

# 数据结构与算法

张铭

<http://db.pku.edu.cn/mzhang/DS/>

北京大学信息科学与技术学院

“数据结构与算法”教学小组

2007年9月10日

©版权所有，转载或翻印必究

# 教学目的...

- “数据结构+算法=程序”
  - 基本数据结构的**ADT**及其应用
  - 合理组织数据, 有效表示数据, 有效处理数据
  - 算法的设计分析技术
- 抽象能力
  - 问题——数据——算法
- 提高程序设计的质量

# 课程的主要内容

- 理论
  - 算法的数学基础
  - 算法的时间和空间度量
- 抽象
  - 排序、检索等重要问题类的有效算法
  - 重要数据结构技术
- 设计
  - 算法的选择、实现和测试

# 实习课目的

配合“数据结构与算法”主课，提高实际动手能力和程序设计的质量

- 基本数据结构

- 线性表(向量、串、栈和队列)、二叉树、树、图等

- ADT、STL

- 综合应用程序

- 排序、检索、文件、索引等技术

- 程序设计实践和技巧

# 实习课程内容(1/2)

- C++ 编程技术补充
  - 标准模板库 **STL** 的基本概念
  - C++ 流处理
- 程序设计实践和技巧
  - 风格、设计和实现
  - 界面、排错
  - 测试、性能和可扩展性

# 实习课程内容(2/2)

- 基本算法
  - 枚举法、贪心法
  - 递归、回溯、搜索与分支限界
  - 分治法、动态规划
- 问题建模
  - 数学建模、软件模型
- 数据结构的应用

主题	组长	组员
面向对象技术	毛琛	吴迪
<b>STL和C++</b>	钱昊	巨程
调试和测试技术	赖博彦	陈学轩 丁羽 陈醒 王瑞超
递归和回溯	王子琪	谭裕韦
图、搜索、剪枝	姚金宇	黄柏彤 陈琪
动态规划	杨涛	金鑫 李昂 周金果
算法优化	李森	贾由
数学建模技术	汪瑜婧	雷涛 罗睿辞 王尧
数据结构与算法的应用	陈志杰	冯熙铉 张策

# 实习课程进度(1/2)

- 9月12日 第一周 数据结构与算法实习简介
- 9月19日 第二周 算法（一）：穷举法
- 9月26日 第三周 算法（二）：回溯法
- 10月3日 第四周 国庆放假
- 10月10日 第五周 算法（三）：贪心法, 算法优化
- 10月17日 第六周 程序设计实践（一）：风格、设计和实现，面向对象技术
- 10月24日 第七周 程序设计实践（二）：**STL**的基本概念和常用容器
- 10月31日 第八周 算法（四）：分治法



# 实习课程进度(2/2)

- 11月7日 第九周 算法（五）：动态规划
- 11月14日 第十周 习题讨论，布置大实习，讨论项目管理
- 11月21日 第十一周 程序设计实践（三）：界面和排错
- 11月28日 第十二周 程序设计实践（四）：测试、性能和可扩展性
- 12月5日 第十三周 问题建模专题讨论
- 12月12日 第十四周 图的应用
- 12月19日 第十五周 数据结构应用
- 12月26日 第十六周 上机题讲评，期末总复习

# 主课教学考核

- 期中**20 %**
- 期末**20 %**
- 高级数据结构**20%**
- 平时(考勤+课堂)**20 %**
- 书面作业、上机作业**15 %**
- 态度**5%**

# 考 勤

- 不要迟到早退、旷课
  - ◆ 有事提前请假
- 申请“课堂免修”（自学）？
  - ◆ 同样交作业、上机题、考试
  - ◆ 不建议
    - ◆ 学习需要环境
- ◆ 考勤措施：随堂小测试

# 作业要求

- 1. 主课只有书面作业(每周4道)
  - “我保证本次作业由我本人独立完成, 没有抄袭他人作业”
  - 不需要调试
- 2. 实习课3道大综合实习, 7道ACM
  - “诚实代码”
  - 要调试
  - 要提交上机报告

# 按时提交作业，严禁抄袭

- 每周第一次课课间交书面作业(或之前ftp提交电子版)
- 计分标准**10**分，期末加权。规则：
  1. 准时提交，满分可达**10**分（个别加分）；
  2. 延迟**3**天之内提交，满分可达**7**分；
  3. 延迟**7**天之内提交，满分可达**3**分；
  4. **7**天之后提交或不交，得分 - **5**分。
  5. 抄袭得 - **20**分。

# 作业提交期限的说明

- 1. 有利于助教及时批改、讲解
- 2. 有利于同学及时讨论复习
  
- 破例申请——要在**deadline**前提出
  - 1. 个别有困难的同学
  - 2. 生病或事假

# 诚信

- 端正学习态度、调动学习兴趣
  - ◆ 提倡讨论，但**严禁抄袭**
    - ◆ 可以讨论思路
    - ◆ 但要亲自动手实现
  - ◆ 发现抄袭，**严肃处理**
    - ◆ 抄袭者和被抄袭者本次作业或上机题计双倍倒扣分，即得 - **20**分
    - ◆ 以后的作业题会得到重点检查
    - ◆ 严重的期评将给予不及格处理

# 书面作业提交要求

- 写学号、名字
- 写“**XX**保证没有抄袭他人作业”的诚实保证。
- 写算法分析、注释
- 注意算法格式(层次嵌套、不同功能块之间留空)
- 否则，**计零分**或根据抄袭情况**倒扣分**。



# 学习过程中的问题

- 问题1：进度快，听不懂
  - 别人是牛人，自己是菜鸟
  - 同学发言听不懂
- 建议：主动学习
  - 建立信心——我是精英，我能行
  - 预习——聪明鸟先飞
  - 提问式学习——置疑、创新
  - 复习——总结、提高、实践应用
    - 以考题的形式来看书，做习题

# 学习过程中的问题

- 问题2：算法思想能懂，但代码不懂
  - 还没有入门，经验不够
- 建议：多读代码
  - 先弄懂思想，再看代码
  - 抽象——具体——抽象
    - 弄懂算法的大框架——抽象
    - 要多用实例走几遍，特例检测边界——具体
    - 再总结算法的主要功能块——抽象
  - 经典算法代码不仅要懂，还要自己能写出来
    - 不是背，是用自己的风格重写
  - 对经典算法的灵活
    - 修改问题条件，仿写

# 学习过程中的问题

## ■ 问题3：编写的算法质量不高

- 数学模型建立不了
  - 本质上是数学思维能力，抽象能力
  - 递归思想
- 边界处理不周到
- 有思想写不出代码

## ■ 建议：多写代码

- 事先想明白算法思想
- 编写结构清晰的程序
  - 顺序、条件选择、循环、函数
- 不要过于依赖编译器

# 学习过程中的问题

## ■ 问题4：不能独立完成作业

- 抄同学的——作弊
- 不深入考虑，就问——没有收获
- 翻书，照着代码修改——考试有问题

## ■ 建议：独立完成作业

- 先看一遍教材、或讲义
- 不懂的地方找同学请教，或看视频
- 还不懂，在**bbs**上提问，或找责任助教

# 学习过程中的问题

## ■ 问题5：不好意思与同学交流

- 怕打搅
- 怕抄袭嫌疑

## ■ 建议：主动交流、善于交流

- 沟通、交流是重要的情商
  - 增进感情、为将来的合作打基础
- 能加深对问题的理解，提高作业质量
- 只要不直接看答案，思想类似不会构成抄袭
- 重要前提：一定要自己先思考，否则效率低

# 学习过程中的问题

- 问题6：不善于利用资源

- 课程网站、课程讨论bbs
- 助教、同学
- 推荐作业

- 建议：高效调动资源

- 多看参考资源
- 多请教、讨论
- 推荐作业
- 研究复习大纲

# 课程资源

- 数据结构与算法（信息学院）
  - ◆ <http://db.pku.edu.cn/mzhang/DS/>
- 数据结构实习（计算机和智能专业强化）
  - ◆ <http://db.pku.edu.cn/mzhang/DS/shixi/index.htm>
- 课程答疑
  - ◆ <http://db.cs.pku.edu.cn/mzhang/ds/bbs/>
  - ◆ 注册：**h-学号xxx**

# 实验班资源

- <http://db.pku.edu.cn/mzhang/ds/honor>

## ■ 实验班作业提交

[ftp://ds\\_honor:ds07honor@fusion.grids.cn](ftp://ds_honor:ds07honor@fusion.grids.cn)

