课后建议

- 学用一门高级语言的类库
- 研究里面的数据结构
- C/C++ STL （Standard Template Library，标准模板库），http://www.cplusplus.com/reference/stl/
- Java

Container class templates

Sequence containers:

array C++11	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list C++11	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Associative containers:

set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

Unordered associative containers:

unordered_set C++11	Unordered Set (class template)
unordered_multiset C++11	Unordered Multiset (class template)
unordered_map C++11	Unordered Map (class template)
unordered_multimap C++11	Unordered Multimap (class template)

第1章　绪论

1.2 基本概念和术语

数据(data): 是对客观事物的符号表示，是所有能输入到计算机中并被计算机程序处理的符号的总称。

例，整数、“对弈树”中的格局

数据元素 (data element/node/record): 数据的基本单位。

例，“对弈树”中的一个格局、书目信息中的一条书目

数据项(item/field): 一个数据元素可由若干个数据项组成。

例，一条书目信息是由书名、作者名、分类等多个数据项组成的

数据项是数据的不可分割的最小单位。

三者之间的关系：数据 > 数据元素 > 数据项

例：学生表 > 个人记录 > 学号、姓名……

数据对象：性质相同的数据元素的集合，是数据的子集。

例，整数数据对象、字母字符数据对象、学生数据对象

数据结构：存在一种或多种特定关系的数据元素的集合。

通常，数据元素都不是孤立存在的，在它们之间存在着某种关系，这种数据元素相互之间的关系称为结构。

基本结构

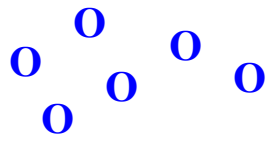
1. 集合：　　数据元素同属关系

2. 线性结构：数据元素之间一对一

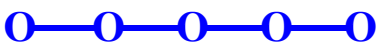
3. 树形结构：数据元素之间一对多

4. 图状结构：数据元素之间多对多

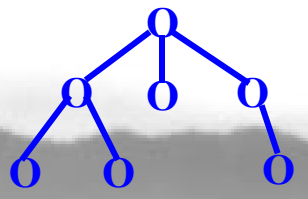
集合



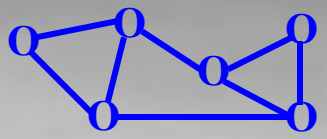
线性



树



图



数据结构的两个层次

逻辑结构：描述了数据元素之间存在的逻辑*关系*。

又称为抽象数据结构、数据的结构。[影响算法*设计*]

例，线性表中的前驱后继关系、树中的父子关系。

存储结构（物理结果）：数据结构在*计算机*内的*表示/存储*方式。

又称为存储数据结构、*物理结构*。[影响算法*实现*]

包括数据元素的表示和逻辑关系的表示

例，数据如何在计算机中存储，如何用位串表示数据？

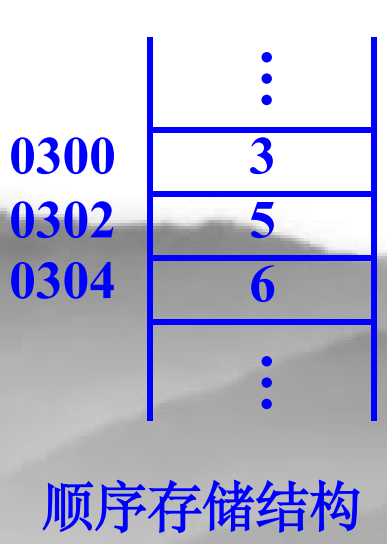
一对一关系如何存储？

数据元素之间关系的表示---逻辑关系

顺序存储结构: 借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系。[顺序映像]

链式存储结构: 借助指示元素存贮地址的指针表示数据元素之间的逻辑关系。[非顺序映像]

例，整数数组a[3]={3，5，6}



顺序存储结构

链式存储结构

C语言编译器 => 虚拟机(VM) => 虚拟存储结构

- **抽象数据类型: (Abstract Data Type, ADT)(=数据类型)**

  一个数学模型以及定义在该模型上的一组操作。

  数据类型将数据结构(数据元素、数据关系)和数据操作封装在一起，构成对象类。

  模块内部给出这些数据的定义、表示及其操作的细节，而在模块外部使用的只是抽象的数据和抽象的操作。

  ADT = {值域} + {值域上的操作}

- **用途：软件重用(reusability)**

一个含抽象数据类型的模块通常应包含定义(形式化/外部)、表示(存储结构/内部)和实现(操作/内部)三部分。

抽象数据类型的形式定义：抽象数据类型是一个三元组

(*D*， *S* ， *P*)

其中：

*D*是数据对象，数据元素的有限集

*S*是*D*上关系的有限集

*P*是对*D*的基本操作的有限集

ADT 抽象数据类型名

{

数据对象：<数据对象的定义>

数据关系：<数据关系的定义>

基本操作：<基本操作的定义>

} *ADT 抽象数据类型名*

对象和关系用伪码描述；操作=<接口，条件/初态，结果>

1.4 算法和算法分析

算法: 是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作。

一个算法通常具有五个重要特性:

- 有穷性　有穷步结束
- 确定性　操作无二义性，唯一执行路径，可重复
- 可行性　操作可以通过有限次已有基本运算实现
- 输入　零个或多个输入
- 输出　一个或多个输出

非数学有穷性!

