山东师范大学 SHANDONG NORMAL UNIVERSITY

《算法设计与分析》
第1章 算法及基础知识(2)
—递归、数据结构与数学基础

山东师范大学信息科学与工程学院
段会川
2014年9月

目录

- 复习
- 递归基础
- 基本数据结构
- 算法分析的数学基础
- 习题
- GCD算法
- 线性搜索算法
- 求最大值算法

上节课回顾：GCD算法

上节课回顾：线性搜索算法

目录

- 复习
- 递归基础
- 基本数据结构
- 算法分析的数学基础
- 习题
- 回顾
- 普适性
- GCD的递归表述
- 基本认知
- n!的递归算法
- 深入的n!算法

1.4 递归—回顾

- 本课程之前所学习的递归

1.4 递归—普适性

- 递归是一种普适性的问题解决方法

1.4 递归—GCD算法的递归表述

1.4 递归—概述

递归技术是设计和描述算法的一种强有力的工具，它在算法设计与分析中起着非常重要的作用，采用递归技术编写出的程序通常比较简洁且易于理解，并且证明算法的正确性要比相应的非递归形式容易得多。因此在实际的编程中，人们常采用该技术来解决某些复杂的计算问题。有些数据结构如二叉树，结构本身就具有递归特性；此外还有一类问题，其本身没有明显的递归结构，但用递归程序求解比其他方法更容易编写程序，如八皇后问题、汉诺塔问题等。鉴于该技术的优点和重要性，在介绍其他算法设计方法之前先对其进行讨论。

1.4.1 认知递归

子程序（或函数）直接调用自己或通过一系列调用语句间接调用自己，称为递归。直接或间接调用自身的算法称为递归算法。递归的基本思想就是“自己调用自己”，体现了“以此类推”、“重复同样的步骤”这样的理念。实际上，递归是把一个不能或不好解决的大问题转化为一个或几个小问题，再把这些小问题进一步分解成更小的小问题，直至每个小问题都可以直接解决。

通常，采用递归算法来求解问题的一般步骤是：

（1）分析问题、寻找递归关系。找出大规模问题和小规模问题的关系。换句话说，如果一个问题能用递归方法解决，它必须可以向下分解为若干个性质相同的规模较小的问题。

（2）找出停止条件，该停止条件用来控制递归何时终止，在设计递归算法时需要给出明确的结束条件。

（3）设计递归算法、确定参数，即构建递归体。

递归算法的运行过程包含两个阶段：递推和回归。递推指的是将原问题不断分解为新的子问题，逐渐从未知向已知推进，最终达到已知的条件，即递归结束的条件。回归指的是从已知的条件出发，按照递推的逆过程，逐一求值回归，最后达到递推的开始处，即求得问题的解。

1.4.2 n!的递归算法及其复杂度的递推分析法

- **算法名称：n!的递归计算factorial**
- **输入：n**
- **输出：n!**
- **1: factorial(n)**
- **2: if n=0 then**
- **3: return 1**
- **4: else**
- **5: return n*factorial(n-1)**

1.4.2 深入探索n!的计算算法

目录

- 复习
- 递归基础
- 基本数据结构
- 算法分析的数学基础
- 习题

- 顺序表
- 链表
- 栈
- 队列
- 树
- 图
- 集合

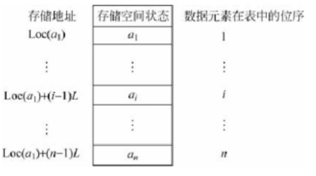
1.5 基本数据结构—顺序表

把线性表的元素按逻辑次序依次存放在一组地址连续的存储单元，用这种方法存储的线性表简称为顺序表。

顺序表的特点是逻辑上相邻的数据元素其物理位置也相邻，表中第 i 个元素 a_i 的存储位置可用一个简单且直观的公式来表示：

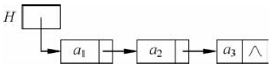
$$\mathrm{LOC}(a_i) = \mathrm{LOC}(a_1) + L \times (i-1) \quad 1 \leqslant i \leqslant n$$