- 用户名: **ds\_honor**
- 密码: **ds07honor**



# 教材

- 许卓群、杨冬青、唐世渭、张铭，《数据结构与算法》，高等教育出版社，**2004年 7月**。
- 张铭、赵海燕、王腾蛟，《数据结构与算法——学习指导与习题解析》，高等教育出版社，**2005年 10月**。

# 参考书目

- 张铭、刘晓丹译，《数据结构与算法分析》(C++第二版、Java版)，电子工业出版社，2002年。（用1998年的第一版也可以，也可以直接用英文原版）
- 许卓群，《数据结构》，高等教育出版社，1988。

# 教学参考书

- Donald E. Knuth, The Art of Computer Programming, Addison Wesley. Vol. 1, Vol 3. 国防工业出版社影印。（苏运霖译）
- Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest, Clifford Stein, Inroduction to Algorithms, MIT Press. 高等教育出版社影印。
- William Ford, 《Data Structure with C++》，清华大学出版社

# 教学参考书(续)

- 殷人昆, 《数据结构——用面向对象方法与C++描述》, 清华大学出版社, 1999。
- 张乃孝, 裘宗燕, 《数据结构——C++与面向对象的途径》, 高等教育出版社, 2001。
- 严蔚敏, 《数据结构》 第二版(Pascal和语言版都可以), 清华大学出版社。
- 严蔚敏, 《数据结构题集》, 清华大学出版社

# 主课助教

- 王位春：学院课程总助教
  - 13810026968
- 黄燕京
- 吴定明
- 邢国峰(实习课总助教)

# 第一章 概论

- 为什么要学习数据结构
- 什么是数据结构
- 抽象数据类型
- 算法的特性及分类
- 算法的效率度量
- 数据结构的选择和评价
- 总结

# 为什么要学习数据结构

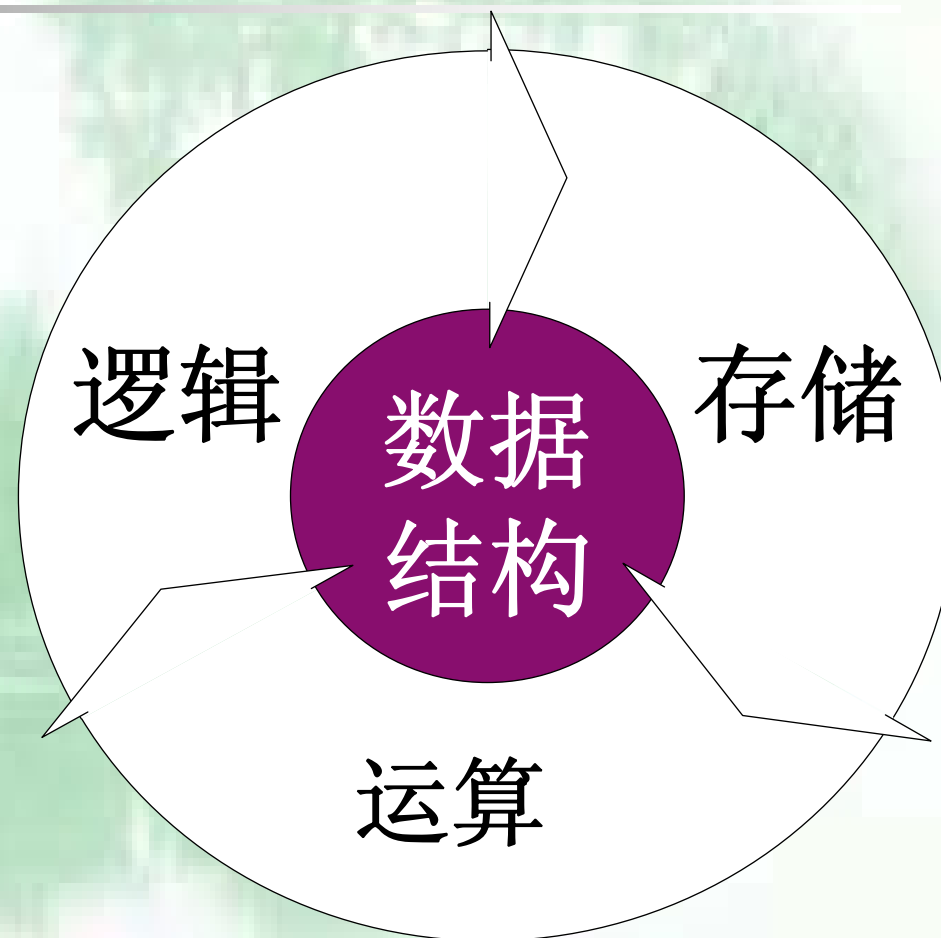


- 计算机软件与理论学科的专业基础课程
- 后续专业课程学习的必要知识与技能准备
  - 编译技术要使用栈、散列表及语法树
  - 操作系统中用队列、存储管理表及目录树
  - 数据库系统运用线性表、多链表、及索引树
  - .....
- 增强求解复杂问题的能力

# 什么是数据结构 (data structure)



- 数据逻辑结构
- 数据的存储结构
- 数据的运算





# 数据的逻辑结构

- 二元组:  $(K, R)$ 
  - $K$ 是由有限个结点组成的集合
  - $R$ 是定义在集合 $K$ 上的一组关系, 其中每个关系(relation)  $r$  ( $r \in R$ ) 都是 $K \times K$ 上的二元关系
  - 数据结构中, 只讨论 $R = \{r\}$

# 常见的逻辑关系

- 线性结构
- 树形结构
- 图结构
- 文件结构

图 $\supseteq$ 树 $\supseteq$ 二叉树 $\supseteq$ 线性表

# 结点的类型

- 基本数据类型

- 整数类型(integer)

- 实数类型(real)

- 布尔类型(boolean)

- 在C++语言中一般使用整数0表示

- false**，用非0表示**true**

# 结点的类型

- 基本数据类型

- 字符类型(char): 用单个字节（8bit，最高位bit为0）表示ASCII字符集中的字符。
  - 汉字符号需要使用2个字节（每个字节的最高位bit为1）的编码
  - Unicode, GB, Big5, HZ

# 结点的类型

- 基本数据类型
  - 指针类型(**pointer**): 指向某一内存单元的地址
    - 32 bit机器, 4个字节表示一个指针
    - 64bit的机器, 8指针
  - 指针操作
    - 分配地址
    - 赋值 (另一个指针的地址值, **NULL**空值)
    - 比较两个指针地址
    - 指针增减一个整数量

# 结点的类型

- 复合数据类型

- 基本数据类型/复合类型

- 数组 **int A[100];**

- 结构 **typedef struct {} B;**

- class C { };**



# 结点和结构

- 数据结构的设计一层一层地进行
  - 先明确数据结点，及其主要关系 $r$
  - 对于复杂情况，再分析下一层次的逻辑结构
- 自顶向下的分析设计方法

# 结构的分类

- 对  $(K, R)$  的分类，用  $R$  的性质来刻画
  - 线性结构 (linear structure)
  - 树型结构 (tree structure)
  - 图结构 (graph structure)

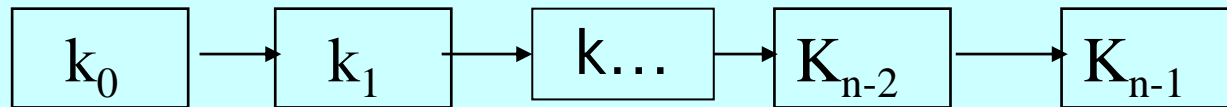


# 线性结构

- 应用最广泛，关系 $r$  是一种线性关系
  - ‘前后关系’
  - ‘大小关系’
- 关系 $r$ 有向，全序性和单索性等约束条件
  - 全序性：全部结点两两皆可以比较前后（关系 $r$ ）
  - 单索性：每一个结点 $y$ 都存在唯一的直接后继结点 $z$ 
    - $y \rightarrow z$
    - $x \rightarrow \dots \rightarrow z$
    - 则 $x \rightarrow \dots y \rightarrow z$
  - $x = y$  ?

