

# Recursive Algorithm Design

Instructor: Shizhe Zhou

Course Code:00125401

# Polynomial Evaluation

问题 给定一串实数  $a_n, a_{n-1}, \dots, a_1, a_0$ , 以及一个实数  $x$ , 计算多项式  $P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  的值。

- 1. 直接估值:  $(n+1) * n / 2$  次乘法,  $n$  次加法.
- 2. 从低次到高次递归:  $2n$  次乘法,  $n$  次加法.
- 3. 反向递归: 1 次乘法, 1 次加法.

Horner  
法则

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 = ((\dots ((a_n x + a_{n-1}) x + a_{n-2}) \dots) x + a_1) x + a_0$$

# max derived graph

- A critical point: if  $n=k+1$ , it has to be a complete graph.
- Code:
- <http://staff.ustc.edu.cn/~szhou/course/algorithmsFundamentals/maxsubgraph.zip>

# Find largest set with 1-1 map

**问题** 给定一个集合  $A$  和一个从  $A$  到自身的映射  $f$ , 寻找元素个数最多的一个子集  $S \subseteq A$ ,  $S$  满足: (1)  $f$  把  $S$  中的每一个元素映射到  $S$  中的另一元素 (即  $f$  把  $S$  映射到它自身), (2)  $S$  中没有两个元素映射到相同的元素 (即  $f$  在  $S$  上是一个一对一函数)。

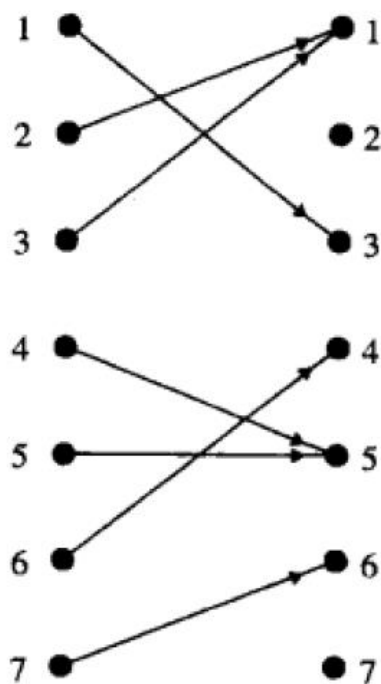


图 5.2 从一个集合到自身的映射 (两侧是相同的集合)

**算法** *Algorithm Mapping* ( $f, n$ )

**输入:**  $f$  (数组, 其元素值为 1 到  $n$  的数值)

**输出:**  $S$  (1 到  $n$  的整数集合的一个子集,  $f$  是  $S$  上的一对一函数)

**begin**

$S := A$ ; { $A$  是由 1 到  $n$  的整数组成的集合}

**for**  $j := 1$  to  $n$  **do**  $c[j] := 0$ ;

**for**  $j := 1$  to  $n$  **do** increment  $c[f[j]]$ ;

**for**  $j := 1$  to  $n$  **do**

**if**  $c[j] = 0$  **then** put  $j$  in Queue;

**while** Queue is not empty **do**

    remove  $i$  from the top of the queue;

$S := S - \{i\}$ ;

    decrement  $c[f[i]]$ ;

**if**  $c[f[i]] = 0$  **then** put  $f[i]$  in Queue

**end**

不一定非要用队列

使得简化过的问题和原问题相同

图 5.3 算法 *Mapping*

# 社会名流问题

**问题** 给定一个  $n \times n$  连结矩阵，确定是否存在一个  $i$ ，其满足在第  $i$  列所有项（除了第  $i_i$  项）都为 1，并且第  $i$  行所有项（除了第  $i_i$  项）都为 0。

- <http://staff.ustc.edu.cn/~szhou/course/algorithmsFundamentals/celebrity.zip>

### 算法 *Celebrity (Know)*

**输入:** *Know* (一个  $n \times n$  的布尔数组)

**输出:** *celebrity*

**begin**

*i* := 1 ;

*j* := 2 ;

*next* := 3 ;

{第一阶段, 通过消除只留下一个候选者}

**while** *next* ≤ *n* + 1 **do**

**if** *Know*[*i*, *j*] **then** *i* := *next*

**else** *j* := *next* ;

*next* := *next* + 1 ;

    {*i* 和 *j* 的其中之一被消除}

**if** *i* = *n* + 1 **then**

*candidate* := *j*

**else**

*candidate* := *i* ;

{现在检查候选者确实是社会名流}

*wrong* := *false* ;

*k* := 1 ;

*Know*[*candidate*, *candidate*] := *false* ;