不!

算法设计的要求：

- 正确性　　正确反映需求(通过测试)
- 可读性　　有助于理解、调试和维护
- 健壮性　　完备的异常和出错处理
- ✓ 高效率+低存储　　时间、空间的要求

设算法的问题规模为n；

频度：语句重复执行的次数称为该语句的频度，记$f(n)$。

对算法各基本操作的频度求和，便可得算法的时间复杂度。

但实际中我们所关心的主要是一个算法所花时间的数量级，即取算法各基本操作的最大频度数量级。

时间复杂度：算法执行时间度量,记$T(n)=O(\ \text{maxlevel}(f(n))\ )$

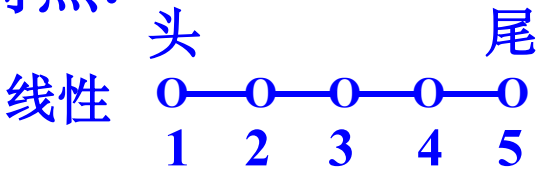
$$f(n) = 1 + n + n^2 + n^3$$

$$T(n) = O(\ n^3\ )$$

时间复杂度常用大O符号表述，不包括这个函数的低阶项和首项系数。使用这种方式时，时间复杂度可被称为是渐近的，它考察当输入值大小趋近无穷时的情况。

第2章　线性表

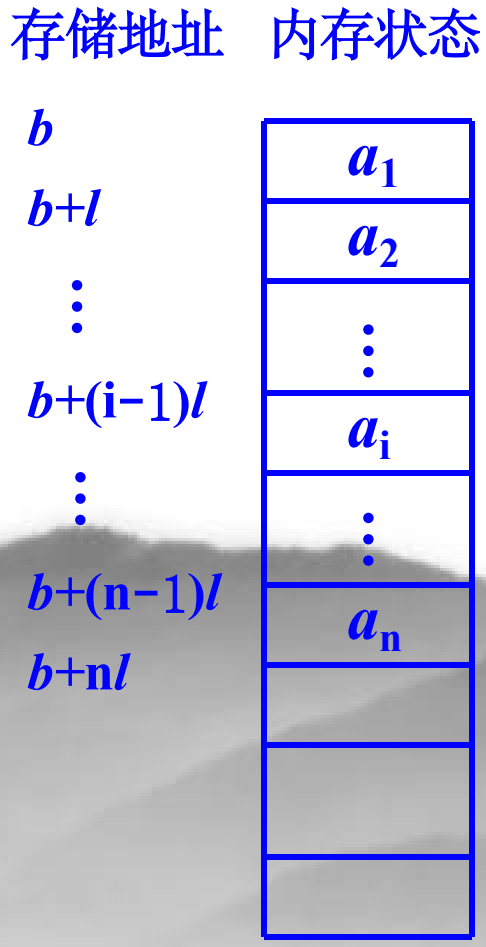
线性结构特点：



- 唯一头元素
- 唯一尾元素
- 除头元素外，均有一个直接前驱
- 除尾元素外，均有一个直接后继

2.2 线性表的顺序表示和实现

线性表的顺序表示指的是用一组地址连续的存储单元依次存储线性表的数据元素。



设每个元素需占用 l 个存储单元

$LOC(a_i)$表示元素a_i的存储地址

则$LOC(a_1)$是第一个数据元素a_1的存储地址，也是整个线性表的起始地址

$$LOC(a_{i+1}) = LOC(a_i) + l$$

$$LOC(a_i) = LOC(a_1) + (i - 1)l$$

@dou：一定要会计算某个元素的存储地址!

假设p_i是在第 i 个元素之前插入一个新元素的概率

则长度为 n 的线性表中插入一个元素所需移动元素次数的期望值为：$E_{is} = \sum_{i=1}^{n+1} p_i (n - i + 1)$

设在任何位置插入元素等概率，$p_i = \frac{1}{n+1}$

$$E_{is} = \frac{1}{n+1} \sum_{i=1}^{n+1} (n - i + 1) = \frac{n}{2} \qquad O(n)$$