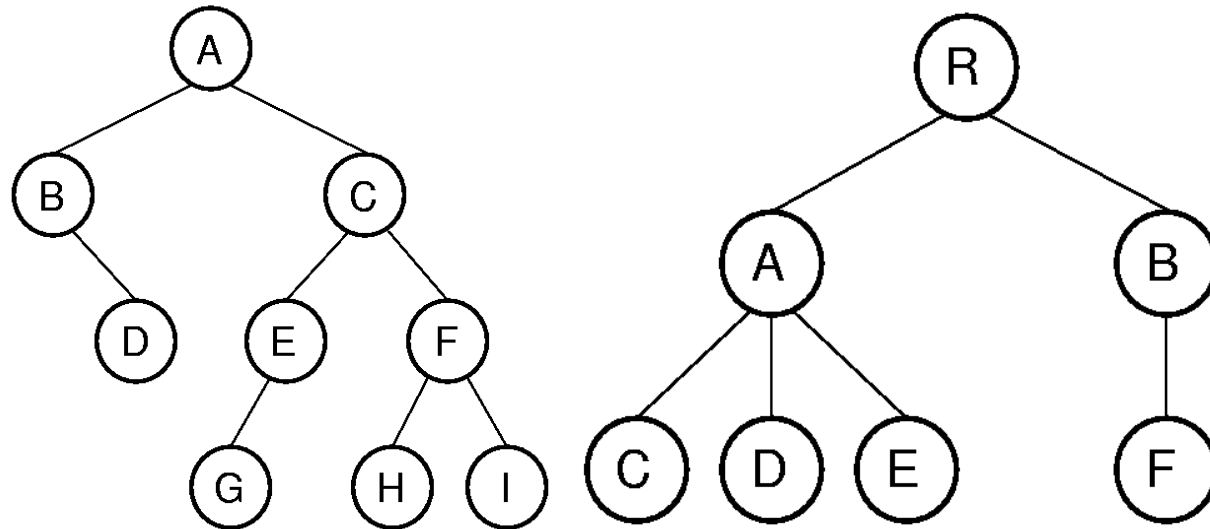
# 线性结构

- 图示法（注意与链表的图示区别开来）



# 树型结构

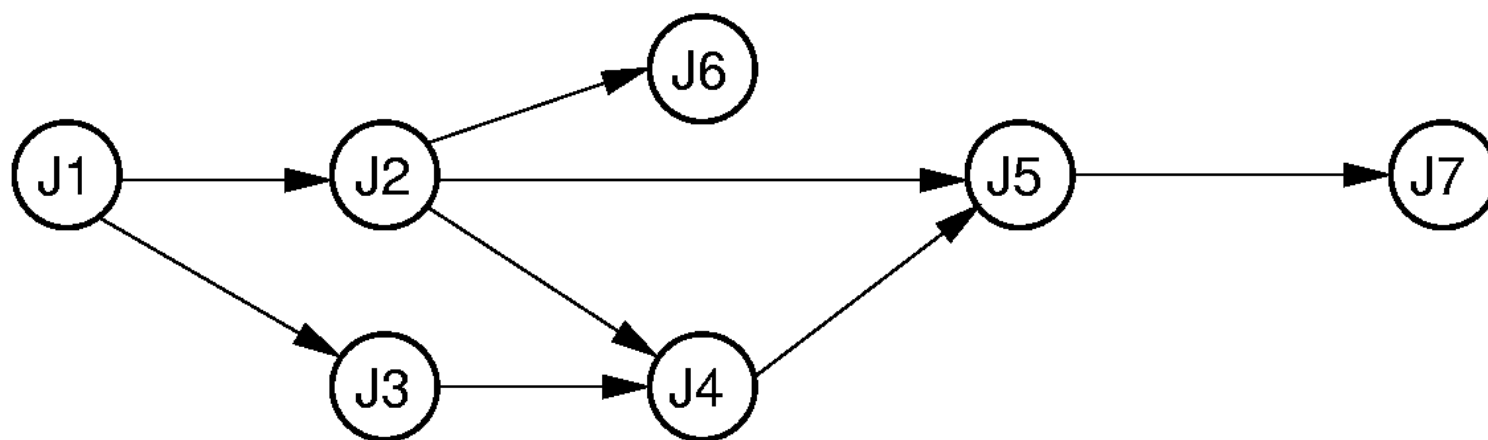
- 其关系 $r$ 为层次关系，或称‘父子关系’、‘上下级关系’
- 每一个结点可以有**多于一个的‘直接下级’**，但是它只能有**唯一的‘直接上级’**。
- **根（root）**结点没有父结点



# 图结构

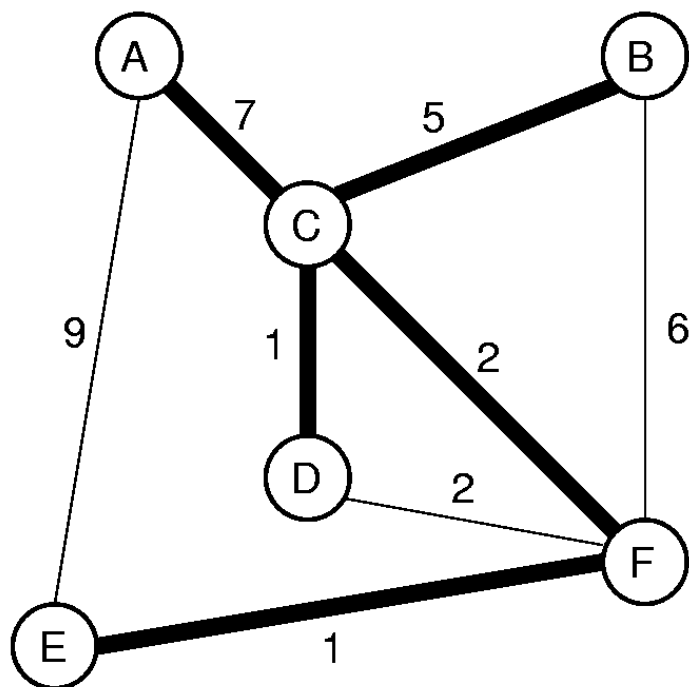
图：  $B=\{K, R\}$ ,  $R=\{r\}$ ,  $K$ 中结点相对于 $r$ 前驱和后继个数都没有限制。

有向图



# 图结构

无向图



左图逻辑结构 $B=(K, R)$ ，其中

$K = \{A, B, C, D, E, F\}$

$R = \{r\}$ ,  $r = \{ \langle A, C \rangle, \langle C, A \rangle,$   
 $\langle A, E \rangle, \langle E, A \rangle, \langle B, C \rangle, \langle C, B \rangle,$   
 $\langle B, F \rangle, \langle F, B \rangle, \langle C, D \rangle, \langle D, C \rangle,$   
 $\langle C, F \rangle, \langle F, C \rangle, \langle D, F \rangle, \langle F, D \rangle,$   
 $\langle E, F \rangle, \langle F, E \rangle \}$

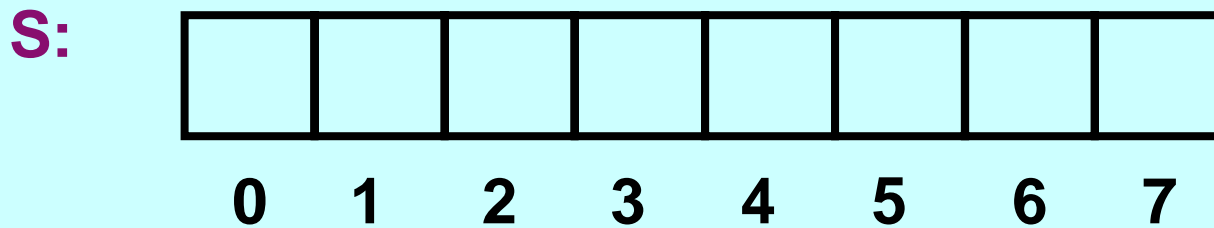
可以简写为:

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

# 数据的存储结构

- 映射：逻辑  $\rightarrow$  存储
- 对于数据逻辑结构  $(K, r)$ , 其中  $r \in R$ 
  - 从  $K$  到存储器  $M$  的单元的映射:  $K \rightarrow M$ 
    - 每一个结点  $k \in K$  都对应于唯一的区域  $c \in M$ 。
  - 关系元组  $(k_1, k_2) \in r$  ( $k_1, k_2 \in K$  映射为存储单元的地址指向关系
- 四种存储结构：顺序、链接、索引、散列
- 空间时间权衡
  - 尽量少的空间，尽量快的运算速度

