数据结构与算法

(一)

1 数据结构与算法概述

杜育根

ygdu@sei.ecnu.edu.cn

教 学 目 的

- 理解数据结构的概念，理解数据结构的逻辑和存储结构；
- 理解算法的概念和算法的基本特性，了解算法复杂度的度量方法；
- 理解线性数据结构，理解顺序存储和链式存储的存储方法；
- 描述栈和队列、串和数组这几个线性数据结构的概念；
- 了解非线性的数据结构，了解树、二叉树以及图的概念和数据结构；
- 理解排序的概念，描述插入、选择、冒泡和快速排序的算法；
- 理解查找的概念，描述顺序查找和折半查找的算法，并能够比较它们
- 理解递归的概念，能够在实践中了解递归的应用。

学习内容

- 数据结构的基本概念
- 算法的描述、流程图的使用以及算法的复杂度的衡量
- 顺序存储和链式存储的方法
- 栈、队列、串和数组的概念和用法
- 二叉树数据结构
- 查询、排序和递归算法

程序设计的本质

- 计算机科学家沃斯（N.Wirth）提出的:

“程序=数据结构+算法”

- 揭示了程序设计的本质：对实际问题选择一种好的数据结构，加上设计一个好的算法，而好的算法很大程度上取决于描述实际问题的数据结构。算法与数据结构是互相依赖、互相联系的。

第一节 数据结构概述

1. 《数据结构》研究的对象

(1) 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

☞**数据结构正是讨论非数值类问题的对象描述、信息组织方法及其相应的操作**

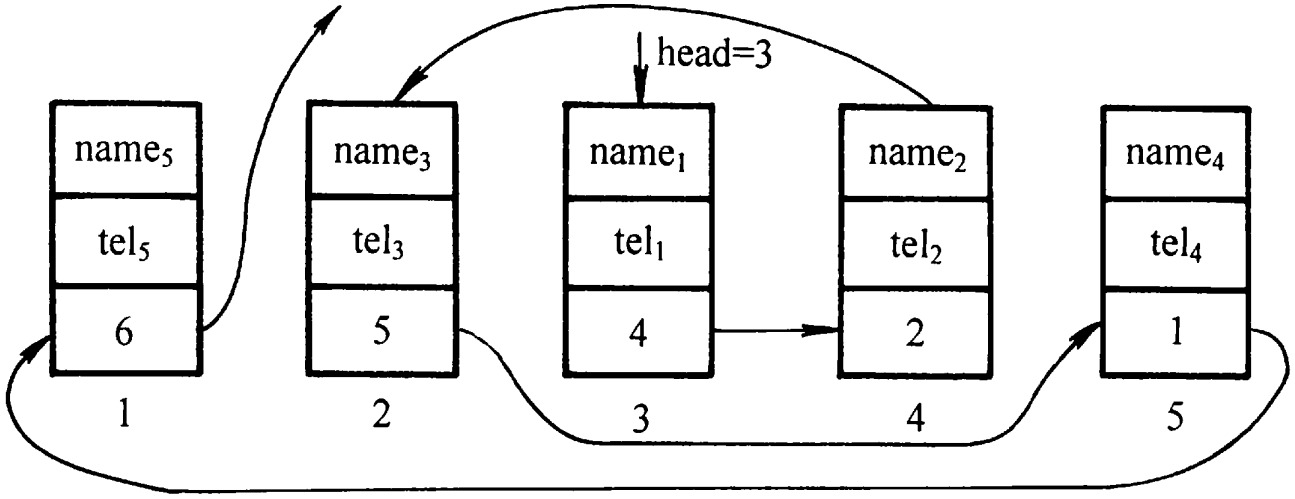
数据结构学科的定义：主要是为研究和解决如何使用计算机处理非数值问题而产生的理论、技术和方法。

例1

设有一个电话号码薄，有N个人的姓名和电话号码。要求设计一个程序，按人名查找号码，若不存在则给出不存在的信息。

姓　　名	$name_1$	$name_2$	$name_3$	……	$name_n$
电话号码	tel_1	tel_2	tel_3	……	tel_n

(a) 顺序存储



(b) 链式存储

2. 数据结构相关概念

1.基本概念和术语

数据元素、结点、数据项、关键字或主关键字、 次关键字、数据对象、数据结构

2.数据结构

特性相同的数据元素构成的集合中，如果在数据元素之间存在一种或多种特定的关系，则称之为数据结构 。

Data-Structure=(D,R)

其中，D是数据元素的有限集，R是D上关系的有限集。

3. 四类基本的数据结构

- **集合结构。**在集合结构中，数据元素间的关系是“属于同一个集合”。集合是元素关系极为松散的一种结构，各元素间没有直接的关联。
- **线性结构。**该结构的数据元素之间存在着一对一的关系。
- **树型结构。**该结构的数据元素之间存在着一对多的关系。
- **图形结构。**该结构的数据元素之间存在着多对多的关系，图形结构也称作网状结构。

(a)集合结构

(b)线性结构

(c)树形结构

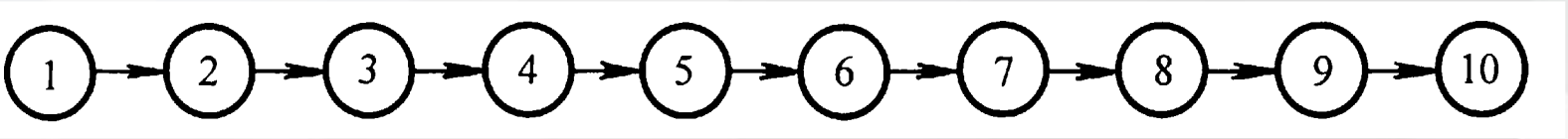
(d)图结构

[例2]

线性数据结构=(D,S)

D={1,2,3,4,5,6,7,8,9,10}

S={<1,2>,<2,3>,<3,4>,<4,5>,<5,6>,<6,7>,<7,8>,
<8,9>,<9,10>}



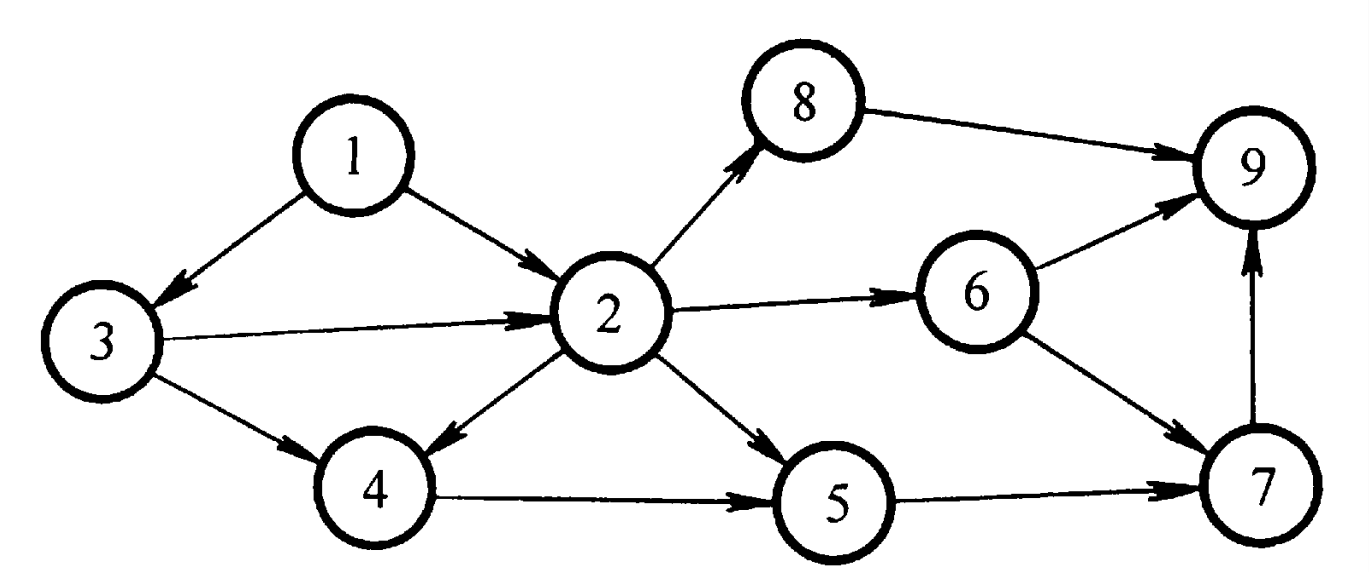
例3

图形数据结构=(D,R)

