

# 数据结构

## ▼ 数据结构

- 第1章 绪论
- 第2章 线性表
- ▼ 第3章 栈与队列

- 栈
- 循环队列

## ▼ 第4章 串

- 串的块链存储表示
- 二叉链表&线索二叉树

## ▼ 第6章 树和二叉树

- 二叉树的性质
- 二叉树和森林间的转换

## ▼ 第7章 图

- 图的定义
- 最小生成树
- 拓扑排序
- 最短路径

## ▼ 第9章 查找

- 分块查找法(索引顺序表的查找)
- B-树
- ▼ 哈希表
  - 开放定址法&线性探测再散列

## ▪ 邻接表

## ▼ 第10章 内部排序

- 快速排序
- 堆排序

## ▼ 其它

- 静态链表
- 后缀表达式

# 第1章 绪论

以下代码时间复杂度为 $O(\log n)$

```
int = 1;
while(i<=n)
    i = i * 2;
```

## 第2章 线性表

- 线性表是顺序存储结构是一种**随机存取**的存储结构

## 第3章 栈与队列

### 栈

结构体:

```
typedef struct{
    SElemType *base;
    SElemType *top;
    int stacksize;
}SqStack;
```

入栈push: \*S.top++ = e

判断栈满: S.top - S.base >= S.stacksize

出栈pop: e = \*S.top--

判断栈空: S.top == S.base

### 循环队列

初始化建栈: Q.front = Q.rear = 0

插入新元素: Q.rear = (Q.rear + 1) % MAXSIZE

删除头元素: Q.front = (Q.front + 1) % MAXSIZE

判断队满: if((Q.rear + 1) % MAXSIZE == Q.front)

## 第4章 串

### 串的块链存储表示

存储密度 = 结点数据本身所占的存储量 / 结点结构所占的存储总量

# 二叉链表&线索二叉树

- 在有N个结点的二叉链表中必定存在N+1个空链域

lchild	LTag	data	RTag	rchild
--------	------	------	------	--------

LTag = 0 lchild指示结点的左孩子

LTag = 1 lchild指示结点的前驱

RTag = 0 rchild指示结点的右孩子

RTag = 1 rchild指示结点的后继

以这种结点结构构成的二叉链表作为二叉树的存储结构,叫做**线索链表**,其中指向结点前驱和后继的指针,叫做**线索**.加上线索的二叉树称之为**线索二叉树**(Threaded Binary Tree).

## 第6章 树和二叉树

### 二叉树的性质

- 二叉树第*i*( $i \geq 1$ )层上至多有 $2^{i-1}$ 个结点
- 深度为k的二叉树至多有 $2^k - 1$ 个结点

### 二叉树和森林间的转换

- 树转换成二叉树
  - (1) 兄弟结点间加线
  - (2) 除长子(左孩子)外的孩子去线
  - (3) 层次调整(孩子靠左)

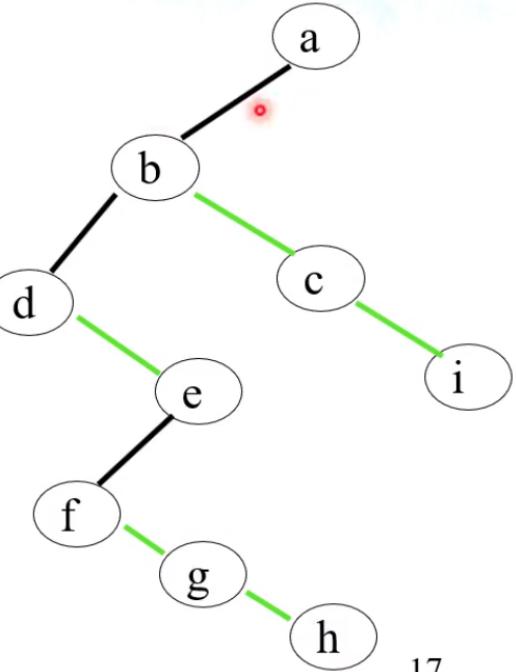
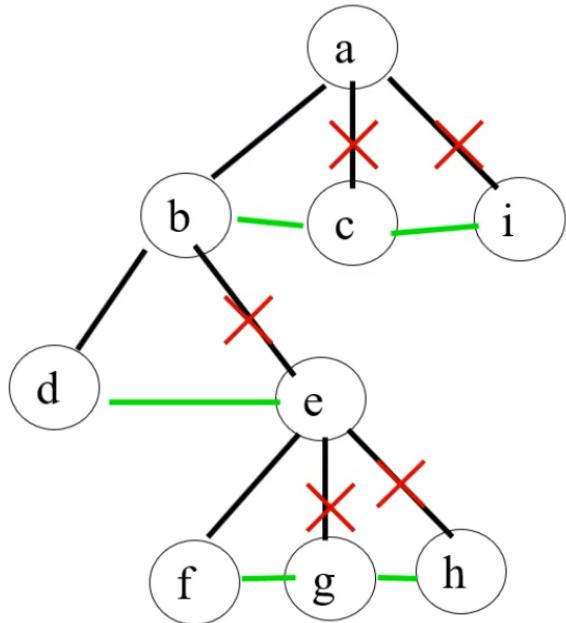
回顾1：树如何转为二叉树？