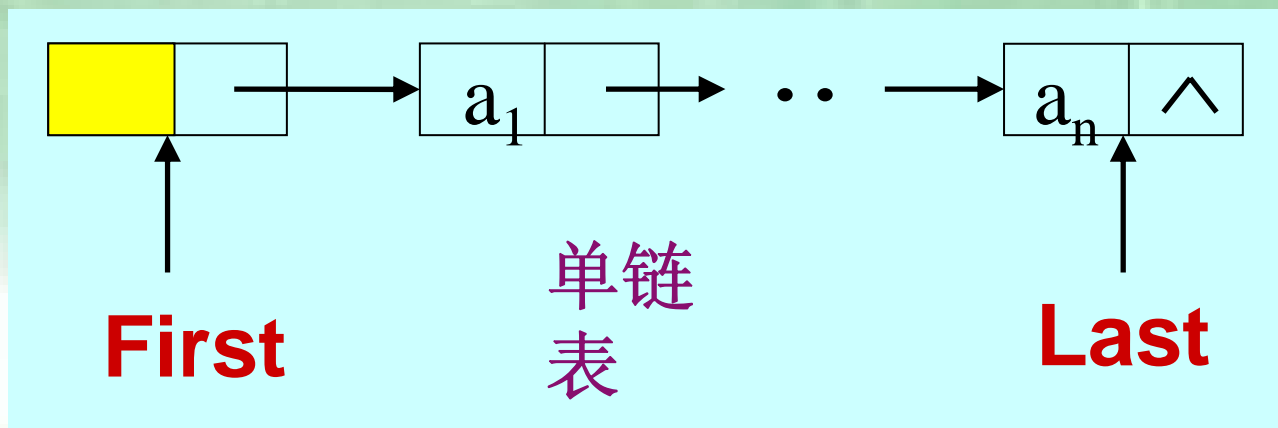
# 顺序存储——数组



- 把一组结点存储在按地址相邻的顺序存储单元里
  - 结点间的逻辑后继关系用存储单元的自然顺序关系来表达
- 紧凑存储结构
- 紧凑性可以用‘存储密度’来度量：
  - 存储密度( $\leq 1$ ) = 数据本身存储量 / 整个结构占用存储量

# 链接（linked）的方法

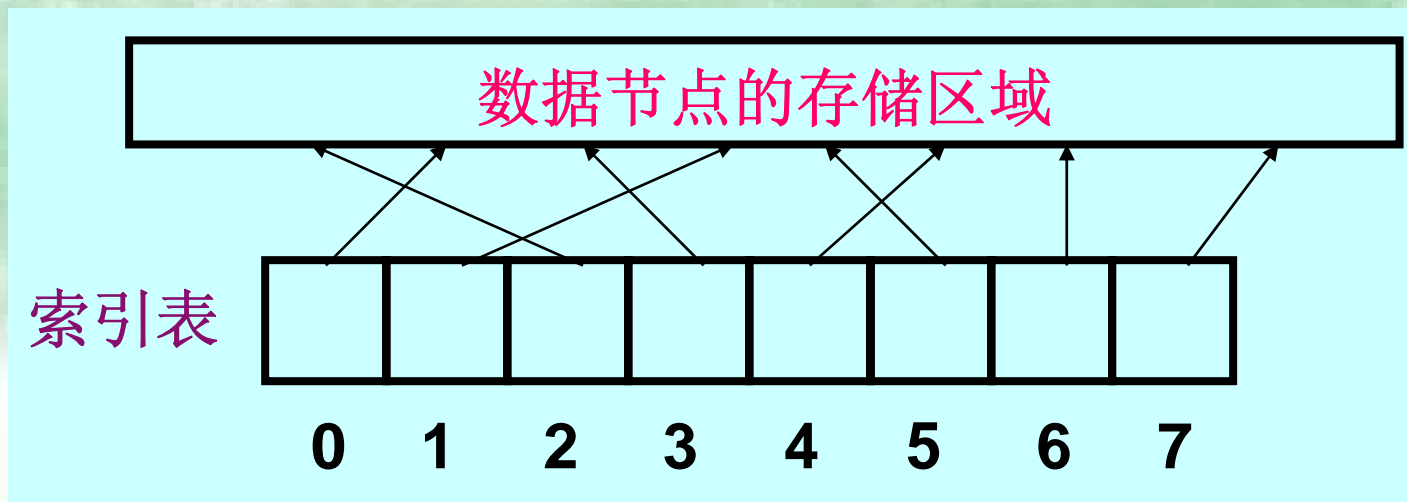
- 指针的指向：结点间逻辑后继关系
- 两部分：数据字段、指针字段
- 链索：多个相关结点的依次链接就会形成
- 存储密度： $\rho = n \times E / n(P + E) = E / (P + E)$