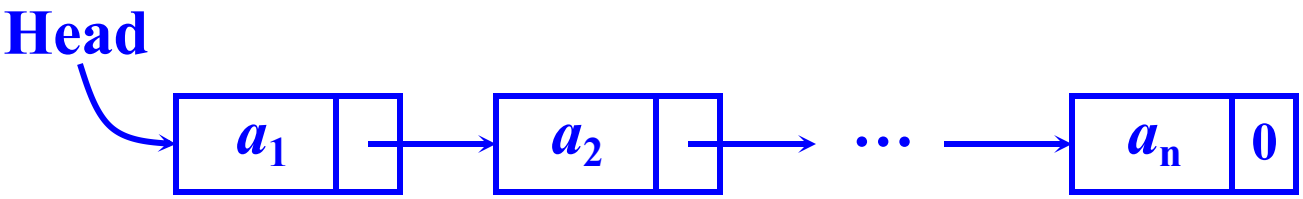
假设q_i是删除第 i 个元素的概率

则长度为 n 的线性表中删除一个元素所需移动元素次数的期望值为：$E_{dl} = \sum_{i=1}^{n} q_i (n - i)$

设删除任何位置的元素等概率，$q_i = \frac{1}{n}$

$$E_{dl} = \frac{1}{n} \sum_{i=1}^{n} (n - i) = \frac{n - 1}{2} \qquad O(n)$$

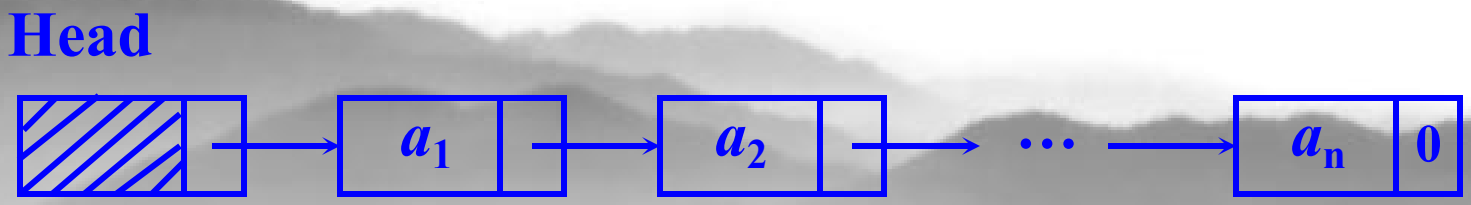
2.3.1 线性单链表



Head: 头指针，指向链表中第一个结点。

0: 空指针，有时也表示为“NULL”或“∧”。

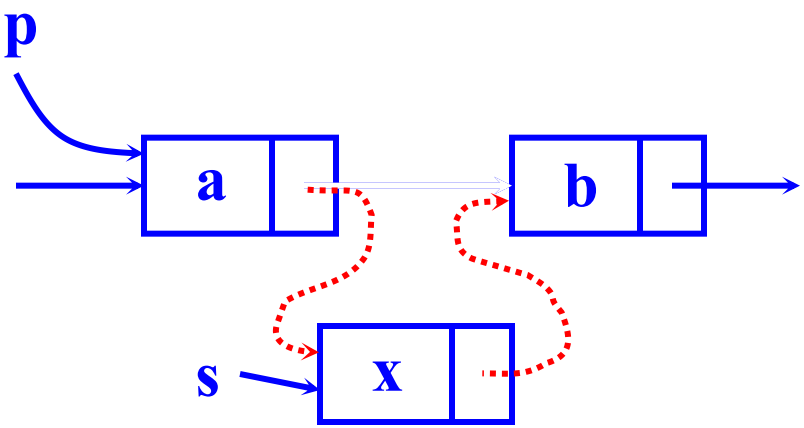
头结点: 通常需要在链表第一个结点之前附加一个结点，用于记录线性表的某些性质信息(如长度)。