方法：加线—抹线—旋转

兄弟相连

长兄为父

孩子靠左



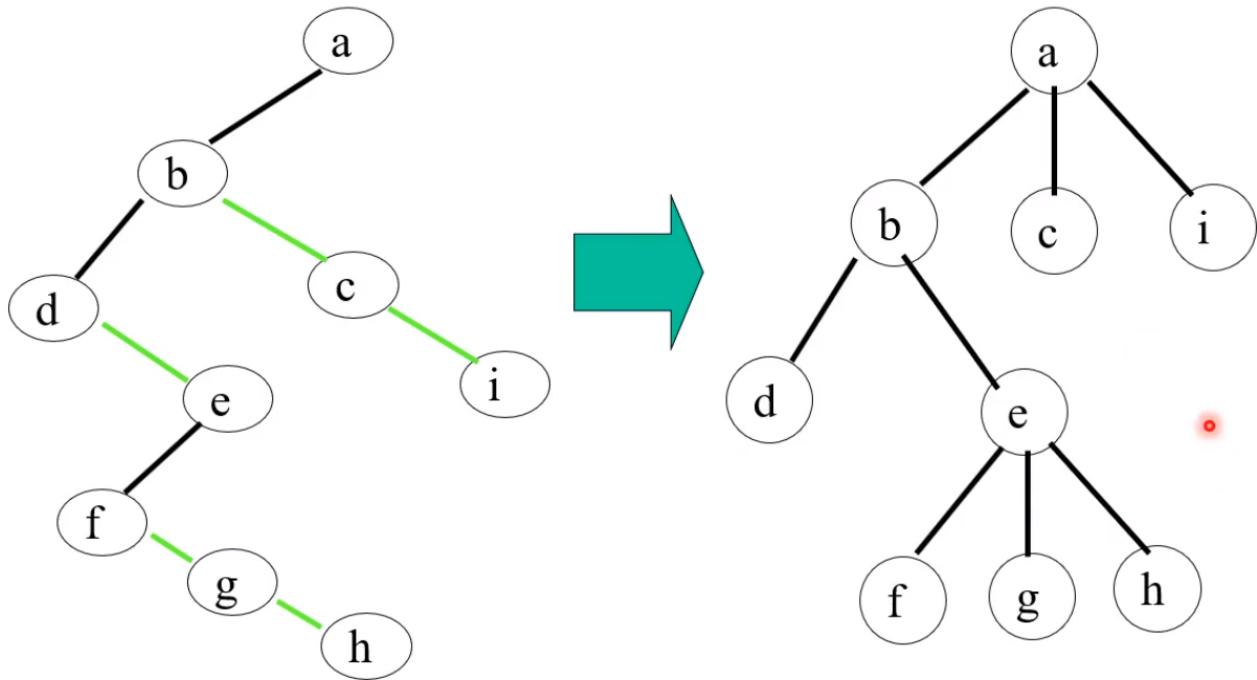
17

- 二叉树转换成树

- (1) 所有右孩子变成兄弟(不包括右孩子的左孩子)
- (2) 将兄弟与左孩子的双亲连线
- (3) 层次调整

## 回顾2：二叉树怎样还原为树？

要点：逆操作，把所有右孩子变为兄弟！



- 森林转换成二叉树

如果  $F = \{T_1, T_2, T_3, \dots, T_m\}$  是森林，按以下规则转换成一棵二叉树  $B = (\text{root}, \text{LB}, \text{RB})$

- (1) 若  $F$  为空，即  $m = 0$ ，则  $B$  为空树；
- (2) 若  $F$  非空，即  $m \neq 0$ ，则  $B$  的根  $\text{root}$  即为森林中第一棵树的根  $\text{ROOT}(T_1)$ ； $B$  的左子树  $\text{LB}$  是从  $T_1$  中根结点的子树森林  $F_1 = \{T_{11}, T_{12}, T_{13}, \dots, T_{1m}\}$  转换而成的二叉树；其右子树  $\text{RB}$  是从森林  $F' = \{T_1, T_2, T_3, \dots, T_m\}$  转换而成的二叉树