D={1, 2, 3, 4, 5, 6, 7, 8, 9}

R={<1,2>,<1,3>,<2,4>,<2,5>,<2,6>,<2,8>,<3,2>,

<3,4>, <4,5>,<5,7>,<6,7>,<6,9>,<7,9>,<8,9>}

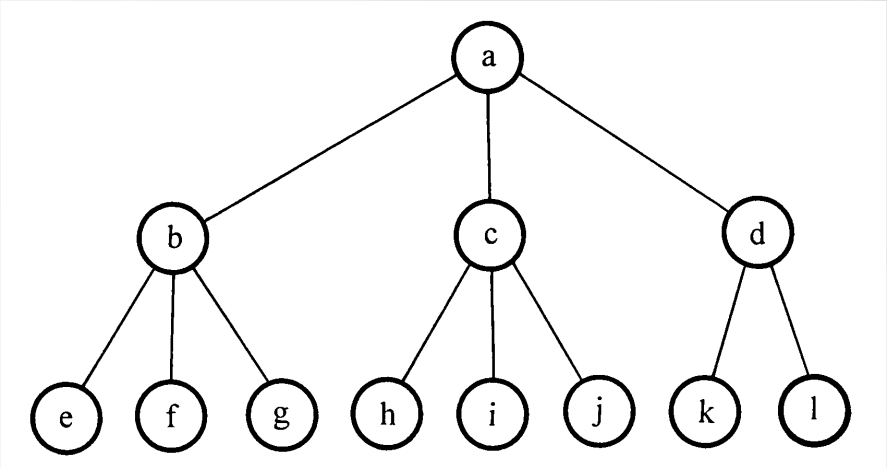


例4

树形结构 = （D,R)

D={a, b, c, d, e, f, g, h, i, j, k, l}

R={<a,b>,<a,c>,<a,d>,<b,e>,<b,f>,<b,g>,<c,h>,<c,i>,<c,j>, <d,k>,<d,l>}



4. 数据结构的分类

（1）按数据结构的性质划分

数据的逻辑结构——数据元素之间的逻辑关系

（**设计算法—— 数学模型**）

数据的物理结构——数据结构在计算机中的映像

（**存储结构，算法的实现**）

（2）按数据结构的操作来划分

静态结构——经过操作后，数据的结构特征保持不变（如数组）。

半静态结构——经过操作后，数据的结构特性只允许很小变迁（如栈、队列）。

动态结构——经过操作后，数据的结构特性变化比较灵活，可随机地重新组织结构（如指针）。

4. 数据结构的分类（续）

（3）按数据结构在计算机内的存储方式来划分

顺序存储结构——借助元素在存储器的相对位置来表示数据元素之间的逻辑关系。

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之间的逻辑关系

索引存储结构——在存储结点的同时，还建立附加的索引表，索引表中的每一项称为索引项，形式为：关键字，地址。

散列存储结构——根据结点的关键字直接计算出该结点的存储地址。

说明：**四种存储方法可结合起来对数据结构进行存储映像。**

第二节 算法概述

1. 算法的概念及特征

算法：对问题求解的描述，为解决问题给出的一个确定的、有限长的操作序列。

算法具有以下五个重要的特征：

1）**有穷性：**一个算法必须保证执行有限步之后结束。

2）**确切性：**算法的每一步骤必须有确切的定义。

3）**输入：**一个算法有0个或多个输入，以刻画运算对象的初始情况，所谓0个输入是指算法本身定除了初始条件。

4）**输出：**一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法没有实际意义。

5）**可行性：**算法原则上能够精确地运行，而且人们用笔和纸做有限次运算后即可完成。

2、算法的描述:

1）流程图

2）伪代码——类程序设计语言

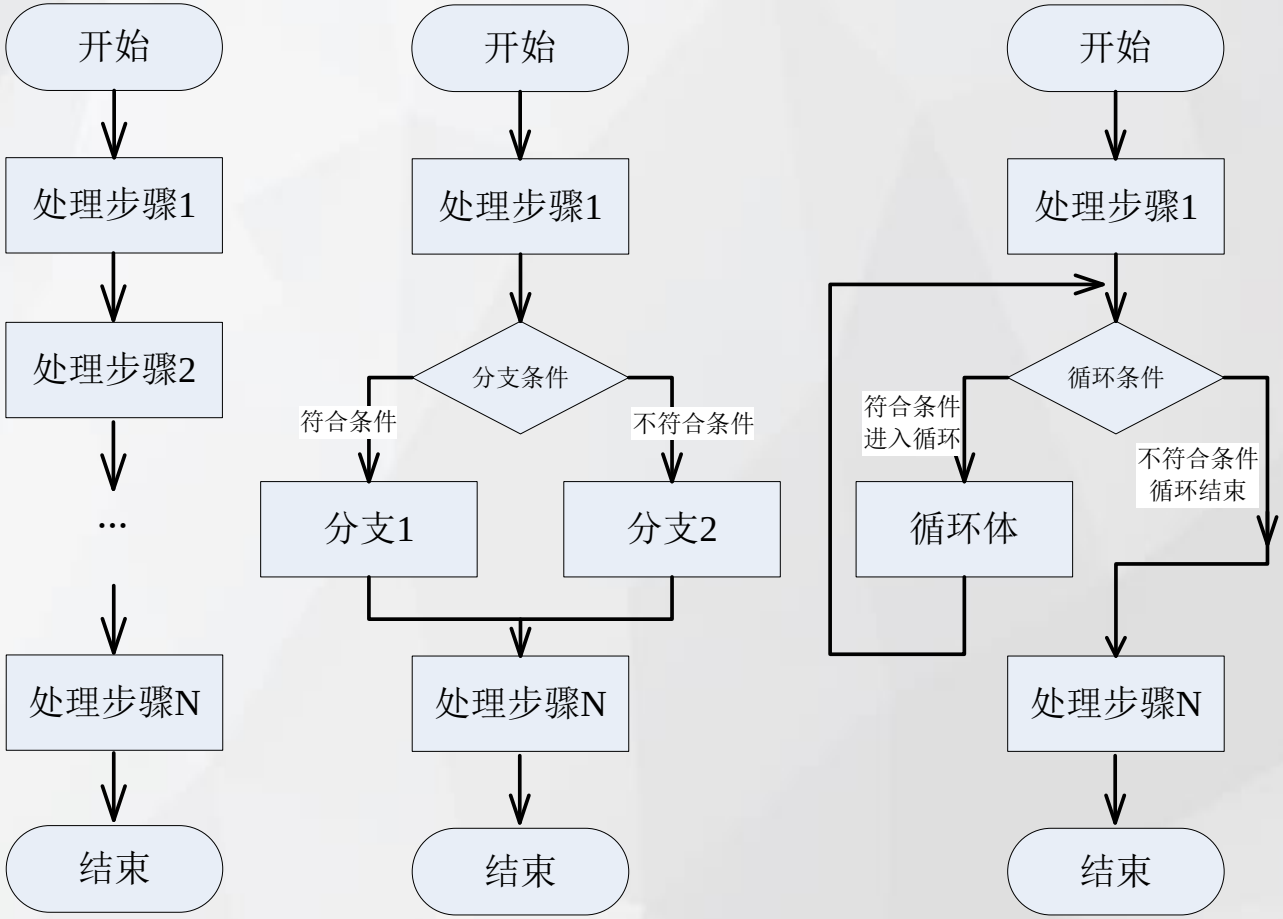
3、算法的基本结构：

1）顺序结构

2）分支结构

3）循环结构

算法基本结构示意图



算法基本结构示意图

4. 算法效率衡量方法与准则

时间复杂度： 指算法从开始执行到处理结束所需要的总时间。

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

空间复杂度： 指算法从开始执行到处理结束所需的存储量空间的总和。

$$S(n) = O(g(n))$$

5. 算法与数据结构的关系:

- 计算机科学家沃斯（N.Wirth）提出的:

  “算法+数据结构=程序”

  揭示了程序设计的本质：对实际问题选择一种好的数据结构，加上设计一个好的算法，而好的算法很大程度上取决于描述实际问题的数据结构。算法与数据结构是互相依赖、互相联系的。