优点： 数据元素的插入、删除相对方便

在a，b之间插入元素x：

务必记住
操作顺序



s->next = p->next

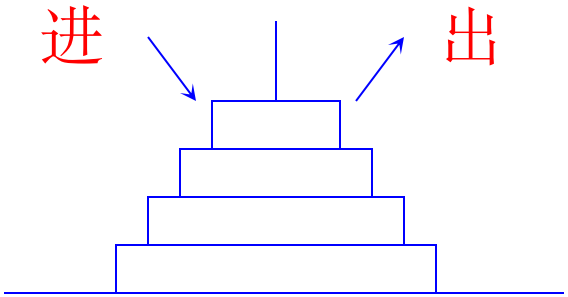
p->next = s

其他重点

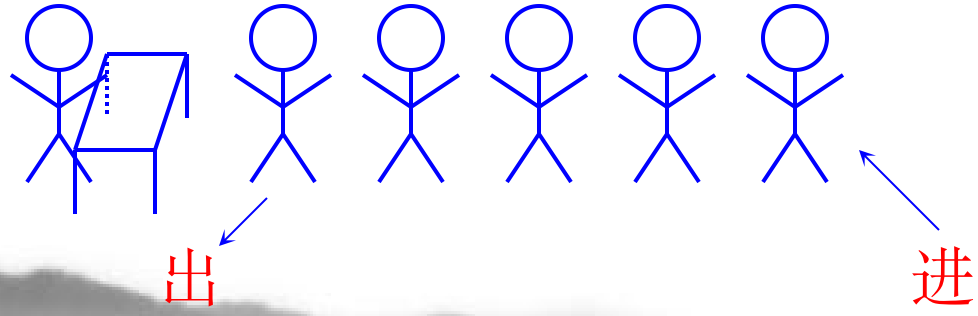
- 单链表、循环链表、双向链表等的特点，给定特定需求，选择合适的链表

第3章　　栈和队列

汉诺塔



排队
买票



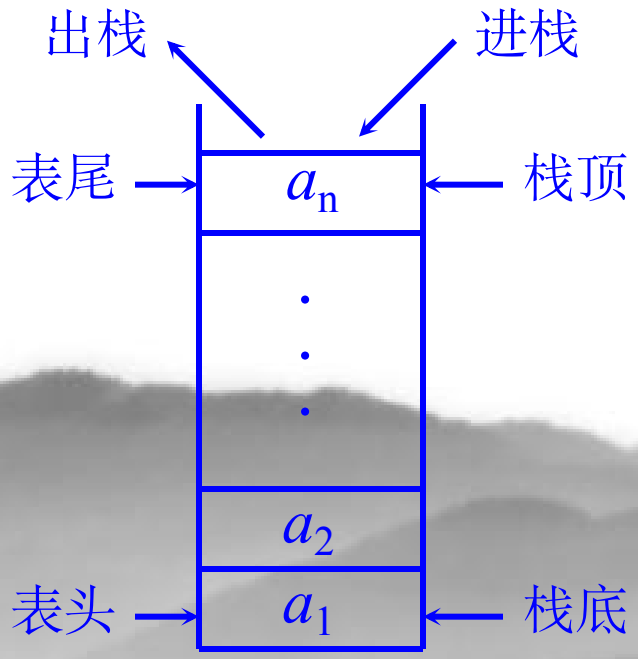
- 栈和队列是两种重要的线性结构
- 栈和队列是操作受限的线性表

3.1 栈

3.1.1 栈的定义

栈是限定仅在表尾进行插入或删除操作的线性表。

通常，表头端称为栈底，表尾端称为栈顶。



a_1为栈底元素

a_n为栈顶元素

按a_1、a_2…、a_n顺序进栈

按a_n…、a_2、a_1顺序出栈

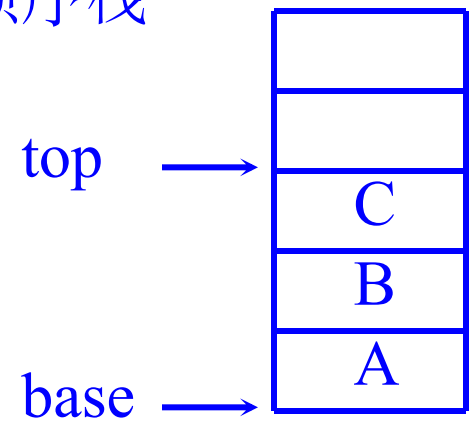
栈称为后进先出线性表(LIFO)

或先进后出（FILO）

3.1.2 栈的表示和实现

顺序栈 vs. 链式栈

1. 顺序栈



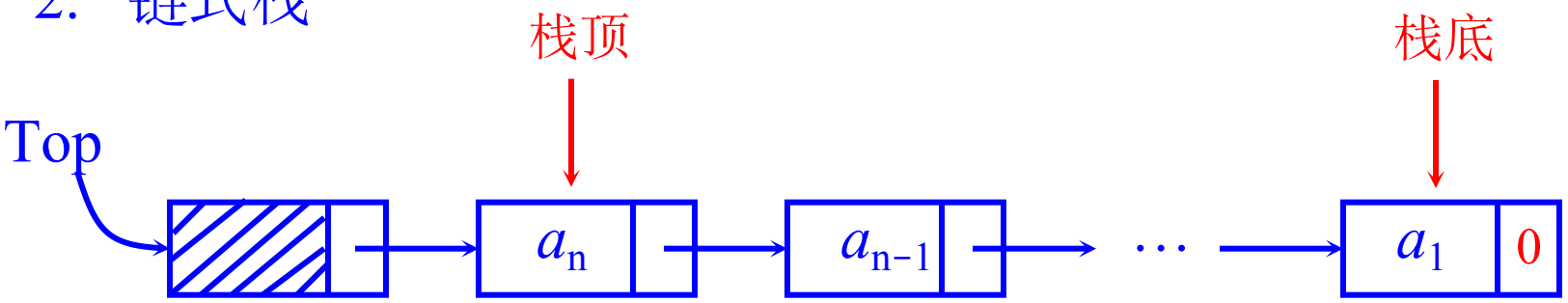
栈底指针——base

栈顶指针——top

栈容量

```
typedef   struct  {
    SElemType     * base ;
    SElemType     * top ;
    int               stacksize ;  //当前可用量大小
} SqStack ;
```

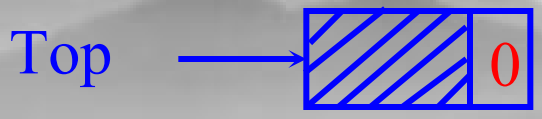
2. 链式栈




```
typedef   struct   SNode  {
        SElemType          data ;
        struct  SNode      * next ;
} SNode ,   * LinkStack ;
```

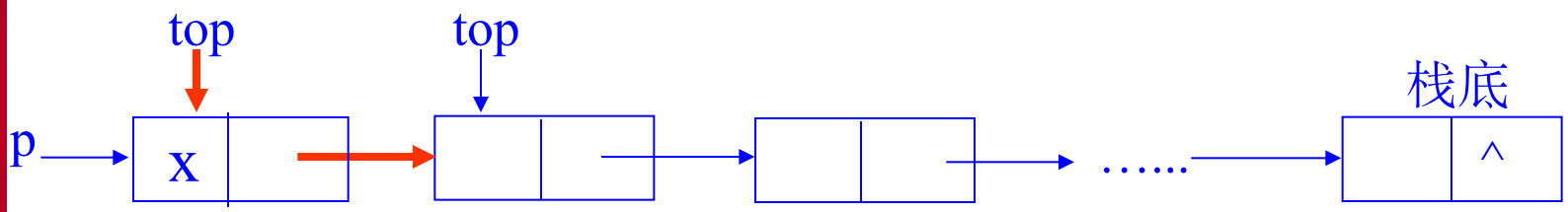
为什么栈顶在头?

