

数据结构与算法

第一章 绪论

2022-2-22

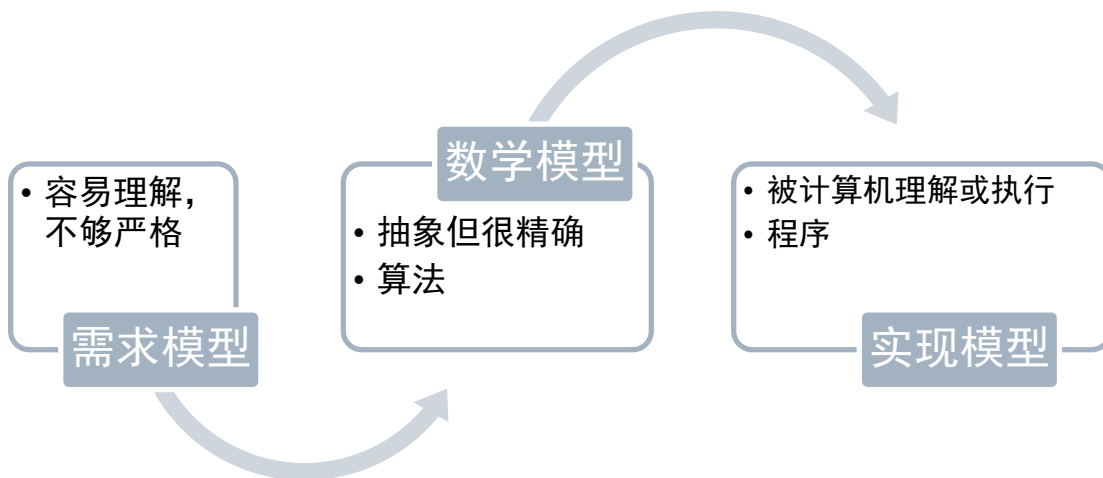
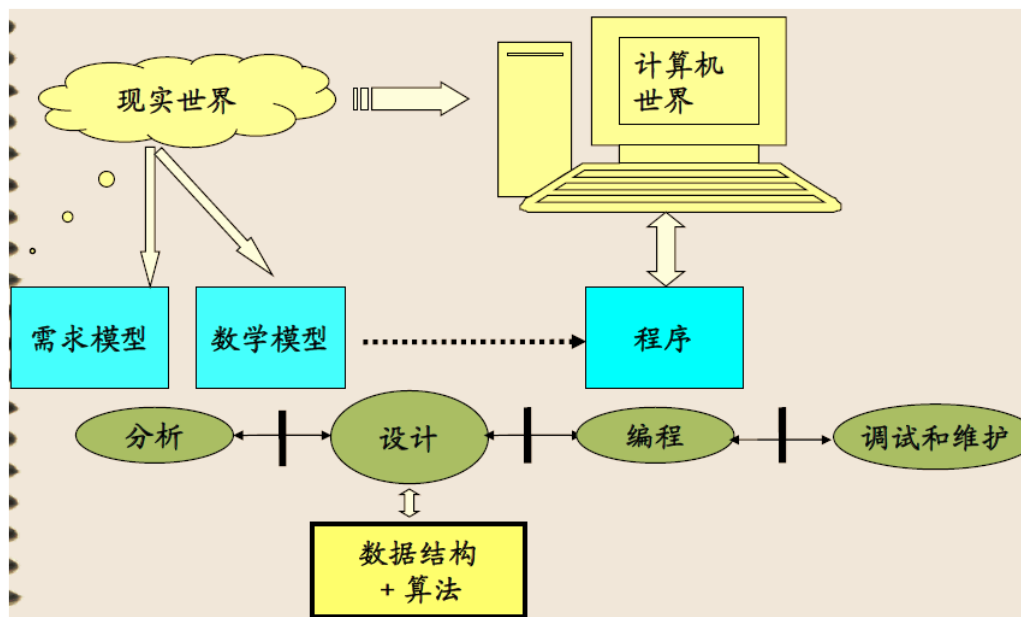
张史梁

slzhang.jdl@pku.edu.cn

第一章 绪论

- 数据结构在问题求解中的作用
- 数据类型与数据结构
- 算法特性与算法评价
- 总结

计算机求解问题

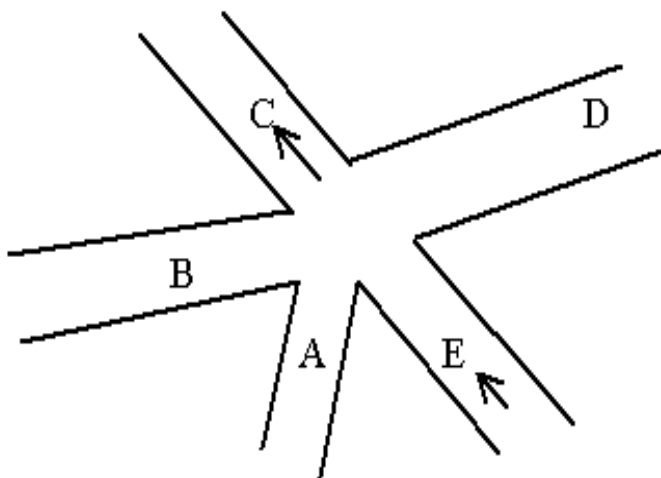


计算机求解问题的过程

- **分析阶段**：弄清所要解决的问题是什么，并用一种语言清楚地描述出来。
- **设计阶段**：建立程序系统的结构，重点是**数据结构**的设计和**算法**的设计。
- **编码阶段**：采用适当的程序设计语言，编写出可执行的程序。
- **测试和维护**：发现和排除在前几个阶段中产生的错误，经测试通过的程序便可投入运行，在运行过程中还可能发现隐含的错误和问题。

多叉路口交通信号灯的管理:

(一) 问题分析: 首先需要分析一下所有车辆的行驶路线的冲突问题。这个问题可以归结为对车辆的可能行驶方向作某种分组, 对分组的要求是使任一个组中各个方向行驶的车辆可以同时安全行驶而不发生碰撞。



(C和E单行, 其它双行)

图 1.1 一个交叉路口的模型

根据这个路口的实际情况可以确定13个可能通行方向:

A→B, A→C, A→D,
B→A, B→C, B→D,
D→A, D→B, D→C,
E→A, E→B, E→C,
E→D。

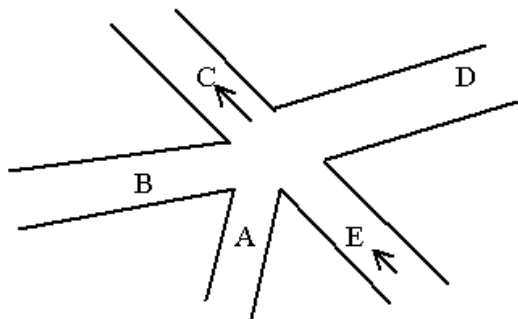


图 1.1 一个交叉路口的模型

为了叙述方便，我们下面把 $A \rightarrow B$ 简写成 AB ，并且用一个小椭圆把它框起来，在不能同时行驶的路线间画一条连线（表示它们互相冲突），便可以得到下图

