.$O(n\log_2 n)$　D. $O(\log_2 n)$

作业选讲

3. 以下算法中加下划线的语句的执行次数为（ B ）。

```
int m=0,i,j;
for(i=1; i<=n;i++)
for(j=1;j<=2*i;j++)
  m++;
```

A. n　　B. n(n+1)　　C.n+1　　D.n^2

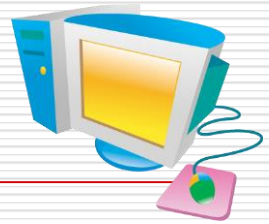
作业选讲

4. 以下算法的时间复杂度为（ C ）。

```
count t=0;
for(k=1;k<=n;k*=2)
for(j=1;j<=n;j++)
    count++;
```

A. $O(\log_2 n)$　B. $O(n)$　C.$O(n\log_2 n)$　D. $O(n^2)$

作业选讲

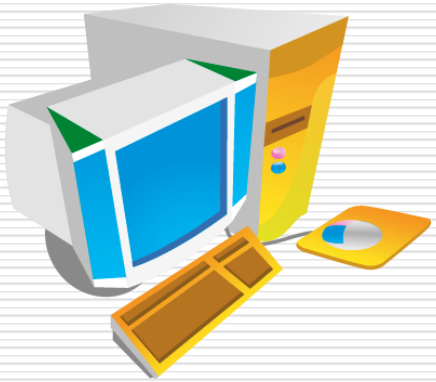


5.设n为正整数。试确定下列各程序段中前置以记号@的语句的频度为（ B ）。

```
i=1; k=0;
while(i<=n-1){
    @ k+=10*i;
      i++;
}
```

A. n　　B. n-1　　C.n+1　　D. n+2

第一章　结束



Thank you!