空表:

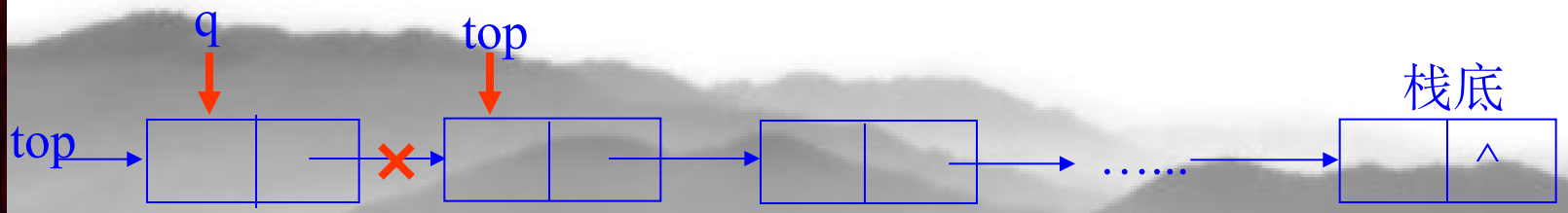


课后练习：实现

入栈算法



出栈算法



三、表达式求值

表达式的组成

- 操作数(operand)：常数、变量。
- 运算符(operator)：算术运算符、关系运算符和逻辑运算符。
- 界限符(delimiter)：左右括弧和表达式结束符。

算术运算的规则

- 先乘除后加减
- 先左后右
- 先括弧内后括弧外

例如：4+2*3-10/5

=4+6-10/5

=10-10/5

=10-2

=8

算法的思想：

- 设置两个工作栈
  - 运算符栈OPTR，运算符栈的栈底为表达式的起始符#。
  - 操作数栈OPND，操作数栈为空栈。
- 依次读入表达式中的每个字符
  - 若是操作数则进OPND栈；
  - 若是运算符，则和OPTR中的栈顶元素做比较再操作。
    - 若运算符优先级高于OPTR中的栈顶元素，则运算符入栈；
    - 若运算符优先级低于OPTR中的栈顶元素，则从OPND栈顶弹出两个操作数，与OPTR中的栈顶元素做运算，并将运算结果入OPND；
    - 若运算符优先级等于OPTR中的栈顶元素(括号或者#），则将OPTR中的栈顶元素出栈。
- 直至表达式求置完毕。

中缀表达式转后缀表达式的算法

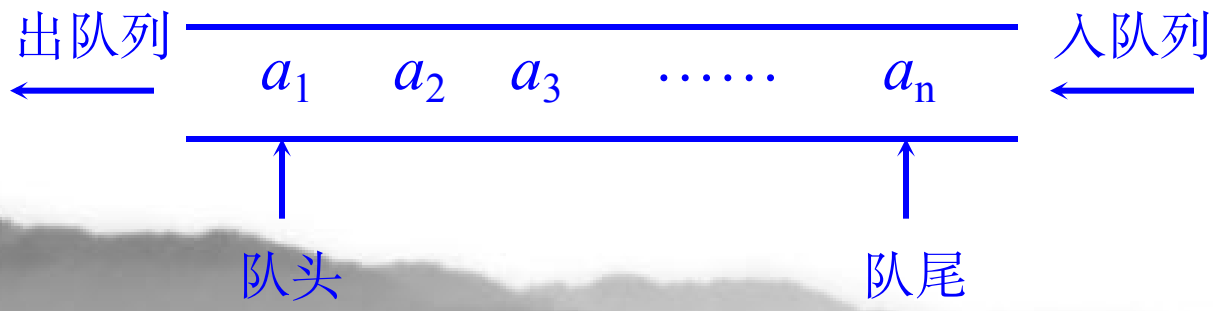
- 遇到数字：直接输出
- 遇到'('：压栈
- 遇到')'：持续出栈，如果出栈的符号不是'('则输出，否则终止出栈。
- 遇到符号则判断该符号与栈顶符号的运算优先级，如果栈顶符号的运算优先级高，则出栈并输出，直到优先级相等或栈为空；如果栈顶符号的运算优先级低于或等于当前符号的运算优先级，则将当前符号压栈。
- 处理完字符串后将栈中剩余的符号全部输出。