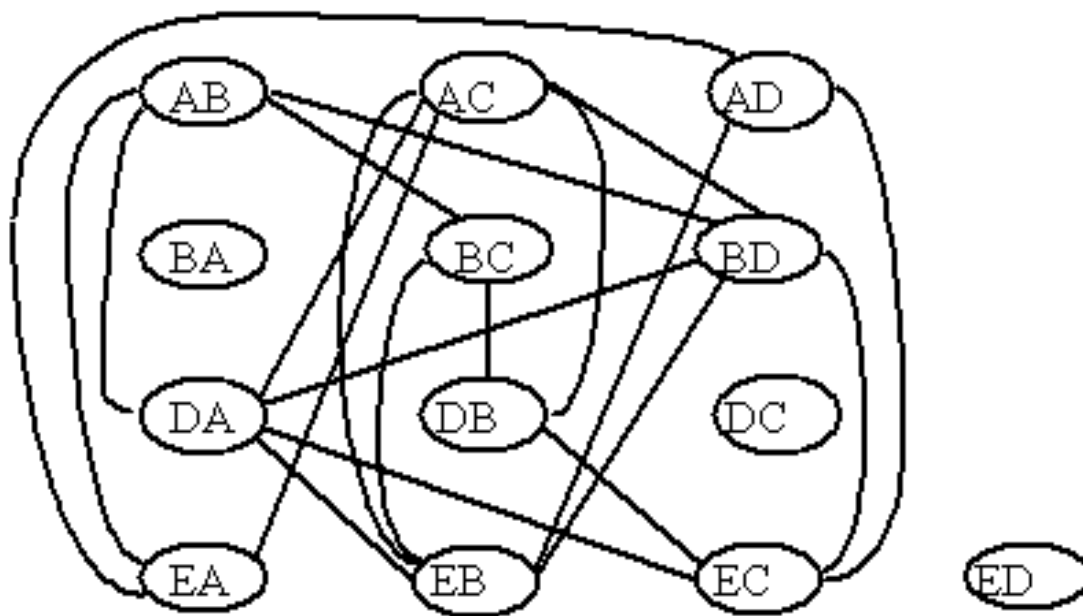


图 1.2 交叉路口的图式模型

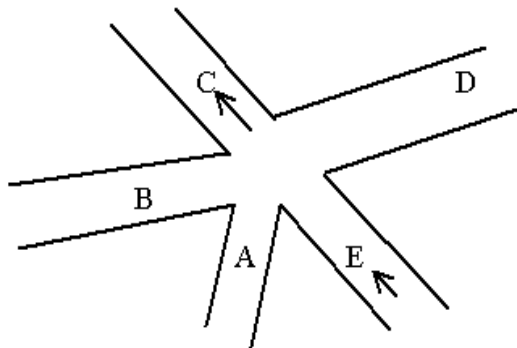


图 1.1 一个交叉路口的模型

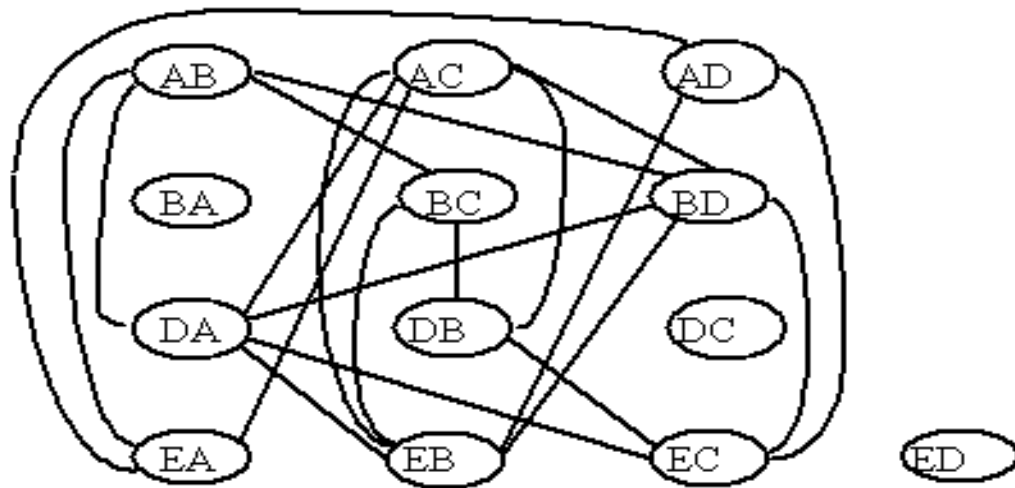


图 1.2 交叉路口的图式模型

这样做就把要解决的问题借助图的模型变成了另一个抽象问题：要求将图 1.2 中的结点分组，使有线相连(互相冲突)的结点不在同一个组里。

- 可行解：仅满足安全性的普通解。
- 最优解：分组数最少的解。
- 次优解：分组数接近最优解的可行解

着色问题

- 如果把上图中的一个结点理解为一个国家，结点之间的连线看作两国有共同边界，上述问题就变成著名的“着色问题”
- 即求出要几种颜色可将图中所有国家着色，使得任意两个相邻的国家颜色都不相同。

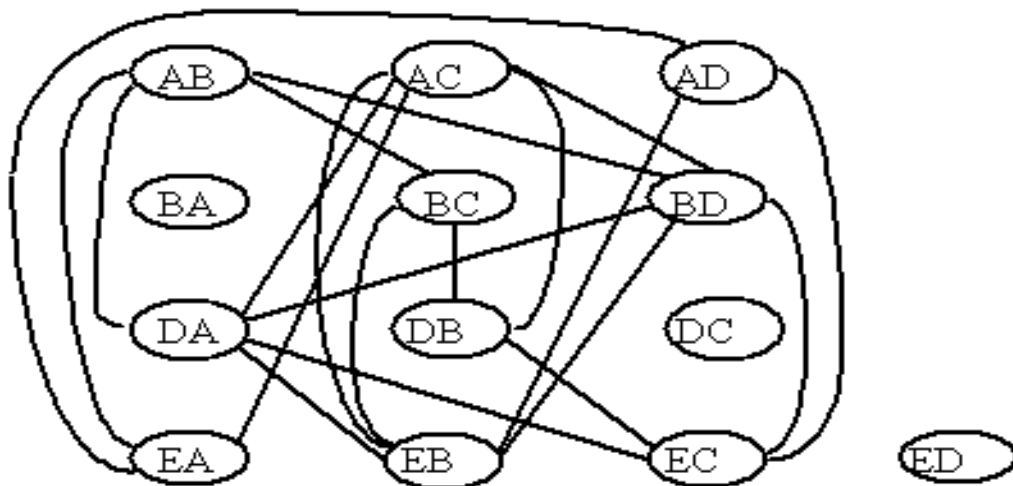
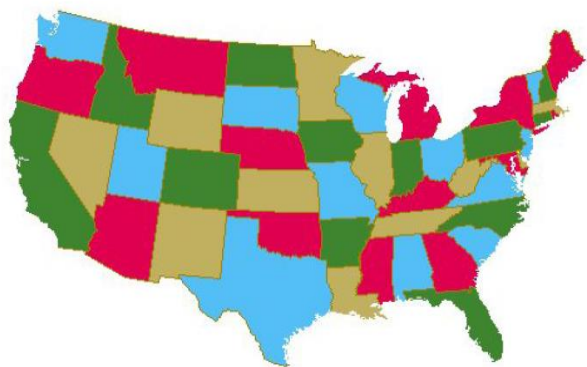


图 1.2 交叉路口的图式模型

多叉路口交通信号灯的管理:

(二) 算法设计

□ 穷举法:

- 从分为1、2、3...组开始考察, 逐个列举出所有可能的分组方案, 检查这样的分组方案是否满足要求。
- 首先满足要求的分组, 自然是问题的最优解。
 - 将所有结点放在一个组里是否可行(一种颜色), 考察组内结点是否有连线, 没有则成功;
 - 否则放在两个组里是否可行, 穷举出所有可能的分组方案, 以此类推三组、四组...
- 对结点少的问题(称为“规模小的”问题)还可以用
- 对规模大的问题, 由于求解时间会随着实际问题规模的增长而指数性上升, 使计算机无法承受。

多叉路口交通信号灯的管理:

□ 贪心法

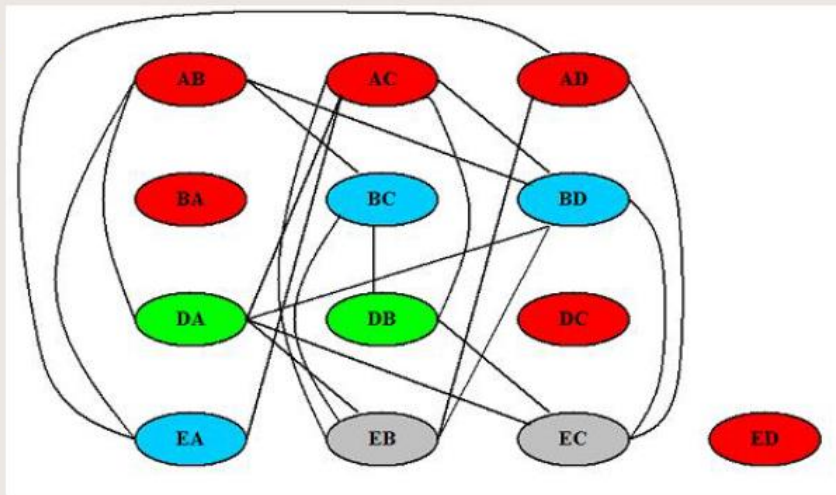
- 先用一种颜色给尽可能多的结点上色
- 然后用另一种颜色在未着色结点中给尽可能多的结点上色
- 如此反复直到所有结点都着色为止。

红色: AB AC AD BA DC ED

蓝色: BC BD EA

绿色: DA DB

灰色: EB EC



多叉路口交通信号灯的管理:

(三) 程序实现

□ 数据类型选择

- 使用什么数据类型来表示地图，以及使用什么样的数据类型表示一组国家等。
- 如：使用一个图结构表示地图；使用国名（图中结点名）的集合表示国家的分组。

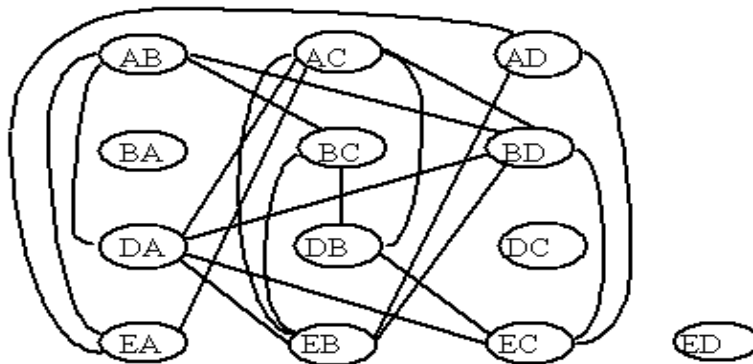


图 1.2 交叉路口的图式模型

程序描述

- 设需要着色的图是G，集合V1包括所有未被着色的结点，着色开始时V1是G所有结点集合。
- NEW表示已确定可以用新颜色着色的结点集合。
- 贪心法的伪码描述：

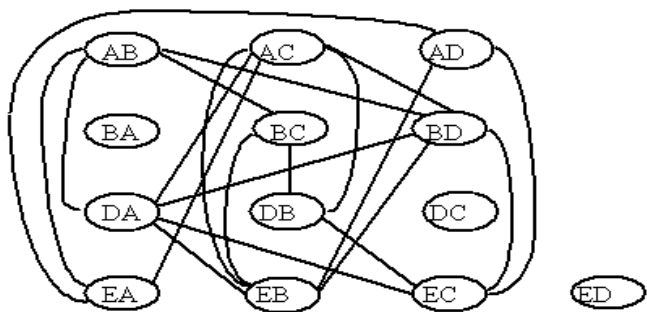


图 1.2 交叉路口的图式模型

```
// 找到一组可以着相同颜色的节点集合  
置NEW为空集合;  
for 每个  $v \in V1$  do  
    if  $v$ 与NEW中所有结点间都没有边  
        从V1中去掉 $v$ ;  
        将 $v$ 加入NEW;
```

- 着色程序可以反复调用这段伪码，直到V1为空
- 每次调用选择一种新颜色，这段伪码执行的次数就是需要的不同颜色个数。

编程

```
int colorUp(Graph G) //输出所需的颜色数量和每种颜色对应的节点
{
    int color = 0;
    V1 = G.V;
    while(!isEmpty(V1)) //如果V1非空
    {
        emptySet(NEW); //清空NEW
        while (v ∈ V1 && notAdjacentWithSet(NEW, v, G))
        {
            addToSet(NEW, v); // v加入NEW
            removeFromSet(V1, v); //将v从V1中删除
        }
        printSet(NEW); // 输出当前NEW中的节点
        ++color; // 所需颜色数量+1
    }
    return (color);
}
```

多叉路口交通信号灯的管理：

（四）测试阶段

- 输入不同类型的值，测试结果是否出错
- 重点考察**边界值**（输入或输出范围的边界）

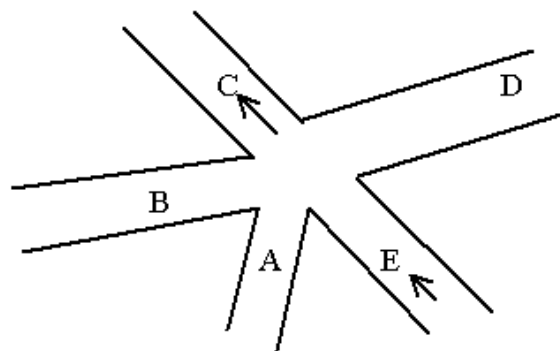


图 1.1 一个交叉路口的模型

问题求解中数据结构的作用

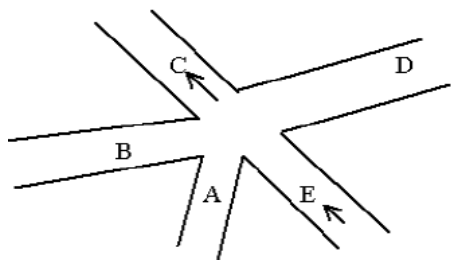


图 1.1 一个交叉路口的模型

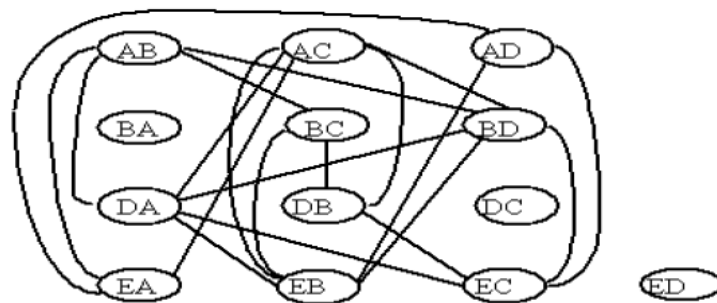


图 1.2 交叉路口的图式模型

□ 用计算机求解问题

- 首先**分析问题**的需求，抽象出需求（问题）模型，
- 然后**设计适当的数据结构和有关的算法**，
- 最后采用计算机编程语言精确地描述所需要的数据和算法，实现程序。

问题求解中数据结构的作用

- 常见的计算机语言通常只提供基本的数据类型（整数、实数等）和一些数据构造手段（如数组、结构、指针等），
- 复杂数据结构（图、树、集合等）的设计以及相应的操作必须由用户自己实现。
- 在数据结构确定以后，可以进一步根据设计的数据结构进行算法精化。

第一章 绪论

- 数据结构在问题求解中的作用
- 数据类型与数据结构
- 算法特性与算法评价
- 总结

基本数据类型

- 整数类型(integer)
- 实数类型(real)
- 布尔类型(boolean)
 - C++语言中0表示false，非0表示true
 - 也支持false，true 保留字
- 字符类型(char)
 - ASCII用单个字节表示字符
 - 汉字符号需用2个字节编码