其中，L 是每个元素占用的存储单元的个数；$\mathrm{LOC}(a_1)$ 是第一个元素 a_1 的存储地址（简称为基地址）；$\mathrm{LOC}(a_i)$ 是元素 a_i 的存储地址。



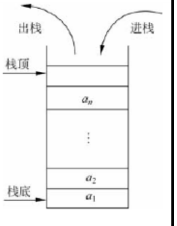
1.5 基本数据结构—链表

- 线性链表/单链表
- 循环链表
- 双向链表



1.5 基本数据结构—栈

和线性表类似，栈也有两种存储表示方法，即顺序栈和链栈。顺序栈指的是栈的顺序存储结构，是利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素。设置栈顶指针为 top，它指示栈顶元素在顺序栈中的位置；栈底指针为 base，它始终指向栈底的位置。初始时，top 指向栈底，每当插入一个元素时，top++；删除栈顶元素时，top--，因此，非空栈中的 top 始终在栈顶元素的下一个位置。



1.5 基本数据结构—队列

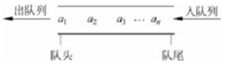


图 1-6　队列的示意图

队列的示意图如图 1-6 所示。

和线性表类似，队列也有两种存储结构，即顺序存储结构（循环队列）和链式存储结构（链队列）。

和顺序栈类似，在队列的顺序存储结构中，除了用一组地址连续的存储单元依次存放从队头到队尾的元素之外，尚需设置两个指针 front 和 rear，它们分别指示队列的头元素和尾元素。初始时，令 front＝rear＝0，每当插入 1 个新的元素，rear 增加 1；每当删除 1 个新的元素，front 增加 1；当 front＝rear 时，队列为空。这种操作方式会导致一个新的情况：如果插入和删除的元素越来越多，rear 的值将一直增加，最终达到所分配存储单元的最大值，当要继续插入新的元素时，队尾已没有空间了。但如果 front>0，此时第 0 个单元至第 front－1 个单元的存储空间并未占用。那么，该如何利用这部分可用空间呢？一个较为巧妙的办法是将顺序队列构造为一个环状的空间，称为循环队列。这样，空间就可以循环利用了。

1.5 基本数据结构—树

定义 5　树是由 $n(n \geqslant 0)$ 个结点组成的有限集。在任意一棵非空树中，有且仅有一个特定的结点称为该树的根结点，当 $n>1$ 时，根结点之外的其余结点可分为 $m(m \geqslant 0)$ 个互不相交的有限集合 $T_1, T_2, \cdots, T_m$，其中每个集合本身又是一棵树，称为根的子树。树的示意图如图 1-7 所示。

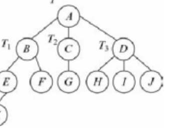


图 1-7　树的示意图

图 1-7 是一棵由 11 个结点组成的树 T。其中 A 是根结点，其余结点分为三个互不相交的有限子集：$T_1=\{B, E, F, K\}$，$T_2=\{C, G\}$，$T_3=\{D, H, I, J\}$，T_1、T_2、T_3 都是根 A 的子树，这三棵子树的根结点分别是 B、C、D，每棵子树本身也是一棵树，可继续划分。例如子树 T_3 以 D 为根结点，子树的其余结点又可分为三个互不相交的子集 $T_{31}=\{H\}$，$T_{32}=\{I\}$，$T_{33}=\{J\}$，而 T_{31}、T_{32} 和 T_{33} 又是只有一个根结点的树。

