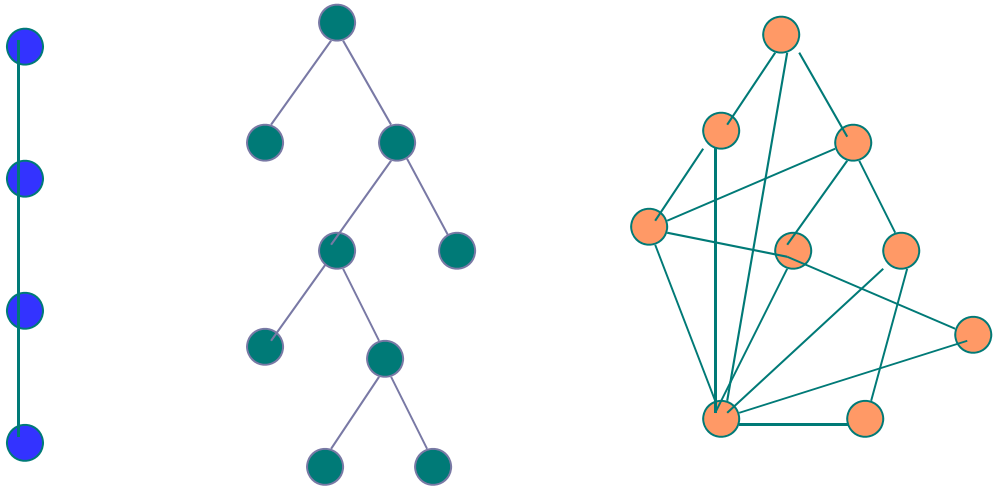
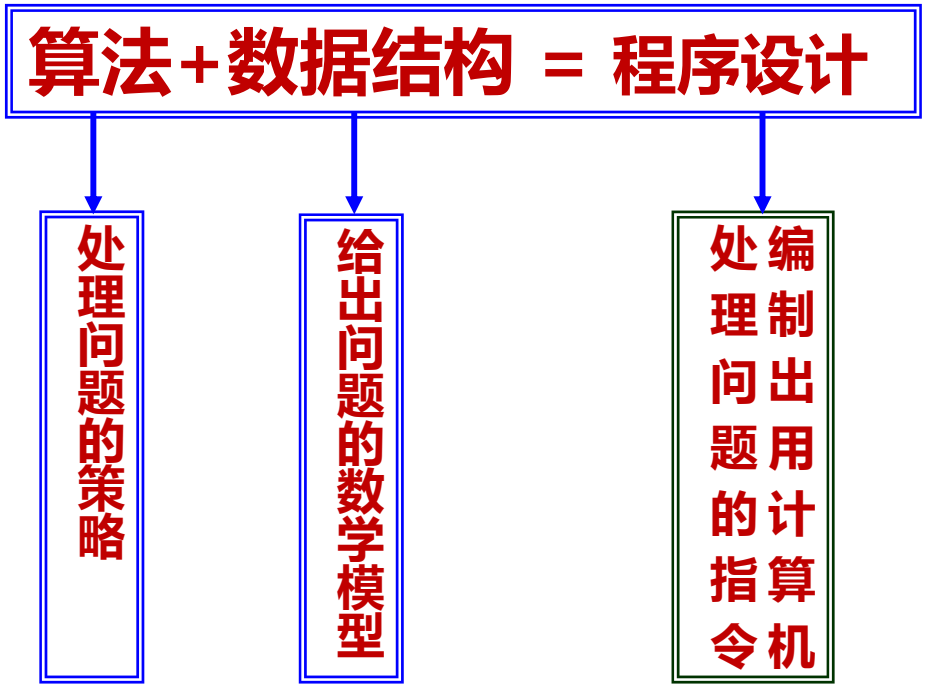
数据结构

华中科技大学计算机学院



Niklaus Wirth：沃思（图灵奖获得者）

Algorithm + Data Structures = Programs



本 课 程 的 任务

- 基本数据结构的定义、特性、运算与算法
  - ▲ 线性结构

    线性表；栈, 队列, 双队列；数组, 串
  - ▲ 非线性结构

    树, 二叉树；图, 网
- 数据结构的存储结构与实现

  选择存储结构, 设计算法
- 查找算法：顺序, 折半, 分块, 哈希, 二叉排序树等
- 排序算法：直接插入，堆排序，2-路归并，快速排序等
- 基本应用与综合应用

基本要求

- 阅读教材与参考书、听课、记笔记
- 完成一定数量的书面作业

教材和参考书

- 严蔚敏，吴伟民，数据结构（C语言版），清华大学出版社

- Alfred V. Aho, John E.Hopcroft（Born in 1939, Turing Award Winner ）, Jeffrey D. Ullman, Data Structures and Algorithms, 清华大学出版社

第一章　绪　　论

1.1　什么是数据结构

1.2　基本概念和术语

1.3　算法和算法分析

1.1 什么是数据结构

为了编写出一个“好”的程序，必须分析待处理的对象的特性以及各处理对象之间之间存在的关系，这就是”数据结构”这门学科形成和发展的背景.

