数据结构：概论

Data Structure

主讲教师：屈卫兰

Office number：基地203

Email：5604293@qq.com

第1章　数据结构

- 1.1 数据结构的原则
- 1.2 抽象数据类型和数据结构
- 1.3 问题、算法和程序

数据结构

- 大多数计算机程序的主要目标是存储信息和尽快地检索信息。因此，研究数据结构和算法就成了计算机科学的核心问题。课程目的就是怎样组织信息，以便支持高效的数据处理。
  - 介绍常用的数据结构
  - 引入并加强“权衡”(tradeoff)的概念，每一个数据结构都有其相关的代价和效益的权衡
  - 评估一个数据结构或算法的有效性。

1.1 数据结构的原则

- 数据结构的地位：
  - 计算机专业本科生必修的学位课程
  - 计算机专业研究生入学考试必考科目
  - 计算机软件技术资格和水平考试内容
  - 全国计算机等级考试（三级、四级）内容
  - 为操作系统、数据库系统、编译原理、计算机网络等后续课程提供了必要的知识基础
  - 为提高程序设计能力、逻辑思维能力和分析问题、解决问题的能力提供了必要的技能训练
- 数据结构：按照逻辑关系组织起来的一批数据，按一定的存储方法把它存储在计算机中，并在这些数据上定义了一个运算的集合。

程序设计的目标

1．设计一种容易理解、编码和调试的算法

2．设计一种能有效利用计算机资源的算法

目标1主要涉及到的是软件工程原理；

本课程主要讲的是与目标2有关的问题

效率度量

- 算法分析：
  - 估算一种算法或者一个计算机程序的效率的方法
  - 算法分析可以度量一个问题的内在复杂程度

学习数据结构的必要性

- 一个数据结构就是一类数据的表示及其相关操作。
- 一个数据结构被认为是一组数据项的组织或者结构。
- 存储一组数据项，数据结构执行所有必需的运算：查找、打印或排序，或者更改数据项的值。选择不同的数据结构可能会产生很大的差异。

资源限制

- 资源限制包括空间和时间。
- 算法的代价(cost)是指这种算法消耗的资源量。
- 选择数据结构的步骤：
  - 分析问题，以确定任何算法均会遇到的资源限制。
  - 确定必须支持的基本操作，并度量每种操作所受的资源限制。
  - 选择最接近这些开销的数据结构。

〖Example〗 Given (possibly negative) integers $A_1, A_2, \ldots, A_N$, find the maximum value of $\sum_{k=i}^{j} A_k$.

Max sum is 0 if all the integers are negative.

Algorithm 1

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
          int  ThisSum,  MaxSum,  i,  j,  k;
/* 1*/    MaxSum = 0;   /* initialize the maximum sum */
/* 2*/    for( i = 0; i < N; i++ )  /* start from A[ i ] */
/* 3*/        for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 4*/            ThisSum = 0;
/* 5*/            for( k = i; k <= j; k++ )
/* 6*/                ThisSum += A[ k ];  /* sum from A[ i ] to A[ j ] */
/* 7*/            if ( ThisSum > MaxSum )
/* 8*/                MaxSum = ThisSum;  /* update max sum */
              }  /* end for-j and for-i */
/* 9*/    return  MaxSum;
}
```

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

Algorithm 2