- 一个算法总是建立在一定数据结构上的；反之，算法不确定，就无法决定如何构造数据。

第三节 线性结构

1.线性表

2.栈和队列

3.串和数组

1. 线性表

(1) 线性表的定义

线性表是n(n>=0)个数据元素的**有限序列**，表中各个元素具有相同的属性，表中相邻元素间存在“**序偶**”关系。

记做：$(a_1,a_2,.....a_{i-1},a_i,a_{i+1},...,a_{n-1},a_n)$

其中，a_{i-1}称为a_i 的**直接前驱元素**，a_{i+1}是a_i的**直接后继元素**

线性表的长度：表中的元素个数 n

位序：i称元素a_i在线性表中的位序

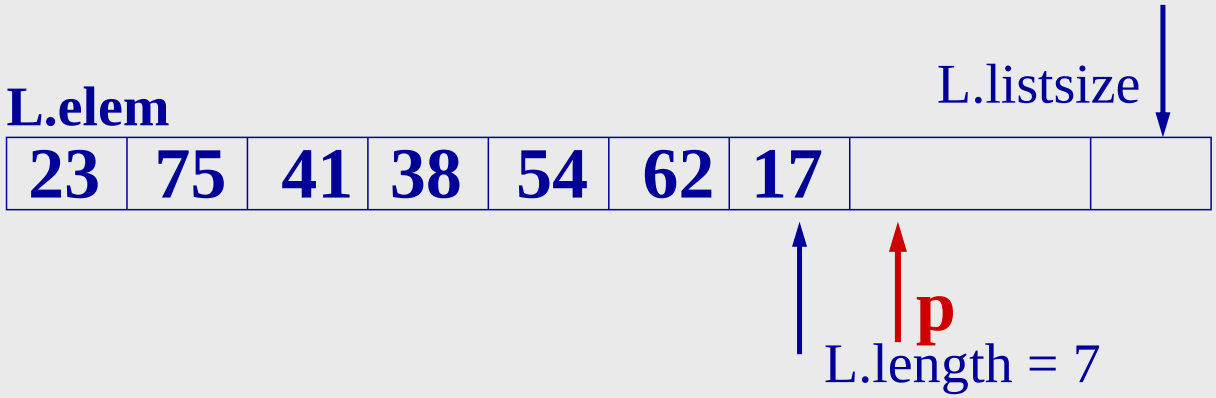
(2) 线性表的顺序表示和实现

线性表的顺序存储是指在内存中用地址连续的一块存储空间顺序存放线性表的各元素，用这种存储形式存储的线性表称其为**顺序表**。

顺序表——线性表的顺序存储表示

```
Const LIST_INIT_SIZE=100;          (C++规范)
Const LISTINCREMENT=10;
#define LIST_INIT_SIZE 100          (C规范)
Typedef  Struct {
  Elemtype        * elem;
  int             length;
  int             listsize;
  int             incrementsize;
}SqList;
```

[例5]：顺序表



e = 38　　返回值 = 4

e = 23　　返回值 = 1

(3) 线性表的链式表示和实现

链表是通过一组任意的存储单元来存储线性表中的数据元素的，为建立起数据元素之间的关系，对每个数据元素a_i，除了存放数据元素的自身的信息a_i之外，还需要和a_i一起存放其后继a_{i+1}所在的存贮单元的地址，这两部分信息组成一个**“节点”**。

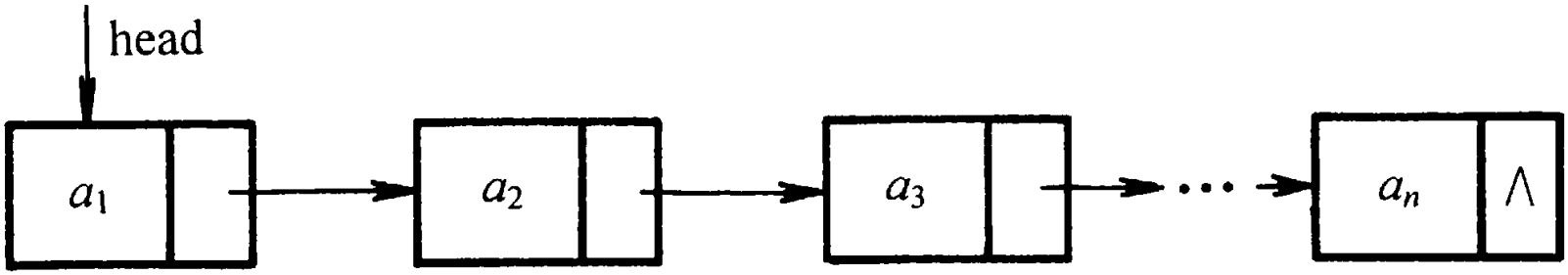
单链表——线性表的链式存储表示

数据域（data）和指针域（next）

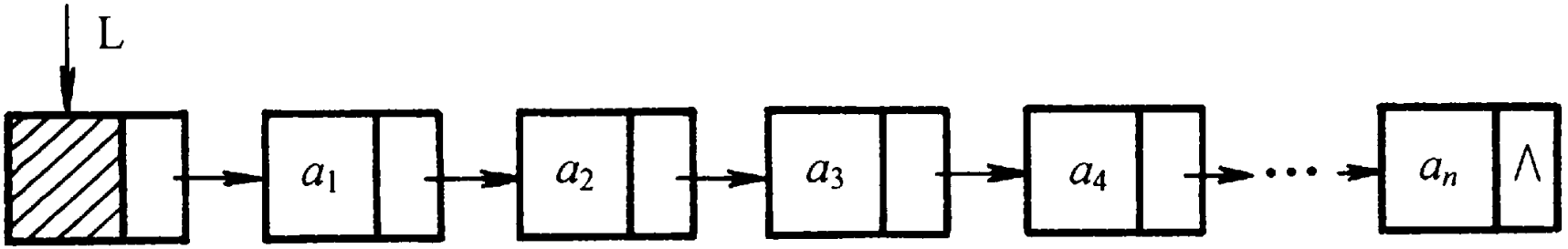
- **存储表示**

```
typedef struct Lnode{
    ElemType        data;
    struct Lnode          *next;
}Lnode, *LinkList;
```

单链表表示意图



(a) 不带头结点的单链表



(b) 带头结点的单链表

双向链表（循环链表）—线性表的链式存储表示

概念：两个指针，分别指向前驱元素和后继元素

```
typedef struct DuLnode{
    ElemType data;
    struct DuLnode   *prior;
    struct DuLnode   *next;
}DuLnode, *DuLinkList;
```

2. 栈和队列

（1）栈的定义

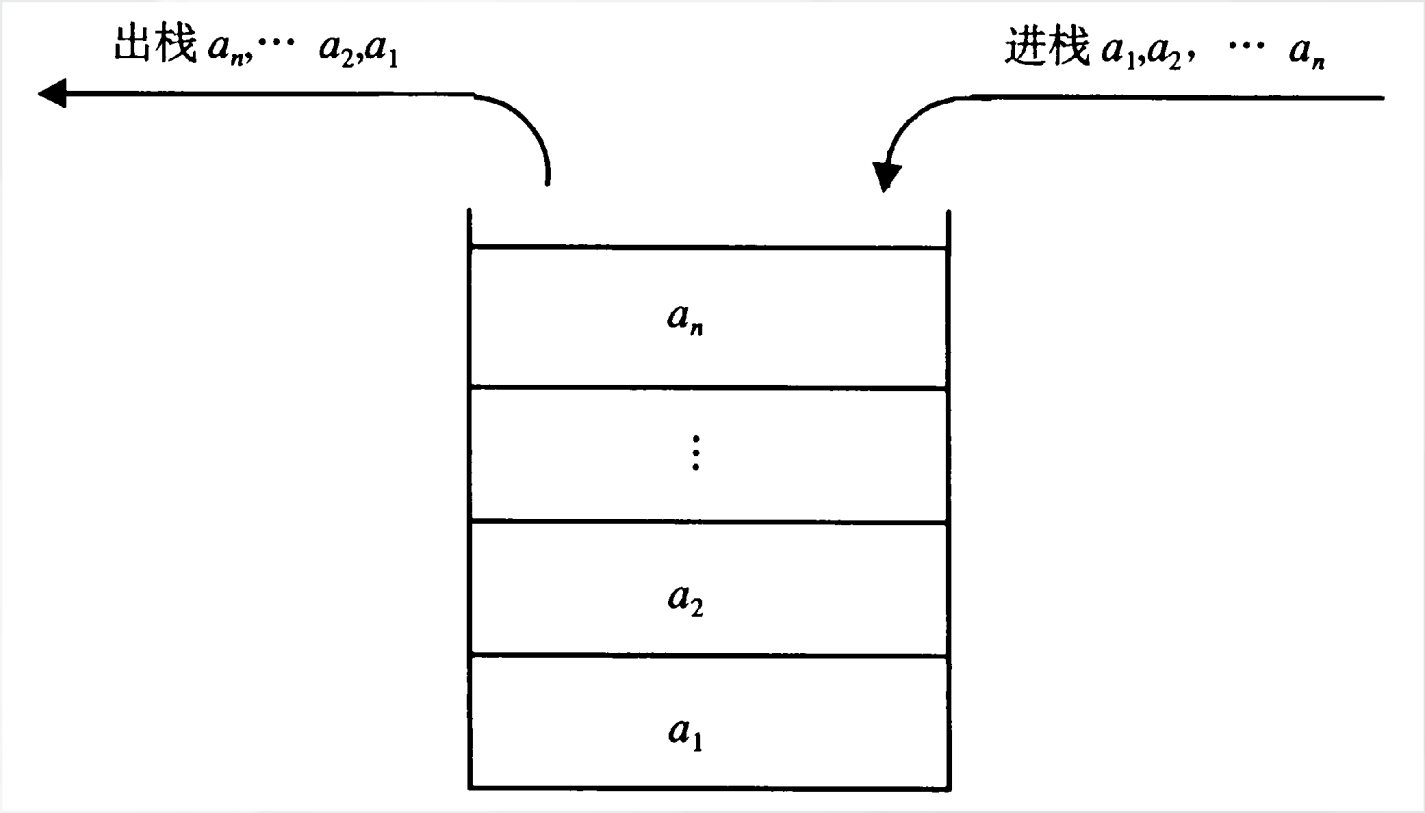
栈（Stack）是限定只能在表得一端进行插入和删除操作得线性表，又称限定性线性表结构。