用计算机解决具体问题时的步骤：

从具体问题抽象出一个适当的：

数学模型→设计一个解此数学模型的算法→编程序，

进行测试，调整直至得到最终解答。

而建立数学模型是分析具体问题的过程，包括：

- 分析具体问题中操作对象
- 找出这些对象间的关系，并用数学语言描述

这就是构建数据结构的过程。

数学模型分两类：

1）数值计算类：

例：根据三条边，求三角形面积。

假定：三条边依次为a, b, c三个实型数，

满足：a>0 ， b> 0 , c>0 , a+b>c , b+c>a， c+a>b,

则 $s=\frac{a+b+c}{2}$

$area=\sqrt{s*(s-a)*(s-b)*(s-c)}$

2）非数值计算类:

例一:图书馆的书目检索系统自动化

算法：？　　查找图书某一方面的信息

模型：？　　按照某一方面信息建立的表格

图书检索自动化

- 按图书的某些特征项建索引(如编号、书名、作者、出版日期等)
- 根据查询要求，按某一索引项进行排序
  - 操作对象：是一本书的基本信息，如：编号、书名、作者、出版日期
  - 对象间关系：按某一索引项的线性排序关系 如：按编号进行排序
- 本例表示了一种数据结构---线性数据结构

（线性结构）图书目录文件示例

001	高等数学	樊映川	S01	…
002	理论力学	罗远祥	L01	…
003	高等数学	华罗庚	S01	…
004	线性代数	栾汝书	S02	…
⋮	⋮	⋮	⋮	⋮

高等数学	001, 003, …
理论力学	002, …
线性代数	004, …
⋮	

樊映川	001, …
华罗庚	003, …
栾汝书	004, …
⋮	

L	002, …
S	001, 003, …
⋮	

数据：书目信息

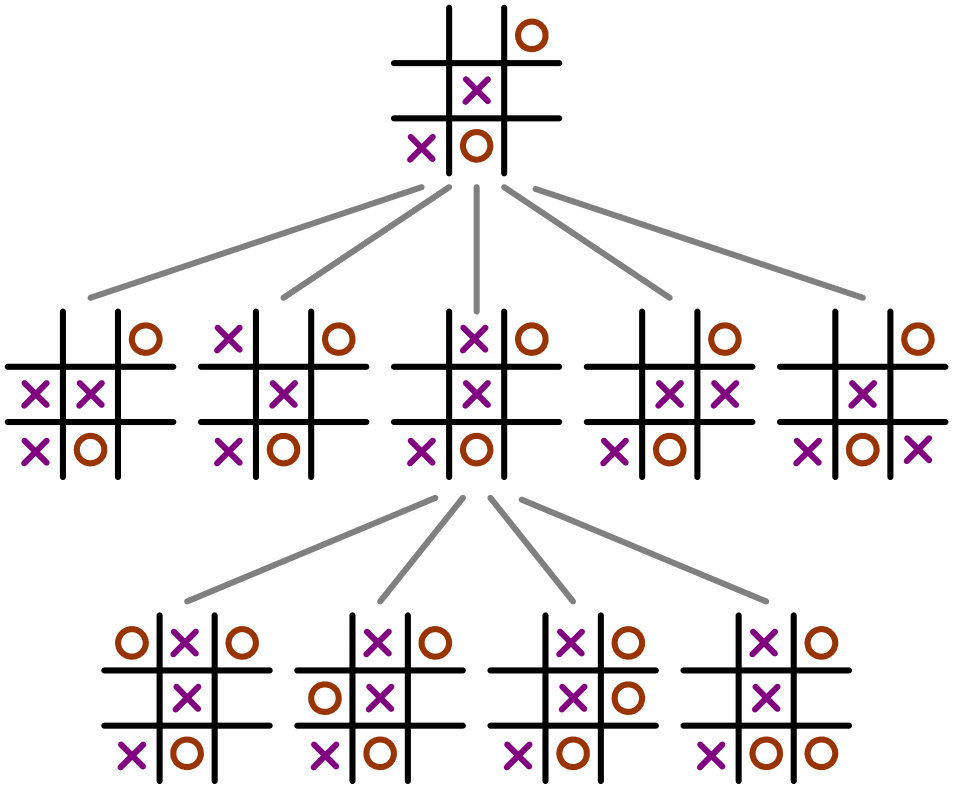
结构：顺序关系——线性结构

例二：计算机和人对弈问题

算法：？　　对弈的规则和策略

模型：？　　棋盘及棋盘的格局

（树结构）　井字棋对奕“树”