```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
        int  ThisSum,  MaxSum,  i,  j;
/* 1*/  MaxSum = 0;   /* initialize the maximum sum */
/* 2*/  for( i = 0; i < N; i++ )  {   /* start from A[ i ] */
/* 3*/      ThisSum = 0;
/* 4*/      for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 5*/          ThisSum += A[ j ];  /* sum from A[ i ] to A[ j ] */
/* 6*/          if ( ThisSum > MaxSum )
/* 7*/              MaxSum = ThisSum;  /* update max sum */
            }  /* end for-j */
        }  /* end for-i */
/* 8*/  return  MaxSum;
}
```

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

时间效率

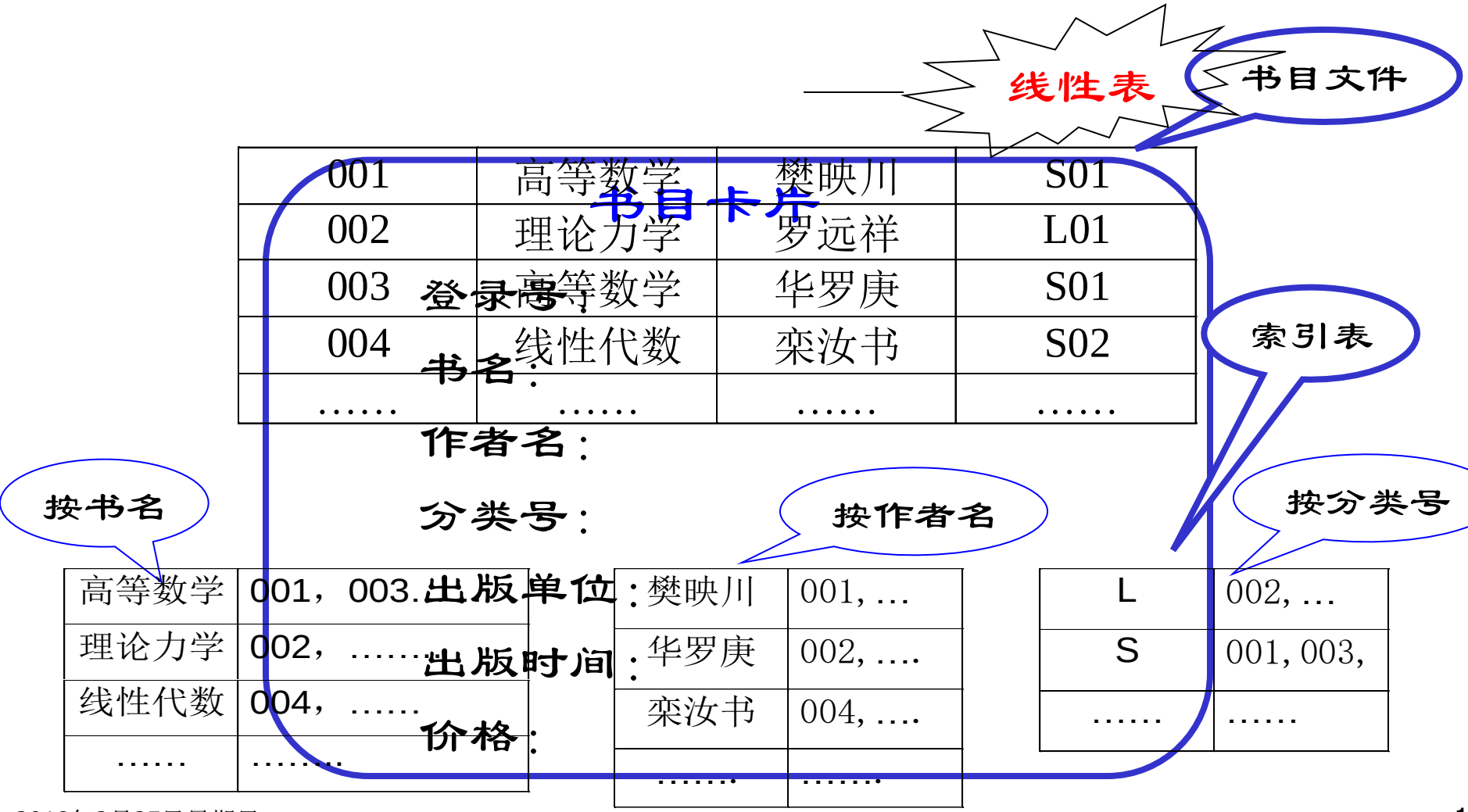
- 时间效率
  - 数据集：100万个数
  - 计算机每秒运行10亿条指令
  - 算法1：

    $(10^6)^3/10^9 = 10^9$秒≈11574天≈31.7年
  - 算法2：

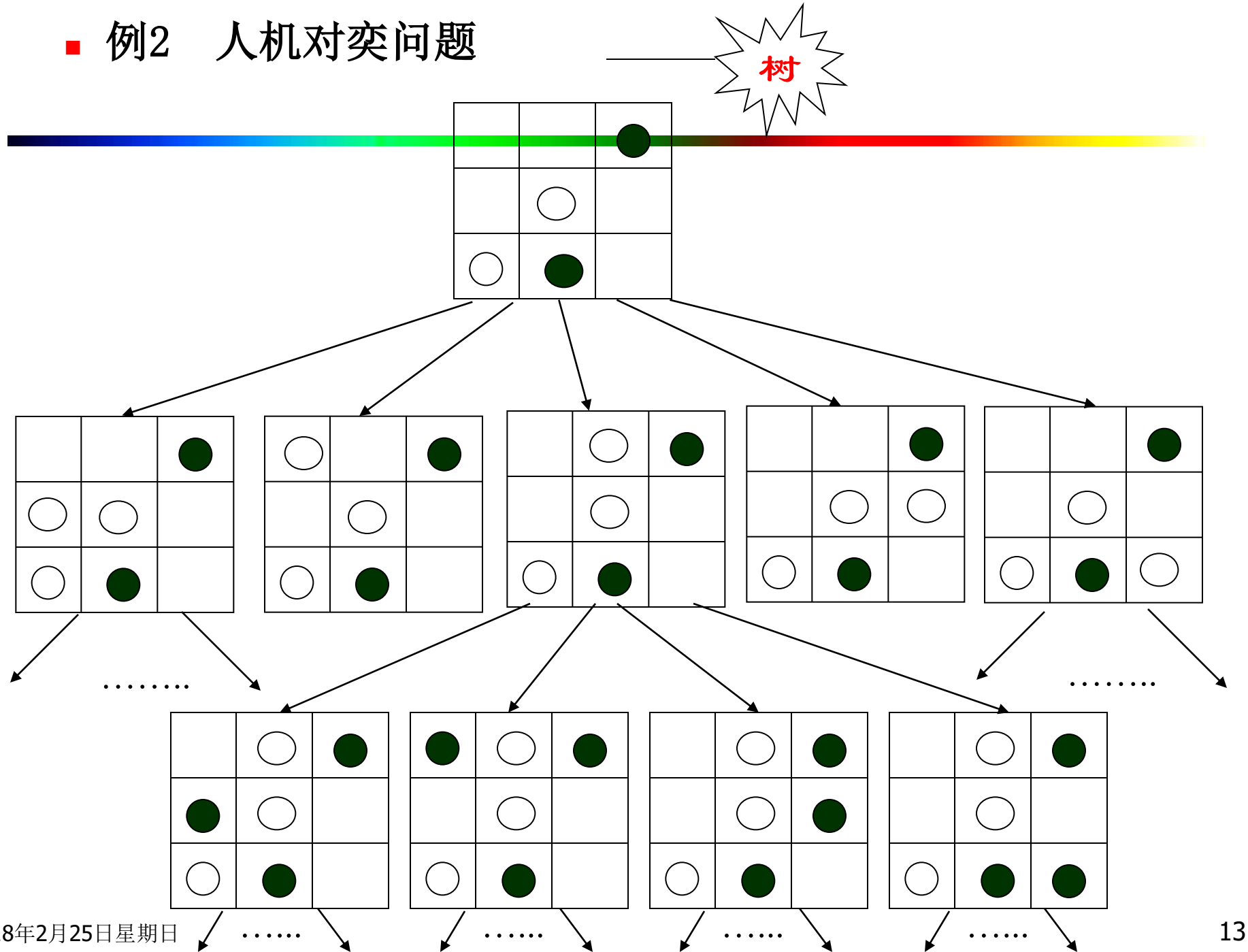
    $(10^6)^2/10^9 = 10^3$秒≈16.7分钟

举例

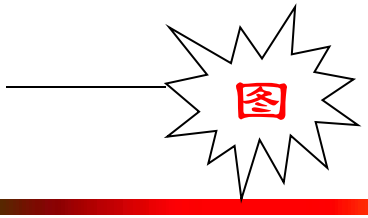
- 例1 书目自动检索系统

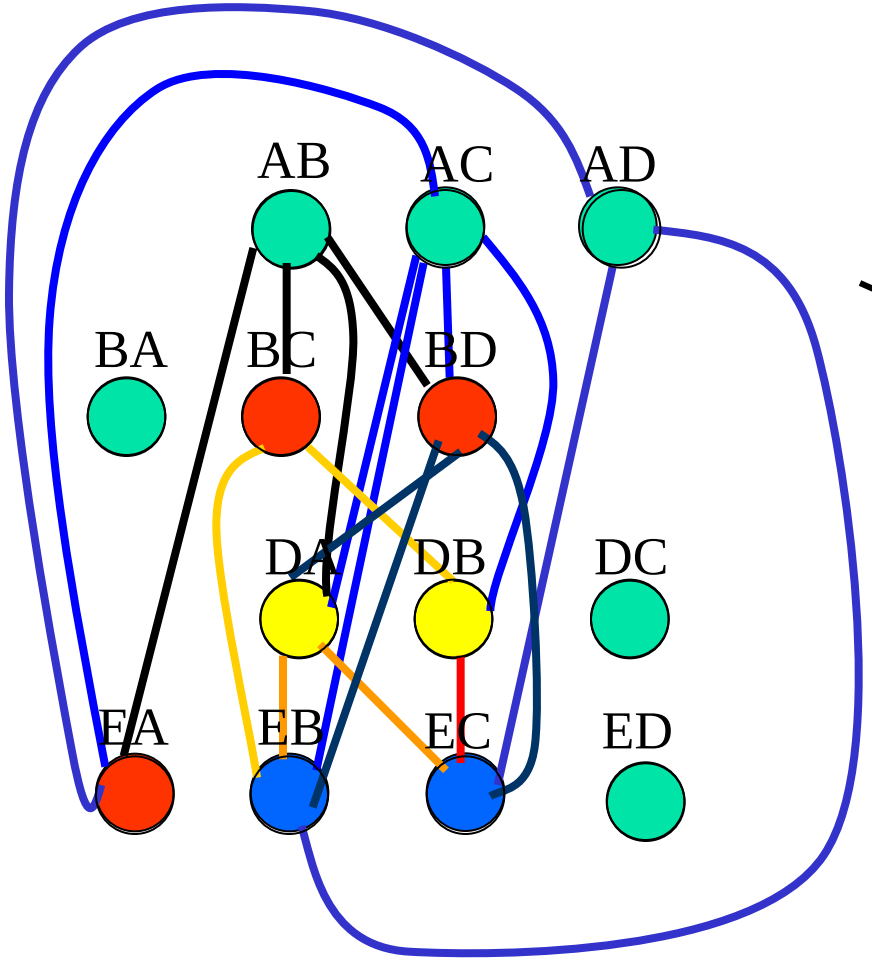


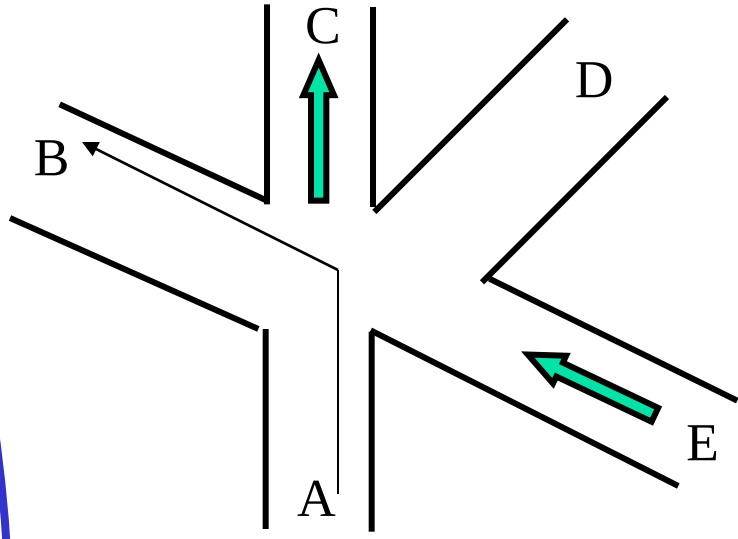
- 例2　人机对奕问题



例3 多叉路口交通灯管理问题







1.1.1 数据的逻辑结构

- 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:

  Data_Structure = {K, R}