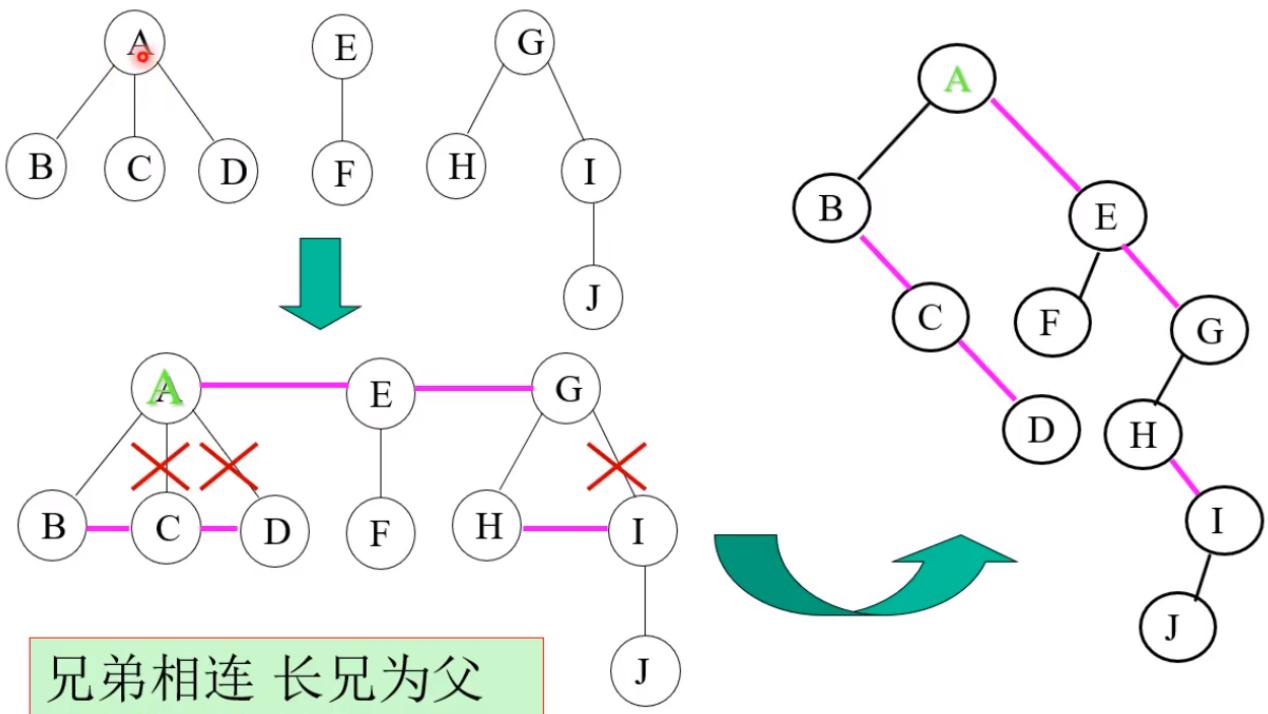
- 做题方法1：

- (1) 森林中的每棵树各自转换成二叉树
- (2) 将所有的二叉树转换成一棵二叉树（如下）
  - (2.1) 第一棵树不变
  - (2.2) 将第二棵树作为第一棵树的根的右孩子
  - (2.3) 将第三棵树作为第二棵树的根的右孩子，以此类推

- 做题方法2：

## 森林转二叉树举例：

(用法二，森林直接变兄弟，再转为二叉树)



- 二叉树转换成森林

如果  $B=(\text{root}, LB, RB)$  是一棵二叉树, 按以下规则转换成森林  $F=\{T_1, T_2, T_3, \dots, T_m\}$

- (1) 若  $B$  为空, 则  $F$  为空;
- (2) 若  $B$  非空, 则  $F$  中第一棵树  $T_1$  的根  $\text{ROOT}(T_1)$  即为二叉树  $B$  的根  $\text{root}$ ;  $T_1$  中根结点是子树森林  $F_1$  是由  $B$  的左子树  $LB$  转换而成的森林;  $F$  中除  $T_1$  之外其余树组成的森林  $F'=\{T_2, T_3, \dots, T_m\}$  是由  $B$  的右子树  $RB$  转换而成的森林

- 做题方法:

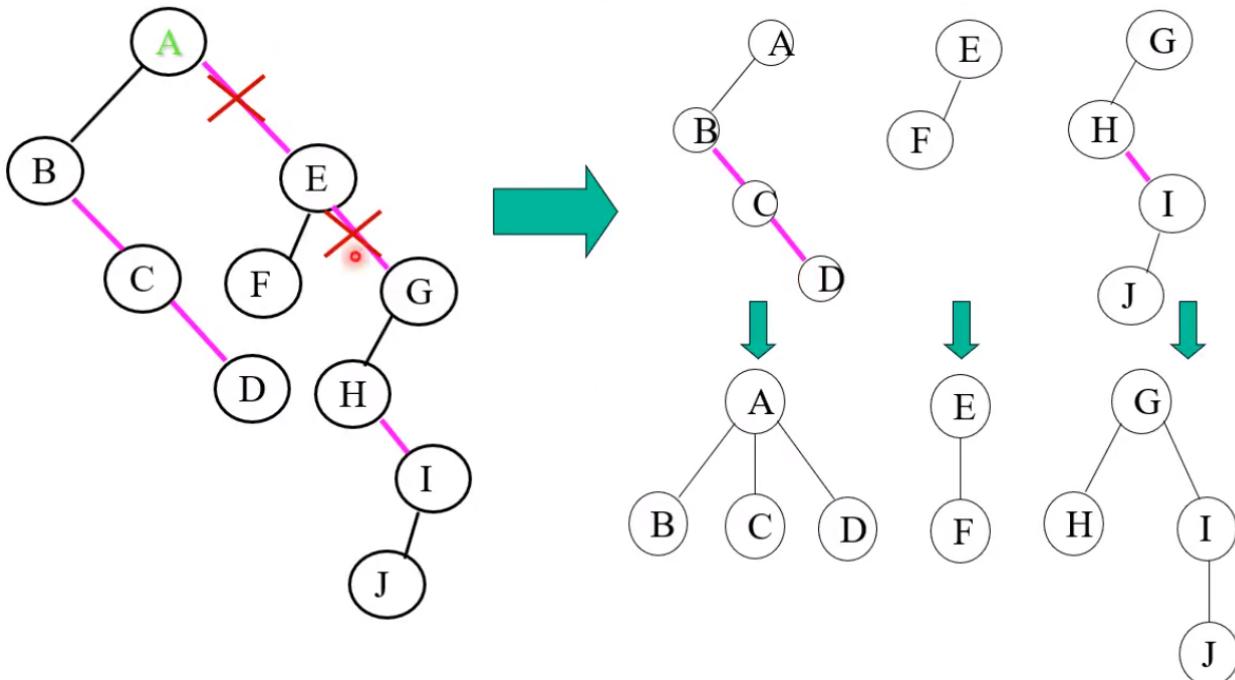
二叉树能否转化成森林看是否有右孩子? 有  $\checkmark$  则可以转化成森林, 没有  $\times$  则只能转化成树

- (1) 根与右孩子去线
- (2) 右孩子的右孩子间去线, 以此类推
- (2.1) 得到  $n$  棵二叉树, 将二叉树转换成树

## 讨论2：二叉树如何还原为森林？

即  $B = \{\text{root}, LB, RB\} \longleftrightarrow F = \{T_1, T_2, \dots, T_m\}$

要点：把最右边的子树变为森林，其余右子树变为兄弟



- 先序遍历森林(森林非空)
  - (1)访问森林中第一棵树的根结点;
  - (2)先序遍历第一棵树根结点的子树森林;
  - (3)先序遍历除第一棵树之后剩余的树构成的森林;
- 中序遍历森林(森林非空)
  - (1)中序遍历森林第一棵树的根结点的子树森林;
  - (2)访问第一颗树的根结点;
  - (3)中序遍历除第一棵树之后剩余的树构成的森林;

## 第7章 图

### 图的定义

- 无向图
  - 完全图: 任意顶点间存在边(直接)
    - 边的取值范围:  $0 \sim n*(n-1)/2$
  - 连通无向图: 任意顶点间存在路径(间接)
    - $n$ 个顶点的连通无向图, 其边的个数至少为  $n-1$

- 有向图

- 有向完全图

- 边的取值范围:  $0 \sim n^*(n-1)$

- 强连通图

- $n$ 个顶点的强连通图, 有向边的个数至少为 $n$

## 最小生成树

- 普里姆算法 (Prim)

- 以某顶点为起点,选取其权值最小的邻边

- 克鲁斯卡尔算法 (Kruskal)

- 选取图中权值最小的边, 不构成连通分量

## 拓扑排序

(1) 在有向图中选一个没有前驱的顶点且输出之

(2) 从图中删除该顶点和所有以它为尾的弧

(3) 重复以上两步, 直至全部顶点均已输出, 或者当前图中不存在无前驱的顶点为止(存在环)

## 最短路径

单源点最短路径指给定带权有向图G和源点 $V_0$ , 求从 $V_0$ 到G中其余各顶点的最短路径

- 迪杰斯特拉(Dijkstra)

- (1) S集合的初态只包含 $V_0$ , T集合的初态为网中除了 $V_0$ 之外的所有顶点

- (2)按各顶点与 $V_0$ 间**最短路径长度递增的次序**(1,2,3,...),逐个把T集合中的顶点加入到S集合中去  
使得从 $V_0$ 到S集合中各顶点的路径长度始终不大于从 $V_0$ 到T集合中各顶点的路径长度