1.5 基本数据结构—树

父结点：若一个结点有子树，则该结点为父结点(或称为双亲结点)。

孩子结点：若某结点有子树，则其子树的根为该结点的孩子结点。

兄弟结点：同一个结点的孩子结点。

层次：结点的层次是从根结点开始定义的，根结点的层次为 1，其余结点的层次是其父结点的层次加 1。若某结点在第 i 层，则其子树的根就在第 $i+1$ 层。

结点的深度：结点所在的层次减 1。

树的高度(深度)：树中结点的最大深度。

度：结点拥有的子树数目称为该结点的度，即一个结点的孩子数目就是该结点的度。

叶子结点：度为 0 的结点。

森林：森林是 $m(m\geqslant0)$ 棵互不相交的树的集合，对树中每个结点而言，其子树的集合即为森林。

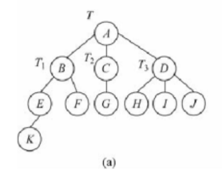
二叉树：二叉树是一种非常重要的树型结构，它的特点是每个结点至多只有两棵子树，并且，二叉树的子树有左右之分，其次序有时不能任意颠倒。

1.5 基本数据结构—树的存储结构

① 顺序存储结构——双亲表示法。

双亲表示法是以一组连续的空间来存储树中的结点，并将树中所有结点从上到下、从左至右依次编号，并用该编号作为结点在顺序存储中的位置。

树中每个结点包含两个域，即数据域(data) 和双亲域(parent)。数据域存放结点的数据，双亲域存放双亲(父)结点在顺序存储中的位置编号。通过 parent 域，顺序存储的各个结点仍然保持正确的前驱-后继关系。



(a)

数组下标

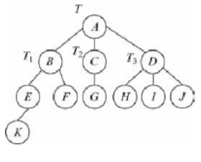
0	A	−1
1	B	0
2	C	0
3	D	0
4	E	1
5	F	1
6	G	2
7	H	3
8	I	3
9	J	3
10	K	4

(b)

1.5 基本数据结构—树的存储结构

② 链式存储结构——孩子表示法。

把每个结点的孩子结点排列起来，看成是一个线性表，且以单链表作为存储结构，则 n 个结点有 n 个孩子链表(叶子结点的孩子链表为空表)。而 n 个头指针又组成另外一个线性表。为了便于查找，可采用顺序存储结构。

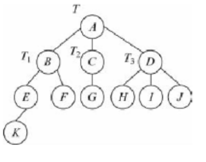


1.5 基本数据结构—树的存储结构

③ 链式存储结构——孩子兄弟表示法。

又称二叉树表示法或二叉链表表示法，即以二叉链表作为树的存储结构，链表中结点的两个域分别指向该结点的第一个孩子结点和下一个兄弟结点。

利用这种存储结构易于实现找结点孩子等操作。如果为每个结点增加一个 parent 域，则同样能方便地实现对双亲的操作。

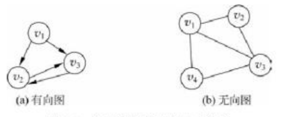


1.5 基本数据结构—图

定义 6 图是一种数据结构，可以用二元组表示，图中的数据元素通常称为顶点(或结点)，数据元素之间的关系称为边，其形式化定义为：$G=(V,E)$。

其中，集合 V 是顶点集，即它是顶点的有穷非空集合；集合 E 是边集，即它是 V 中两个顶点之间的关系的有穷集。

在图 G 中，若 $<v,w>\in E$，则 $<v,w>$ 表示从 v 到 w 的一条弧，v 称为弧尾或始点，w 称为弧头或终点，此时的图称为有向图。若 $<v,w>\in E$ 必有 $<w,v>\in E$，即 E 是对称的，则以无序对 (v,w) 代替这两个有序对，表示 v 和 w 之间的一条边，此时的图称为无向图。下面通过实例(如图 1-9 所示)讲述一下这两种图的区别及相应的形式化定义方法。

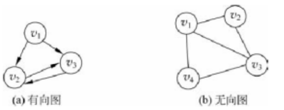


(a) 有向图　　(b) 无向图

图 1-9　有向图及无向图的示意图

1.5 基本数据结构—图

面通过实例(如图 1-9 所示)讲述一下这两种图的区别及相应的形式化定义方法。



(a) 有向图　　(b) 无向图

图 1-9　有向图及无向图的示意图

在图 1-9(a)中，每条边的方向是用从始点指向终点的箭头表示的，因此该图是有向图。该有向图的形式化定义为：$G_1=(V_1,E_1)$；顶点集 $V_1=\{v_1,v_2,v_3\}$；边集 $E_1=\{<v_1,v_2>,<v_1,v_3>,<v_2,v_3>,<v_3,v_2>\}$；注意 $<v_3,v_2>$ 与 $<v_2,v_3>$ 表示两条不同的边。

