2012 年版
数据结构导论

02142

学校：邮　电

作者：马明洋

计算机及应用专业：20230929

数据结构导论 02142

1. 概论

1.1. 基本概念和术语

1.1.1. 数据，数据元素，数据项

(1) 数据：存储 处理 的对象

(2) 数据元素：数据的基本单位，多个数据项组成（每行数据，多个字段）

(3) 数据项：多个字段（域） 。

数据结构：存在一个多个特定关系的数据元素集合，包括数据的逻辑结构，数据的存储结构，数据的基本运算。

1.1.2. 数据的逻辑结构

- 指数据元素之间的逻辑关系（关联方式 或 ‘邻接方式’）

(1) 集合

(2) 线性结构

(3) 树形结构

(4) 图结构

1.1.3. 数据的存储结构（物理结构）

数据的逻辑结构在机器上的实现称为数据的存储结构，一般包含两个部分

(1) 存储数据元素

(2) 数据元素之间的关联方式

1.1.4. 运算

指逻辑结构上的操作，定义了一组基本运算

- 建立，查找，读取。插入，删除

1.2. 算法及描述

运算的实现 指该 运算的算法（基本概念）

一个算法规定了问题的处理步骤和及执行顺序，能在有限的时间内被解决。

1.3. 算法分析

(1) 正确性：正确实现

(2) 易读性：易于理解交流，调试，修改，补充

(3) 健壮性：输入非法数据，算法也能适当反应或处理

(4) 时空性：时间性能（效率） 和 空间性能（效率）

1.3.1. 时间复杂度

- $O(1)$ ：常数阶
- $O(\log_2 n)$ ：对数阶（n 的几次方 等于 2）
- $O(n)$ ：线性阶
- $O(n^c)$ ：多项式阶（多方，平方 常见：$O(n^2)$ ）
- $O(C^n)$ ：指数阶（常见：$O(2^n)$ ）

$O(\log_2 n) < O(n) < O(n^c) < O(C^n)$

1.3.2. 空间复杂度

- 一个算法所耗费的空间

1) 代码所占空间

2) 输入数据所占空间

3) 辅助变量所占空间

2. 线性表

2.1. 基本概念

(1) 线性结构。n(n>=0)个数据元素（结点）组成的有穷序列。

(2) 节点个数 n 为表长，n=0 为空表

(3) n!=0 ，表示为 $(a_1,a_2,a_3,...a_n)$ ，a_1 为起始结点 a_n 为终端结点。

(4) 头结点没有直接前驱，尾结点没有直接后驱。

(5) 同一个线性表所有结点的数据元素具有相同的特性

2.2. 顺序存储

2.2.1. 线性表顺序存储的类型定义

(1) 将表中所有结点依次存放在计算机一组连续的储存单元

(2) 数据元素在线性表的邻接关系决定他们存储空间的的存储位置

2.2.2. 基本运算实现

- 算法中，会将线性表定义在数组中，下标：第 i 个元素 (i-1)

插入（插入位置 后元素后退），删除（删除元素 后元素前进），定位（while 遍历）

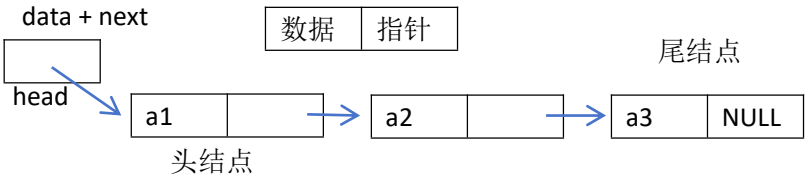
2.2.3. 顺序表的实现算法

- 元素平均移动次数约为 n-1/2 ，时间复杂度为 O(n)

2.3. 链接存储

存储结构为链式，常见有：单链表，循环链表，双向循环链表

2.3.1. 单链表的类型定义（便于实现运算，在第一个节点前添加头结点）



2.3.2. 线性表基本运算在单链表实现（while 遍历，i 为 fori，x 为数据）

- 初始化：head->next=NULL （空单链表只有一个节点，指针域为 NULL）
- 求表长：p->next !=NULL（while 判断是否尾结点） p=p->next （指针后移）
- 读表数据：p!=NULL（不可遗漏，while 判断不是尾节点），p=p->next return p
- 定位：p!=NULL p->data!=x（while 判断 x 是否为给定值）return i
- 插入：（找到第 i 个节点，在之前插入数据 x 的新节点）

(1) If (i==1) q==head; else q=GetLinklist(head, i-1);

(2) If (q==NULL) exit (“找不到”)

```
else { p=malloc( sizeof(Node)); p->data=x;
        p->next=q->next;   q->next=p;     (不能写反)
        }
```