# 索引（indexing）的方法

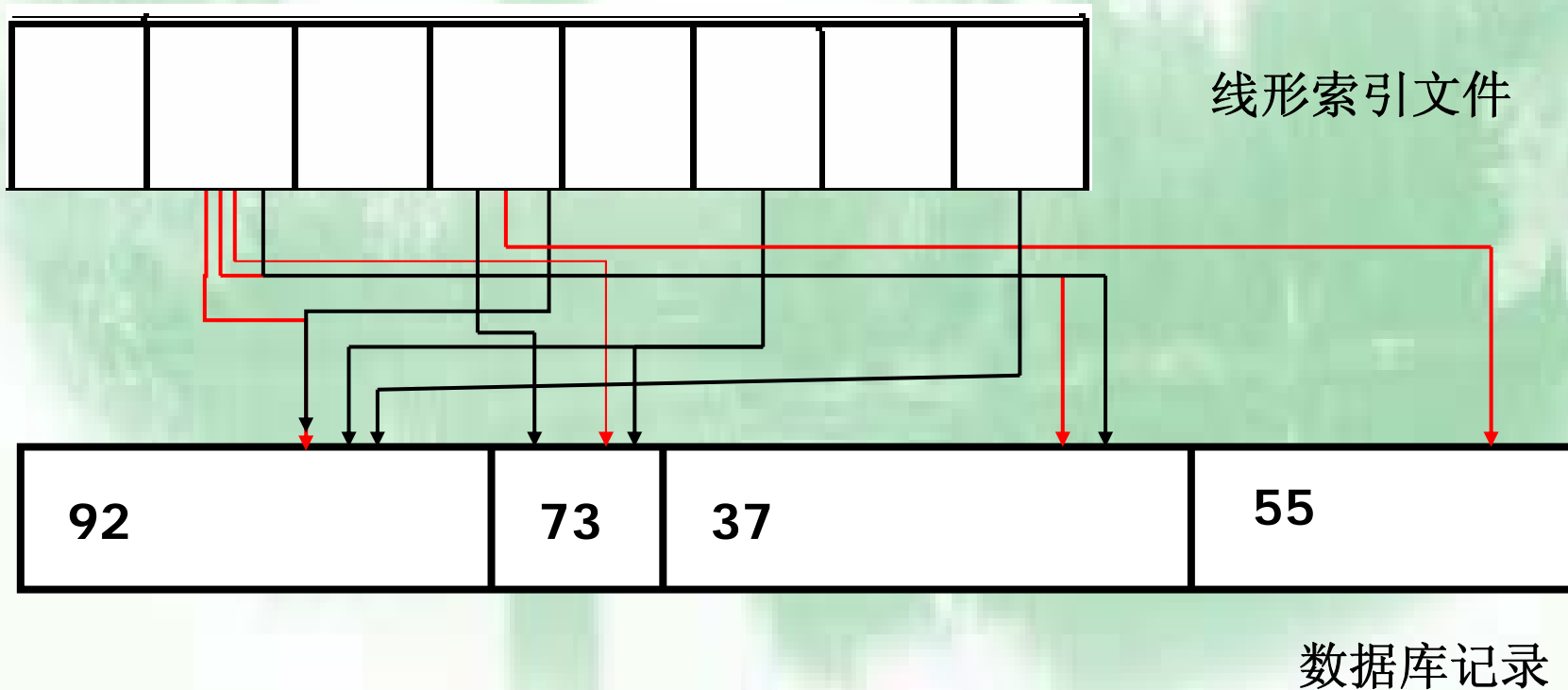
- 索引法是顺序存储法的一种推广，它也使用整数编码来访问数据结点位置



- 索引函数  $Y: Z \rightarrow D$

- 结点的整数索引值  $z \in Z$

- 结点的存储地址  $d \in D$

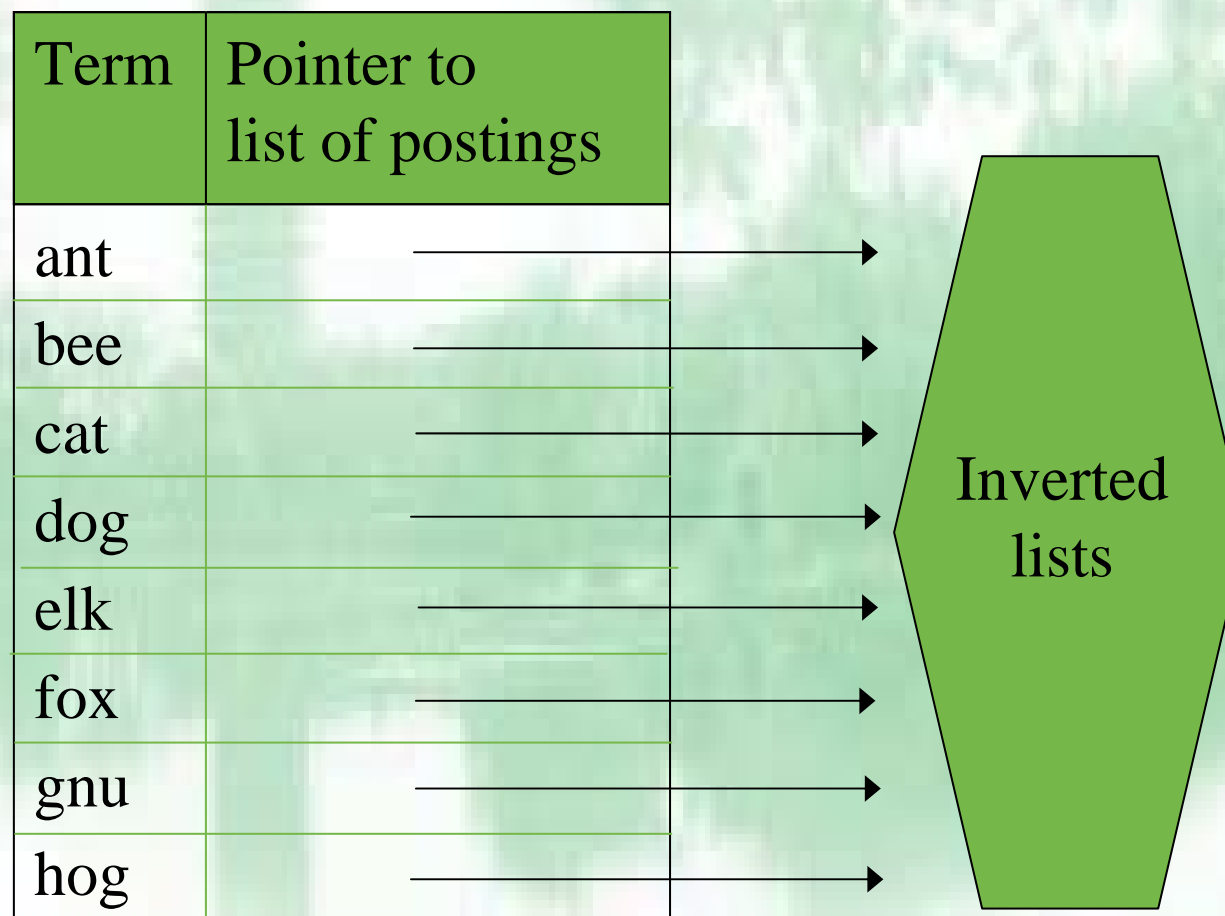


# 倒排文件

<i>Word</i>	<i>Postings</i>	<i>Document</i>	<i>Location</i>
<b>abacus</b>	<b>4</b>	<b>3</b>	<b>94</b>
		<b>19</b>	<b>7</b>
		<b>19</b>	<b>212</b>
		<b>22</b>	<b>56</b>
<b>actor</b>	<b>3</b>	<b>2</b>	<b>66</b>
		<b>19</b>	<b>213</b>
		<b>29</b>	<b>45</b>
<b>aspen</b>	<b>1</b>	<b>5</b>	<b>43</b>
<b>atoll</b>	<b>3</b>	<b>11</b>	<b>3</b>
		<b>11</b>	<b>70</b>
		<b>34</b>	<b>40</b>



# 倒排文件与线性索引



# 散列（hashing）的方法

- 散列是索引方法的一种延伸和扩展
  - 更快的检索速度
  - 受数组元素地址计算启发
    - $\text{Loc}(A[i]) = \text{Loc}(A[0]) + i * \text{sizeof}(\text{Element});$
- 散列函数是将字符串 $s$ 映射到非负整数 $z$ 的一类函数 $h: S \rightarrow Z$  ,  
对任意的  $s \in S$ , 散列函数  $h(s)=z, z \in Z$

# 散列（hashing）的方法

- 散列函数 $h(s)$ 除了它取非负整数值外，关键的问题：
  - 恰当地选择散列函数
  - 如何建造散列表
  - 在构建散列表的中间解决‘碰撞’的办法

# 散列（hashing）的方法

0	77		→ 110
1			
2			
3	14		
4			
5			
6			
7	7		→ 95
8			
9	75		
10			

$$H(k) = k \% 11$$



# 数据的运算

- 逻辑结构和存储结构都相同，但运算不同，则数据结构不同
  - 例如，栈与队列
- 对于一种数据结构，常见的运算
  - 建立、清除数据结构
  - 插入、删除、修改某个数据元素
  - 排序、检索




# 排序问题

- **Google**等搜索引擎返回结果排级
- 图书馆员书目编号、上架
- 各种排名
  - 《泰晤士报》 大学排名
  - 《福布斯》 富豪榜
  - 保研成绩排名
  - .....
- **Windows**资源管理器，文件查看
- .....



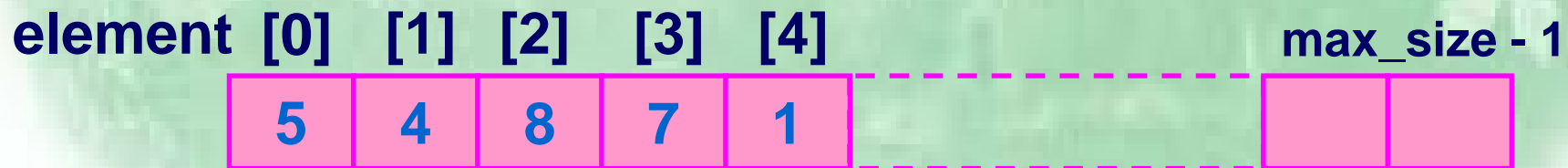
# 抽象数据类型ADT

- 抽象数据类型是定义了一组运算的数学模型
  - 剥离存储与实现细节
  - 在适当的抽象层次上考虑程序的结构和算法
  - 封装和信息隐蔽

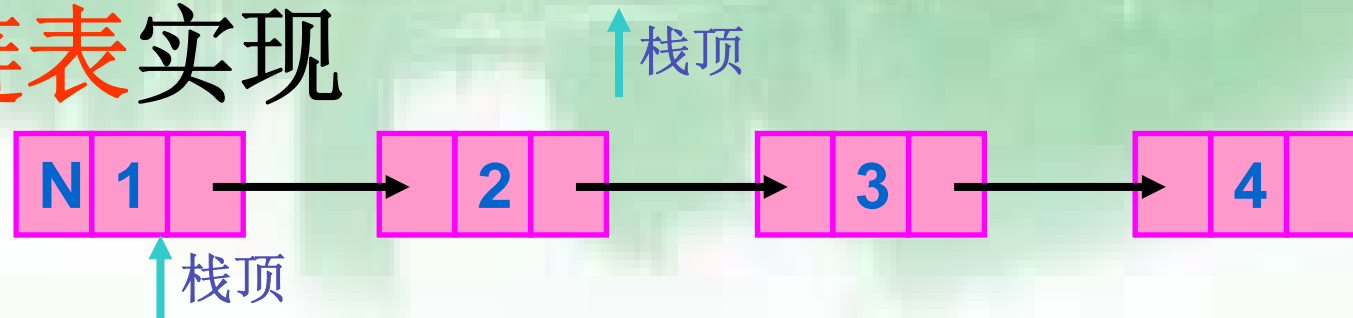
- 
- 逻辑、存储、运算三者不同，都可以看作不同数据结构
  - 忽略存储，强调逻辑、运算则为 **ADT**
  - 不特别指明，允许使用 **STL**

# 栈的不同存储实现

- 插入、删除、检索都只能被限制在  
同一端的线性表**先进后出LIFO**
- **顺序**实现



- **链表**实现



# 栈(stack)

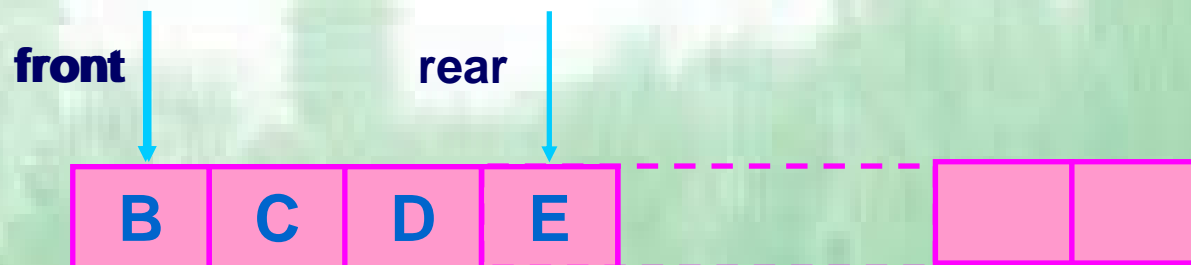
栈：限制访问端口的线性表：LIFO表

- Push: 元素插入称为栈的‘压入’
- Pop: 删除称为栈的‘弹出’
- Top: 表首被称为‘栈顶’