3.4 队列

3.4.1 队列的定义

队列是一种先进先出(FIFO)的线性表。

队列只允许在一端进行插入，而在另一端进行删除。



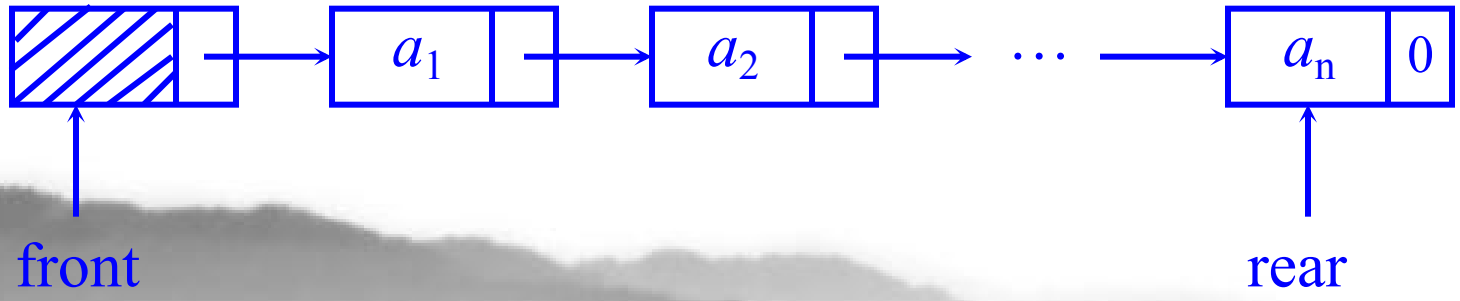
3.4.2 队列的表示和实现

链队列、顺序队列、循环队列

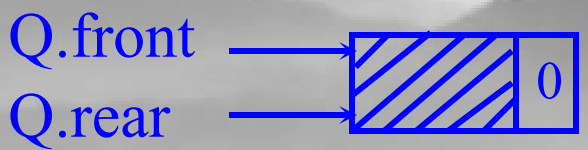
1. 链队列—链式存储结构

必须具备指示队头和队尾的指针(头指针、尾指针)。

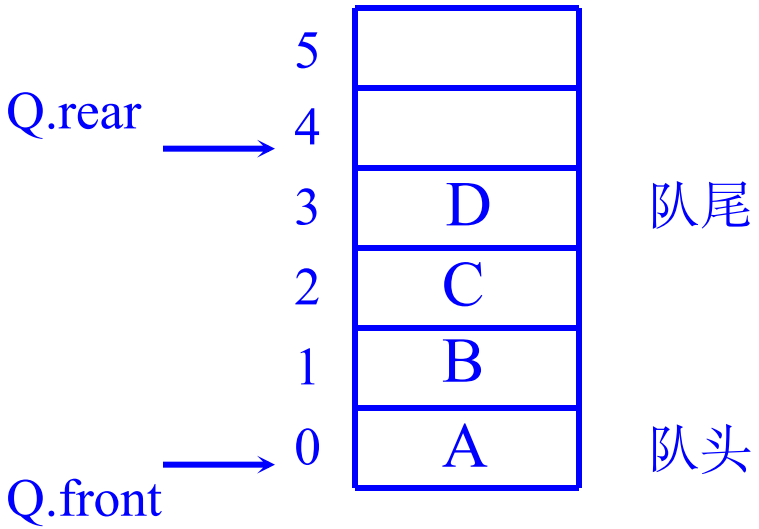
使用*头结点*



空队列



2. 顺序队列—顺序存储结构



用一组地址连续的存储单元依次存放队头到队尾的元素。

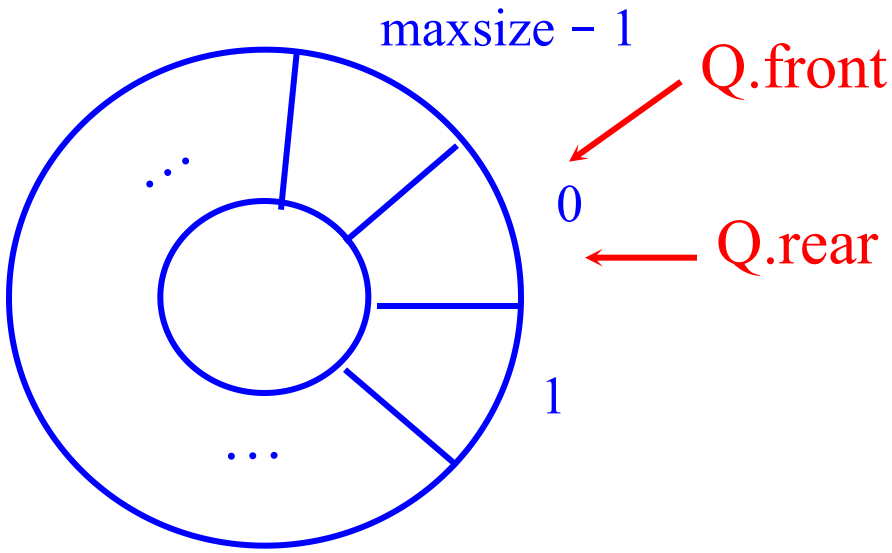
指针 front 、rear 分别指示队头和队尾下一个元素。

令 front = rear = 0 表示空队列，rear=MAXSIZE 表示队满。

每插入一新元素，rear 增 1，每删除一元素，front 增 1。

3. 循环队列—顺序存储结构

将顺序队列改造为一个环状的空间。



指针 front 、rear 分别指示队头和队尾下一个元素。

令 front = rear = 0 表示空队列。

每插入一新元素，rear = (rear + 1) % maxsize，每删除一元素，front = (front + 1) % maxsize 。　　// % : 求余

第 4 章　串

现象：

计算机非数值处理的对象基本上都是字符串数据。

例，学生管理系统：学生学号、姓名、性别、院系等；

图书管理系统：图书编号、书名、作者、简介等；

4.3 子串定位---模式匹配

1. 顺序串上的子串定位运算index(S，T)

子串定位运算通常称为串的“**模式匹配**”，是串处理中最重要的运算之一。设串s="$a_1a_2\ldots a_n$"，串T="$b_1b_2\ldots b_m$"($m\leq n$)，子串定位是在主串S中找出一个与子串T相同的子串，通常把主串S称为**目标**，把子串T称为**模式**。匹配有两种结果出现：