  其中，K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。
- 常见逻辑关系有：线性(线性表，栈，队列，向量，字符串，多维数组，广义表)、树、图、文件（本质上是线性结构，而其索引则往往用树型）

线性结构

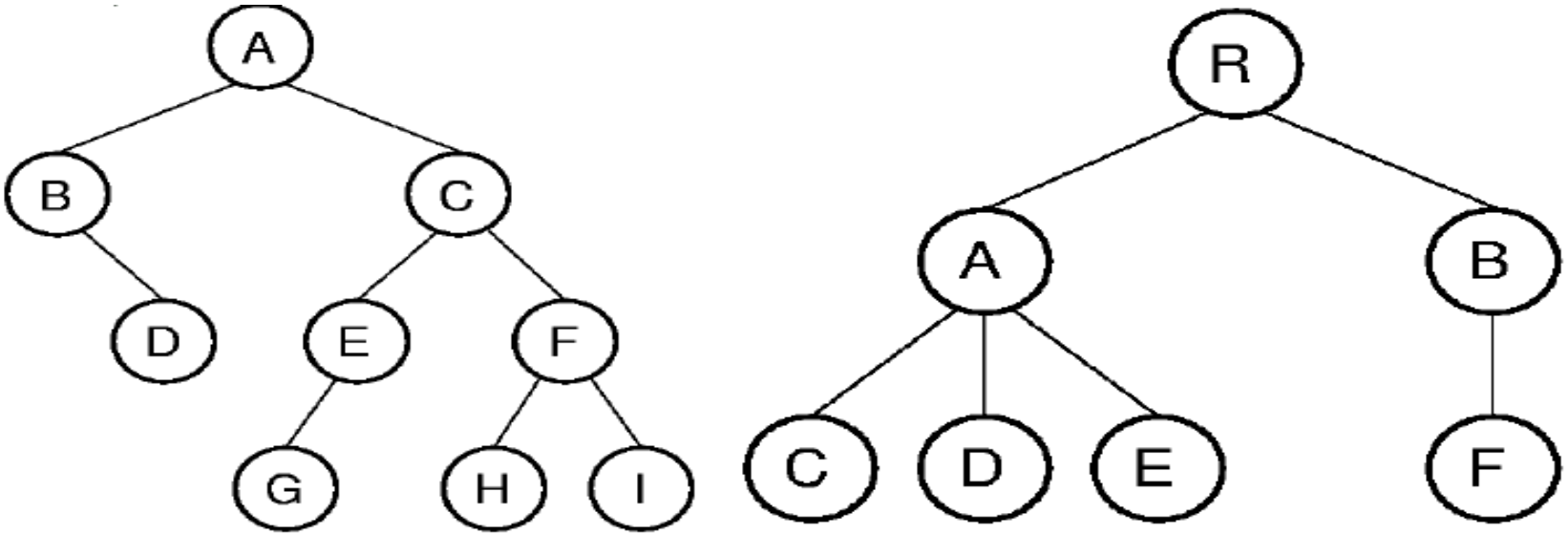
- **线性结构：**各结点最多只有一个直接前驱和一个直接后继，且只有一个结点无直接前驱，且只有一个结点无直接后继。
- 线性表的逻辑结构B=(K, R)，其中

  $K=\{k_0, k_1, \ldots, k_{n-1}\}$

  $R=\{r\}$，$r=\{\langle k_{i-1}, k_i\rangle \mid K_i \in K, 1<i<n\}$
- 图示法（注意与链表的图示区分开来）

树形结构

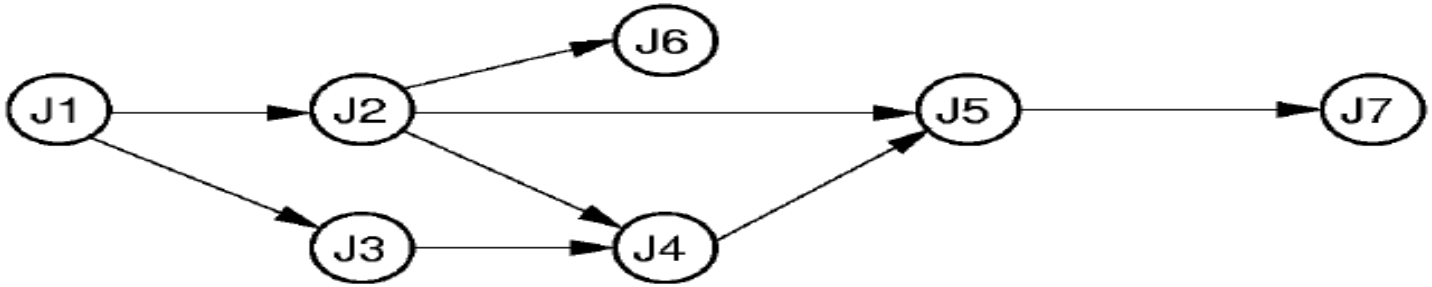
树形结构： B={K, R}, R={r}, r满足下列条件：有且仅有一个称为根的结点没有前驱；其它结点有且仅有一个前驱；对于非根结点都存在一条从根到该结点的路径。



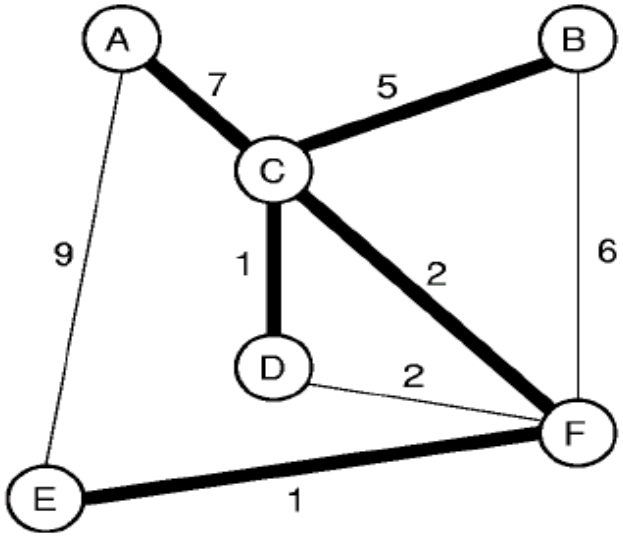
图