终点	1	2	3
$V_1$	$\infty$	$\infty$	$\infty$
$V_2$	$\langle V_0, V_2, 100 \rangle$	$\langle V_0, V_3, V_2, 40 \rangle$	
$V_3$	$\langle V_0, V_3, 30 \rangle$		
集合S	$\{V_0, V_3\}$	$\{V_0, V_3, V_2\}$	

终点	1	2	3	4	5
B	$\langle A, B, 3 \rangle$				
C	$\infty$		$\langle A, B, C, 15 \rangle$	$\langle A, B, E, 14 \rangle$	
D	$\langle A, D, 15 \rangle$	$\langle A, D, 15 \rangle$	$\langle A, D, 15 \rangle$		
E	$\langle A, E, 10 \rangle$	$\langle A, E, 10 \rangle$			
F	$\infty$	$\infty$	$\langle A, B, E, F, 18 \rangle$	$\langle A, B, E, C, F, 16 \rangle$	$\langle A, B, E, C, F, 16 \rangle$
集合S	$\{A, B\}$	$\{A, B, E\}$	15 $\{A, B, E, C\}$	10 $\{A, B, E, C, D\}$	6 $\{A, B, E, C, D, F\}$

对如下带权有向图，采用迪杰斯特拉(Dijkstra)算法求顶点 A 到其余各顶点的最短路径的顺序是什么？ (10 分)

```

graph LR
    A((A)) -- 3 --> B((B))
    A((A)) -- 15 --> C((C))
    A((A)) -- 15 --> D((D))
    A((A)) -- 10 --> E((E))
    B((B)) -- 3 --> C((C))
    B((B)) -- 12 --> D((D))
    B((B)) -- 6 --> E((E))
    B((B)) -- 5 --> F((F))
    C((C)) -- 4 --> D((D))
    D((D)) -- 8 --> E((E))
    D((D)) -- 9 --> F((F))
    E((E)) -- 2 --> F((F))

```

## 第9章 查找

### 分块查找法(索引顺序表的查找)

1. 对每个子表(或称块)建立一个索引项, 其中包括两项内容:

关键字项: 其值为该子表内的最大关键字

指针项: 指示该子表的第一个记录在表中位置

2.  $ASL = 1/2(n/s + s) + 1$

n: 表长

s: 每块含有s个记录

## B-树

m阶的B-树满足如下特性(m阶指最多有m棵子树):

1. 树中每个结点最多有m棵子树
2. 若根不是叶子结点, 则至少有2棵子树
3. 除根以外的所有非终端结点至少有 $m/2$ (向上取整)棵子树
4. 所有非终端节点都包含以下信息数据:

(n, A<sub>0</sub>, K<sub>1</sub>, A<sub>1</sub>, K<sub>2</sub>, A<sub>2</sub>, ..., K<sub>N</sub>, A<sub>N</sub>)

其中:

A<sub>i-1</sub>所指子树中所有结点的关键字均小于K<sub>i</sub>

$A_n$ 所指子树中所有结点的关键字均大于  $K_n$

$n$ 为关键字个数: (向上取整) $m/2 \leq n \leq m-1$

- 所有叶子结点都出现在同一层次并且不带信息,实际上这些结点不存在,指向这些结点的指针为空

## 哈希表

### 开放定址法&线性探测再散列

- $H_i = (H(key) + d_i) \text{ MOD } m$ , 其中  $i = 1, 2, 3, \dots, k$  ( $k \leq m-1$ )

$H(key)$ 为哈希函数;  $m$ 为哈希表表长;  $d_i$ 为增量序列;

增量序列的取法:

- (1)线性探测再散列:  $d_i = 1, 2, 3, \dots, m-1$ ;
- (2)二次探测再散列:  $d_i = 1^2, -1^2, 2^2, -2^2, \dots, \pm k^2$  ( $k \leq m/2$ )
- (3)伪随机探测再散列:  $d_i$ =伪随机数序列

## 邻接表

adjvex	nextarc	info
--------	---------	------

- 邻接点域(adjvex): 指示与顶点  $V_i$ 邻接的点在图中的位置
- 链域(nextarc): 指示下一条边或弧的结点
- 数据域(data): 存储和边或弧相关的信息,如权值等