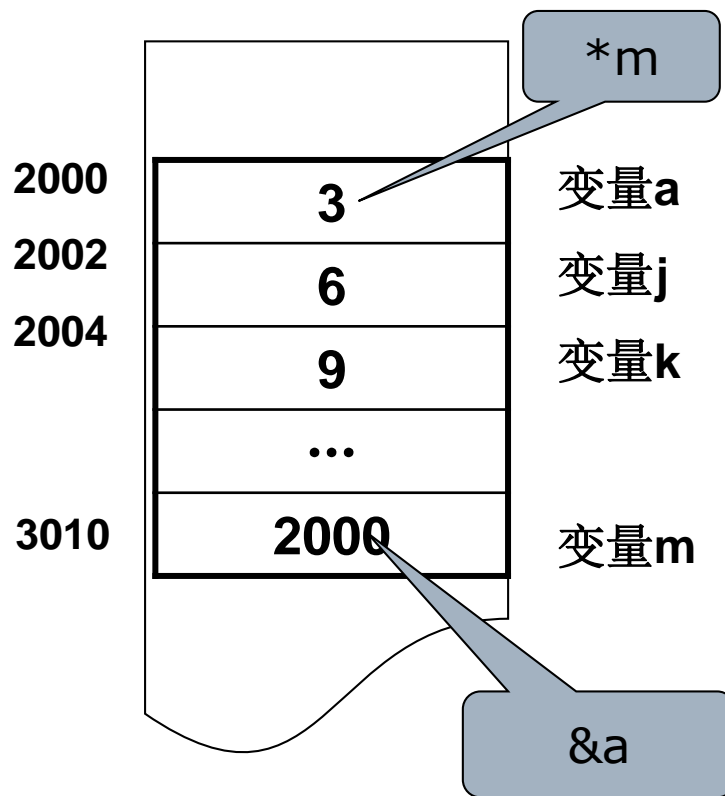
指针类型

□ 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
- 64 bit的机器, 8个字节

□ 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



复合数据类型

- 基本数据类型/复合类型组成的复杂结构
- 例如,
 - 数组: `int A[100];`
 - 结构: `typedef struct{ } B;`
 - 类: `class C { };`

基本概念和术语

□ 数据（Data）：

- 指能够被计算机识别、存储和加工处理的信息载体。
- 在计算机科学中是所有能输入到计算机中并能被计算机程序处理的符号的总称。

□ 数据结构：

- 计算机中表示（存储）的、具有一定逻辑关系和行为特征的一组数据。
- 涉及数据之间的逻辑关系、数据在计算机中的存储表示和在这种结构上一组能执行的操作（运算）

数据结构三要素

□ 逻辑结构:

- 定义了结构中的基本元素之间的相互关系。
- 如：图关系，树关系

□ 存储结构:

- 给出了结构中的基本元素及其之间关系的存储方式，包括元素的表示和关系的表示。
- 如：图、树在内存中存储图

□ 数据的运算:

- 这个结构具有的**行为特征**，体现为在存储结构上具体实现的算法。
- 如：判断两个节点是否在图中相连，树上节点间的从属关系

逻辑结构

□ 逻辑结构(Logical Structure)

- 具体问题的数学抽象，反映事物组成和逻辑关系

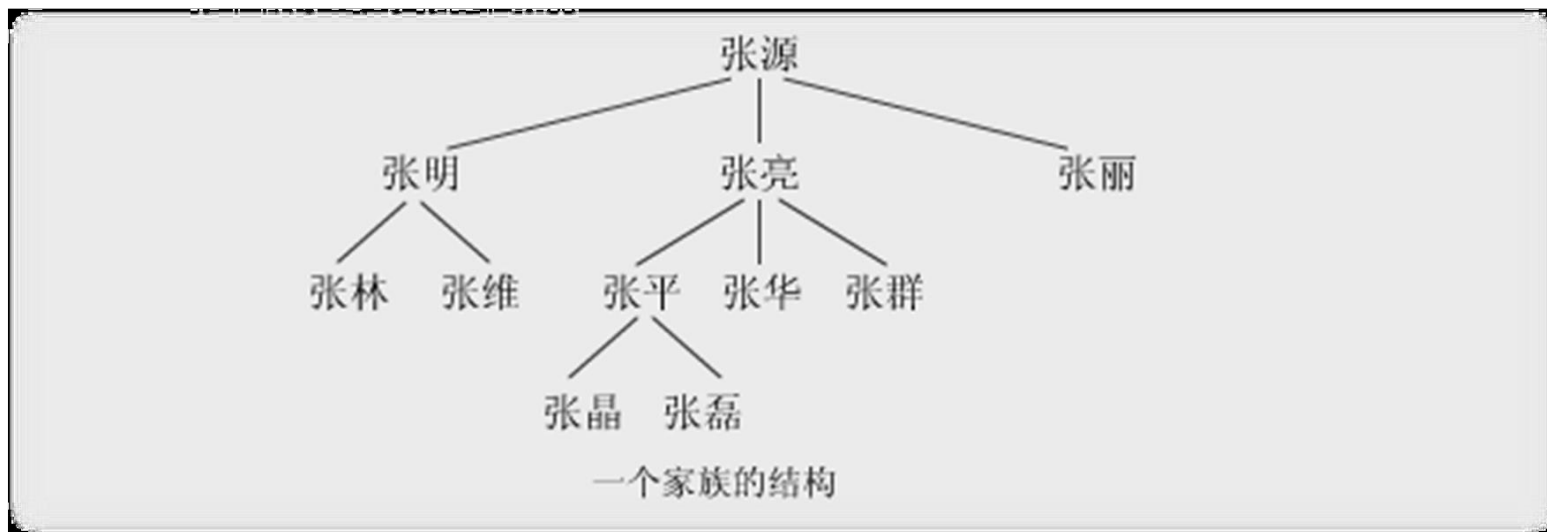
□ 逻辑结构的表示

- 用一组数据（表示为结点集合 K ），及数据间的二元关系（关系集合 R ）表示： (K, R)
 - ✓ K 由有限个结点组成，代表一组有明确结构的数据集
 - ✓ R 是定义在集合 K 上的一组关系，每个关系 $r \in R$ 都是 $K \times K$ 上的二元关系，描述结点间的逻辑关系
 - ✓ 若 $\langle k_1, k_2 \rangle \in R$, 则称 k_1 为 k_2 的前驱, k_2 为 k_1 的后继。没有前驱的结点为开始结点, 没有后继的结点为终端结点。

举例

□ 家族成员-树状数据结构

- 每个成员个体作为结点（实体），全部成员组成结点集 K
- 家族中各类亲属关系就是一组关系 R
 - 其中如父子关系 r 、兄弟关系 r^* 、和妯娌关系 r' 等



节点的类型

□ 基本数据类型

- 整数类型(integer)、实数类型(real)、字符类型(char)和指针类型(pointer)

□ 复合数据类型

- 由基本数据类型组合而成的数据结构类型
—如数组、结构类型等
- 复合数据类型本身，又可参与定义更为复杂的结点类型

□ 结点的类型[可以根据应用的需要来灵活定义](#)

结构（关系）类型

- 逻辑结构（ \mathbf{K}, \mathbf{R} ）的分类，讨论重点在关系集 \mathbf{R} 上
- 用 \mathbf{R} 的性质来刻画数据结构的特点，并进行分类
 - 集合结构（set structure）， \mathbf{R} 为空
 - 线性结构（linear structure）
 - 树型结构（tree structure）
 - 图结构（graph structure）

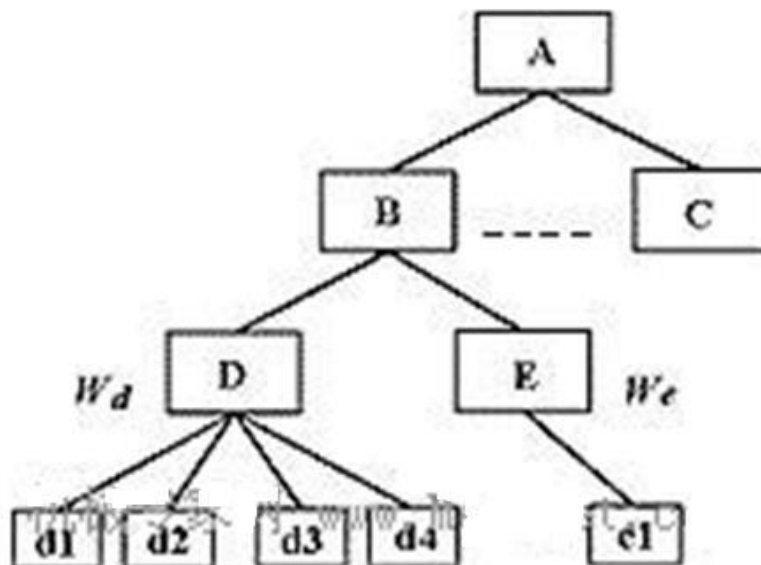
线性结构

- K中每个结点最多只有一个前驱和一个后继
- 数据元素之间仅存在一对一的关系（唯一后继、唯一前驱）
- 线性表，栈，队列

0	1	2	3	4	5	...	N-2	N-1	N
----------	----------	----------	----------	----------	----------	------------	------------	------------	----------

树形结构

- 亦称层次结构，每个结点可有多于1个‘直接后继’，但只有唯一的‘直接前驱’
- 最高层结点称为根（**root**）结点，无父结点
- 二叉树、三叉树...