2.4. 运算单链表实现

2.4.1. 建表

2.4.2. 删除重复节点

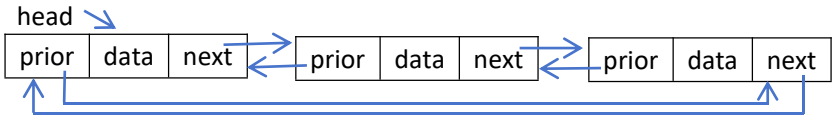
2.5. 其他链表

2.5.1. 循环链表

在单链表中让尾节点的指针域指向头结点可以构成循环链表（算法与单链表相似）

2.5.2. 双向循环链表

在每个节点在设置一个指向直接前驱的指针域 prior 每个节点有两个指针



访问前驱结点和后驱节点事件负责度均为 O(1) ，具有对称性

p = p->prior->next = p->next->prior （三个都是访问的自己）

2.6. 顺序实现与链接实现的比较

时间复杂度	查找	定位	插入，删除
顺序表	O(1)	O(n)	O(n)
单链表	O(n)	O(n)	O(n)

3. 栈、队列、数组

3.1. 栈

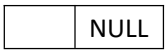
3.1.1. 基本概念

- 先进后出，后进先出

3.1.2. 顺序实现

初始化，进栈，出栈（判断是否下溢->栈空：EmptyStack(stk) ）

3.1.3. 链接实现

称为链栈，可以用带头节点的单链表实现 LS-> 

首节点就是栈项定点，LS->next 指向栈顶节点（LS 为头结点），尾节点为栈底节点

新增节点始终插入到头结点之后

3.1.4. 简单应用 和 递归

简单应用：进栈 ABCDEFG，出栈一 ABCDEFG，出栈二 GFEDCBA，出栈三 ABCGFED

递归：函数嵌套函数计算，直到函数计算结果满足要求

3.2. 队列

3.2.1. 基本概念

- 先进先出，后进后出 的线性表，新加入不允许插队，只能按顺序出队

3.2.2. 顺序实现

由一个一维数组及两个分别指示队首和队尾元素的变量组成

队列首指针 **front** 和队列尾指针 **rear**

- 入队列：SQ.rear=SQ.rear+1;SQ.data[SQ.rear]=x
- 出队列：SQ.front=SQ.front+1;

◆会出现问题：

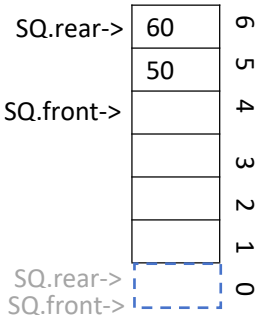
当队列入队满后，

并且全部出队，

新入队时会判断

队尾指针已满

此现象为假溢出



- 循环队列

环状队列，当 SQ.rear=maxsize-1 时（maxsize 为队列总容量 0~6=7），后续有空间，则令 SQ.rear=0，把 SQ.data[0]作为新的队尾。

此无法判断队列满或空

则：空余一个队列元素，满时有一个元素为空，空队列时 front==rear

- **循环队列判断条件**

(1) 队列满：**((CQ.rear=1)%maxsize==CQ.front)** 成立

(2) 队列空：**(CQ.rear==CQ.front)** 成立

(3) 数组进出队列：**rear=(rear+1)%max　front=(front+1)%max**

3.2.3. 链接实现

- 用一个带有头结点的单链表 表示队列，称为链队列
- 头指针指向头结点，头结点 next 指向队列首节点，尾指针指向队列尾节点

3.2.4. 简单应用

3.3. 数组

3.3.1. 逻辑结构和基本运算

- 由一组相同类型的数据元素组成，并储存在一组连续的存储单元中。
- 若一维数组中的数据元素是一维数组，称为二维数组。
- 一个 n 维数组可以看成元素为 n-1 维数组的线性表。

3.3.2. 存储结构

- 一维数组的内存单元地址是连续的，二维数组可以有两种存储方法：列序为主，行序为主，
- 二维数组 a[m][n]，每个元素占 k 储存单位，行为主序，讨论 a[i][j]

(1) i 行之前，每行有 n 个元素；第 i 行有 j+1 个元素

(2) 总共有 n*i+j+1 个元素

(3) 第一个元素与 a[i][j]相差 n*i+j+1-1 个位置

(4) a[i][j] 的位置为 loc[i,j] = loc[0,0] + (n*i+j)*k

3.3.3. 矩阵的压缩存储

3.3.3.1. 特殊矩阵

(1) **对阵矩阵** （矩阵展示如 两侧 for 循环的 九九乘法表）

- 方阵 A 满足：$a_{ij} = a_{ji}$　0<=i , j<=n-1　(n-1 为数组个数，起始是 0)
- 可将 n2 个元素压缩储存到含有 n(n+1)/2 个元素的一维数组中
- **k 为下标，计数 a[i][j]**