图： B={K, R}, R={r}, K中结点相对于r前驱和后继个数都没有限制。

有向图



无向图



左图逻辑结构B=(K, R)，其中K={A,B,C,D,E,F}，R={r}，

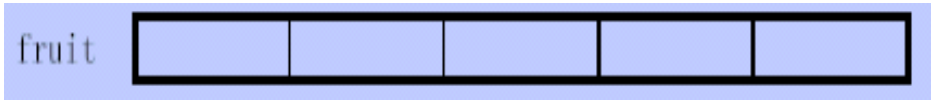
r={ (A,B),(B,F),(C,D),(C,F),(F,D) }

1.1.2　数据的存储结构

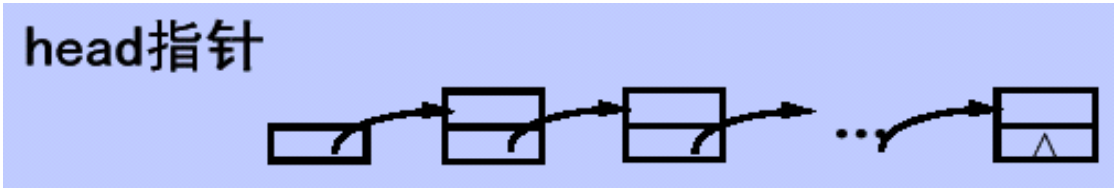
定义：若B=（K，R）是一个数据结构，有一个映射S：K→M，对于每一个k∈K，都有唯一的z∈M（存储区域），使得S（k）=z，同时这个映射S应具有明显地或隐含地体现关系R的能力，则称对逻辑结构进行了存储。

存储结构

- 顺序方法：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



- 链接方式：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



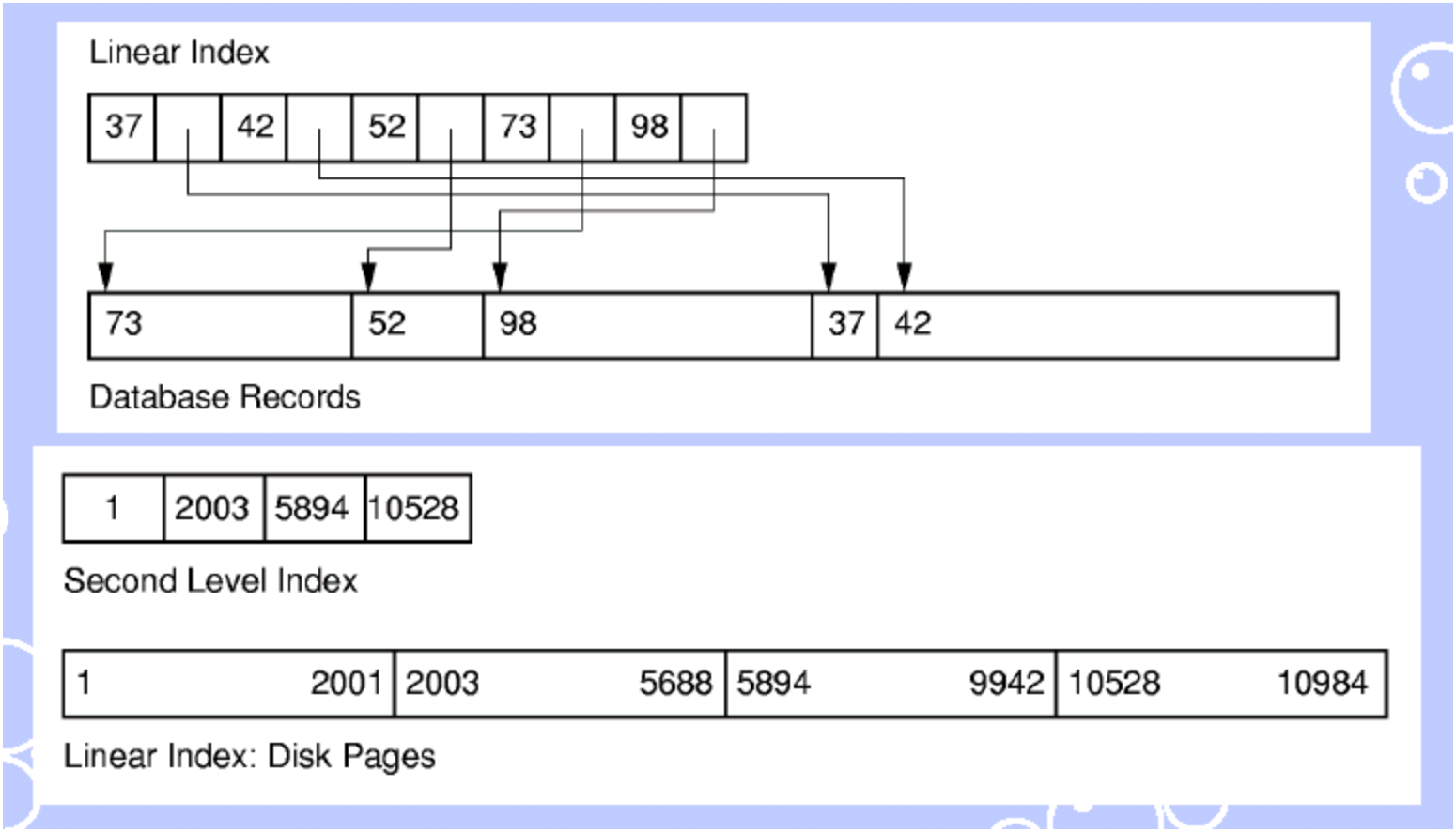
- 索引方法：
- 散列方法：

存储密度(≤ 1) =数据本身存储量/整个结构占用存储量