图 1-9(b)中所示的每条边都是没有方向的，因此该图为无向图。该无向图的形式化定义为：$G_2=(V_2,E_2)$；顶点集 $V_2=\{v_1,v_2,v_3,v_4\}$；边集 $E_2=\{(v_1,v_2),(v_1,v_3),(v_1,v_4),(v_2,v_3),(v_3,v_4)\}$；注意 (v_2,v_1) 与 (v_1,v_2) 表示同一条边。

1.5 基本数据结构—关于图的重要术语

① 邻接点和相关边。

对于无向图 $G=(V, E)$，如果边$(v, w)\in E$，则称顶点 v 和 w 互为邻接点，称边(v, w)依附于顶点 v 和 w，即边(v, w)是与顶点 v 和 w 相关联的边。

对于有向图 $G=(V,E)$，如果$<v,w>\in E$，则称顶点 v 邻接到顶点 w，顶点 w 邻接于顶点 v，而弧$<v,w>$是与顶点 v 和 w 相关联的。

② 路径和回路。

路径：在无向图 $G=(V,E)$中，从顶点 v 到顶点 v'的路径是一个顶点序列($v=v_{i,0}$, $v_{i,1}$, …, $v_{i,m}=v'$)，其中$(v_{i,j-1}, v_{i,j})\in E$，$1\leqslant j\leqslant m$。如果 G 是有向图，则路径也是有向的，顶点序列应满足$<v_{i,j-1}, v_{i,j}>\in E$，$1\leqslant j\leqslant m$。路径的长度为路径上的边(或弧)的数目。

第一个顶点和最后一个顶点相同的路径称为回路或环。环的判断也是一个非常重要的概念，在 Kruskal 算法中就涉及了对环的判断。另外，有向无环图是描述含有公共子式的表达式、一项工程及系统的进行过程的有效工具。

1.5 基本数据结构—关于图的重要术语

③ 权、网。

有时候图中的边或弧具有和其相关的数据，如表示一个顶点到另一个顶点的距离和费用，称这些相关的数据为边或弧的权，而这种带权的图称为网。网示意图如图 1-10 所示。

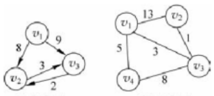

(a) 有向图　(b) 无向图

图 1-10　网示意图

④ 连通、生成树。

在无向图 G 中，如果从顶点 v 到顶点 v'有路径，则称 v 和 v'是连通的。如果图中任意两个顶点都是连通的，则称 G 是连通图。

在有向图 G 中，如果对于每一对 $v,w\in V$，$v\neq w$，从 v 到 w 和从 w 到 v 都存在路径，则称 G 是强连通的。

一个连通图的生成树是一个极小连通子图，它含有图中全部顶点，但只能构成一棵树的 $n-1$ 条边。同样，生成树是一个很重要的概念，例如在求解如何在最节省费用的前提下在 n 个城市间构造通信联络网时，就用到了最小生成树的概念。在该概念的指导下，问题的解决将变得非常容易。

1.5 基本数据结构—图的邻接矩阵存储

定义 7　假设 $G=(V, E)$是一个有 n 个顶点的图，规定各顶点的序号依次为 1,2,3,…, n，则 G 的邻接矩阵是一个具有如下定义的 n 阶方阵：

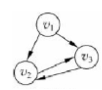
$$A[i,j]=\begin{cases}1 & 若 <v_i,v_j>\in E 或者(v_i,v_j)\in E\\ 0 & 反之\end{cases}$$

对于带权图(或网)，可以将上述定义改为：

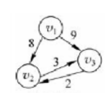
$$A[i,j]=\begin{cases}W_i & 若 <v_i,v_j>\in E 或者(v_i,v_j)\in E\\ \infty & 反之\end{cases}$$