（2）栈的结构特点和操作

- 栈顶(Top)、栈底(Bottom),先入后出(LIFO)
- 栈的基本操作

InitStack（&S）
DestroyStack(&S)
ClearStack(&S)
StackEmpty（S）
StackLength（S）
GetTop(S,&e)
Push(&S, e)
Pop(&S,&e)
StackTraverse(S)

堆栈结构示意图



堆栈结构示意图

2. 栈和队列

(3)队列的定义

队列（Queue）是限定只能在表得一端进行插入在表的另一端进行删除操作的线性表。

(4)队列的结构特点和操作

- 队列头(front)、队列尾(rear)，先入先出(FIFO)
- 队列的基本操作

InitQueue(&Q)
DestroyStack(&S)
ClearQueue(&Q)
QueueEmpty(Q)
QueueTraverse(Q)
QueueLength(Q)
GetHead(Q,&e)
EnQueue(&Q,e)
Dequeue(&Q,&e)

3 串和数组

（1）串的定义和表示方法

- 串定义

串（即字符串）是一种特殊的线性表，它的数据元素仅由一个字符组成

字符串，由零个或多个字符组成的有限序列。

$$S = “a_0a_1.....a_{n-1}”$$

串的长度：n

空串：n=0，Null String

子串与主串，子串的位置（从0开始）

串的比较：最大相等前缀子序列

串的表示方法

- **定长顺序存储表示**

  两种表示方法：

  1)下标为0的数组存放长度 (pascal)

  ```
  typedef  unsigned char SString[MAXSTLEN+1] ;
  ```

  2)在串值后面加 ‘\0’ 结束 (C语言)

- **堆分配存储表示**

  串变量的存储空间是在程序执行过程中动态分配的，程序中出现的所有串变量可用的存储空间是一个共享空间，称为“堆”。

3. 串和数组（续）

（2）数组的定义和操作

- **数组定义**

  数组是一个具有固定格式和数量的数据有序集，每一个数据元素有唯一的一组下标来标识。数组可以看作线性表的推广。数组作为一种数据结构其特点是结构中的元素本身可以是具有某种结构的数据，但属于同一数据类型。

- **二维数组定义**

  其数据元素是一维数组的线形表。

- **N维数组定义**

  其数据元素是N－1维数组的线形表。

数组的操作

initarray(&A,n,bound1,bound2...boundn) ——初始化

Destroyarray(&A) —— 删除数组

value(A,&e,index1,index2.....indexn) —— 赋值

assign(&A,e,index1,index2.....indexn) —— 分配数组

数组的存储方式和表示

- 数组元素的两种存储方式

  行主序存储

  列主序存储

- 数组中元素在内存映象中的关系:

  二维数组A[m][n]

  $LOC[i,j]=LOC[0,0]+(i*n+j)*L$

  三维数组B[p][m][n]