i>=j :　k = (i+1)i/2+j

I<j :　k = (j+1)j/2+i

(2) 三角矩阵

以主对角线为界，上或下半部分是一个固定的值

3.3.3.2. 稀疏矩阵

t 个非零元素，t<<m*n 时，为稀疏矩阵

4. 树 和 二叉树

4.1. 基本概念

4.1.1. 树的基本概念

线性表最多只有一个直接后继，树形结构中的一个节点可以有多个直接后继。

- 树是 n(n>=0)个节点的有限集合

(1) n=0 时，称为空树。

(2) n>0 时，有且只有一个根节点，其余节点 m 个不相交的非空集合，每个集合

都是一课树，称为很的子树。

4.1.2. 树的相关术语

(1) 结点的度：树上任一结点所拥有的子结点的数量

(2) 叶子：度为 0 的结点，称为叶子结点或终端结点

(3) 树的度：一棵树的所有结点的度的最大值为该树的度

4.2. 二叉树

4.2.1. 二叉树的基本概念

- 二叉树是 n(n>=0)个元素的有限集合，该集合或者为空，或者由一棵及两棵互不相交的左子树和右子树组成，其中左右子树均称为二叉树
- 二叉树任意节点都有两颗子树（左右子树都可为空），**左右有次序，保证唯一**

4.2.2. 二叉树的基性质

(1) 二叉树第 i(i>=0)层上至多有 2^{i-1} 个结点

(2) 深度为 k(k>=1)的二叉树至多有 2^k-1 个结点

(3) 任何二叉树度为 0 的结点（叶子）有 n_0 个，度为 2 的结点个数为 n_2：$n_0 = n_2 + 1$

■ **满二叉树**：深度为 k(k>=1)且有 2k-1 个结点的二叉树

◆ **完全二叉树**：对满二叉树按从上到下，从左到右的顺序编号，并在最后一层删除部分结点（最后一行仍有节点），删除的结点编号是连续的，且包含最大编号的结点

(1) 含有 n 个结点的完全二叉树深度为 $< \log_2 n > + 1$（< x >表示不大于 x 的最大整数）

(2) 完全二叉树编号，编号为 i (1 <= i <= n) 的结点，最大为 n，则结点 A:

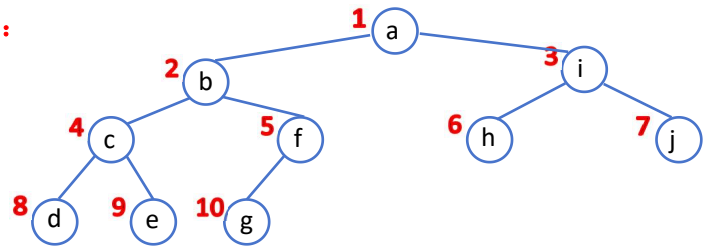
对非完全二叉树不成立，需要设立虚节点进行编号再计算

1) 若 i=1，则 A 结点为根，若 i >1，则 A 的双亲为 **< i/2 >**

2) 若 2*i > n，则 A 无孩子，否则左孩子编号为 **2*i**

3) 若 2*i + 1 > n，则 A 无右孩子，否则右孩子编号为 **2*i + 1**

完全二叉树：



完全二叉树顺序存储：

Btree:		a	b	i	c	f	h	j	d	e	g
下标:	0	1	2	3	4	5	6	7	8	9	10

4.3. 二叉树存储结构

4.3.1. 二叉树的顺序存储结构

将二叉树按编号次序，可以存储到一维数组中

4.3.2. 二叉树的链式存储结构

指向左孩子指针域	数据域	指向右孩子指针域
ichild	data	rchild

4.4. 二叉树遍历

4.4.1. 遍历的递归实现

(1) 先序遍历：（访问到那个节点 就是 程序获取到那个节点）

◆ 获取级别：（ 深度优先 > 根优先 > 左次之 > 右最后 ）访问每结点

根节点开始，优先访问左子树（深度优先）

访问左子树结点子树（左优先 左空则访问右），子树非空接着下访子树

若左右子树都为空，返回上一层并访问右子树：

若上一层右子树不为空，则下访子树（左优先，深度优先）

上一层右子树为空，接着返回上一层

(2) 中序遍历：（返回 才是 程序先获取到的）

◆ 获取级别：（ 深度优先 > 左优先 > 根次之 > 右最后 ）访问每结点

设 根左右 三个结点为 当前树 （根为首）

访问根节点左子树，优先访问当前树的左子树（左优先，深度优先）

左不为空则接着访问当前树的左子树（设~左树变根树）

当前树的左子树为空则返回当前根节点，再访问当前树的右子树

右树为空，返回上一级根树，并访问上一级右树

右树非空，访问右树左子树（左优先，深度优先）

(3) 后续遍历：（返回 才是 程序先获取到的）

◆ 获取级别：（ 深度优先 > 左优先 > 右次之 > 根最后 ）访问每结点

根节点开始，优先访问左子树（深度优先）

访问左子树结点子树（左优先 左空则访问右），子树非空接着下访子树

左右子树为空，返回当前根节点，并访问当前根上一级节点右子树

右子树全部访问完成，返回右子树节点，再返回当前根节点

4.4.2. 层次遍历

◆ 二叉树从根节点开始,逐层向下遍历,每一层从左向右遍历

4.4.3. 遍历的非递归实现

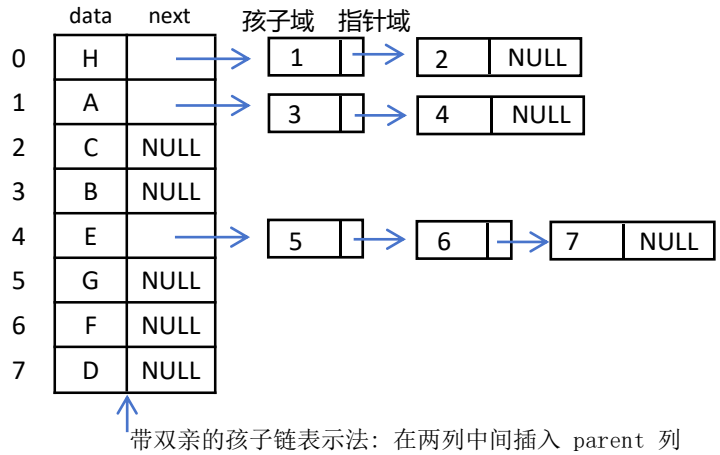
先中后序遍历实现

4.5. 树和森林

4.5.1. 树的存储结构

4.5.1.1. 孩子链表表示法:

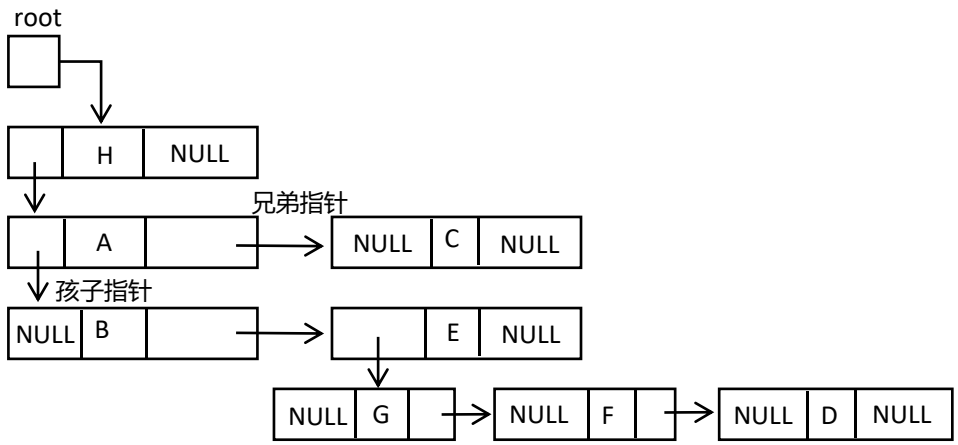
表头数组: Link



带双亲的孩子链表示法：在两列中间插入 parent 列

并将该节点的双亲位数填入 根节点为-1, 如 H 后为-1,A 后为 0

4.5.1.2. 孩子兄弟链表示法:



4.5.1.3. 双亲表示法:

	data	parent
0	H	-1
1	A	0
2	C	0
3	B	1
4	E	1
5	G	4
6	F	4
7	D	4

4.5.2. 树，森林，二叉树 的关系

4.5.2.1. 树转换成二叉树

1) 将所有兄弟结点连接
2) 保留第一个兄弟结点与父结点的连接,断开其他兄弟结点与父结点的连接,然后以根节点为轴心顺时针旋转 45 度

4.5.2.2. 深林转换成二叉树

1) 将每棵树转换成二叉树
2) 将每第二棵树的根结点都与第一个数的根的子结点为兄弟关系连接根节点

4.5.2.3. 二叉树转换成森林

1) 断开二叉树上所有结点与该结点的右孩子的连接, 知道断开树的根结点没有右孩子
2) 将断开树的子孩子右结点(不论层) 都与根结点相连, 断开原父子关系

4.5.3. 树和森林的遍历

4.5.3.1. 树的遍历

1. 先序遍历
   访问根结点 ；依次先序遍历根结点子树 $T_1 \dots T_m$
2. 后序遍历
   依次后序遍历根结点子树 $T_1 \dots T_m$ ；访问根结点
3. 层次遍历
   先访问根结点；依次向下层访问 每层结点(i 层已访问,则访问 i+1 层结点)