其中，W_i 表示$<v_i,v_j>$弧或(v_i, v_j)边上的权值。

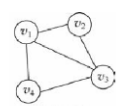
1.5 基本数据结构—图的邻接矩阵存储



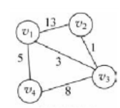
$$\begin{bmatrix}0 & 1 & 1\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}$$



$$\begin{bmatrix}\infty & 8 & 9\\ \infty & \infty & 3\\ \infty & 2 & \infty\end{bmatrix}$$



$$\begin{bmatrix}0 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 1\\ 1 & 0 & 1 & 0\end{bmatrix}$$

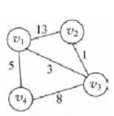


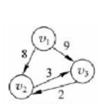
$$\begin{bmatrix}\infty & 13 & 3 & 5\\ 13 & \infty & 1 & \infty\\ 3 & 1 & \infty & 8\\ 5 & \infty & 8 & \infty\end{bmatrix}$$

1.5 基本数据结构—图的邻接表存储

邻接表是图的一种链式存储结构。在邻接表中，图中每个顶点对应一个单链表，第 i 个单链表中的结点表示依附于顶点 v_i 的边(对于有向图是以顶点 v_i 为尾的弧)。每个结点由三个域组成，邻接点域指示与顶点 v_i 邻接的点在图中的位置，链域指示下一条边或弧的结点，数据域存储与边或弧相关的信息。每个链表设置一个表头结点，该结点中的链域指向链表中第一个结点，数据域用来存储 v_i 的名称或其他相关信息。

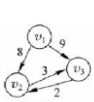
由于在邻接表中，为了求某个顶点的入度则必须遍历所有链表。因此，为了提高入度计算的效率，通常为图建立一个逆邻接表。逆邻接表就是链表结点代表以 v_i 为弧头的弧。





1.5 基本数据结构—图的邻接表存储

由于在邻接表中，为了求某个顶点的入度则必须遍历所有链表。因此，为了提高入度计算的效率，通常为图建立一个逆邻接表。逆邻接表就是链表结点代表以 v_i 为弧头的弧。



1.5 基本数据结构—集合的表示

表示集合的方法有很多，表示方法的不同将造成查找等运算的算法也不同，所以集合的表示方法将直接影响集合运算的效率。在计算机应用中，集合有以下几种表示方法：位向量、线性表、搜索树、跳表和散列表，这里简单介绍前两种。

(1) 只考虑集合 U 的子集，用位串来表示集合。如果集合 U 具有 n 个元素，那么 U 的任何子集 S 能够用一个长度为 n 的位串来表示，称为位向量。当且仅当 U 的第 i 个元素包含在 S 中时，向量中第 i 个元素为 1。例如：$U=\{1,2,3,4,5,6,7,8,9\}$，那么 $S=\{2,3,5,7\}$ 可以用位串 011010100 表示。这种表示法使得实现非常快速的标准集合运算成为可能，但是所需的存储空间较大。

(2) 用线性表来表示集合元素。当然，只有对有限集合这个方法才是可行的。

目录

1.6 常用数学公式—对数公式

定义 9 令 b 是大于 1 的实数，x 是实数。如果对某些正实数 y，有 $y=b^x$，那么 x 称为 y 以 b 为底的对数，记为：$x=\log_b y$。其中，b 称为对数的底数，y 称为真数。

关于对数公式，有下列性质：

(1) 负数和零没有对数。

(2) 1 的对数是 0；即 $\log_b 1=0$。

(3) 底数的对数是 1，即 $\log_b b=1$。

(4) $\log_b b^n=n$。

(5) $b^{\log_b n}=n$。

两个特殊对数：以 10 为底的对数称为常用对数，即 N 的常用对数记做 $\lg N$；以无理数 e(e=2.71828…)为底的对数称为自然对数，N 的自然对数记做 $\ln N$；以 2 为底 N 的对数简记为 $\log N$。