{虚构的变量通过检验}

**while** not *wrong* and *k* ≤ *n* **do**

**if** *Know*[*candidate*, *k*] **then** *wrong* := *true* ;

**if** not *Know*[*k*, *candidate*] **then**

**if** *candidate* ≠ *k* **then** *wrong* := *true* ;

*k* := *k* + 1 ;

**if** not *wrong* **then** *celebrity* := *candidate*

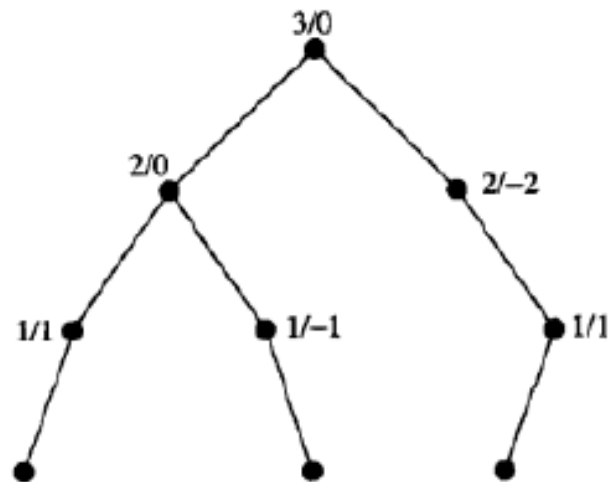
**else** *celebrity* := 0 {没有社会名流}

**end**

图 5.4 算法 *Celebrity*

# Balance Factor for BST tree

问题 给定一棵  $n$  个节点的二叉树  $T$ ，计算其所有节点的平衡因子。





# Find maximum consecutive subsequence

**问题** 给定一个实数序列  $x_1, x_2, \dots, x_n$  (不必是正数), 寻找一个 (连续的) 子序列  $x_i, x_{i+1}, \dots, x_j$ , 使得其数值之和在所有连续子序列数值之和中是最大的。

算法 *Maximum\_Consecutive\_Subsequence* ( $X, n$ )

输入:  $X$  (大小为  $n$  的数组)

输出:  $Global\_Max$  (最大连续子序列之和)

*begin*

$Global\_Max := 0;$

$Suffix\_Max := 0;$

*for*  $i := 1$  *to*  $n$  *do*

*if*  $x[i] + Suffix\_Max > Global\_Max$  *then*

$Suffix\_Max := Suffix\_Max + x[i];$

$Global\_Max := Suffix\_Max$

*else if*  $x[i] + Suffix\_Max > 0$  *then*

$Suffix\_Max := x[i] + Suffix\_Max$

*else*  $Suffix\_Max := 0$

*end*

# Knapsack problem

**问题** 给定一个整数  $K$  和  $n$  个不同大小的物品，第  $i$  个物品的大小为整数  $k_i$ ，寻找一个物品的子集，它们的大小之和正好为  $K$ ，或者确定不存在这样的子集。

### 算法 *Knapsack* ( $S, K$ )

**输入:**  $S$  (存储物品尺寸的大小为  $n$  的数组) 和  $K$  (背包的大小)

**输出:**  $P$  (一个二维数组, 如果对于前  $i$  个元素和大小为  $k$  的背包存在解, 则  $P[i, k].exist = true$ , 同时若该解包括了第  $i$  个元素, 则  $P[i, k].belong = true$ )  
{关于算法改进的提示参见习题 5.15}

**begin**

$P[0, 0].exist := true$  ;

**for**  $k := 1$  **to**  $K$  **do**

$P[0, k].exist := false$  ;

{不需要对  $P[i, 0], i \geq 1$  进行初始化, 因为其值将由  $P[0, 0]$  计算得到}

**for**  $i := 1$  **to**  $n$  **do**

**for**  $k := 0$  **to**  $K$  **do**

$P[i, k].exist := false$  ; {默认值}

**if**  $P[i-1, k].exist$  **then**

$P[i, k].exist := true$  ;

$P[i, k].belong := false$

**else if**  $k - S[i] \geq 0$  **then**

**if**  $P[i-1, k - S[i]].exist$  **then**

$P[i, k].exist := true$  ;

$P[i, k].belong := true$

**end**

图 5.10 算法 *Knapsack*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$k_1=2$	0	-	I	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$k_2=3$	0	-	O	I	-	I	-	-	-	-	-	-	-	-	-	-	-
$k_3=5$	0	-	O	O	-	O	-	I	I	-	I	-	-	-	-	-	-
$k_4=6$	0	-	O	O	-	O	I	O	O	I	O	I	-	I	I	-	I