4.5.3.2. 森林的遍历

1. 先序遍历
   1) 访问森林中第一棵树的根结点
   2) 先序遍历森林中第一个树的根结点子树组成的森林
   3) 先序遍历其他森林
2. 中序遍历
   1) 中序遍历森林中第一棵树的根结点组成的森林
   2) 访问根结点
   3) 中序遍历其他森林

4.6. 判定树和哈曼夫树

4.6.1. 分类与判定树

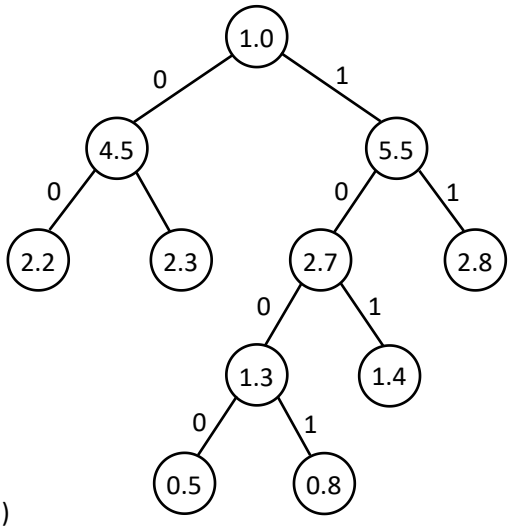
从根结点开始判断 是或否 Yes Or No
No 走左树接着判断 Yes 走右树接着判断

4.6.2. 哈夫曼树和哈夫曼算法

1) 将给出每个权(大小/概率)分别作成只有根结点的森林
2) 取最小的两个(根结点)权作为二叉树的左右子树，根结点为两个权之和
3) 将第 2)部产生的二叉树加入森林，若剩下的权仍大于一棵二叉树，重复第 2)部

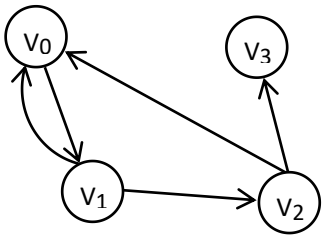
4.6.3. 哈夫曼编码

频率	字符编码
0.5	1000
0.8	1001
1.4	101
2.2	00
2.3	01
2.8	11



每个结点的哈曼夫编码:
从根结点到该结点所经过的权(路径)
权:
路径值(左为 **0** , 右为 **1**)

5. 图



有向图

5.1. 基本概念

5.1.1. 应用背景

最低的网络通信，最便捷的城市交通

5.1.2. 定义 和 术语

- 图 G 由两个集合 V 和 E 组成, G=(V,E)
- V 是顶点的有穷非空集合　E 是边的集合　；　边是 V 中顶点的偶对

1. 有向图 < > ：**顶点偶对指向有序(有方向，箭头)**

   有向图从顶点 v 到顶点 w 有一条边

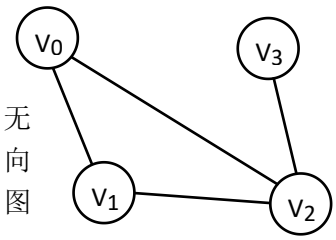
   弧，弧头，弧尾 ：　v w 为弧 ；　v 弧头 ；　w 弧尾

   即:　v, w∈V　<v, w>∈E

2. 无向图 ( ) ：**顶点偶对指向无序(无方向，连接线)**

   无向图从顶点 v 和顶点 w 间有一条边

   即:　v, w∈V　(v, w)∈E



无向图

3. 完全无向图

   任何两点之间有边的无向图

   n 个顶点的完全无向图的边数为　$C_n^2 = n(n-1)/2$

4. 完全有向图

   任何两点之间有弧的有向图

   n 个顶点的完全有向图的边数为　$P_n^2 = n(n-1)$

5. 权 ：带权图 ：图的边附带数值 ，值 就叫做权
6. 顶点的度
   1) 度 ：顶点 v 的度 为与该顶点相关联的边的数目；记为　D (v)
   2) 入度(有向图)：以顶点 v 为终点的 弧 的数目；记为 ID(v)
   3) 出度(有向图)：以顶点 v 为始点的 弧 的数目；记为 OD(v)
7. 子图 ：设 G=(V,E)；有 G`=(V`,E`) ，V`为 V 子集 ；　E`为 E 子集
8. 路径，路径长度 ：顶点 V1 到顶点 Vi

   无向图: (V1, V2), (V2, V3), (V,3 V4), (V4, Vi)

   有向图: <V1, V2>, <V2, V3>, <V3, V4>, <V4, Vi>
9. 简单路径 ：序列中不出现重复路径
10. 回路 ：路径中 第一个顶点和最后一个顶点相同的路径
11. 简单回路 ：除了头尾顶点,其他顶点不重复的回路
    - ◆ 无向图 称为 连通
12. 连通图 ：无向图中任意两点都有路径，称为连通图
13. 连通分量 ：存在多个不连通 顶点，可以从图中分出多个 连通分量