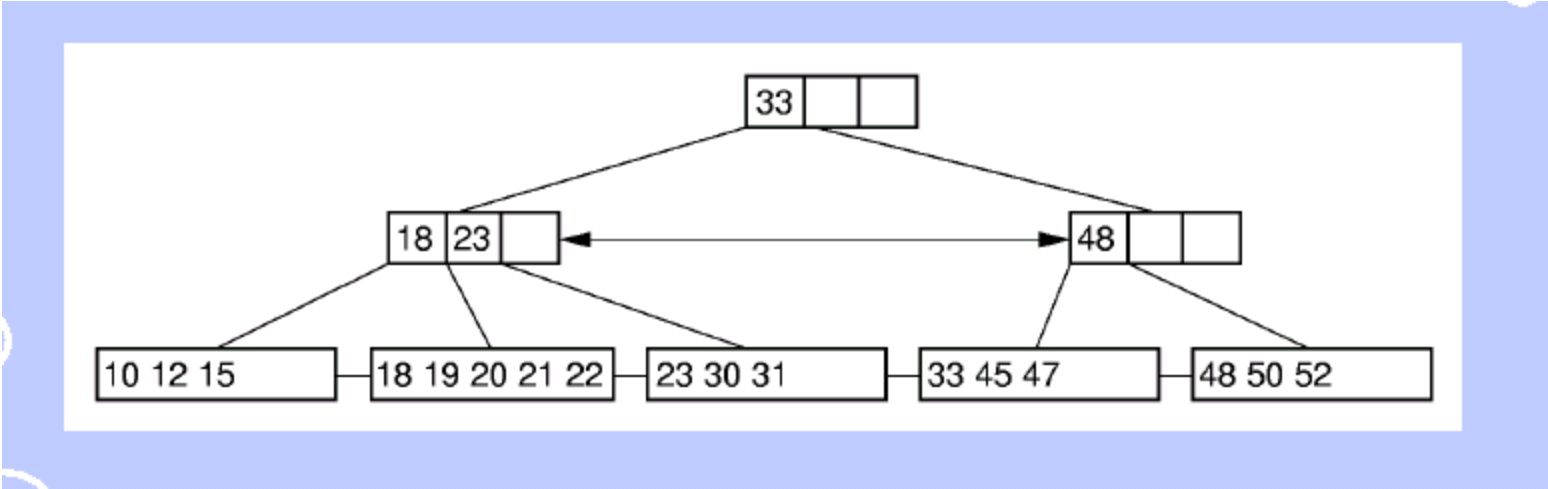
链表： $\rho = n \times E/n(P+E) = E/(P+E)$

n表示线性表中当前元素的数目

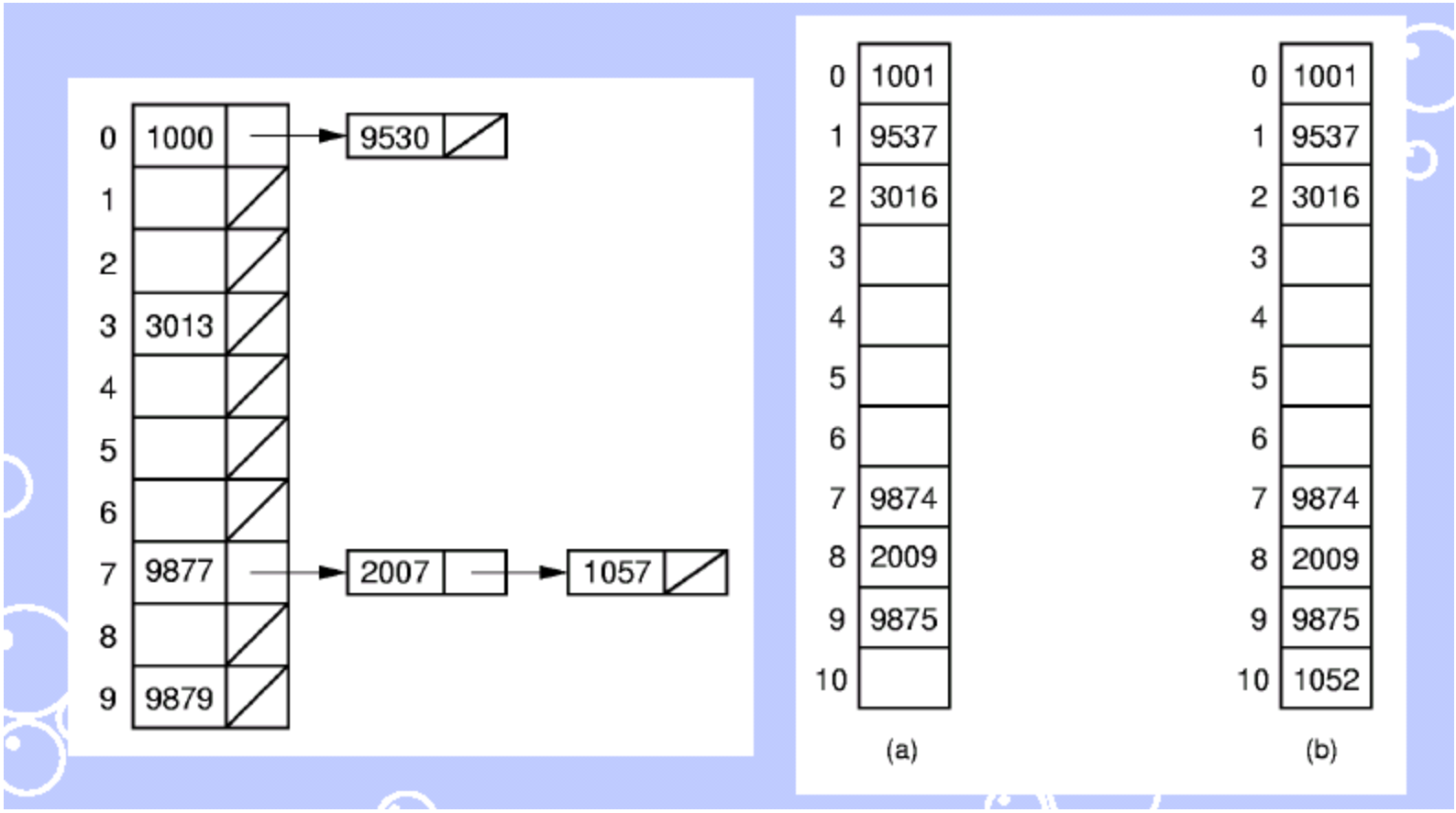
索引存储


Linear Index
37
42
52
73
98
73
52
98
37
42
Database Records
1
2003
5894
10528
Second Level Index
1
2001
2003
5688
5894
9942
10528
10984
Linear Index: Disk Pages

索引存储



散列存储


0 1000
9530
1
2
3 3013
4
5
6
7 9877
2007
1057
8
9 9879
0 1001
1 9537
2 3016
3
4
5
6
7 9874
8 2009
9 9875
10
(a)
0 1001
1 9537
2 3016
3
4
5
6
7 9874
8 2009
9 9875
10 1052
(b)

抽象数据类型和数据结构

- 数据抽象与过程抽象，实现信息隐蔽和局部化
  以一个严格定义的过程接口的方式，在数据结构上提供一个抽象．数据的实现和处理细节被隐蔽了．