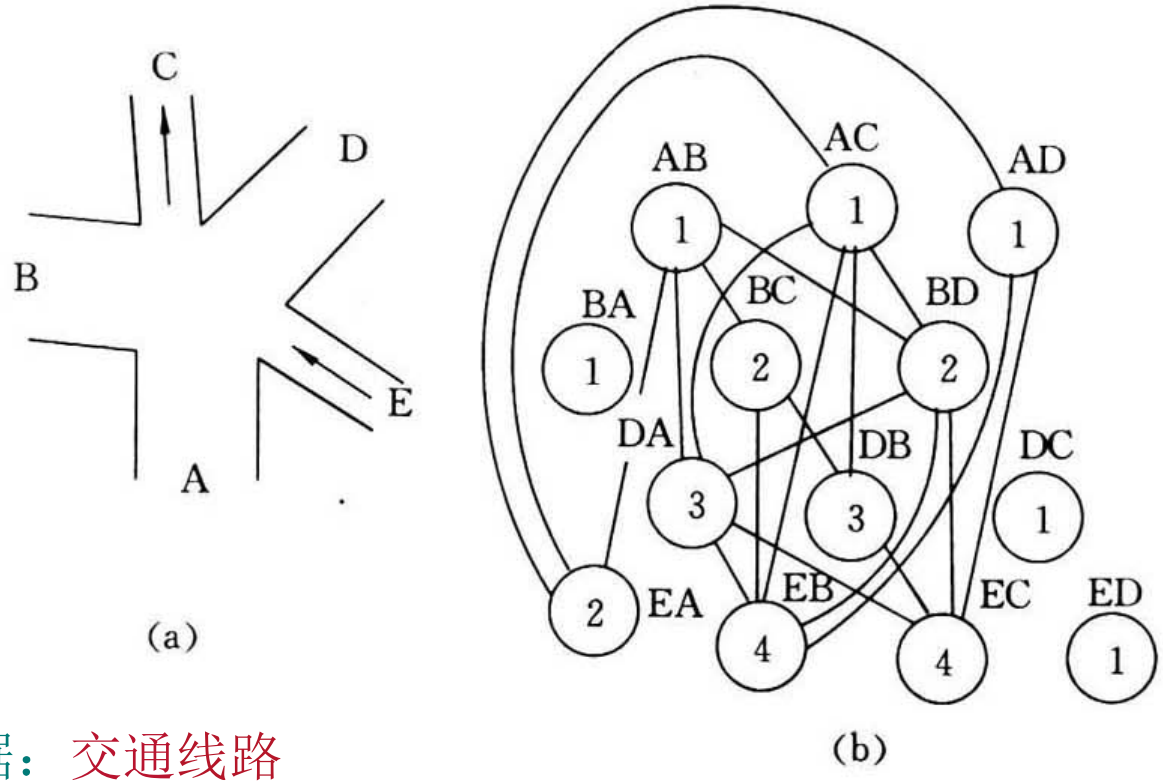
数据：棋盘格局

结构：层次关系——树结构

例三：多叉路口交通灯的管理

- 设置交通灯要满足以下几个要求：
  - 没有不通的道路
  - 行驶的车辆不发生相互碰撞
  - 保证路口达到最大车流量
- 对这个实际问题进行抽象，找出对象及其间的关系，从而建立数据结构模型：
  - 操作对象：通路
  - 关系：行驶冲突关系---不能同时通行

（图结构）　　五叉路口交通管理示意图



数据：交通线路

结构：网状关系——图结构

- 问题转换、抽象
  - 在上述图中，可以同时通车的道路对应的顶点染以相同颜色，有线段相连的顶点染以不同的颜色，且使所用颜色数量尽量少。
  - 至此这个实际问题便转换成图论中经典问题：

    图的染色问题---顶点的颜色与交通灯的颜色相对应。

- 检查数学模型与原问题的对应性
  - 图中已列出所有可能的通路 （路路通）；
  - 具有冲突关系的顶点（通路）颜色不同 （不发生碰撞）；
  - 颜色少，则每一种颜色（交通灯色）具有更多的车辆通行（流量最大）。
- 本例例示了另一种数据结构---图状数据结构

概括地说：

数据结构是一门讨论“描述现实世界实体的数学模型(非数值计算)及其上的操作在计算机中如何表示和实现”的学科。

数据结构是介于数学、计算机硬件、计算机软件三者之间的一门核心课程，在计算机科学中，数据结构不仅是一般程序设计的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

程序设计的实质是对确定的问题选择一种好的数据结构，加上设计一种好的算法。

1.2 基本概念和术语

1.数据（data）：

所有能输入到计算机中并被计算机程序加工、处理的符号的总称。

如：整数、实数、字符、声音、图象、图形等。

2.数据元素(data element)：

数据的基本单位。（元素、记录、结点、顶点）

在计算机程序中通常作为一个整体进行考虑和处理。

3.数据项(data item)：

是数据的不可分割的最小单位。如：姓名、年龄等

一个数据元素可由一个或多个数据项组成。

如：(姓名、年龄)