图型结构

□ 图结构亦称网络结构

- 交通网、因特网、社会网络等

□ K中结点的前驱、后继结点的个数都不作限制

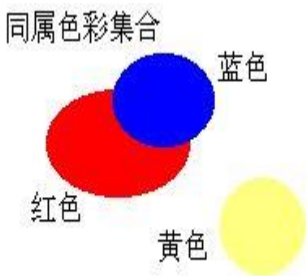
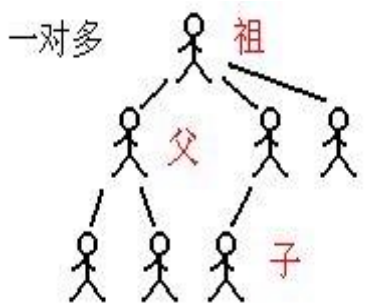

□ 树型结构和图型结构的区别是

- 每个结点是否仅仅属于一个直接前驱

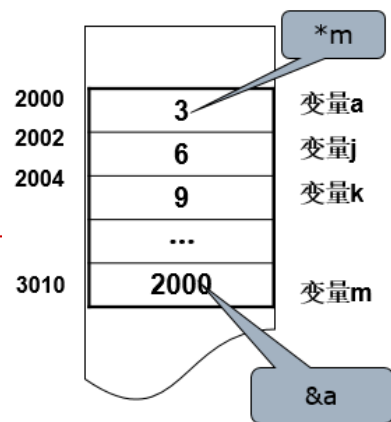
■ 线性结构和树型结构的区别是

- 每个结点是否仅仅有一个直接后继

逻辑结构

	集合	线性结构	树形结构	图状或网状结构
特征	元素间为松散的关系	元素间为严格的一对一关系	元素间为严格的一对多关系	元素间为多对多关系
示例	<p>同属色彩集合</p> 	<p>如银行排队叫号时队列中的各元素</p>	<p>一对多</p> 	<p>多对多</p> 

存储结构



□ 数据的存储结构（Storage Structure）

- 亦称物理结构, 是逻辑结构在计算机中的物理存储表示

□ 计算机主存储器

- **空间相邻**: 存储空间是一种具有**非负整数**地址编码的、**空间相邻的单元**集合, 其基本存储单元是**字节**
- **随机访问**: 计算机指令具有**按地址随机访问**存储空间内任意单元的能力, 访问不同地址所需时间相同

□ 四种基本存储映射方法

- 顺序、链接、索引、散列

1、顺序方法

□ 顺序存储

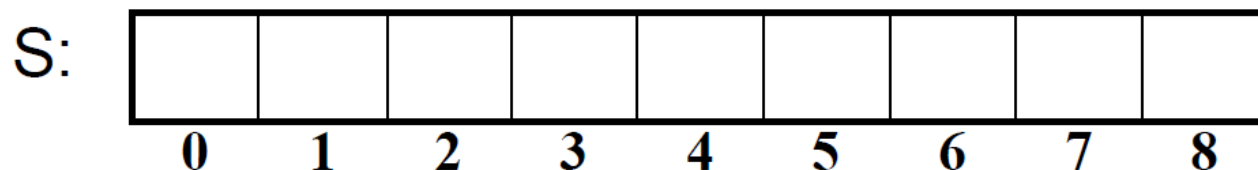
- 它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现
- 数组是顺序存储法的具体实例

□ 顺序存储是紧凑存储结构

- 紧凑指存储空间除存储有用数据外，不存储其他附加信息
- 存储密度：存储结构所存储‘有用数据’和该结构（包括附加信息）整个存储空间大小之比

举例

顺序存储结构：一维数组



二维数组：支持整数编码访问

M_{00}	M_{01}	M_{02}
M_{10}	M_{11}	M_{12}
M_{20}	M_{21}	M_{22}

$$M[i][j] = M[0][0] + (k*i + j) * (\text{元素尺寸})$$

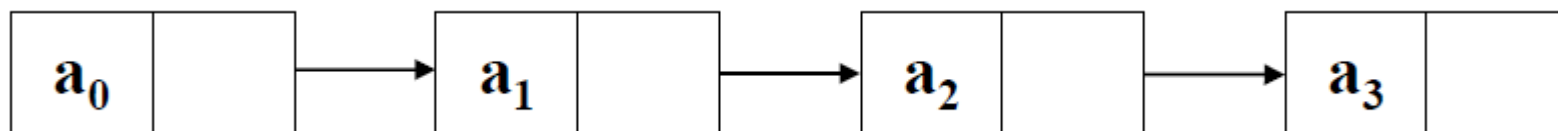
2、链接方法

□ 链接法

- 利用存储结构中**指针指向**来表示两个结点间逻辑关系
- 它**不要求逻辑上相邻的结点在物理位置上亦相邻**，结点间的逻辑关系是由附加的指针字段表示的。

□ **可以表达任意的逻辑关系 r** ，将数据结点分为两部分：

- **数据字段**：存放结点本身的数据
- **指针字段**：存放指针，链接到某个后继结点，指向存储单元的开始地址
- 多个相关结点的依次链接就会形成**链表**



2、 链接方法

□ 优点： 增删容易

- 顺序方法对于经常增、删结点情形，往往遇到困难
- 链接方法结合动态存储可以解决这些复杂的问题

□ 缺点： 定位困难

- 访问结点必须知道该结点的指针
- 否则需要沿着链接指针逐个搜索，花费时间较大
- 存储额外的指针

3、索引方法

目 录

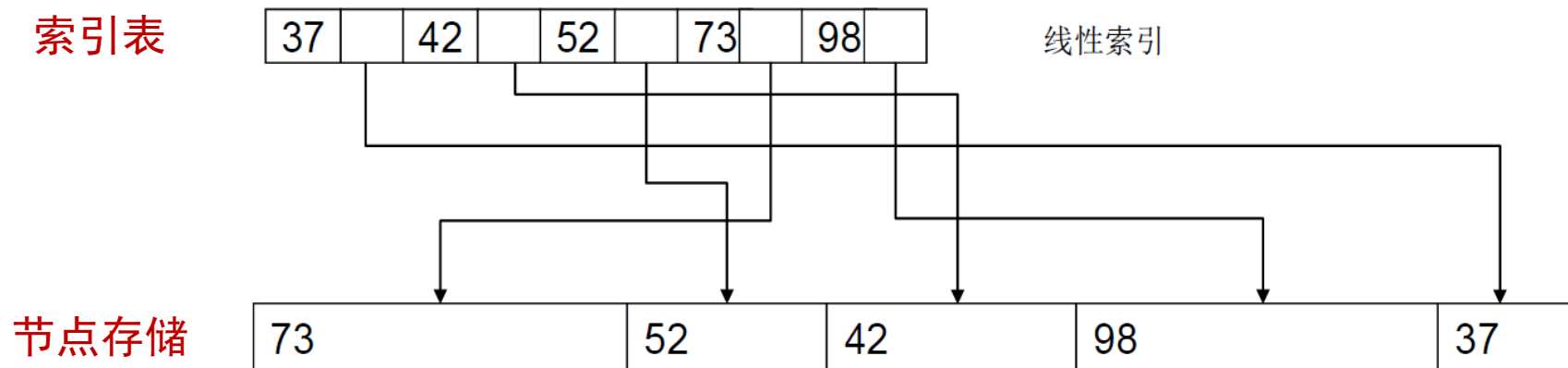
关键字

数据结构与算法实验教程	i
内容提要	ii
前 言	iii
目 录	iv
第 1 章 数据结构与算法教学实施方案	1
1.1 “数据结构与算法”的理论体系	1
1.1.1 课程的基本定位	2
1.1.2 知识体系	3
1.2 “数据结构与算法”学习重点	6
1.2.1 概论	6
1.2.2 线性表	7
1.2.3 栈与队列	9
1.2.4 字符串	10
1.2.5 二叉树	11
1.2.6 树	13
1.2.7 图	14
1.2.8 内排序	16
1.2.9 文件与外排序	18
1.2.10 检索	19