- 抽象数据类型，定义了一组运算的数学模型．
  与物理存储结构无关，使软件系统建立在数据之上（面向对象）

抽象数据类型（Abstract Data Type，ADT）

- 数据(data)是描述客观事物的数字、字符以及其他能输入到计算机中并可被计算机程序处理的符号的集合。
- 数据元素(data element)是数据的基本单位，即数据集合中的个体。它是计算机的最小可存取单位，在计算机程序中通常被作为一个整体进行考虑和处理。有时一个数据元素可由若干个数据项(data item)组成。数据项是数据的不可分割的最小单位，也是数据集合的最小可命名单位。
- 类型(type)是一组值的集合。例如，布尔类型由两个值TRUE和FALSE组成；整数也构成一个类型。

ADT

- 数据类型(data type)是指一个类型以及定义在这个类型上的一组操作。例如整数类型是某个区间上的整数集合(区间大小依赖于不同的计算机)，定义在其上的操作为加、减、乘、除和取模等算术运算。
- 抽象数据类型(abstract data type，简称ADT)是对数据类型的逻辑形式的规范化描述。一个ADT的定义并不涉及它的实现细节，这些实现细节对于ADT的用户是隐藏的。隐藏实现细节的过程称为封装(encapsulation)。

线性表的ADT