4.数据对象(data object)

由性质相同(类型相同)的数据元素组成的集合。

数据对象是数据的一个子集。

例1 由4个整数组成的数据对象

D1={20,-30,88,45}

例2 由正整数组成的数据对象

D2={1,2,3,...}

例3 由26个字母组成的数据对象

D3={A,B,C,...,Z}

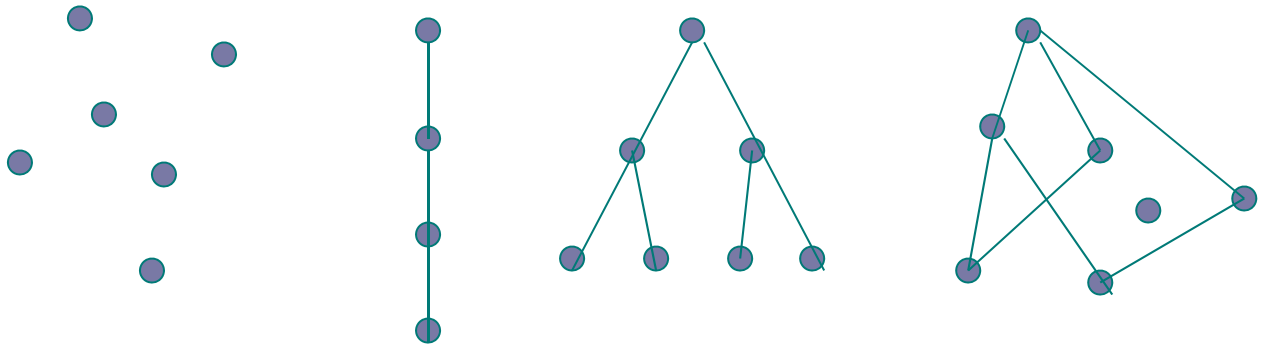
其中：D1，D3是有穷集，D2是无穷集。

5. 数据结构(data structure)

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。

四类基本结构：

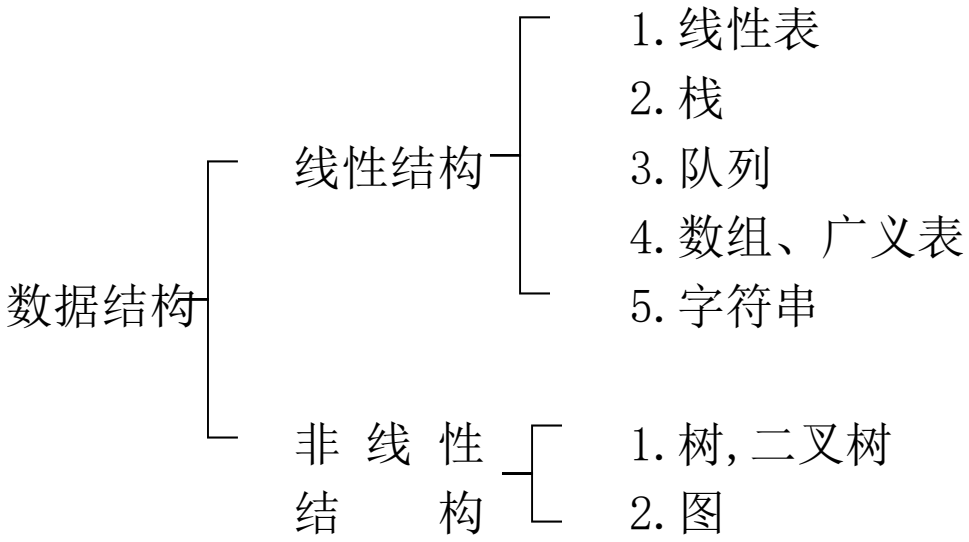


集合　　线性结构　　树形结构　　图状结构

6. 数据的逻辑结构

各数据元素之间的逻辑关系。

数据结构(逻辑结构)分类



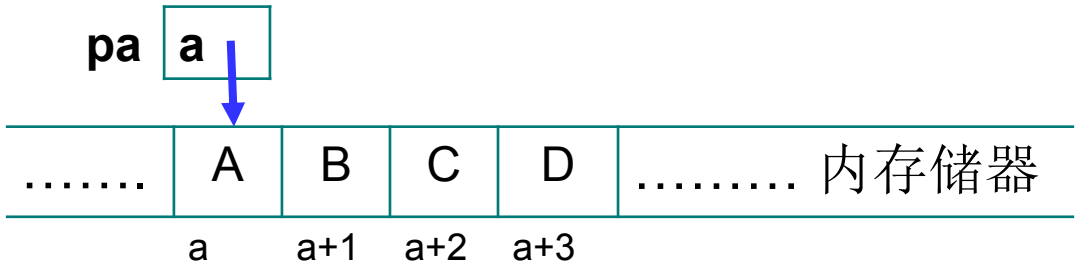
7. 数据的存储结构

数据结构在计算机存储器中的映象(mapping)。

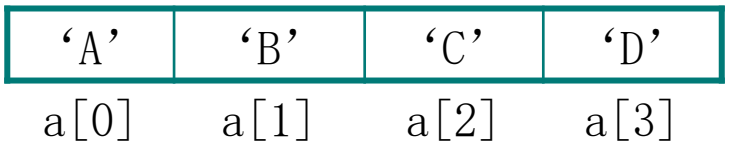
存储结构也称为：存储表示,物理结构,物理表示。

(1)顺序存储结构(向量,一维数组)