    连通分量 是无向图中 极大连通子图
    - ◆ 有向图 称为 强连通
14. 强连通图 ：有向图中,任意两个顶点都互相有到对方的路径
15. 强连通分量 ：有向图的 极强大连通子图 称为 强连通分量
16. 生成树，生成深林:一个连通图的生成树,是该图全部顶点的一个极小连通子图

若连通图 G 的顶点个数为 n,则 G 的生成树的边数为 n-1

若 G 的一个子图 G`的变数大于 n-1, 则 G`中一定有环, 变数少于 n-1, G`一定不连通

5.2. 图的存储结构

5.2.1. 邻接矩阵

通常用二维数组来实现矩阵　连接为 1 否为 0

有向图　和　无向图　的 邻接矩阵 M1 和 M2：　带权图:M3,设每边权为 10

- **无向图 邻接矩阵**是一个**对称矩阵**

M1

始 \ 终	V_0	V_1	V_2	V_3
V_0	0	1	0	0
V_1	1	0	1	0
V_2	1	0	0	1
V_3	0	0	0	0

M2

	V_0	V_1	V_2	V_3
V_0	0	1	1	0
V_1	1	0	1	0
V_2	1	1	0	1
V_3	0	0	1	0

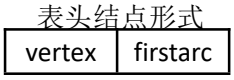
M3

	V_0	V_1	V_2	V_3
V_0	∞	10	∞	∞
V_1	10	∞	10	∞
V_2	10	∞	∞	10
V_3	∞	∞	∞	∞

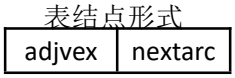
5.2.2. 邻接表

顺序存储与链式存储相结合的存储方法

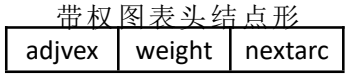
表头结点形式



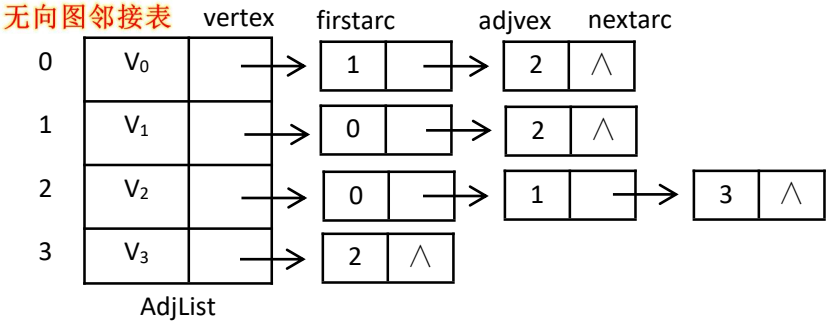
表结点形式



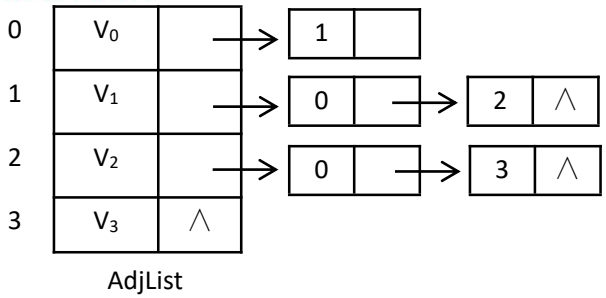
带权图表头结点形



无向图邻接表



有向图邻接表



有向图逆邻接表:

将路径顺序颠倒，变为谁到该顶点

若一个无向图有 n 个顶点, e 条边, 那么他的邻接表需要 n 个头结点和 2e 个表结点

在边稀疏 **(e<<n(n-1)/2)** 情况下, 邻接表 比 矩阵存储 省空间

- **无向图:** 顶点的**度** V_i 为 第 i 个单链表中的结点数
- **有向图:** 顶点的**出度** V_i 为 第 i 个单链表中的结点数

5.3. 图的遍历

5.3.1. 连通图的深度优先搜索

时间复杂度为 $O(n^2)$

5.3.2. 连通图的广度优先搜索

访问所有连接顶点,再访问第一个顶点的连接结点...