```
class list {// List
   class ADT
public:
list(const int
   =LIST_SIZE);
~list();
void clear();
void insert(const ELEM&);
void append(const ELEM&);
ELEM remove();
void setFirst();
void next();
 void prev();
int length() const;
void setPos(const int);
void setValue(const
   ELEM&);
ELEM currValue() const;
bool isEmpty() const;
bool isInList() const;
bool find(const
   ELEM&); };
```

ADT的例子（1）

- 集合与集合的并、交、差运算可以定义为一个ADT
  - 一个ADT包含哪些操作由设计者根据需要确定
  - 例如，对于集合， 如果需要， 也可以把判别一个集合是否为空集或两个集合是否相等作为集合上的操作

ADT的例子（2）

- 驾驶汽车的主要操作包括控制方向、加速和刹车
  - 几乎所有的汽车都通过转动方向盘控制方向，通过踩油门加速，通过踩车闸刹车
  - 汽车的这种设计可以看作是具有“控制方向盘”、“加速”和“刹车” 操作的一个ADT
  - 两辆汽车可以用截然不同的方式实现这些操作
    - 但是大多数司机都能够驾驶许多不同的汽车
    - 因为ADT提供了一致的操作方法

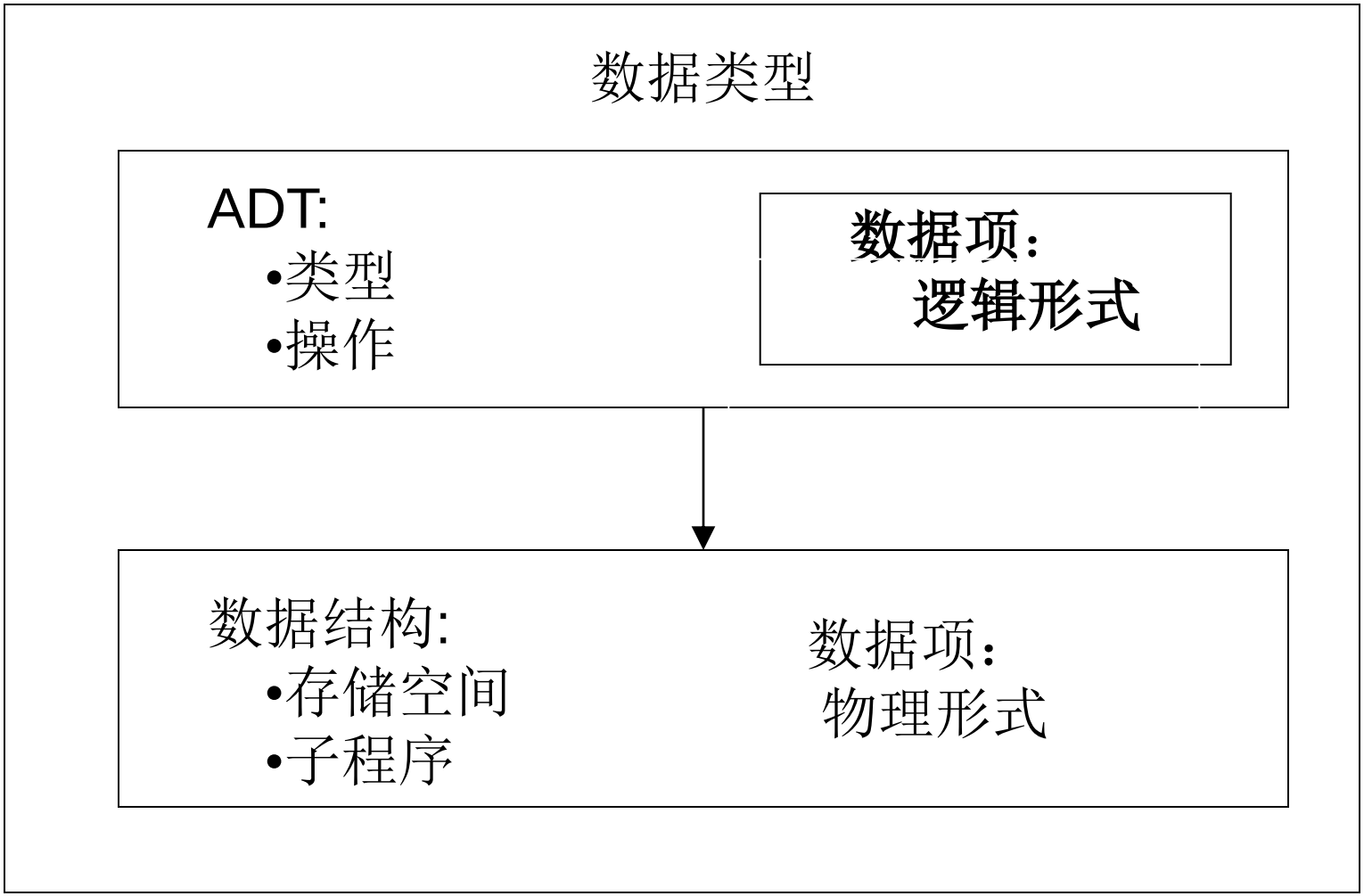
ADT的例子（3）

- 一个整数线性表的ADT应包含下列操作
  - 把一个新整数插入到线性表的末尾
  - 返回线性表中当前整数的个数
  - 重新初始化线性表
  - 删除线性表中特定位置上的整数

逻辑形式和物理形式

- 数据项有逻辑形式（logical form）和物理形式（physical form）两个方面
  - 逻辑形式：用ADT给出的数据项的定义
  - 物理形式：数据结构中对数据项的实现
- 实现一个ADT时，是处理相关数据项的物理形式
- 在程序中其他地方使用ADT时，涉及到相关数据项的逻辑形式

数据项、抽象数据类型和数据结构之间的关系



问题、算法和程序

问题

- 是一个需要完成的任务
  - 对应一组输入有一组相应的输出
- 问题的定义不应包含有关怎样解决问题的限制
- 问题的定义应该包含对任何可行方案所需资源的限制
  - 任何计算机程序只能使用可用的主存储器和磁盘空间
  - 必须在合理的时间内完成运行

问题（续）

- 可以把问题看作数学函数
  - 函数(function)是输入(即定义域，domain)和输出(即值域，range)之间的一种映射关系
  - 函数的输入可以是一个值或是一些信息
  - 这些值组成的输入称为函数的参数(parameter)
  - 参数的一组选定值称为问题的一个实例(instance)
  - 不同的实例可能产生相同的输出，但是对于问题的任何一个实例，只要把它作为给定输入，每次函数计算得到的输出就必然相同

算法

- 是指令的有限序列，其中每一条指令表示一个或多个操作
- 如果将问题看作函数，那么算法就是把输入转换为输出
  - 把输入映射到输出
- 一个问题可以用多种算法来解决

算法的基本特性

- 有限性　一个算法必须总是在执行有限步之后结束
- 确定性　算法中的每一步都必须具有确切的含义，不会产生二义性
- 可行性　算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的
- 输入　一个算法有零个或多个输入，这些输入取自某个特定的对象集合
- 输出　一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量

程序

- 是使用某种程序设计语言对一个算法的具体实现
- 不是所有的计算机程序都是算法
  - 算法必须可以终止
  - 操作系统是一个程序，而不是算法

算法的代价

- 是算法运行所需要的计算机资源的量
  - 需要的时间资源的量称为时间代价
  - 需要的空间(即存储器)资源的量称为空间代价
  - 这个量应集中反映算法所采用的方法的效率，而与运行该算法的硬件、软件环境无关
- 算法的代价是算法的效率的度量
- 引入算法的代价这个概念是为了比较解决同一问题的不同算法的效率之间的差异