例. 线性表L＝（'A','B','C','D'）；

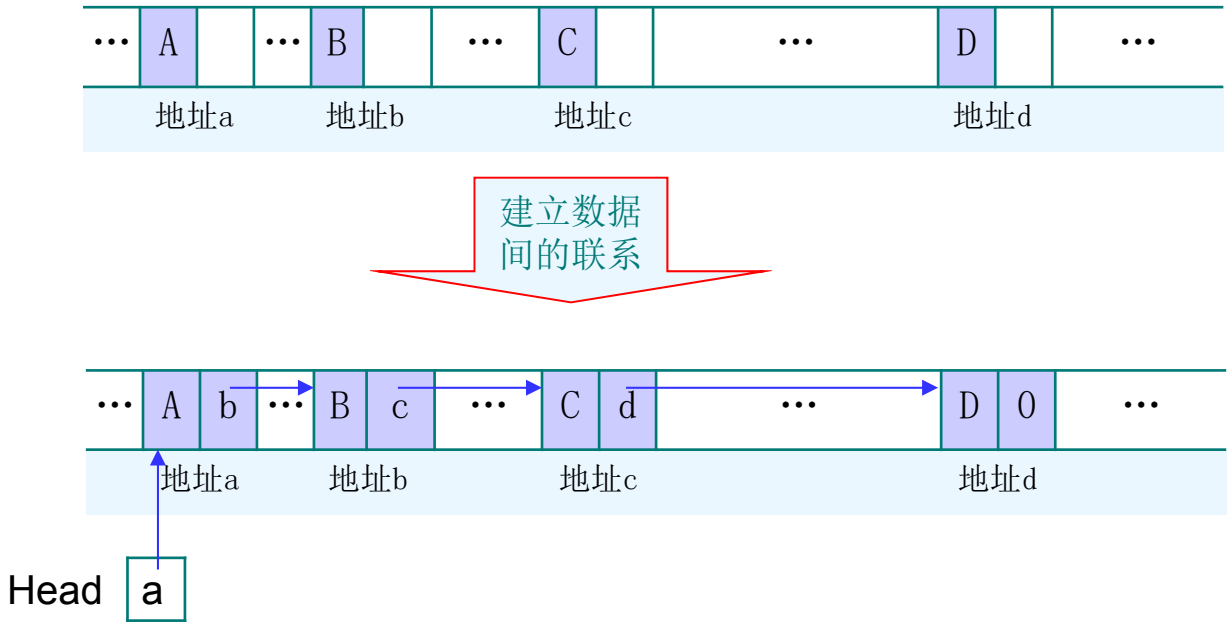


C语言实现方法：char a[4]={'A','B','C','D '};



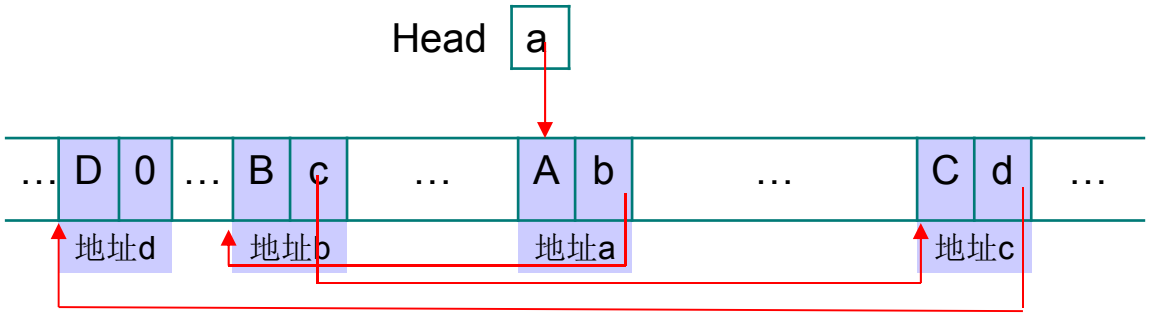
(2) 非顺序存储结构(链接表)