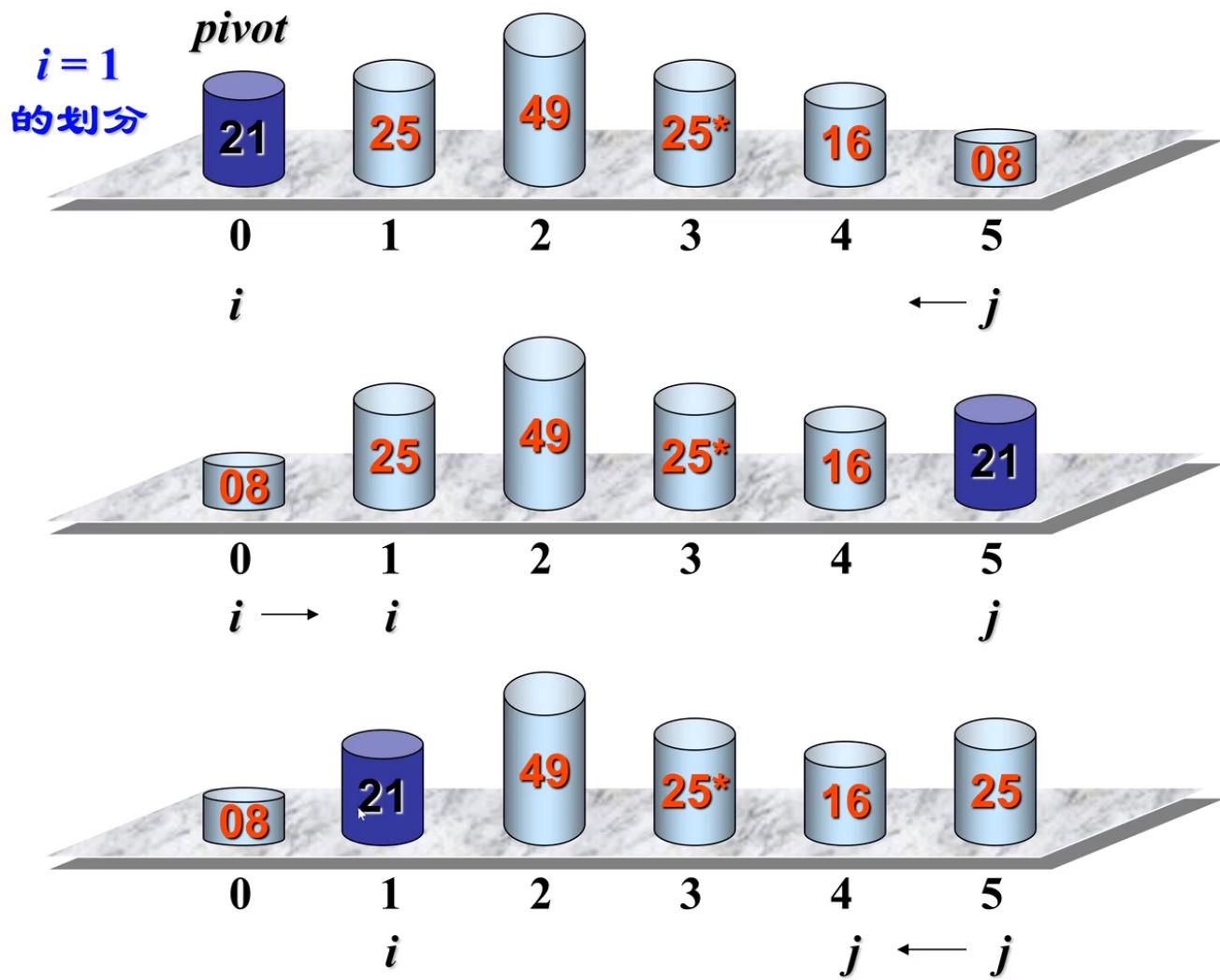
若无向图中有  $n$ 个顶点,  $e$ 条边, 则它的邻接表需要  $n$ 个头结点和  $2e$ 个表结点