数据
存储区

3、索引方法

- 除建立存储结点信息外，还建立**附加的索引表来标识结点的地址**。
- 索引方法是要建造一个由整数域 Z 映射到存储地址域 D 的函数/索引表 $Y: Z \rightarrow D$ ，把结点的整数索引值 $z \in Z$ 映射到结点的存储地址 $d \in D$ 。 Y 称为索引函数/索引表

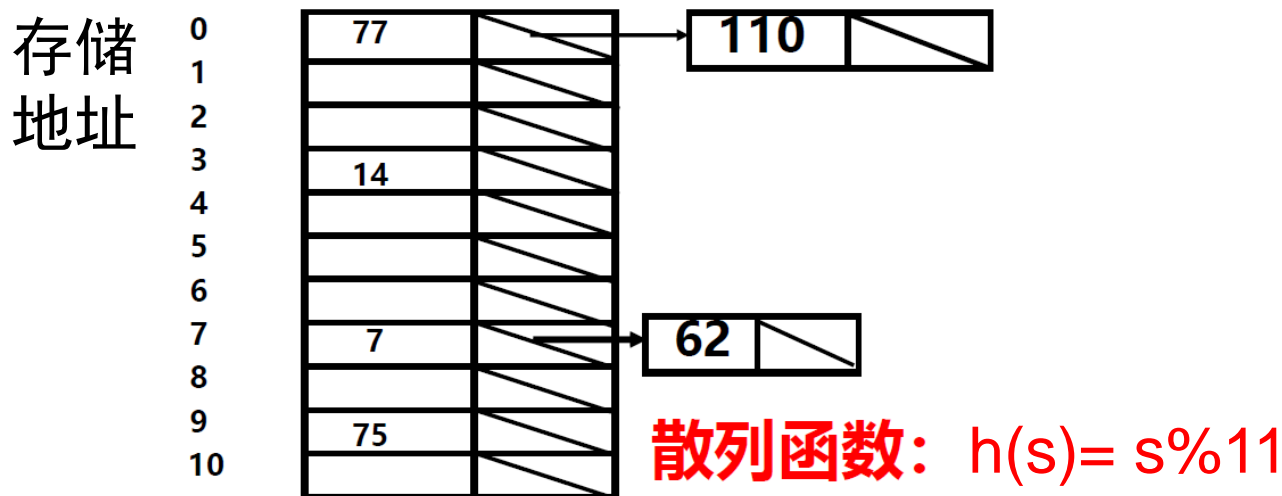


4、散列方法 Hashing

- 就是根据结点的关键字直接计算出该结点的存储地址。
- 散列是索引方法的扩展，常用于海量数据索引
- **散列函数**：将关键码 s 映射到非负整数 z

$$h: S \rightarrow Z$$

对任意的 $s \in S$, 散列函数 $h(s)=z, z \in Z$



数据的运算

- 定义在逻辑结构上的一系列操作以及这些操作在存储结构上的实现；
 - 常用的运算：检索、插入、删除、定位、修改、排序等；
 - 数据的运算是定义在逻辑结构上的，而具体实现是基于存储结构
-

本课程中讨论的各种数据结构皆按照三个方面进行：

- 逻辑定义（逻辑结构）
- 存储结构
- 各种运算的实现

小节

□ 数据结构

➤ 逻辑结构

– 节点

- 基本数据类型

整数、实数、布尔、字符，
指针

- 复合数据类型

– 结构（关系）

- 线性结构
- 树型结构
- 图型结构

➤ 存储结构

– 顺序的方法

– 链接的方法

– 索引的方法

– 散列的方法

➤ 数据运算

第一章 绪论

- 数据结构在问题求解中的作用
- 数据类型与数据结构
- 算法特性与算法评价
- 总结

算法的概念

- ❑ 算法是对特定问题求解方法和步骤的一种描述，
- ❑ 它是指令的一组有限序列，其中每个指令表示一个或多个操作。
- ❑ 选择了一个恰当的数据结构，还需要一个好的算法，再进行编程实现，才能更好地解决问题。

数据结构+算法=?

- Pascal之父、结构化程序设计先驱Niklaus Wirth
最著名的一本书，叫作
《数据结构+ 算法= 程序设计》
- 程序设计的实质
 - 为计算机处理问题编制一组“指令”
- 需要解决两个问题：算法和数据结构
 - **数据结构** = 问题的数学模型（表示数据）
 - **算法** = 处理问题的策略（操作数据）

数据结构+算法=程序

- 表达了数据结构与算法的内在联系，及其在程序中的地位
 - 程序：在特定逻辑结构和存储表示的数据基础上，算法具体实现的描述
 - 数据结构与算法是程序设计中相辅相成、不可分割的两个方面

算法的性质

□ 输入

□ 输出

□ 有穷性

- 算法的执行必须在有限步内结束
- 换句话说，算法不能含有死循环

□ 确定性

- 算法的每一步有确切的定义
- 算法描述中的下一步应执行的步骤必须明确

□ 可行性

- 算法的每个动作，都可以由机器或人准确的完成
- 算法是一个工作序列，其中每一步都是力所能及的一个动作

描述算法的工具

- 自然语言，如施工步骤
- 数学语言，如数学解题步骤
- 约定的符号描述：流程图、伪代码
- 计算机高级语言描述：C、Pascal、C++、Java...

算法的分类

- 穷举法[万能低效]
- 贪心法[分步完成，局部最优试图得到整体最优，着色问题算法中有举例分析]
- 分治法[问题规模缩小，分而治之。如折半检索等，快速排序]
- 动态规划[问题分解（缩小规模），得到各个分解结果，再自底往上求最后结果]
- 回溯法[深度优先搜索]
- 分支界限法[广度优先搜索]
- ...