例. 单链表：分配不一定连续的空间

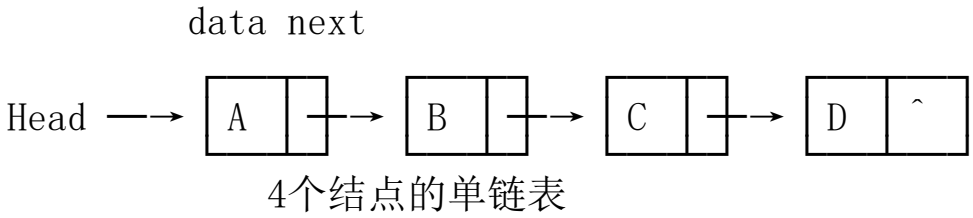


(2) 非顺序存储结构(链接表)

例. 单链表：存放数据的空间顺序可任意



一般表达形式：



4个结点的单链表

8.数据类型（data type)

是一个值的集合和定义在这个值上的一组操作的总称。

(1)原子类型(如：int,char,float等)

(2)结构类型(如：数组,结构,联合体等）

9.抽象数据类型(Abstract Data Type)

与计算机的实现无关的数据类型。

形式定义：

ADT 抽象数据类型名

{ 1.数据对象；

2.数据关系:一个或多个关系；

3.一组基本操作/运算

} ADT 抽象数据类型名

注意：ElemType是抽象元素类型。

1.3 算法和算法分析

1. 算法定义：

求解一个特定任务的指令的有限序列。

例. 求a[0]..a[n-1]中n个数的平均值(假定n>0)。

```
float average(float a[ ],int n)
{ int i; float s=0.0;    //累加器赋初值
  for (i=0; i<n; i++)
     s=s+a[i];           //a[i]累加到s中
  s=s/n;                 //计算平均值
  printf("ave=%f",s);    //输出
  return(s);             //返回
 }
```

其中：a,n为输入量；s为输出量。

2. 算法的5个特征：

(1) 有穷性：在有限步(或有限时间)之后算法终止。

例.{ i=0; s=0;
while (i<=10)
{ s+=i;
i++; } }

(2) 确定性：无二义性。

(3) 可行性：算法中的操作都是已经实现的基本运算执行有限次来实现的。

(4) 输入：有0或多个输入量。

(5) 输出：至少有一个输出量。

3. 算法设计要求：

(1) 正确性

a. 无语法错误；

b. 对n组输入产生正确结果；

c. 对特殊输入产生正确结果；

d. 对所有输入产生正确结果。

(2) 可读性：“算法主要是为了人的阅读与交流”。

(3) 健壮性：不同的输入都要有相应的反应。

(4) 高效与低存储量

下列描述不符合算法的什么特征和要求？

例1

```
void suanfa1( )
{ int i,s=0;
  for (i=0; i>=0; i++)   //死循环
    s++;                 //不能终止
 }
```

例2

```
float suanfa2( )
{ int x;
  float y;
  scanf("%d", &x);
  y=sqrt(x);             //当x<0时,出错
  return(y);
 }
```

4. 算法的描述工具：

（1） 自然语言；

（2） 程序设计语言；

（3） 流程图（框图）；

（4） 伪码语言；

是一种包括高级程序设计语言的三种基本结构（顺序、选择、循环）和自然语言成分的“面向读者”的语言。

（5） 类C

介于伪码语言和程序设计语言之间的一种表示形式。保留了C语言的精华；不拘泥与C语言的语法细节；同时也添加了一些C++的成分。

特点：便于理解、阅读；能方便的转换成C语言。

目的：便于简明扼要的描述算法；突出算法的思路。

〈1〉预定义变量和类型：

```
#define TRUE     1
#define FALSE    0
#define OK         1
#define ERROR   0
#define INFEASIBLE     -1
#define OVERFLOW      -2
typedef int Status;      //一般用于算法函数的返回类型
typedef char ElemType;  // 此时char等价于Elemtype;
typedef struct node { ElemType     elem;
                      struct node  *next;
                     } NODE, *LINK;
struct node m,*pm;         NODE  n,*pn;
LINK  p;                              // 等价于 struct node *p;
```

<2> 函数： 用以表示算法

函数类型 函数名（函数参数表） {

//算法说明

语句序列

} //函数名

注：

算法说明应包括功能说明，输入，输出；

为提高算法可读性，关键位置加以说明；

明确函数实参和形参的匹配规则，以便能正确使用算法函数。

<3> 其他

算法描述举例:

问题：设一维数组a[0..n-1]中已有n个整数,其中n为常数,试设计算法:求a[]中的最大值。

算法基本思想:

1. maxai=a[0];
2. i=1;
3. 若i<=n-1,则:
   - 3.1 若a[i]>maxai,则 maxai=a[i];
   - 3.2 i++;
   - 3.3 转3
4. maxai为最大值。

C 函数(算法)

```
//求数组a中n个元素的最大值
int a_maxint(int a[],int n)
{ int j,maxai=a[0];                    //最大值的初值
  for (j=1; j<=n-1; j++)
    if (a[j]>maxai)                    //比较元素大小
      maxai＝a[j];                     //新的最大值
  printf("maxai＝%d\n",maxai);          //输出
  return maxai;
}
```