# 第10章 内部排序

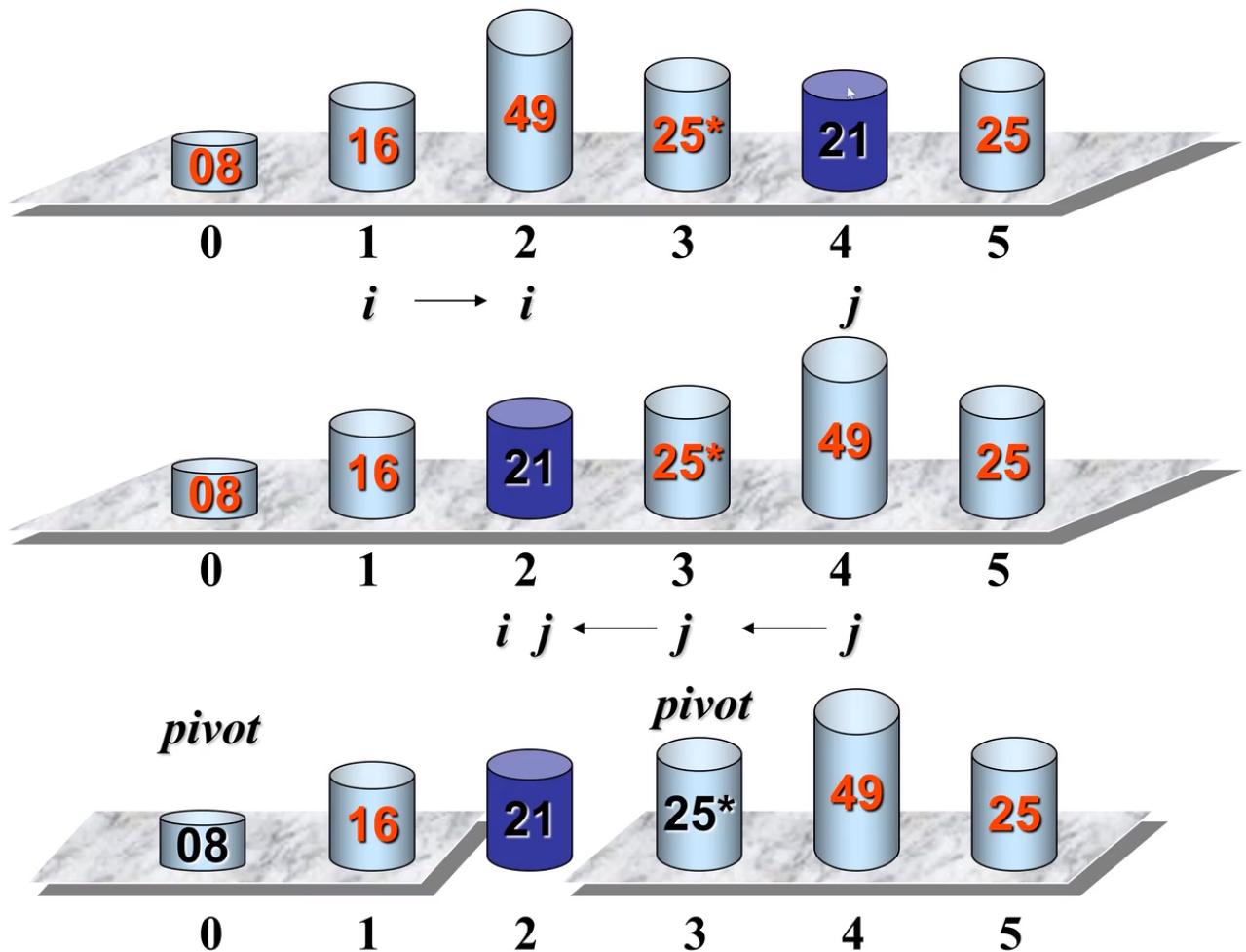
## 各种排序方法的比较

排序方法	比较次数		移动次数		稳定性	附加存储	
	最好	最差	最好	最差		最好	最差
直接插入排序	n	$n^2$	0	$n^2$	√	1	
折半插入排序	$n \log_2 n$		0	$n^2$	√	1	
起泡排序	n	$n^2$	0	$n^2$	√	1	
快速排序	$n \log_2 n$	$n^2$	$n \log_2 n$	$n^2$	✗	$\log_2 n$	$n^2$
简单选择排序	$n^2$		0	n	✗	1	
锦标赛排序	$n \log_2 n$		$n \log_2 n$		√	n	
堆排序	$n \log_2 n$		$n \log_2 n$		✗	1	
归并排序	$n \log_2 n$		$n \log_2 n$		√	n	