  $LOC[i,j,k]=LOC[0,0,0]+(i*m*n+j*n+k)*L$

第三节 非线性结构

1. 树

(1) 树的定义与结构特点

- 树的定义

n(n>=0)个数据元素（结点）的有限集D，若D为空集，则为**空树**。否则：

在D中存在唯一的称为**根**的数据元素；

当n>1时，其余结点可分为m（m>0）个互不相交的有限子集T1，T2，……，Tm，其中每个子集本身又是一颗树，并成为根的**子树**。

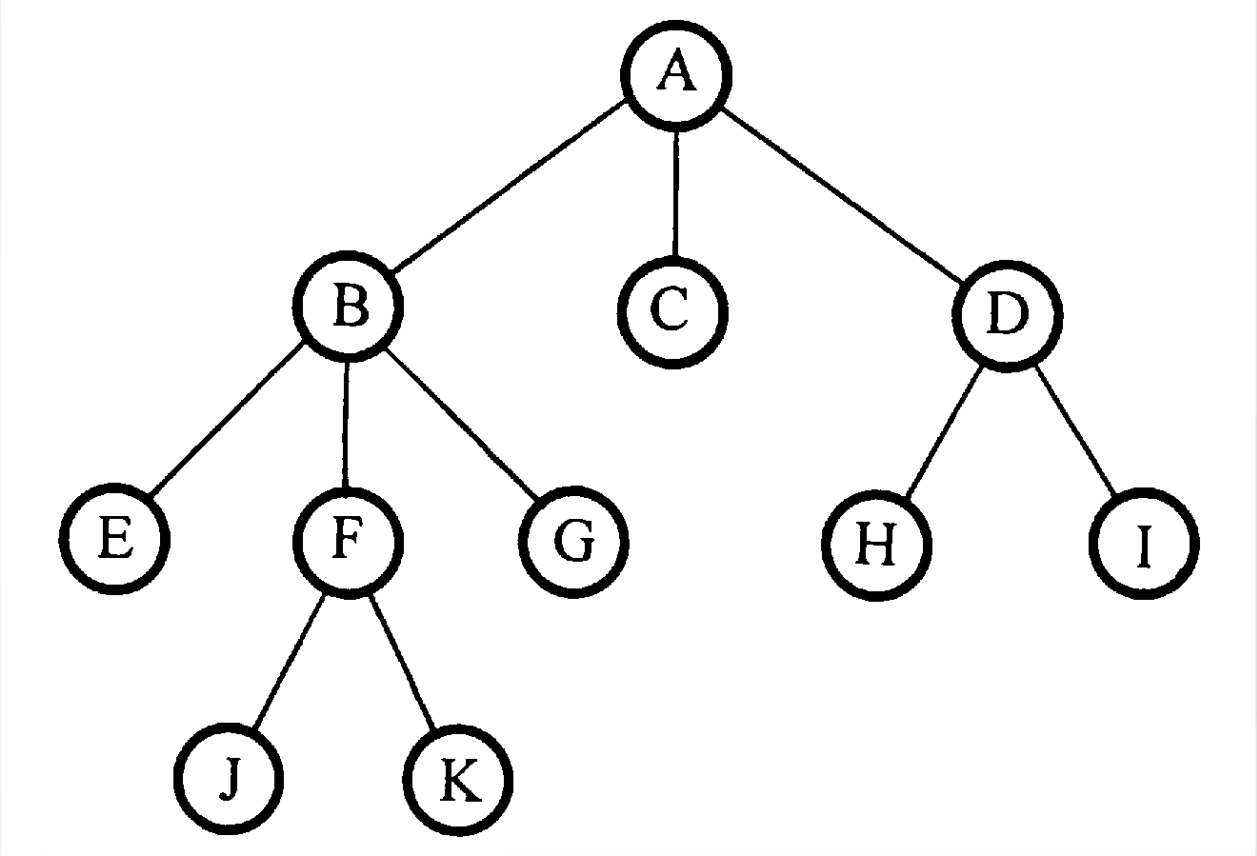
树的结构特点

- **树具有下面两个特点：**

  (1) 树的根节点没有前驱节点，除根节点之外的所有节点有且只有一个前驱节点。

  (2) 树中所有节点可以有零个或多个后继节点。

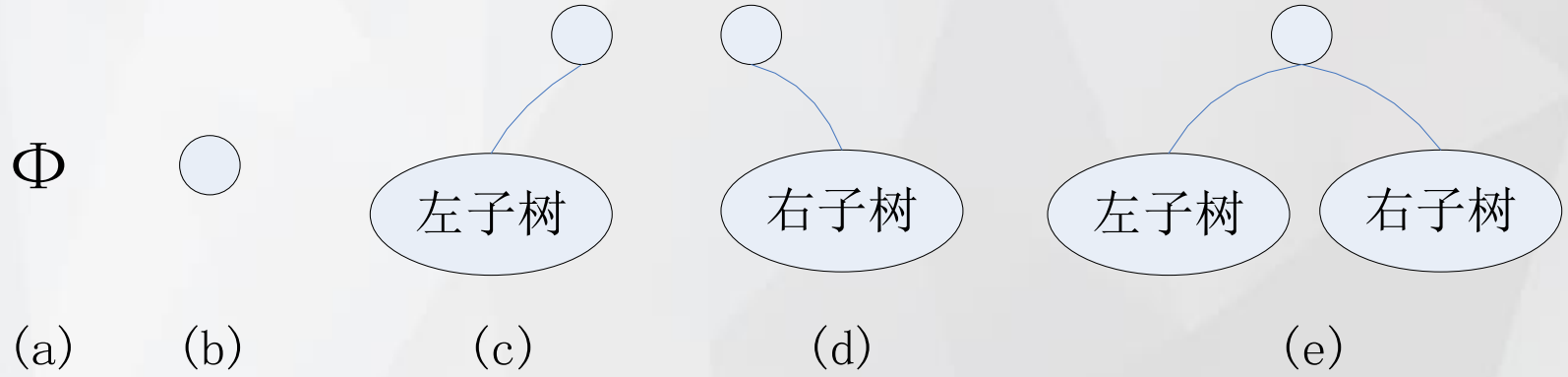
树结构示意图



典型的树结构

二叉树的定义

- 二叉树（Binary Tree）是个有限元素的集合，该集合或者为空、或者由一个称为根(root)的元素及两个不相交的、被分别称为左子树和右子树的二叉树组成。

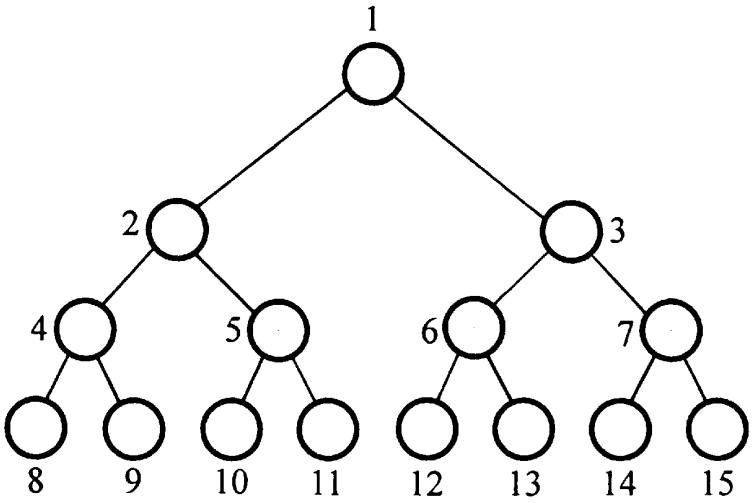


二叉树的五种基本形态

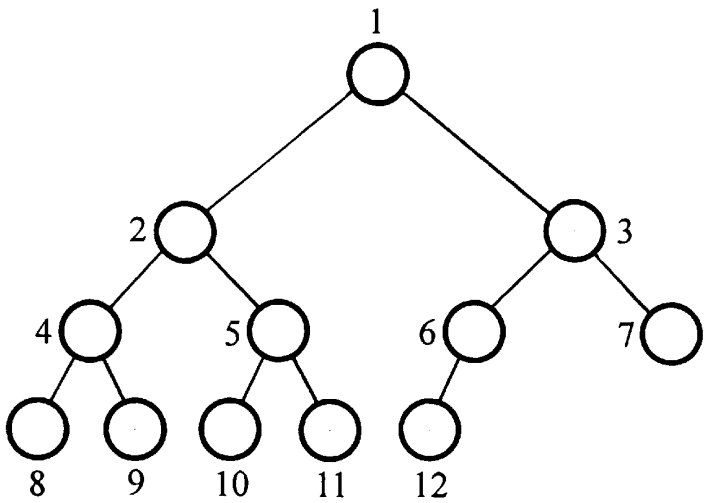
满二叉树和完全二叉树

满二叉树（full binary tree）：所有结点度为2，叶子结点在同一层次。

完全二叉树（complete binary tree）：一棵深度为k的有n个节点的二叉树，对树中的节点按从上至下、从左到右的顺序进行编号，如果编号为i（1≤i≤n）的节点与满二叉树中编号为i的节点在二叉树中的位置相同，则这棵二叉树称为完全二叉树。



(a) 满二叉树



(b) 完全二叉树

树的运算

树的运算主要是**插入节点、删除节点和遍历**等几种。插入节点、删除节点运算改变树的结构，但要求在改变结构的同时，保持树的特性不变，对于二叉树，插入和删除操作后的树仍然是一棵二叉树。这两个操作过于复杂，在专业书籍中介绍，在此不做详细讨论。

树的基本运算操作

- 树的基本运算操作：

  InitTree（&T）

  DestroyTree（&T）

  CreateTree(&T,definition)

  TreeEmpty(T)

  TreeDepth(T)

  Parent(T, e)

  LeftChild（T，e）

  Rightsibling(T,e)

  InsertChild（&T,&p,i,C)

  DeleteChild（&T,&p,i)

  Traverse（T）

2 图

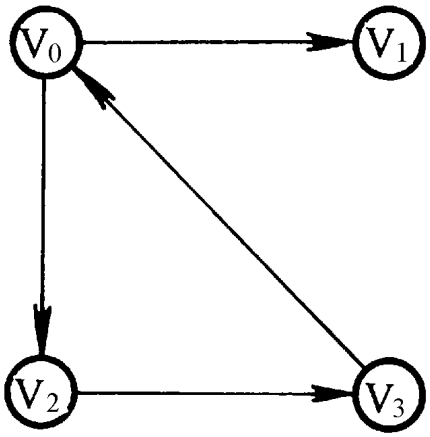
- 图的定义

  图是由一组节点（vertex）的有穷集V(G)和和一组顶点间的连线(arc)的集合E(G)组成的一种抽象数据结构。记做：G = (V，E)。V是数据结构中的数据元素，E是集合上的关系

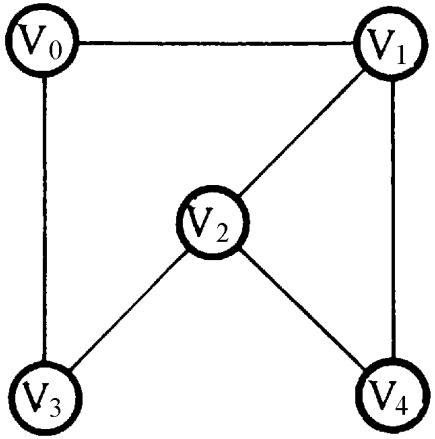
图的相关概念

- 弧(arc)、弧头（终点）、弧尾（起点）：
  <v,w>表示从v到w的弧
- 有向图(digraph)、无向图(undigraph)、边:
  (v,w)代表<v,w>和<w,v>
- 有向网、无向网:
  带权的有向图和无向图
- 完全图（complete graph）：边e为n(n-1)/2
- 有向完全图：弧e为n(n-1)

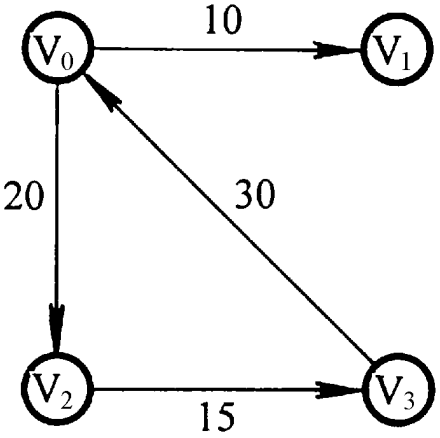
图的示意图



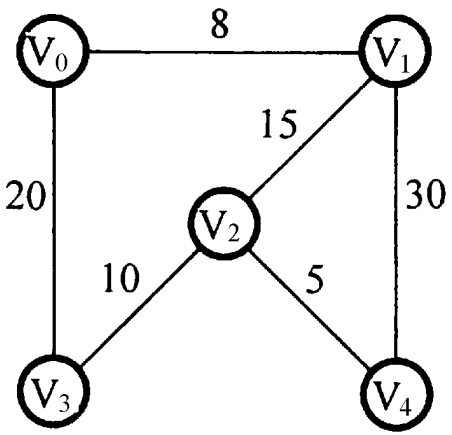
(a) 有向图 G_1



(b) 无向图 G_2



(a) 有向网 G_3



(b) 无向网 G_4

图的基本运算

添加顶点——将一个新顶点插入到图中

添加边——连接一个顶点和一个目标顶点

删除顶点——从一个图里移除一个顶点，同时删除连接顶点的边。

查找顶点——通过遍历图来查找特定的顶点。

图的遍历——指从图中的任一顶点出发，对图中的所有顶点访问一次且只访问一次

说明：图的遍历是图的一种基本操作，图的许多其他操作都是建立在遍历操作的基础之上。

第四节 基本算法

1. 排序

（1）排序的定义与相关概念

- **排序：** 将一组杂乱无章的数据排列成一个按关键字有序的序列。
- **数据表(datalist):** 它是待排序数据对象的有限集合。
- **关键字(key):** 通常数据对象有多个属性域，即多个数据成员组成，其中有一个属性域可用来区分对象，作为排序依据。该域即为关键字。每个数据表用哪个属性域作为关键字，要视具体的应用需要而定。即使是同一个表，在解决不同问题的场合也可能取不同的域做关键字。