若S中有模式为T的子串，就返回该子串在S中的位置，当S中有多个模式为T的子串，通常只要找出第一个子串即可，这种情况称为匹配成功，

若S中无模式为T的子串，返回值为零，称为匹配失败。

KMP(D.E.Knuth-J.H.Morris-V.R.Pratt)算法

时间复杂度可以达到O(m+n)

当 S[i] ≠ T[j] 时，T从i沿S后滑。而已经得到的结果：

S[i-j+1..i-1] == T[1..j-1]

若已知 T[1..k-1] == T[j-(k-1)..j-1]

则有 S[i-(k-1)..i-1] == T[1..k-1]

则可以从T[k]继续和S[i]比较。

定义：模式串的*next*函数

$$next[j]=\begin{cases}0 & \text{当}j=1\text{时}\\ \mathrm{Max}\{k \mid 1<k<j \text{ 且 } 't_1t_2\cdots t_{k-1}'='t_{j-k+1}\cdots t_{j-1}'\} & \\ 1 & \text{其它情况}\end{cases}$$

eg.

T[]= a b a a b c a c

---

next[j]= 0 1 1 2 2 3 1 2

还有一种特殊情况需要考虑：

例如：

S = 'aaabaaaabaaaabaaaabaaaab'

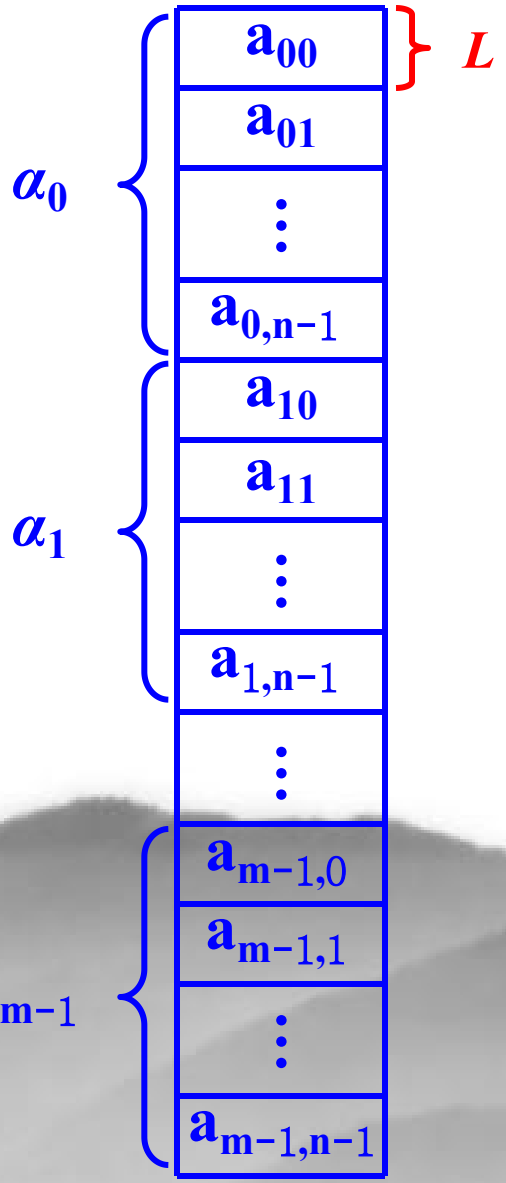
T = 'aaaab'

next[j]=01234

nextval[j]=00004

第5章　数组和广义表

对于数组，一旦规定了维数和维界，如何计算数组元素的存储位置。



设数组以行序为主序。

设二维数组 $A(m \times n)$

其数组元素 a_{ij} 的存储位置为

$$LOC(i, j) = LOC(0, 0) + (n \times i + j)\, L$$

其中，$LOC(0, 0)$是a_{00}的存储位置；

L是每个数组元素占用的存储单元数；

例，$LOC(1, 1) = LOC(0, 0) + (n \times 1 + 1)\, L$

推广到一般情况，可得到 n 维数组数据元素存储位置的映象关系

$$LOC(j_1, j_2, \ldots, j_n) = LOC(0,0,\ldots,0) +$$

$$(b_2 \times \ldots \times b_n \times j_1 + b_3 \times \ldots \times b_n \times j_2 + \ldots + b_n \times j_{n-1} + j_n)L$$

$$= LOC(0,0,\ldots,0) + \sum_{i=1}^{n} c_i j_i$$

其中 $c_n = L$，$c_{i-1} = b_i \times c_i$，$1 < i \leq n$。

称为 n 维数组的映象函数。可以推导出**数组元素的存储位置是其下标的线性函数(1:1)。**

压缩存储是指为多个值相同的元素只分配一个存储空间；对零元素不分配空间。

- 对称矩阵
- 三角矩阵
- 对角矩阵
- 稀疏矩阵

稀疏矩阵的压缩存储 -- 理解和掌握其概念和数据结构

- 三元组顺序表
- 行逻辑链接顺序表
- 十字链表

矩阵乘法不是重点考查内容

广义表是**递归**定义的**线性结构**，

$$LS = (\alpha_1, \alpha_2, \cdots, \alpha_n)$$

其中：α_i 为原子或广义表

例如： A = ( )

B = (e)

C = (a,(b,c, d))

D = (A, B, C)

E = (a, E) = (a, (a, (a, ⋯ ,))))

6) 任何一个**非空广义表** LS = (α1, α2, …, αn)
均可分解为两部分，即：

表头 Head(LS) = α1

表尾 Tail(LS) = (α2, …, αn)