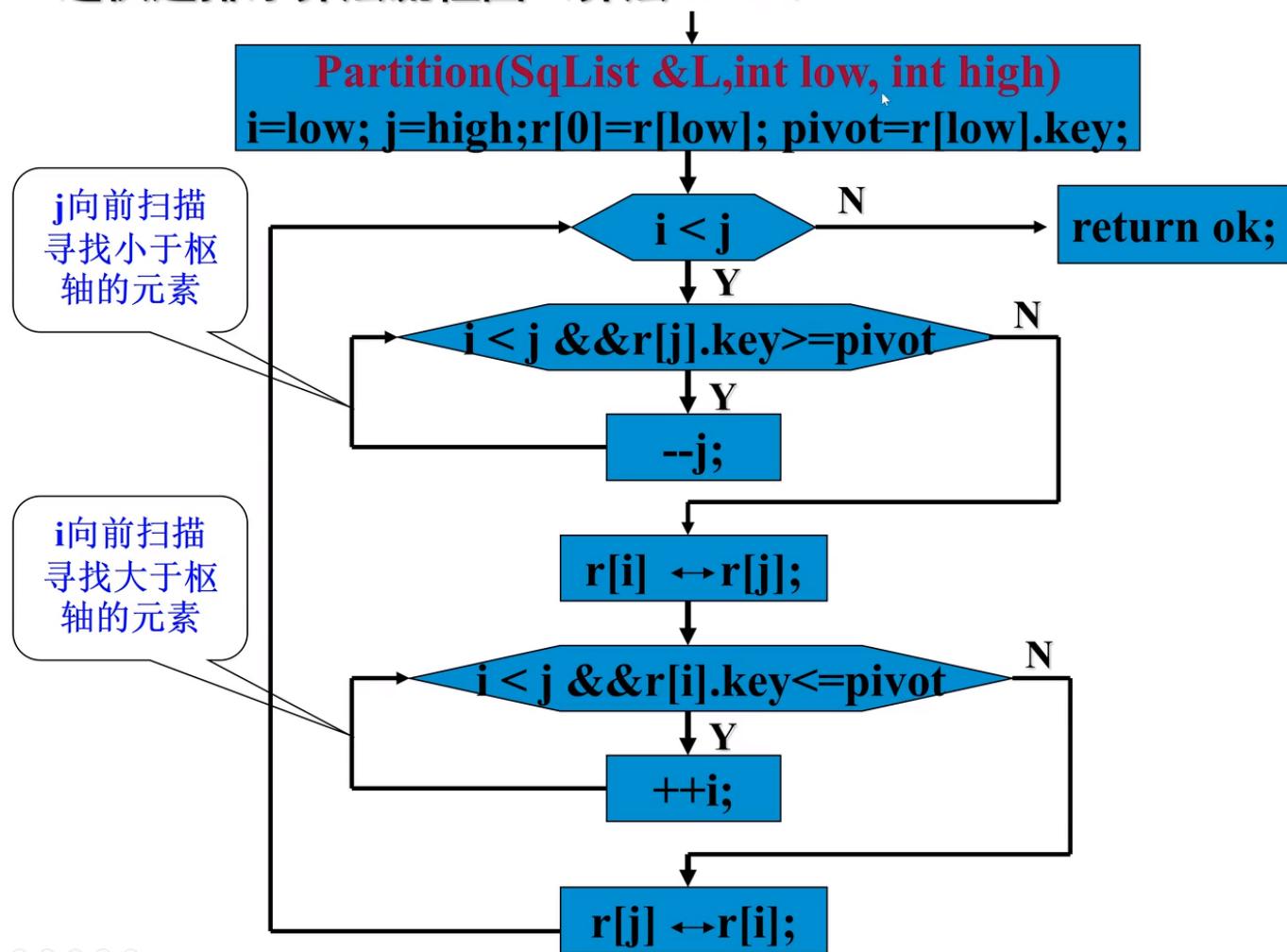
# 快速排序



↓ ↓ ↓ ↓ ↓



# 一趟快速排序算法流程图（算法10.6a）



## 快速排序的算法

```
void QuickSort ( datalist &list, int low, int high ) {  
    //对表list中list[ low~high] 进行快速排序  
    if ( low < high ) {  
        pivot = Partition ( list, low, high ); //一分为二  
        QuickSort ( list, low, pivot-1);  
        //在左子区间进行快速排序  
        QuickSort ( list, pivot+1, high );  
        //在右子区间进行快速排序  
    }  
}
```

注：调整QuickSort先后次序可以有助于减少递归深度！

## 堆排序

- 需要一个记录大小的辅助空间, 每个待排序记录仅占一个存储空间
- 最坏情况下的时间复杂度:  $O(n \log_2 n)$

## 其它

- 在一个稀疏矩阵中, 每个非零元素 $a_{ij}$ 所对应的三元组表示为( $i, j, a_{ij}$ )

## 静态链表

- 静态链表是用数组来实现链式存储结构, 目的是方便在不设指针类型的高级程序设计语言中使用链式结构。

优点是和动态链表一样, 删除和插入元素时间复杂度低;

不足是和数组一样, 需要提前分配一块较大的空间。

# 后缀表达式

例:  $a^*(b+c)-d$

转化为后缀表达式 (逆波兰式) :

从左至右依次遍历中缀表达式各个字符 (需要准备一个字符栈存储操作符和括号)

1、字符为 运算数 :

直接送入后缀表达式 (注: 需要先分析出完整的运算数)。

2、字符为 左括号 :

直接入栈 (注: 左括号入栈后优先级降至最低)。

3、字符为 右括号 :

直接出栈, 并将出栈字符依次送入后缀表达式, 直到栈顶字符为左括号 (左括号也要出栈, 但不送入后缀表达式)。

4、字符为 操作符 :

若栈空, 直接入栈。

若栈非空, 判断栈顶操作符, 若该操作符低于栈顶操作符, 入栈; 否则一直出栈, 并将出栈字符依次送入后缀表达式, 直到栈空或栈顶操作符优先级低于该操作符, 该操作符再入栈。

总结: 只要满足 栈空 或者 优先级高于栈顶操作符 即可停止出栈, 并将该操作符入栈