1.6 常用数学公式—组合公式

定义 10 从 n 个不同元素中取 $m(m\leqslant n)$ 个不重复的元素组成一个子集，而不考虑其元素的顺序，称为从 n 个元素中取 m 个元素的无重组合。组合的全体组成的集合用 C_n^m 表示。

组合公式为：

$$C_n^m=\frac{n!}{m!(n-m)!}=\frac{n(n-1)\cdots(n-m+1)}{m(m-1)\cdots 2}\quad (m\leqslant n)$$

组合公式的性质：

(1) $C_n^m=C_n^{n-m}(m\leqslant n)$。

(2) $C_n^0+C_n^1+C_n^2+\cdots+C_n^n=2^n$。

(3) n 为奇数时，$C_n^0+C_n^2+\cdots+C_n^{n-1}=C_n^1+C_n^3+\cdots+C_n^n=2^{n-1}$，

n 为偶数时，$C_n^0+C_n^2+\cdots+C_n^n=C_n^1+C_n^3+\cdots+C_n^{n-1}=2^{n-1}$。

1.6 常用数学公式—求和公式

(1) 算术级数。

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

(2) 平方和。

$$\sum_{i=1}^{n} i^2=\frac{n(n+1)(2n+1)}{6}=\Theta(n^3)$$

(4) 调和级数。

把调和级数前 n 项之和记为 H_n，则 $H_n=1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}=\sum_{i=1}^{n}\frac{1}{i}$。

1.6 常用数学公式—求和公式

(3) 几何级数。

$$\sum_{i=0}^{n} a^i=\frac{a^{n+1}-1}{a-1}=\Theta(a^n),\quad a\neq 1$$

当 $a=2$ 时，

$$\sum_{i=0}^{n} 2^i=\frac{2^{n+1}-1}{2-1}=\Theta(2^n)$$

当 $a=1/2$ 时，

$$\sum_{i=0}^{n}\frac{1}{2^i}=2-\frac{1}{2^n}<2=\Theta(1)$$

当 $|a|<1$ 时，有如下的无穷级数：

$$\sum_{i=0}^{n} a^i=\frac{1}{1-a}=\Theta(1)$$

1.6 常用数学公式—向下和向上取整

如果 x 是任意实数，则记：

$\lfloor x \rfloor$＝小于或等于 x 的最大整数，简称为 x 的下限，即$\lfloor\ \rfloor$为向下取整公式。

$\lceil x \rceil$＝大于或等于 x 的最小整数，简称为 x 的上限，即$\lceil\ \rceil$为向上取整公式。

例如：$\lfloor \sqrt{3} \rfloor=1$，$\lceil \sqrt{3} \rceil=2$，$\lfloor -\frac{1}{2} \rfloor=-1$，$\lceil -\frac{1}{2} \rceil=0$。

容易证明下面的关系成立：

(1) $\lceil x \rceil=\lfloor x \rfloor$，当且仅当 x 是整数

(2) $\lceil x \rceil=\lfloor x \rfloor+1$，当且仅当 x 不是整数

(3) $\lfloor -x \rfloor=-\lceil x \rceil$

(4) $x-1<\lfloor x \rfloor\leqslant x\leqslant\lceil x \rceil<x+1$

(5) $\lfloor \frac{x}{2} \rfloor+\lceil \frac{x}{2} \rceil=x$

(6) 一个很有用的定理。令 $f(x)$是一个单调递增函数，使得当 $f(x)$是整数时，x 也是整数。那么，有$\lfloor f(\lfloor x \rfloor) \rfloor=\lfloor f(x) \rfloor$并且$\lceil f(\lceil x \rceil) \rceil=\lceil f(x) \rceil$。

目录

习题

1. 写出计算n!的递归算法
2. 写出欧几里得GCD算法的递归描述
3. 给出图1-9(a), 1-9(b), 1-10(a), 1-10(b)中各图的邻接表和邻接矩阵表示
4. 什么是调和函数？其复杂度阶是什么？
5. 举例说明什么是取整函数

The End

Thanks !