5.4. 图的应用

5.4.1. 最小生成树

构造最小生成树算法:

Prim 算法

克鲁斯卡尔方法

- 连通图一次遍历所经过的边的集合及图中所有顶点的集合就构成该图的生成树(不唯一)
- 对于有 n 个顶点的无向图, 所有生成树中都有且仅有 n-1 条边

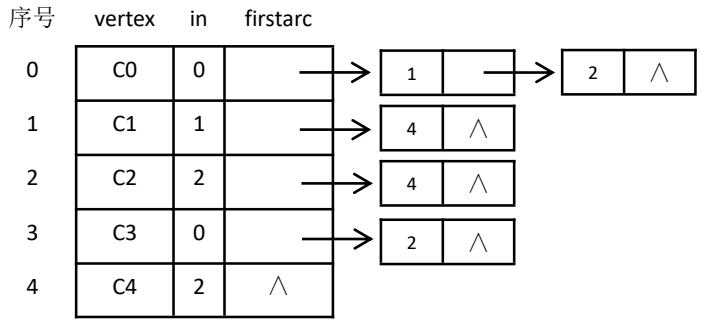
5.4.2. 拓扑排序

拓扑排序就是各个顶点或作业的**依赖关系**

AOV 网 :

1) 首先找出入度为 0 的顶点, 输出该顶点
2) 删除该顶点 和 弧, 将弧头顶点入度 -1
3) 重复 1) 2), 直到入度为 0 的顶点全部输出, **拓扑序列**完成

任何一个**无环 有向图** 其全部顶点可以排列为一个**拓扑序列**

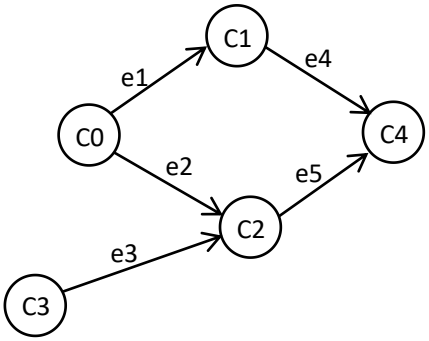


拓扑排序邻接表

拓扑排序邻接表 :

在表头结点增加数据域 in 表示入度

拓扑排序有向图



6. 查找

6.1. 基本概念

根据给定值,在数据集中查找一个键值等于给定值的数据元素

6.2. 静态查找表

6.2.1. 顺序表 上的查找

- 从最后一个元素开始，从后往前依次比较　平均查找长度为: **ASL = n+1/2**

6.2.2. 有序表 上的查找

按照键值大小排序的顺序表称为有序表，可以用效率更高的 **二分查找法**

10	13	17	20	30	55	68	89	95
low				mid				high

查找　17　**low + high /2**　取整数

1)　第一次检索 ：

low + high /2 = (1 + 9)/2 = 5　第 5 个值为 30 --比较-- 17<30 -- high=mid-1

2)　第二次检索 ：

low+(high=mid-1) /2=(1+(5-1))/2=2　第 2 个值 13; 17>13--low=mid+1

3)　第三次检索 ：

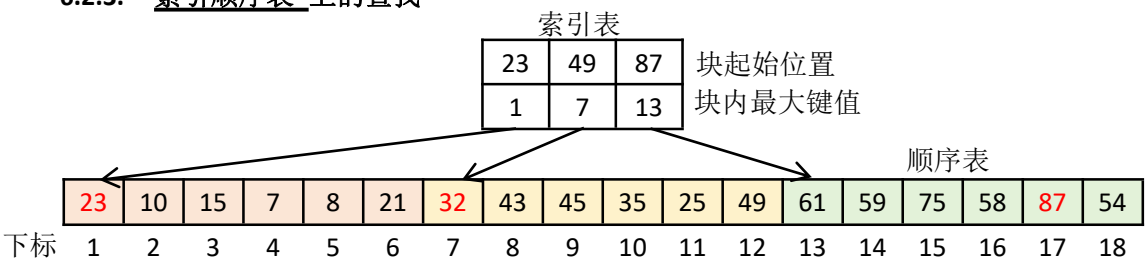
(low=mid+1)+(high=mid-1) /2 =

((2+1) + (5-1)) /2= (3+4)/2 = 3　第 3 个值为 17　已查找完成

- 二分查找法平均查找长度为: **ASL = ((n+1)/n * $\log_2$(n+1)) -1**

当 n 较大时平均查找长度为: **ASL ≈ ($\log_2$(n+1)) -1**

6.2.3. 索引顺序表 上的查找



查找　key=43

1)　先确定待查数据元素所在块 ：23<43<49　(顺序查找)

2)　然后在块内从起始位置顺序查找 ：7 8 9...　8 为 43，查到 key

每个块包含 **s** 个元素，顺序表中元素数目 **n**

- 分块查找的平均查找长度为 ：**ASL = 1/2 * (n/s + s) + 1**

1 000 000 个数据元素, 1000 个块，每块 1000 个数据元素 ：

顺序 ：　≈500 000

二分 ：　≈19

索引顺序 ：　= 1001

效率: 顺序 < 二分 < 索引顺序

要求: 顺序 < 二分 < 索引顺序

6.3. 二叉排序树

二叉排序树的平均查找长度介于 O(n) 和 $O(\log_2^n)$

1) 若他的左子树不空，则左子树上所有结点的键值都小于根结点键值
2) 若他的右子树不空，则右子树上所有结点的键值都大于根结点键值
3) 跟的左右子树也分别为二叉排序树