(1) 排序的定义与相关概念

- **主关键字:** 如果在数据表中各个对象的关键字互不相同，这种关键字即主关键字。按照主关键字进行排序，排序的结果是唯一的。
- **次关键字:** 数据表中有些对象的关键字可能相同，这种关键字称为次关键字。按照次关键字进行排序，排序的结果可能不唯一。
- **排序算法的稳定性:** 如果在对象序列中有两个对象r[i]和r[j]，它们的关键字 k[i] == k[j]，且在排序之前，对象r[i]排在r[j]前面。如果在排序之后，对象r[i]仍在对象r[j]的前面，则称这个排序方法是稳定的，否则称这个排序方法是不稳定的。

(2）衡量排序方法的标准

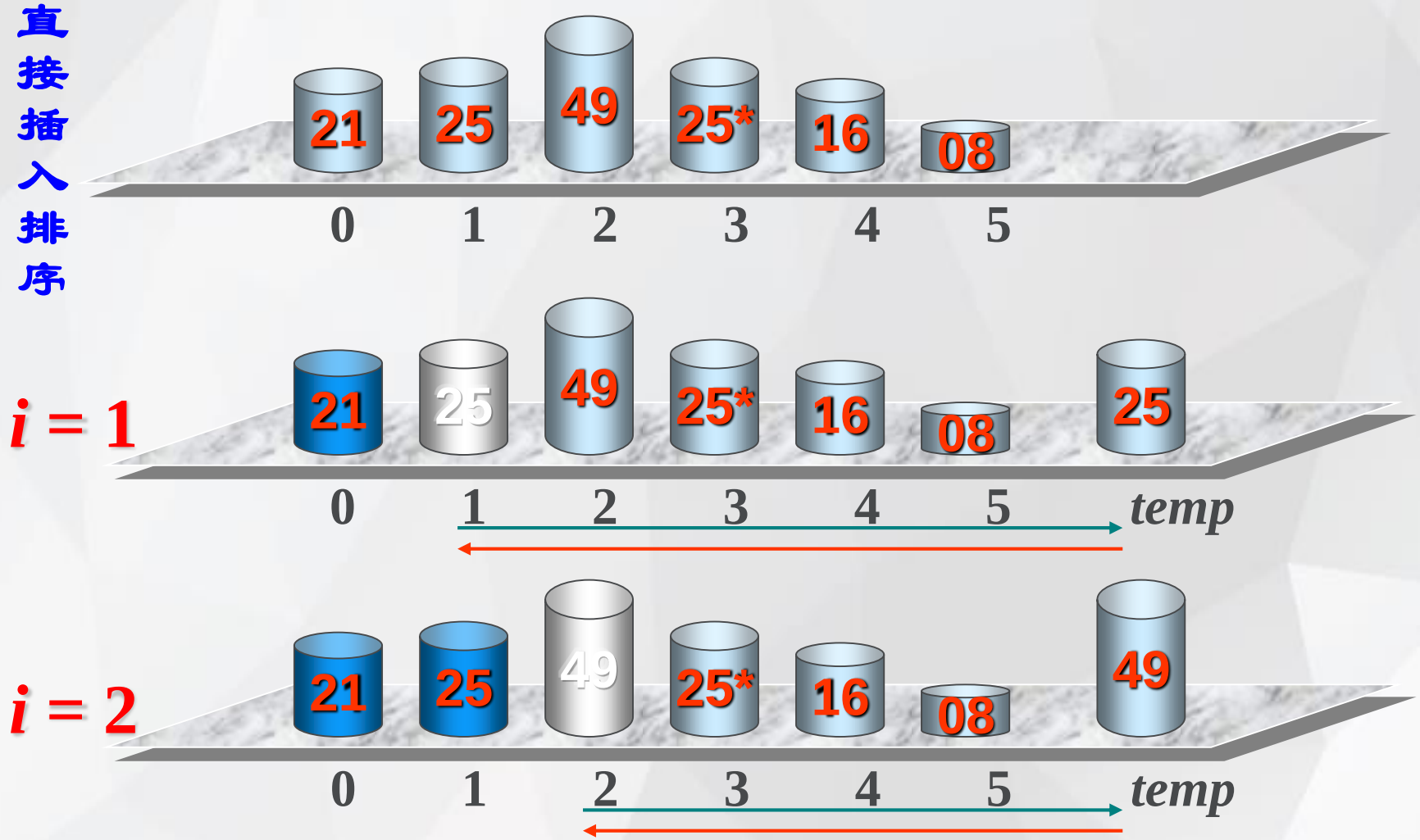
- 排序时所需要的平均比较次数
- 排序时所需要的平均移动
- 排序时所需要的平均辅助存储空间

(3) 插入排序

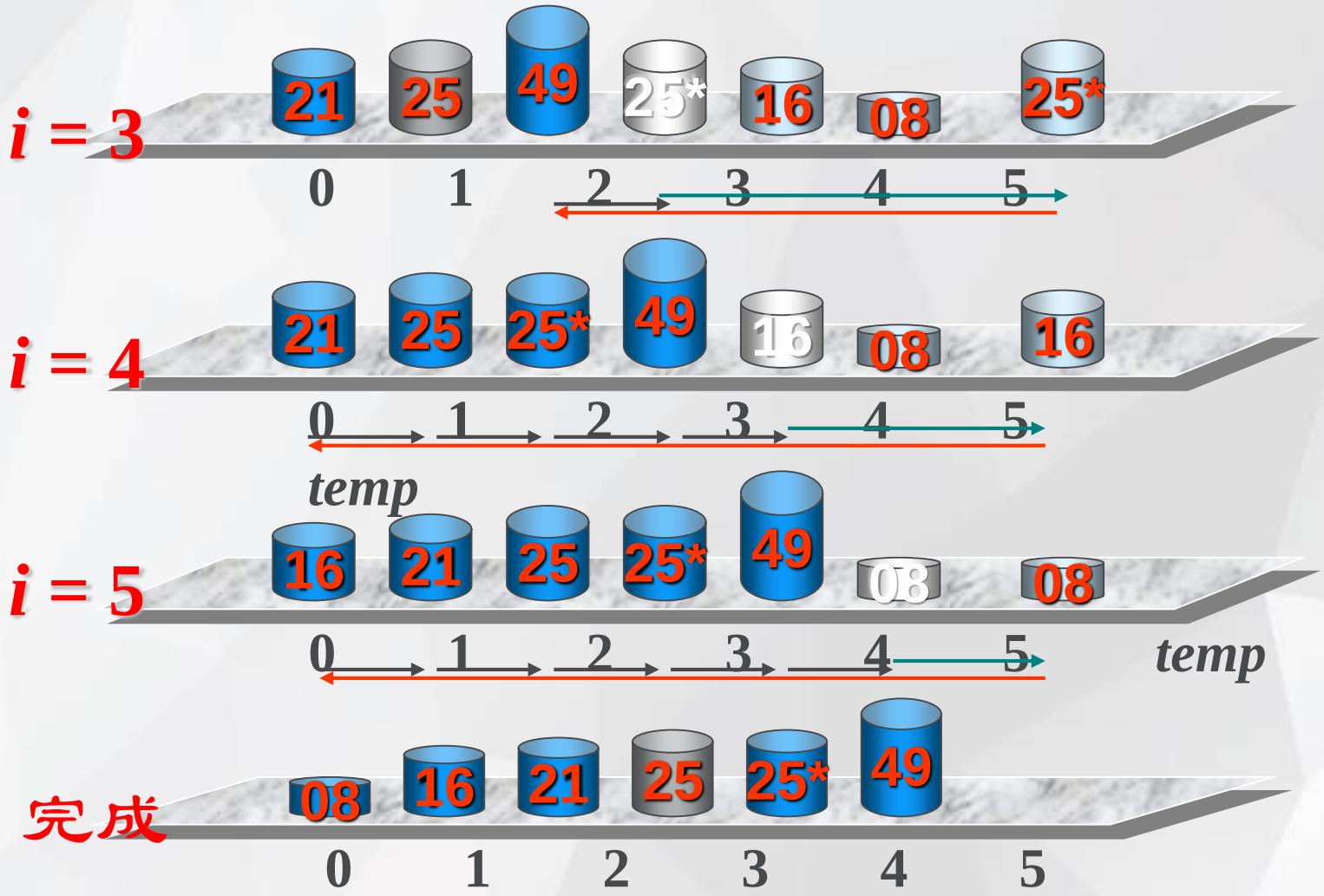
插入排序 (Insert Sorting)的基本方法是：每步将一个待排序的对象，按其关键字大小，插入到前面已经排好序的一组对象的适当位置上，直到对象全部插入为止。