# 队列（queue）

- 插入在一端，删除、检索在另一端的线性表
- 先进先出FIFO



# 栈的抽象数据类型

```
enum Boolean {True,False}  
template <class ELEM> class Stack  
{ // 栈的元素类型为ELEM  
    //栈的存储：可以用顺序表或单链表存储，长  
    //度为定长  
    //栈的运算集为：  
    Stack(int s); //创建栈的实例  
    ~Stack(); //该实例消亡
```



```
void Push(ELEM item);    //item压入栈顶  
ELEM Pop();    //返回栈顶内容，并从栈顶弹出  
ELEM GetTop();    //返回栈顶内容，但不弹出  
void ClearStack();    //变为空栈  
Boolean IsEmpty();    //返回真，若栈已空  
Boolean IsFull();    //返回真，若栈已满  
};
```





# 算法及其特性

- 算法(**algorithms**)是为了求解问题而给出的指令序列
- 程序是算法的一种实现，计算机按照程序逐步执行算法，实现对问题的求解

# 算法及其特性

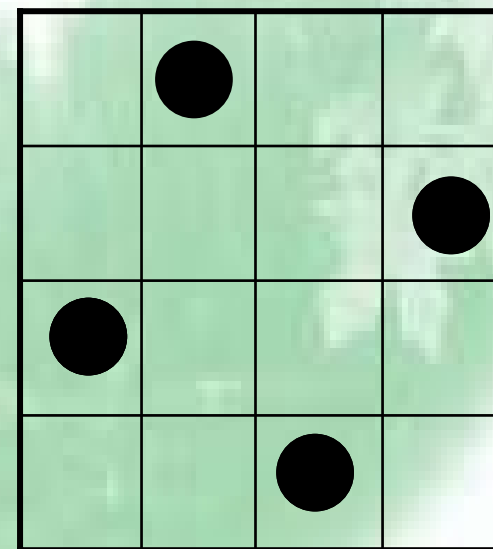
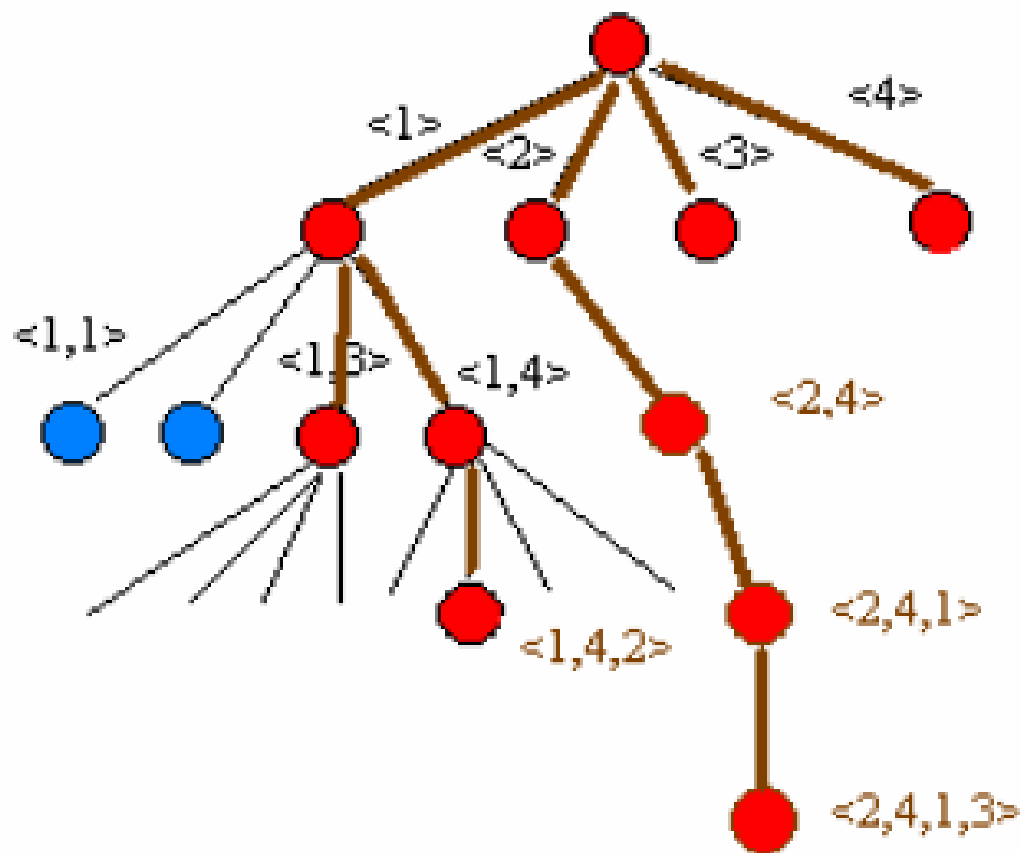
- 通用性
- 有效性
- 确定性
- 有穷性

# 典型算法

- 问题规模,  $n$
- 问题空间
  - 搜索(DFS, BFS)
- 穷举 (百钱买百鸡)
- 贪心(Huffman树)
- 递归
  - 回溯(八皇后)
  - 分治(二分法检索)
- 动态规划(最佳二叉排序树)

# 问题空间

## 四后问题



在棋盘放置4个不会互相攻击的后



# 计算复杂性和算法的效率

- **不可解问题**：能够被准确定义，但却不存在能够解决该问题的算法。
  - 停机问题
  - 程序正确性证明
  - 计算机病毒检测
- **难解问题**：存在着求解算法，但是算法的计算时间是**组合爆炸型**
  - 河内塔
- **NP问题**：在一台非确定机器上并行地以多项式时间求解
  - 最优巡游路径问题（货郎担问题）

# 顺序检索

- 用给定检索值与线性表中各结点的关键码逐个比较。
  - 若找到相当的结点，则检索成功；
  - 否则检索失败(找遍了仍找不到)。
- 存储：可以顺序、链接
- 排序要求：无

# 顺序检索算法

```
■ typedef struct elem {  
    int key;  
    datatype info;  
} ELEM;  
#define UNSUCCESSFUL -1  
// 在线性表array中顺序检索，查找关键码K  
// 返回i为关键码K在表中的下标  
// i值为-1表示检索失败  
int seqsrch(int K, ELEM* array, int n) {  
    int i = n-1;  
    while (i >= 0 && array[i].key != K) i--;  
    return i;  
}
```

# 顺序检索性能分析

## ■ 1) 检索成功

假设检索每个关键码是等概率的,  $P_i = 1/n$

$$\sum_{i=0}^{n-1} P_i \cdot (n-i) = \frac{1}{n} \sum_{i=0}^{n-1} (n-i)$$

$$= \sum_{i=1}^n i = \frac{n+1}{2}$$

## ■ 2) 检索失败

假设检索失败时都需要比较 $n+1$ 次（设置了一个监视哨）



# 二分法检索

对于已排序顺序线性表

- 设数组中间位置为 $mid$ ，相应的元素值记为 $k_{mid}$ 。
  - 如果 $k_{mid} = k$ ，那么检索工作就完成了。
  - 当 $k_{mid} > k$ 时，检索继续在前半部分进行。
  - 相反地，若 $k_{mid} < k$ ，就可以忽略 $mid$ 以前的那部分，检索继续在后半部分进行。
- $k_{mid} \neq k$ 的两种情况，都缩小了一半的检索范围。

# 二分法检索算法

```
// Return the position of an element in a sorted
// array of size n with value K. If none exist, return n.
int binary(int array[], int n, int K) {
    int l = -1;
    int r = n;          // l and r are beyond array bounds
    while (l+1 != r) {  // Stop when l and r meet
        int i = (l+r)/2; // Check middle
        if (K < array[i]) r = i;    // In left half
        if (K == array[i]) return i; // Found it
        if (K > array[i]) l = i;    // In right half
    }
    return n; // Search value not in array
}
```

## 二分法检索图示

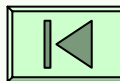
Diagram illustrating an array structure with indices 1 through 9 and corresponding values: 15, 17, 18, 22, 35, 51, 60, 88, 93. The array is divided into three sections: low (indices 1-4), mid (indices 5-7), and high (indices 8-9). Arrows indicate the current pointers: low points to index 1, mid points to index 5, and high points to index 9.