6.4. 散列表

6.4.1. 常用散列法

将键值集合任意键值，经过散列函数映射到地址集合中任意地址概率相等

1) **数字分析法**：将所有键值排列在一起，取其中分布比较均匀的位数
2) **除留余数法**：取不大于键值的正整数 p，将键值除以 p，取余数 (摩于 p(%))
3) **平方取中法**：将键值作平方计算，取其中几位作键值地址
4) **技术转换法**：将键值看成其他进制数，在转换回原进制数据

6.4.2. 散列表的实现

1) **线性探测法**：将键值%，得 d 存放在散列表，若位置已存在，则++

   d+1, d+2, … , m-1 , 0, 1, … , d-1　　(m 为容量(个数))
2) **二次探测法**：将键值%，得 d 存放在散列表，若位置已存在，则 $+k^2$, $-k^2$; ++

   $d=(d+1^2)$, $d=(d-1^2)$, $d=(d+2^2)$, $d=(d-2^2)$, $d=(d+k^2)$, $d=(d-k^2)$
3) **链地址法**：创建散列表，%p 的值为第几行散列表的单链表，并向后追加

   完全避免散列表堆积问题
4) **多重散列法**：将键值%，得 d 存放在散列表，d 冲突时再次进行%并+1
5) **公共溢出区法**：创建两张表,第一张表为单链表，发生冲突时存放在第二张表

6.4.3. 散列表的基本操作算法

7. 排序 (默认 重复数据 后出现的比先出现的大)

7.1. 概述

排序往往为检索服务

1) 内部排序：将记录全部存放在内存进行排序
2) 外部排序：数据量过大，只能在外存进行排序

7.2. 插入排序 (直接插入排序)

每个键值插入后,对数据大小等进行排序，重复数据排在重复数据之后

时间复杂度为 $O(n^2)$

7.3. 交换排序

7.3.1. 冒泡排序

1) 将所有数据存放在一起进行比较
2) 从第一个与第二个开始依次往后进行两个数据之间的比较
3) 服从 A < B，否则交换位置，将大数接着往后比较
4) 第一次结束后将最大值输出到列表最后
5) 重复操作 2) 3) 4)

7.3.2. 快速排序 (分组排序)

1) 将所有数据存放在一起进行比较
2) 默认以第一个数据作为标准，对所有数据进行比较
3) 分成两个组一个标准，左组比标准数小，标准数，右组比标准数大
4) 将左右组重复进行标准分组比较，分成更小的左右组
5) 全部组比较完成之后就是由小到大排序的数据

时间复杂度为 $O(\log_2{}^n)$

7.4. 选择排序

7.4.1. 直接选择排序

1) 将所有数据存放在一起进行比较
2) 选择第 i 个数据，对所有数据进行比较
3) 选出最小数据，交换位置，若第 i 个数据最小则位置不变 i + 1
4) 重复进行选择比较 2) 3)，直到排序完成

不适用数据较大情况 时间复杂度为 $O(n^2)$

7.4.2. 堆排序

1) **建堆：上小下大，此时二叉树顺序比大小是不满足排序条件的**
   1. 将所有数据存放在一起进行建堆，按二叉树顺序创建完全二叉树
   2. 从**最后一个节点**按二叉树顺序进行**倒序比较**，(**结点与父结点**)
   3. **结点比父结点小**则相互交换位置，结点比父结点大则跳过
   4. 最后到**根结点比较**，**根结点与左右孩子**进行比较**(三角形,三个节点)**
   5. 根节点与**其中最小**的进行**交换**，向**下移**动一层，并变成**新的子根结点**
   6. 子根结点**再组成新的**三角结点，重新比较下移，若此时为**最小**，**建堆完成**
2) **堆排序：**
   1. 建堆后的**根结点** (最小)与完全二叉树**最后一个结点**交换(最大)[最小输出]
   2. 将堆重建，**此时树上小下大**，**重新执行第 1 步**，交换最大最小
3) 重复进行选择比较 2) 3)，直到排序完成，**[最小全部输出，输出就是排序顺序]**

时间复杂度为 $O(n \log_2 n)$

7.5. 归并排序

7.5.1. 有序序列的合并

两个序列 $K_h...K_m$ $K_{m+1}...K_n$ 排序后前小后大 时间复杂度为 $O(n-h+1)$

7.5.2. 二路归并排序

将一个序列分成两个有序表,再组合成一个有序表

初始关键字	25	9	78	6	65	15	58	18	45	20
一次归并	9	25	6	78	15	65	18	58	20	45
二次归并	6	9	25	78	15	18	58	65	20	45
三次归并	6	9	15	18	25	58	65	78	20	45
四次归并	6	9	15	18	20	25	45	58	65	78