- **直接插入排序：**当插入第i ($i \geq 1$) 个对象时，前面的V[0], V[1], ..., V[i-1]已经排好序。这时，用V[i]的关键字与V[i-1], V[i-2], ...的关键字顺序进行比较，找到插入位置即将V[i]插入，原来位置上之后的所有对象依次向后顺移。
- **折半插入排序：**设在顺序表中有一 个对象序列 V[0], V[1], ..., V[n-1]。其中，V[0], V[1], ..., V[i-1] 是已经排好序的对象。在插入 V[i] 时，利用折半搜索法寻找 V[i] 的插入位置。

直接插入排序算法


直接插入排序
21
25
49
25*
16
08
0
1
2
3
4
5
i = 1
21
25
49
25*
16
08
25
0
1
2
3
4
5
temp
i = 2
21
25
49
25*
16
08
49
0
1
2
3
4
5
temp

直接插入排序


i = 3
21
25
49
25*
16
08
25*
0
1
2
3
4
5
i = 4
21
25
25*
49
16
08
16
0
1
2
3
4
5
temp
i = 5
16
21
25
25*
49
08
08
0
1
2
3
4
5
temp
完成
08
16
21
25
25*
49
0
1
2
3
4
5

(4) 选择排序

选择排序(Selection Sort)的基本方法是：每一趟（例如第 *i* 趟，*i* = 1, …, *n*-1）在后面的 *n-i+1* 个待排序对象中选出关键字最小的对象，作为有序对象序列的第 *i* 个对象。待到第 *n*-1 趟作完，待排序对象只剩下1个，就不用再选了。

- **简单选择排序 (Simple Selection Sort)**：i从1开始，直到n-1，进行n-1趟排序，第i 趟的排序过程为： 在一组对象r[i] ~ r[n] (i=1,2,…,n-1)中选择具有最小关键字的对象；并和第 i 个对象进行交换
- 与直接插入排序相比，选择排序由于是有选择地选取记录，在插入有序序列时，需要进行的记录移动操作较少，但其总的时间复杂度也是O(n2)。

(5) 冒泡排序

冒泡排序 (Bubble Sort)的基本方法是：设待排序对象序列中的对象个数为 *n*。最多作 *n*-1 趟排序。在第 *i* 趟中顺次两两比较*r*[*j*-1].*Key*和*r*[*j*].*Key*，*j* = *i*, *i*+1, …, *n*-*i*-1。如果发生逆序，则交换*r*[*j*-1]和*r*[*j*]。

冒泡排序典型算法：

```
void BubbleSort(SqList &L)
        { int i, j, tag;
          j = L.length-1;
          do{  tag=1;
               for(i=1; i<=j; i++)
               if( L.r[i+1].key< L.r[i].key )
               { L.r[0]= L.r[i+1];  L.r[i+1]= L.r[i];
                 L.r[i]= L.r[0]; tag=0;
               }
          if( !tag ) j--;
          }  while( !tag &&j );
        return;}
```

(6) 快速排序

快速排序 (Quick Sort)的基本方法是：任取待排序对象序列中的某个对象（例如取第一个对象）作为**枢轴**(pivot)，按照该对象的关键字大小，将整个对象序列划分为**左右两个子序列**：

左侧子序列中所有对象的关键字都小于或等于枢轴对象的关键字

右侧子序列中所有对象的关键字都大于枢轴对象的关键字

- **枢轴对象**则排在这两个子序列中间(这也是该对象最终应安放的位置)。
- 然后分别对这两个子序列重复施行上述方法，直到所有的对象都排在相应位置上为止。

快速排序算法描述

```
QuickSort ( List ) {
    if ( List的长度大于1) {
     将序列List划分为两个子序列
              LeftList 和 Right List;
       QuickSort ( LeftList );
     QuickSort ( RightList );
       将两个子序列 LeftList 和 RightList
          合并为一个序列List;
    }
}
```

注意： 快速排序是一种不稳定的排序方法。对于 n 较大的平均情况而言，快速排序是“快速”的，但是当 n 很小时，这种排序方法往往比其它简单排序方法还要慢。

2. 查找

（1）查找的定义与相关概念

- 查找的定义

查找：

根据给定的某个值，在查找表中确定一个关键字等于给定值的数据元素。若找到表示查找成功，返回该元素详细信息或在查找表中的位置；负责返回NULL。

查找的相关概念

- 查找表：
  由同一类元素或记录构成的集合。对数据元素间的关系未作限定。
- 对查找表的操作有：
  查找某个“特定”的元素是否在表中。
  查找某个“特点”的元素的各种属性。
  在查找表中插入一个元素。
  在查找表中删除一个元素
- 静态查找表、动态查找表
- 关键字

  数据元素中的某个数据项值。可以表示一个数据元素，如可以唯一表示，则为主关键字（primary key）。

查找的结果通常有两种可能

查找成功，即找到满足条件的数据对象。

查找不成功，或查找失败。作为结果，报告一些信息，如失败标志、失败位置等。

查找算法的效率衡量

衡量一个查找算法的时间效率的标准是：**在查找过程中关键字的平均比较次数或平均读写磁盘次数(只适合于外部查找)**，这个标准也称为**平均查找长度*ASL***(*Average Search Length*)，通常它是**查找结构中对象总数 *n* 或文件结构中物理块总数 *n* 的函数**。

- 另外衡量一个查找算法还要考虑算法所需要的存储量和算法的复杂性等问题。
- 在静态查找表中，数据对象存放于数组中，利用数组元素的下标作为数据对象的存放地址。查找算法**根据给定值*x*，在数组中进行查找**。直到找到*x*在数组中的存放位置或可确定在数组中找不到*x*为止。

顺序查找

- 所谓**顺序查找**，又称**线性查找**，主要用于在**线性结构**中进行查找。
- 存储结构：

```
typedef struct{
    ElemType  *elem; // 元素数据结构
    int length; // 查找表长度
    }  SSTable;
```

- 查找过程：从表中最后一个元素开始，顺序用各元素的关键字与给定值*x*进行比较，若找到与其值相等的元素，则查找成功，给出该元素在表中的位置；否则，若直到第一个记录仍未找到关键字与*x*相等的对象，则查找失败。

顺序查找算法描述：

```
Search_Seq(SSTable ST, KeyType key){
        //顺序查找的算法，0号元素为监视哨
      int i;
      ST.elem[0].key=key;
      for (i=ST.length; !EQ(ST.elem[i].key,key);--i);
      return i;
       }
```

顺序查找的平均查找长度基本算法

设查找第 i 个元素的概率为 pi，查找到第 i 个元素所需比较次数为 ci，则查找成功的平均查找长度:

$$ASL_{succ} = \sum_{i=1}^{n} p_i \cdot c_i . \qquad (\sum_{i=1}^{n} p_i = 1)$$

在顺序查找情形，$c_i = n-i+1$，$i = 1, \ldots, n$，因此

$$ASL_{succ} = \sum_{i=1}^{n} p_i \cdot (n - i + 1)$$

折半查找

折半查找：先求位于查找区间正中的对象的下标*mid*，用其关键字与给定值*x*比较:

Element[*mid*].*getKey*( ) = *x*，查找成功；

Element[*mid*].*getKey*( ) > *x*，把查找区间缩小到表的**前半部分**，再继续进行对分查找；

Element[*mid*].*getKey*( ) < *x*，把查找区间缩小到表的**后半部分**，再继续进行对分查找。

- 每比较一次，查找区间缩小一半。如果查找区间已缩小到一个对象，仍未找到想要查找的对象，则查找失败。

折半查找算法过程：

(1) mid= (low+high)/2」

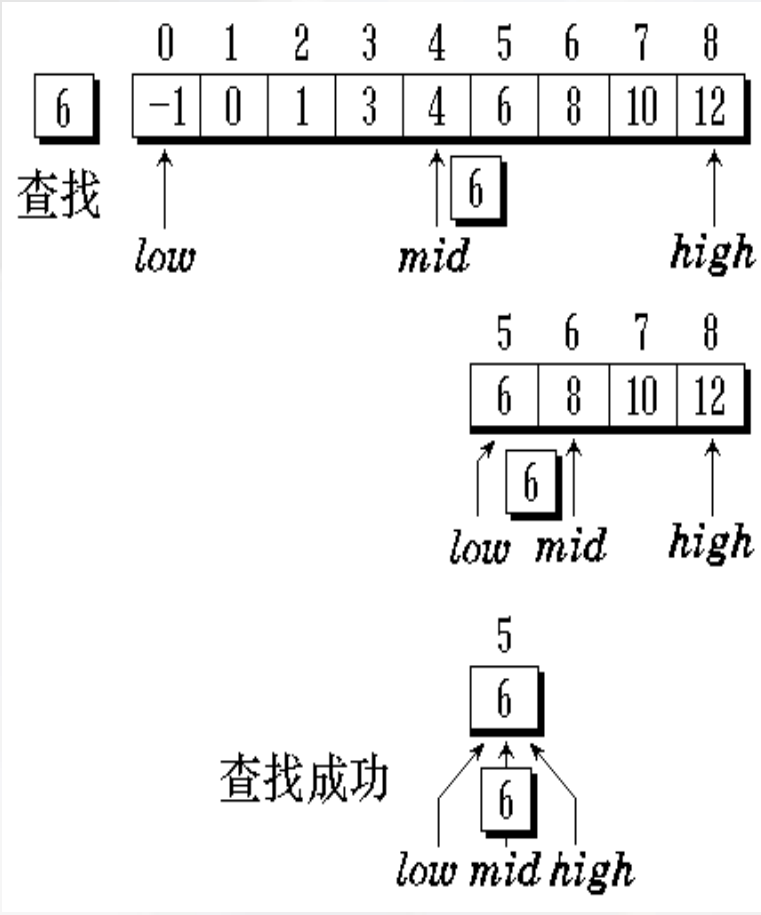
(2) 比较 ST.elem[mid].key = = key?

如果 ST.elem[mid].key = = key，则查找成功，返回mid值

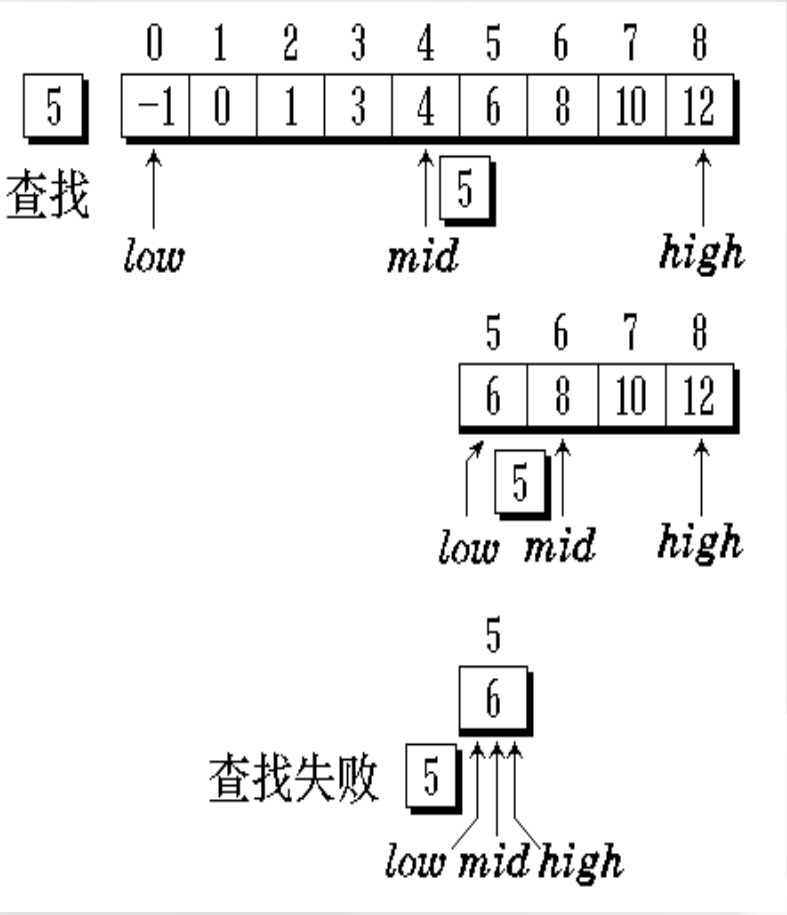
如果 ST.elem[mid].key > key，则置high=mid-1

如果 ST.elem[mid].key < key，则置low=mid+1

(3) 重复计算mid 以及比较ST.elem[mid].key与 key，当low>high 时，表明查找不成功，查找结束。



查找成功的例子



查找失败的例子

第五节 递归

1.递归的定义

- **递归：**递归是指算法在过程中调用自身作为子算法的一种设计方法。
- 递归是设计和描述算法的一种有力的工具，它在复杂算法的描述中被经常采用。能采用递归描述的算法通常具有如下特征：

为求解规模为N的问题，设法将它分解成规模较小的问题，然后从这些小问题的解方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模较大问题的解。

2.递归解决的问题

- 递归算法一般用于解决三类问题：

  (1) 数据的定义是按递归定义的。(例如Fibonacci函数)

  (2) 问题解法按递归算法实现。(例如回溯)

  (3) 数据的结构形式是按递归定义的。(例如树的遍历，图的搜索等)

3.递归算法的优点和缺陷

- 递归算法的优势：

  (1) 对于按递归定义的数据集合处理效率很高。

  (2) 易于将复杂的问题简单化，便于理解。

  (3) 对于特殊的数据结构，例如二叉树、图等具有良好的处理能力。

- 递归算法的缺陷：

  递归引起一系列的函数调用，并且可能会有一系列的重复计算，递归算法的执行效率相对较低。在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。

本章小结

- 数据是计算机化的信息，是计算机可以直接处理的最基本的对象。计算机中的各种应用，都是对数据进行加工处理的过程。要使程序高效率地运行，必须根据数据的特性及数据间的相互关系设计出高质量的数据结构；
- 数据结构通常有：集合结构、线性结构和非线性结构（树和图）。
- 结构中定义的“关系”描述的是数据元素之间的逻辑关系，因此称为数据的逻辑结构。数据结构在计算机中的表示（又称映像）称为数据的物理结构，或称存储结构。
- 数据结构主要研究数据的各种逻辑结构和存储结构，以及对数据的各种操作。因此，主要有三个方面的内容：数据的逻辑结构；数据的物理存储结构；对数据的操作（或算法）。通常，算法的设计取决于数据的逻辑结构，算法的实现取决于数据的物理存储结构。
- 对特定问题求解步骤的一组规格化描述就是算法，也是计算机解题的过程。算法与数据结构是相辅相成的。算法必须具备：正确性、可读性、稳健性和高效性。算法的描述通常使用流程图或伪代码。
- 线性结构用于描述一对一的相互关系，即结构中元素之间只有最基本的联系，现实中的许多事物的关系都是非线性的。

- 线性表是最简单、最基本、也是最常用的一种线性结构，有顺序存储和链式存储两种存储方法。
- 栈和队列是在软件设计中常用的两种数据结构，它们的逻辑结构和线性表相同。其特点在于运算受到了限制：栈按“后进先出”的规则进行操作，队按“先进先出”的规则进行操作。
- 树、二叉树和图是非线性结构的基本数据表示方法；
- 排序运算的概念和基本实现算法。排序是常用运用之一，其实现算法有多种。比较简单的有直接插入排序、简单选择排序、冒泡排序等几种，其中快速排序是性能较好的排序算法；
- 查找运算根据查找列表的结构，有顺序查找和折半查找等多种算法。顺序查找效率较低，但对查找列表没有要求，适用于列表较小的场合；折半查找效率较高，但需要列表是有序表；
- 递归是算法中调用自身的一种算法设计技术。递归的运行效率低，但运用递归方式编写的算法结构简单，便于理解。在实际程序实现时，应将递归算法用非递归形式实现以提高程序的运行效率。