检索关键码18    low=1    high=9    K=18

第一次: mid=5; array[5]=35>18  
high=4; (low=1)

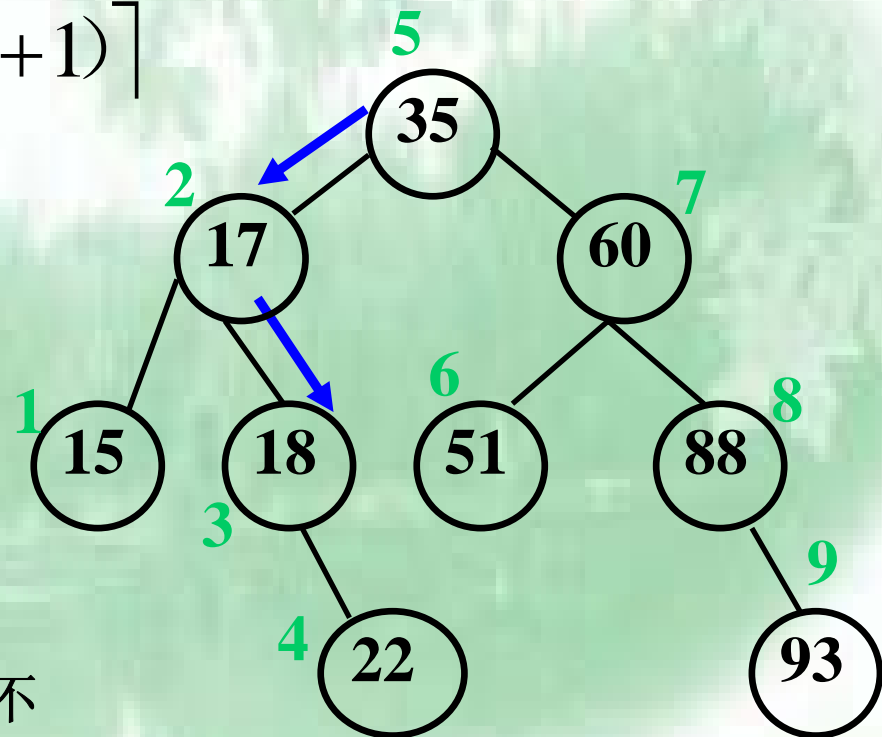
第二次: mid=2; array[2]=17<18  
low=3; (high=4)

第三次: `mid=3; array[3]=18=18`  
`mid=3; return 3`



# 二分法检索性能分析

- 最大检索长度为  $\lceil \log_2(n+1) \rceil$
- 失败的检索长度是  
或  $\lceil \log_2(n+1) \rceil$
- 在算法复杂性分析中
  - $\log n$  是以2为底的对数
  - 以其他数值为底，算法量级不变





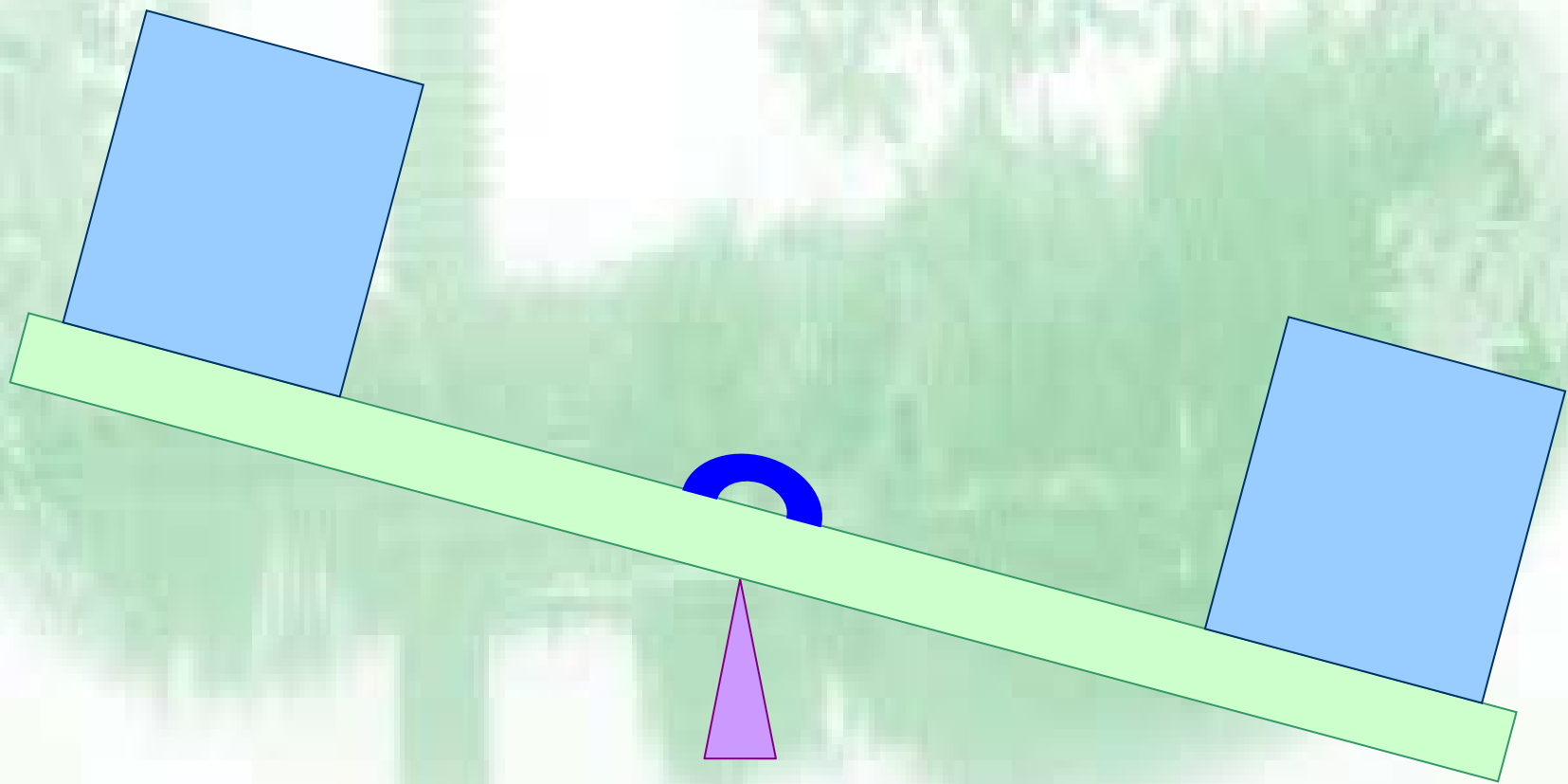
# 算法的度量

- 算法设计的两个目标（除正确性外）
  - 易读、易编码和调试（软件工程）
  - 充分利用计算机资源（算法和数据结构）
- 评估方法
  - 实验（运行程序）
  - 渐进分析
  - 关键资源
  - 影响运行时间的因素
  - 往往与问题的输入规模 $n$ 有关

# 时间/空间权衡

- 数据结构
  - 一定的空间来存储它的每一个数据项
  - 一定的时间来执行单个基本操作
- 代价和效益
  - 空间和时间的限制，时空权衡
  - 程序设计工作量

# 空间时间权衡



# 大O 表示法及其运算规则

- 称某程序的时间(或空间)代价为  $O(f(n))$ 
  - 存在正常数  $c$  和  $n_0$ , 当问题的规模  $n \geq n_0$  后,
  - 该算法的时间(或空间)代价  $T(n) \leq c \cdot f(n)$ ,
- 也称该算法的时间(或空间)增长率为  $f(n)$ 
  - 对于所有足够大的输入数据集 (  $n \geq n_0$  )
  - 算法运行步骤数为  $f(n)$  量级



# 大O 表示法的运算规则

## ■ 单位时间

- 简单布尔或算术运算
- 赋值
- 简单I/O
- 函数返回

## ■ 加法规则: $f_1(n) + f_2(n) = O(\max(f_1(n), f_2(n)))$

- if结构, switch结构

## ■ 乘法规则: $f_1(n) \cdot f_2(n) = O(f_1(n) \cdot f_2(n))$

- for, while, do-while结构

# 时间复杂性

- **sum = 0;**  
**for (i=1; i<=n; i++)**  
**for (j=1; j<=n; j++)**  
**sum++;**
- **$T(n) = c_1 + c_2n^2 \approx O(n^2)$**