5. 算法的时间复杂度:

- 算法(或程序)中基本操作(或语句)重复执行的次数的总和
- 设n为求解的问题的规模,基本操作(或语句)执行次数总和称为语句频度,记作f(n)
- 时间复杂度记作T(n), T(n)=O(f(n))

例1 { int s;
scanf(“%d”,&s);
s++;
printf(“%d”,s);
}

其中：语句频度为：f(n)=f(1)=3

时间复杂度为：T(n)=O(f(n))=O(3)=O(1)

O(1)称成为常量阶/常量数量级

例2 分析下面的算法

```
void sum(int a[],int n)
{ int s=0,i;              // 1次
  for(i=0; i<n; i++)      // n次
    s=s+a[i];             // n次
  printf("%d",s);         // 1次
 }
```

其中：语句频度为： f(n)=1+n+n+1

时间复杂度为：T(n)=O(f(n))=O(2n+2)=O(n)

O(n)称成为线性阶/线性数量级

例3 分析下面的算法

```
1.  void sum(int m,int n)
2.  { int i,j,s=0;                    // 1次
3.    for(i=1; i<=m; i++)             // m次
4.     { for(j=1; j<=n; j++)          // m*n次
5.         s++;                       // m*n次
6.       printf("%d",s);              // m次
7.     }
8.  }
```

其中:$f(m,n)=1+m+2*m*n+m=2mn+2m+1$

当m=n时,$f(n)=2n^2+2n+1$

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

$O(n^2)$ 称成为平方阶/平方数量级

例4 分析下面的算法；当n=5,指出输出结果

```
1. void sum(int n)
2. { int i,j,s=0;                    // 1次
3.   for(i=1; i<=n; i++)              // n次
4.    { for(j=1; j<=i; j++)          // ?次
5.       s++;                        // ?次
6.      printf("%d",s);              // n次
7.    }
8.  }
```

其中：第4行的次数为 1+2+...+n=n(1+n)/2

第5行的次数为 1+2+...+n=n(1+n)/2

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

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

$O(n^2)$ 称成为平方阶/平方数量级

例5

(a) {++x;s=0}

(b) For (i=1;i<=n;++i)

{++x;s+=x;}

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

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

{++x;s+=x;}

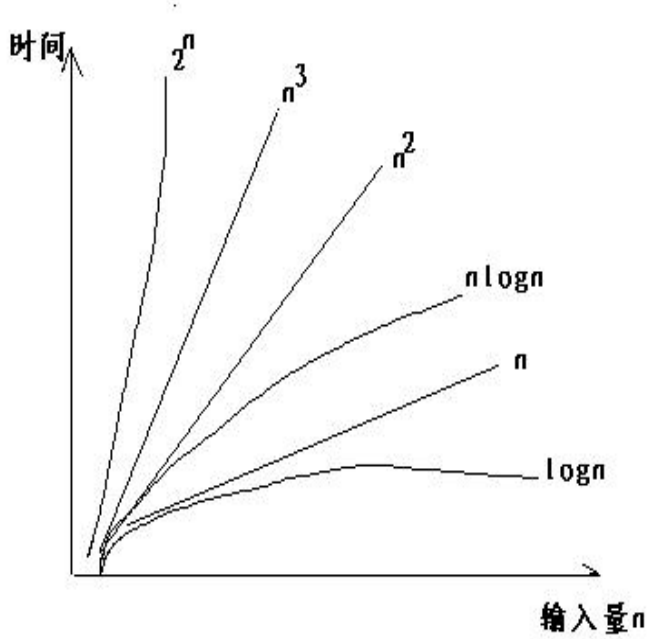
含基本操作“x增1”的语句频度分别为1、n和n^2，则这3个程序段的时间复杂度分别为O(1)、O(n)和$O(n^2)$

时间复杂度曲线

- 常见的时间复杂度：

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

- $O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<(n^2)<O(n^3)<<O(2^n)$



例6 冒泡排序的C语言算法(对数组a中n个数按递增次序排序)

```
1.  void  bubble1(int a[],int n)
2.  { int i,j,temp;
3.    for(i=1; i<n; i++)                    // ? 次
4.      for(j=0; j<n-i; j++)                // ? 次
5.        if (a[j]>a[j+1])                  // ? 次
6.          { temp=a[j];                    // ? 次
7.            a[j]=a[j+1];                  // ? 次
8.            a[j+1]=temp;                  // ? 次
9.          }
10.   for(i=0; i<n; i++)                    // n 次
11.     printf("%d",a[i]);                  // n 次
12. }
```