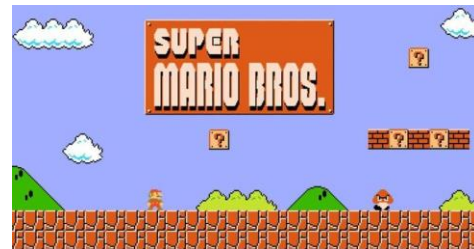
算法的设计要求

□ 如何设计一个好的算法？

- **正确性**：经得起一切输入数据的考验；
- **可读性**：让人阅读得懂，便于交流。注意注释的作用；
- **健壮性**：输入数据错误时，进行必要的处理。不能得到莫名其妙的结果；
- **高效率**：执行时间尽可能地短，对存储要求尽可能地少，既省时又节省空间。

算法的评价

- 同一个问题，可以设计出多个算法，这些算法的优劣如何衡量？
 - **时间效率**：执行算法耗费的时间；
 - **空间效率**：运行算法需要耗费的存储空间，其中主要考虑辅助空间；
 - 理解、阅读、编写、调试的难易等。
- 怎样评价算法的空间代价和时间代价？
 - 事后统计法
 - 事前估算法



40KB

如何评价跑步速度



- 事后测量法
- 事前估计法：根据之前的体能指标

两种统计方法的缺点：

□ 事后统计法

- 必须先运行程序
- 统计量依赖于计算机软硬件等环境因素

□ 事前估算法 - 影响因素比较多

- 算法的策略
- 问题的规模 (n)
- 程序语言
- 编译程序所产生的机器代码的质量
- 机器执行指令的速度

```
① void Bubble_Sort(int a[], int n)    //起泡排序算法
② {
③     int i, change=TRUE;
④     for(i=n-1; i > 1 && change; --i)
⑤     {
⑥         change = FALSE;
⑦         for(j = 0; j < i; j++)
⑧             if(a[j] > a[j+1])
⑨             {
⑩                 swap(&a[j], &a[j+1]);
⑪                 change = TRUE;
⑫             }
⑬     }
⑭ }
```

事前估计法：不同情况下，算法的执行时间（算法中基本运算执行的次数）

- 输入数据集的变化；
- 算法本身处理策略的却别；

算法分析

□ 算法的时间代价(时间复杂性):

- 当问题规模以某种单位由1增至 n 时，对应算法所耗费的时间也以某种单位由 $F(1)$ 增至 $F(n)$ ，这时我们称该算法的时间代价是 $F(n)$ 。
- 时间单位：一般规定为执行一个简单语句（如赋值语句、判断语句等）所用时间。

□ 算法的空间代价(空间复杂性):

- 当问题规模以某种单位由1增至 n 时，解该问题的算法所需占用的空间也以某种单位由 $G(1)$ 增至 $G(n)$ ，这时我们称该算法的空间代价是 $G(n)$ 。
- 空间单位：一般规定为一个简单变量（如整型、实型等）所占存储空间的大小。

大O表示法

如果存在正的常数 c 和 N ,

当问题的规模 $n \geq N$ 后,

该算法的时间(或空间)代价 $T(n) \leq c F(n)$

则

称该算法的时间代价(或空间代价)为 $O(F(n))$,

这时

也称该算法的时间(或空间)代价的增长率为 $F(n)$ 。

如

若 $T(n)=3n^3$, $c=3$, $F(n)=n^3$, 则 $T(n)=O(n^3)$ 。

大O表示法

```
① void Bubble_Sort(int a[], int n)
② {
③     int i, change=TRUE;
④     for(i=n-1; i > 1 && change; --i)
⑤     {
⑥         change = FALSE;
⑦         for(j = 0; j < i; j++)
⑧             if(a[j] > a[j+1])
⑨             {
⑩                 swap(&a[j], &a[j+1]);
⑪                 change = TRUE;
⑫             }
⑬     }
⑭ }
```



实质是一种估计策略

大O表示法计算规则

□ 大O表示法的作用：主要关注复杂性的量级，而忽略量级的系数。

□ 大O表示法常用的计算规则：

1. 加法准则

$$\begin{aligned} T(n) &= T1(n) + T2(n) \\ &= O(f1(n)) + O(f2(n)) = O(\max(f1(n), f2(n))) \end{aligned}$$

2. 乘法准则

$$\begin{aligned} T(n) &= T1(n) \times T2(n) \\ &= O(f1(n)) \times O(f2(n)) = O(f1(n) \times f2(n)) \end{aligned}$$

例子1

考虑如下程序：

```
{  
    action1(⋯.);  
    action2(⋯.);  
}
```

其中action1的时间代价是 $T1(n) = O(n^2)$,

action2的时间代价是 $T2(n) = O(n^3)$ 。

按照加法规则，计算整个程序的时间代价为：

$$T(n) = T1(n) + T2(n) = O(\max(n^2, n^3)) = O(n^3)$$

例子2

考虑如下程序：

```
for(i = 1; i <= n; i++)  
    action3(...);    /*假设action3(…)的执行中不改变i的值*/
```

以操作action3的执行时间为单位，显然这里

for循环的时间代价可以看作是 $T1(n) = O(n)$ ；

假设action3的时间代价是 $T2(n) = O(f(n))$ ，

根据乘法规则整个程序实际的时间复杂性应当是：

$$T(n) = T1(n) \times T2(n) = O(n) \times O(f(n)) = O(n \times f(n))$$

复杂性计算示例

两个 $N \times N$ 矩阵相乘的算法, $C_{n \times n} = A_{n \times n} \times B_{n \times n}$

```
for (i=0; i<n; i++)
```

```
    for (j=0; j<n; j++) {
```

```
        c[i][j] = 0;
```

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

```
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
```

```
    }
```

$$T(n) = O(n \times n \times (1+n))$$

$$= O(n^2 + n^3)$$

$$= O(n^3)$$

问题规模: n

原操作: $c[i][j] += a[i][k] * b[k][j];$

原操作重复执行次数: $f(n) = n^3$

算法的时间复杂度度量

● 常用的时间复杂度按数量级递增排列：

- 常数阶 $O(1)$
- 对数阶 $O(\log_2 n)$
- $O((\log_2 n)^3)$
- 线性阶 $O(n)$
- 线性对数阶 $O(n \log_2 n)$
- 平方阶 $O(n^2)$
- 立方阶 $O(n^3)$
- k 次方阶 $O(n^k)$
- 指数阶 $O(2^n)$

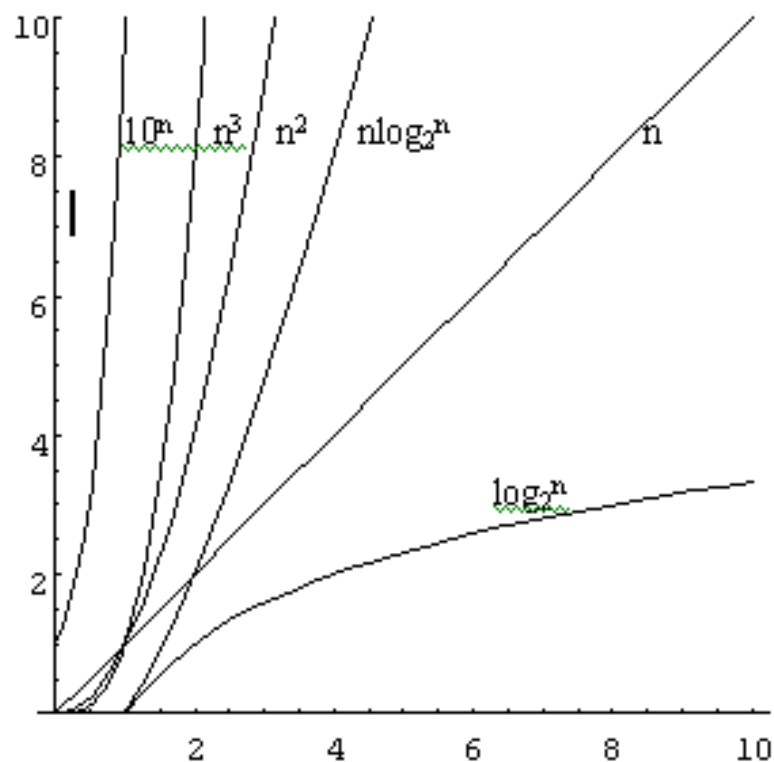


图 1.3 $f(n)$ 函数曲线变化速度的比较

(1) $i=1; k=0;$
 $\text{while}(i<n)$
 $\{ k=k+10*i; i++; \}$

◆ $T(n)=n-1$

∴ $T(n)=O(n)$

◇ 这个函数是按线性阶递增的

(2) $i=1; j=0;$
 $\text{while}(i+j\leq n)$
 $\{ j++; i++; \}$

◆ $T(n)=n/2$

∴ $T(n)=O(n)$

◇ 虽然时间函数是 $n/2$,但其数量级仍是按线性阶递增的。

(3) $x=91; y=1000;$
 $\text{while}(y>0)$
 $\text{If } (x>100)$
 $\{ x=x-10; y--; \}$
 $\text{else } x++;$