# 大 $\Theta$ 表示法及其分析规则

- 大 $\Theta$ 表示法是一种渐进估计，是大 $O$ 表示的扩展
- 对资源开销函数 $T(n)$ ，及其渐进估计式 $F(n)$
- 称 $F(n)$ 为 $T(n)$ 的一个渐进 $\Theta$ 估计，当且仅当
  - 存在正常数 $C_1, C_2$ ，以及正整数 $n_0$
  - 对于任意的正整数 $n > n_0$ ，下面两不等式同时成立：  
 $|T(n)| > C_1 \cdot F(n)$  和  $|T(n)| < C_2 \cdot F(n)$

# 大 $\Theta$ 表示法示例



假定 $T(n) = 100 \cdot n^2 + 5 \cdot n + 500$ ,

令 $F(n) = n^2$ , 这时存在正常数

$C_1 = 100$ ,  $C_2 = 105$ , 以及  $n_0 = 10$

使得当 $n > n_0$ 时,

$|T(n)| > C_1 \cdot F(n)$  和  $|T(n)| < C_2 \cdot F(n)$

同时成立。

上述不等式可以用算术计算验证, 由此说明 $T(n)$ 的 $\Theta$ 估计式等于 $F(n) = n^2$

# 算法效率的基本问题——权衡

对于给定的一类问题

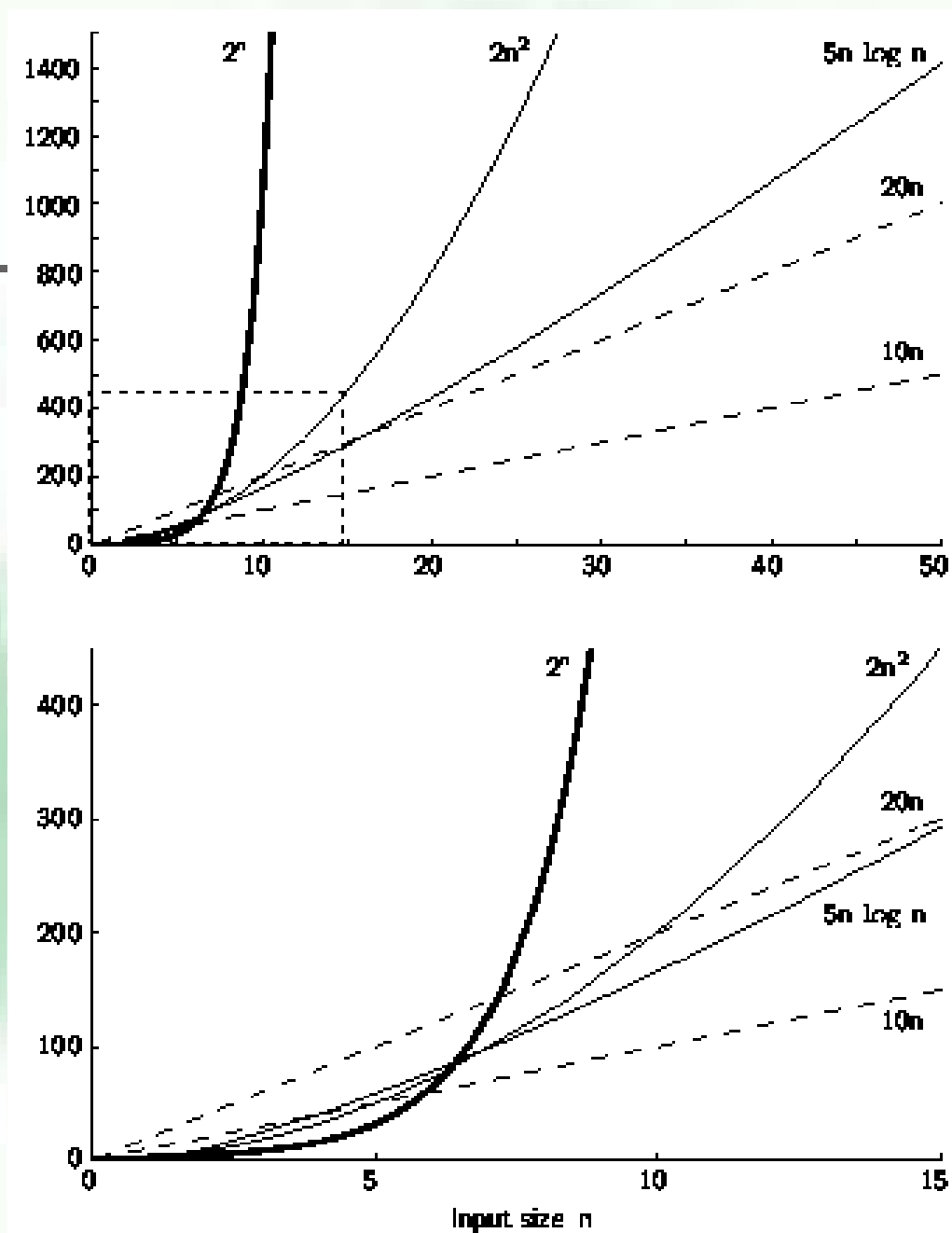
- 算法需要多少存储空间和时间？
- 最好算法的最坏情况是什么？
- 平均来说，算法的运行好到何种程度？
- 算法一般化到何种程度？
- 什么情况下，最好的算法是什么？

## 算法分析技术

# 各增长率函数值的比较

n	log n	n	n log n	$n^2$	$n^3$	$2^n$
1	0	1	0	1	1	2.E+00
10	3	10	33	100	1000	1.E+03
20	4	20	86	400	8000	1.E+06
30	5	30	147	900	27000	1.E+09
40	5	40	213	1600	64000	1.E+12
50	6	50	282	2500	125000	1.E+15
60	6	60	354	3600	216000	1.E+18
70	6	70	429	4900	343000	1.E+21
80	6	80	506	6400	512000	1.E+24
90	6	90	584	8100	729000	1.E+27
100	7	100	664	10000	1000000	1.E+30

# 增长率函数曲线



# 运行时间估计

- 例：假设CPU每秒处理 $10^6$ 个指令，对于输入规模为 $n = 10^8$ 的问题，时间代价为 $T(n) = 2n^2$ 的算法要运行多长时间？
  - 操作次数为 $T(n) = T(10^8) = 2 \times (10^8)^2 = 2 \times 10^{16}$
  - 运行时间为 $2 \times 10^{16} / 10^6 = 2 \times 10^{10}$ 秒
  - 每天有86,400秒，因此需要231,480 天 (634 年)



# 运行时间估计（续）

- 例：假设CPU每秒处理 $10^6$ 个指令，对于输入规模为 $n = 10^8$ 的问题，时间代价为 $T(n) = n/\log n$ 的算法要运行多长时间？
  - $\log n$  在数据结构与算法中一般指 $\log_2 n$ 
    - 其他的底可以换算  $\log_a n = \log_2 n / \log_2 a$
  - 操作次数为 $T(n) = T(10^8) = 10^8 \times \log 10^8 = 2.66 \times 10^9$
  - 运行时间为  $2.66 \times 10^9 / 10^6 = 2.66 \times 10^3$  秒，即**44.33分钟**

# 规定时间内能解决的问题规模

- 假设CPU每秒处理 $10^6$ 个指令，则每小时能够解决的最大问题规模
  - $T(n)/10^6 \leq 3600$
- 对 $T(n) = 2n^2$ ，
  - 即  $2n^2 \leq 3600 \times 10^6$
  - $n \leq 42,426$
- $T(n) = n \log n$ 
  - 即  $n \log n \leq 3600 \times 10^6$
  - $n \leq 133,000,000$

# 快10倍的计算机所能解决的问题规模?



$T(n)$	$n$	$n'$	Change	$n'/n$
$10n$	1,000	10,000	$n' = 10n$	10
$20n$	500	5,000	$n' = 10n$	10
$5n \log n$	250	1,842	$\sqrt{10} n < n' < 10n$	7.37
$2n^2$	70	223	$n' = \sqrt{10}n$	3.16
$2^n$	13	16	$n' = n + 3$	-----

# 最坏、最好和平均情况

- 对于不同的输入情况，算法的时间代价不一样
  - 例如，在一个数组中查找元素K
- 往往分为最坏、最好、平均情况分析



# 数据结构的选择和评价

- 问题模型，数据类型，数据间逻辑关系
- 可扩展性
- 算法的时空开销的优劣



# 总结

- 数据结构的地位与重要意义
- 数据结构的主要内容
- 抽象数据类型的概念
- 算法及其特点
- 算法的有效性度量
- 数据结构的选择

# 谢谢!

---

祝大家学习进步!

<http://db.pku.edu.cn/mzhang/DS/>

张铭: [mzhang@db.pku.edu.cn](mailto:mzhang@db.pku.edu.cn)