思考：在最好情况下，f(n)= ？　　T(n)=O(f(n))= ？

　　在最坏情况下，f(n)= ？　　T(n)=O(f(n))= ？

一般情况：

i值	原序列：											交换范围
		10	3	7	8	5	2	1	4	9	6	
1	第1遍：	3	7	8	5	2	1	4	9	6	10	[0—10-1)
2	第2遍：	3	7	5	2	1	4	8	6	9	10	[0—10-2)
3	第3遍：	3	5	2	1	4	7	6	8	9	10	[0—10-3)
4	第4遍：	3	2	1	4	5	6	7	8	9	10	[0—10-4)
5	第5遍：	2	1	3	4	5	6	7	8	9	10	[0—10-5)
6	第6遍：	1	2	3	4	5	6	7	8	9	10	[0—10-6)
7	第7遍：	1	2	3	4	5	6	7	8	9	10	[0—10-7)
8	第8遍：	1	2	3	4	5	6	7	8	9	10	[0—10-8)
9	第9遍：	1	2	3	4	5	6	7	8	9	10	[0—10-9)

最坏情况：

10 9 8 7 6 5 4 3 2 1

每次比较都发生数据交换。

最好情况：

1 2 3 4 5 6 7 8 9 10

每次比较都不发生数据交换。

```
1.  void  bubble1(int a[],int n)
2.  { int i,j,temp;
3.    for(i=1; i<n; i++)          // n-1 次
4.      for(j=0; j<n-i; j++)     // n-1+n-2+..+1 次
                                    =n(n-1)/2
5.        if (a[j]>a[j+1])      // n(n-1)/2次
6.          { temp=a[j];        // 0 或n(n-1)/2次
7.            a[j]=a[j+1];      // 0 或n(n-1)/2次
8.            a[j+1]=temp;      // 0 或n(n-1)/2次
9.          }
10.   for(i=0; i<n; i++)         // n 次
11.     printf("%d",a[i]);      // n 次
12. }
```

- 在最好情况下，$f(n)= n-1+n(n-1)+2n=n^2+2n-1$
- 在最坏情况下，$f(n)=5n^2/2+n/2-1$
- $T_{最好}(n)= T_{最坏}(n) = O(n^2)$

算法改进思想：

- **每一遍开始时，change=false，**处理时，一旦发生数据交换，就将change修改成true。结束时，若change未变，表示未发生数据交换，即已递增有序。

- 如当前序列为：

$a_0, a_1, \cdots, a_{i-1}, a_i, a_{i+1} \cdots a_{n-1}$.

当前需要处理的范围是0~i，　　$0 \leqslant i \leqslant n-1$

处理前，设置 change=false，

- 若$a_0 \leqslant a_1 \leqslant \cdots \leqslant a_{i-1} \leqslant a_i$，显然不会发生数据交换，所以change未变，即已是递增有序，排序操作可就此结束；否则一定会发生数据的交换，change被修改为true，需要进行下一遍的处理。

例6 改进的冒泡排序的“类C语言”算法时间复杂度分析。

```
1.  void  bubble2(int a[],int n)
2.  { i=n-1;                                    // 1
      do {
3.       change=FALSE;                         // 1 或n-1
4.       for(j=0; j<i; ++j)                    //n-1或n*(n-1)/2
5.         if (a[j]>a[j+1])                    //n-1或n*(n-1)/2
6.           { a[j] <--> a[j+1];               // 0 或n*(n-1)/2
7.             change=TRUE;                    // 0 或n*(n-1)/2
8.           }
9.       } while(change && --i>=1)              // 1 或n-1
10. }
```

- 在最好情况下：$T_{最好}(n)=O(f_{最好}(n))= O(n)$
- 在最坏情况下：$T_{最坏}(n)=O(f_{最坏}(n))= O(n^2)$
- 算法时间复杂度：$T(n)= T_{最坏}(n)= O(n^2)$

例7： 求解：S=1!+2!+.....+n!

算法1：

```
long    sum_of_fac(int n)
 {
  long s=0,t,i,j;
  for(i=1;i<=n;i++)                    //n次
    {
     t=1;                              //n次
     for(j=1;j<=i;j++)                 //n*(n+1)/2次
           t*=j;                       //n*(n+1)/2次
     s+=t;                             //n次
    }
  return (s);                          //1次
 }
```

求i!

$f(n)=n+n+n*(n+1)/2+ n*(n+1)/2+n+1=n^2+4n+1$

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

例7(续）：　求解：S=1!+2!+.....+n!

算法2：

```
  long    sum_of_fac(int n)
   {
    long s=0,t,i;
    t=1;                                  //1次
    for(i=1;i<=n;i++)                     //n次
      {
       t*=i;                              //n次
        s+=t;                             //n次
      }
    return (s);                           //1次
   }
```

求i!

f(n)=1+n+n+n+n+1=3n+2

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

6. 算法的空间复杂度:

- 执行算法所需存储空间的大小
- 记作， S(n)=O(f(n)),n为问题的规模
- 存储空间：寄存指令、常数、变量和输入数据
  对数据进行操作的工作单元
  实现计算所需用的空间