◆ $T(n)=O(1)$

◇ 这个程序看起来有点吓人,
 $x=x-10; y--;$ 总共运行了1000次,但是
我们看到这段程序的运行是和 n 无关
的,就算它再循环一万年,大O表示
法也只是把它看作一个常数阶的函数。

(4) $x=n; \quad // n>1, y \geq 0$
 $\text{while } (x \geq 2^{(y+1)})$
 $y++;$

◆ $T(n)=\log_2 n$

∴ $T(n)=O(\log_2 n)$

◇ 最坏的情况是 $y=0$,
那么循环的次数是 $\log_2 x$ 次,
也就是 $\log_2 n$ 次($n>1$)。
这是一个按对数阶递增的函数。

算法的时间复杂性度量

- 有些算法（如排序等）基本操作的执行次数除了与问题的规模 n 有关外，还与输入数据有关。此时，
 - 用基本操作的平均执行次数衡量算法的时间效率。
 - 也常用算法在输入数据集最坏情况下，基本操作的最多执行次数作为算法的时间效率度量。
- 最好情况下：对同样规模的问题所花费的最小代价
- 最坏情况下：对同样规模的问题所花费的最大代价
- 平均情况下：对同样规模的问题所花费的平均代价
- 实际情况下：介于最好情况与最坏情况之间

顺序找K值

- 顺序从一个规模为 n 的一维数组中找出一个给定的 K 值
- 最佳情况
 - 数组中第一个元素为 K
 - 只需要查找1个元素
- 最差情况
 - K 是数组中最后一个元素
 - 检查数组中所有 n 个元素

顺序找K值 – 平均情况

□ 等概率分布

- K值出现在n个位置的概率均为1/n

□ 查找的平均代价为 $O(n)$

$$\frac{1 + 2 + \dots + n}{n} = \frac{n + 1}{2}$$

顺序找K值 – 平均情况

□ 不等概率分布

- K值出现在1个位置的概率 $1/2$
- 第二个位置概率 $1/4$
- 其他位置概率 $\frac{1-1/2-1/4}{n-2} = \frac{1}{4(n-2)}$

□ 查找的平均代价为 $O(n)$

$$\frac{1}{2} + \frac{2}{4} + \frac{3 + \dots + n}{4(n-2)} = 1 + \frac{n(n+1) - 6}{8(n-2)} = 1 + \frac{n+3}{8}$$

执行次数随输入数据不同而不同

例：冒泡排序法

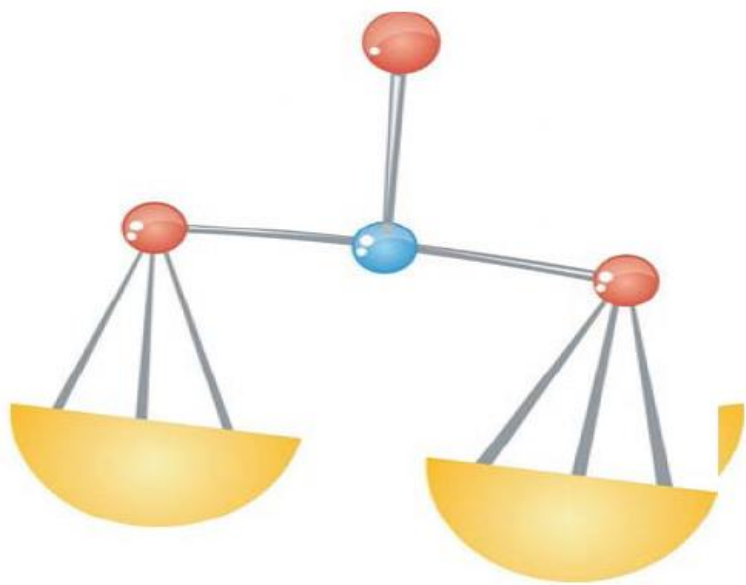
```
void Bubble_Sort(int a[], int n)
{
    int i, change=TRUE;
    for(i=n-1; i >1 && change; --i)
    {
        change = FALSE;
        for(j = 0; j < i; j++)
            if(a[j] > a[j+1])
            {
                swap(&a[j], &a[j+1]);
                change = TRUE;
            }
    }
}
```

$a = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$a = \{9, 8, 7, 6, 5, 4, 3, 2, 1\}$

时间与空间的权衡

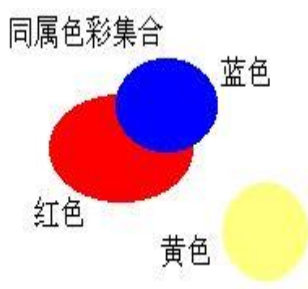
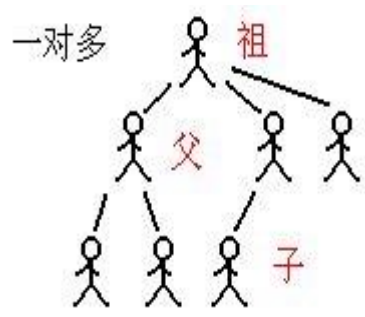
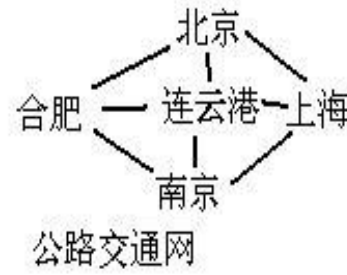
- ❑ 计算机程序中经典矛盾是空间和时间之间的矛盾
- ❑ 增大空间开销可能改善算法的时间开销
- ❑ 可以节省空间，往往需要增大运算时间



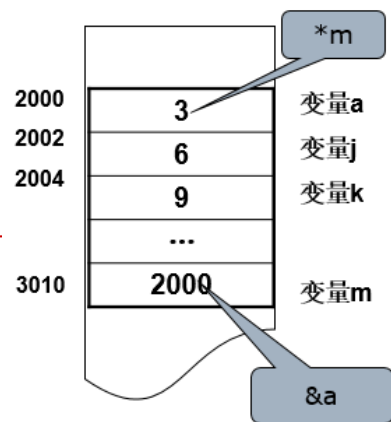
第一章 绪论

- 数据结构在问题求解中的作用
- 数据类型与数据结构
- 算法特性与算法评价
- 总结

逻辑结构

	集合	线性结构	树形结构	图状或网状结构
特征	元素间为松散的关系	元素间为严格的一对一关系	元素间为严格的一对多关系	元素间为多对多关系
示例	<p>同属色彩集合</p> 	<p>如银行排队叫号时队列中的各元素</p>	<p>一对多</p> 	<p>多对多</p>  <p>公路交通网</p>

存储结构



□ 数据的存储结构（Storage Structure）

- 亦称物理结构, 是逻辑结构在计算机中的物理存储表示

□ 计算机主存储器

- **空间相邻**: 存储空间是一种具有**非负整数**地址编码的、空间**相邻的单元**集合, 其基本存储单元是**字节**
- **随机访问**: 计算机指令具有**按地址随机访问**存储空间内任意单元的能力, 访问不同地址所需时间相同

□ 四种基本存储映射方法

- **顺序、链接、索引、散列**

算法

- 算法是对特定问题求解方法和步骤的一种描述，它是指令的一组有限序列，其中每个指令表示一个或多个操作。
- 算法的五个重要特性
 - 输入\输出\有穷性\确定性\可行性
- 算法的评价指标
 - 时间效率：执行算法耗费的时间代价；
 - 空间效率：运行算法需要耗费的存储空间代价

练习

- 按照增值率从低到高的顺序排列以下表达式：
 - $4n^2$, $\log_3 n$, $3n$, $20n$, 2000 , $\log_2 n$, $n^{2/3}$
 - 并给 $n!$ 在其中找个适当的位置

斯特林公式

$$\lim_{n \rightarrow +\infty} \frac{n!}{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n} = 1$$