例如： D = (E, F) = ((a, (b, c)), F)

Head(**D**) = E　　Tail(**D**) = (F)

Head(**E**) = a　　Tail(**E**) = ((b, c))

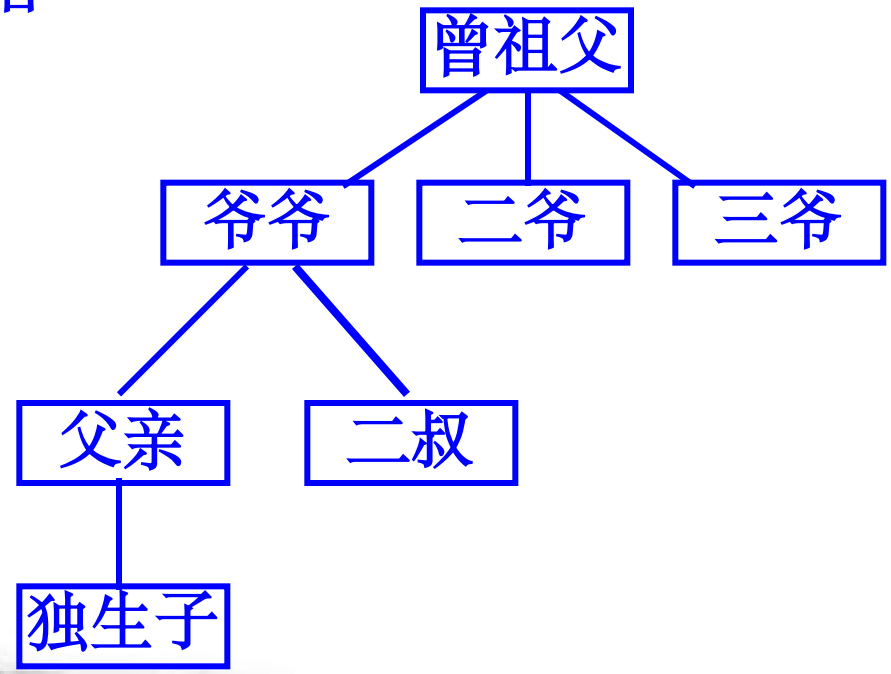
Head(**((b, c))**) = (b, c)　Tail(**((b, c))**) = ( )

Head(**(b, c)**) = b　Tail(**(b, c)**) = (c)

Head(**(c)**) = c　　Tail(**(c)**) = ( )

第六章　树和二叉树

族谱

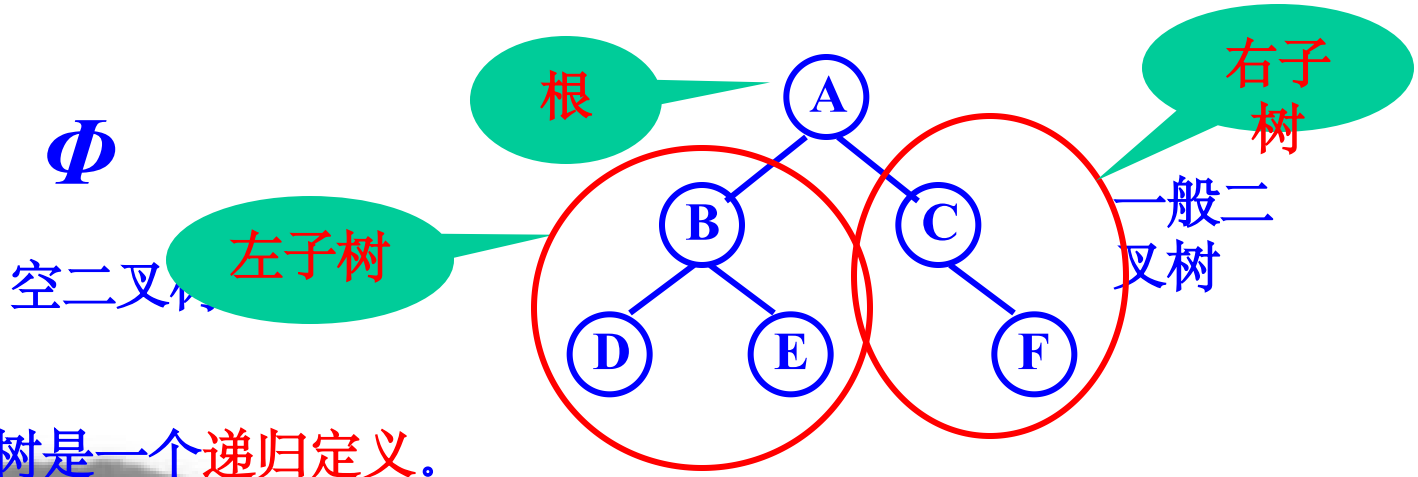


树是以分支关系定义的层次结构。

6.2 二叉树

6.2.1 二叉树(Binary Tree)的定义

二叉树是 n(n≥0) 个结点的有限集，它或者是空集，或者是由一个根和称为左、右子树的两个互不相交的二叉树组成。



二叉树是一个递归定义。

树的子树次序不作规定，二叉树的两个子树有左、右之分。

树中结点的度没有限制，二叉树中结点的度只能取 0、1、2。

性质2：深度为 k 的二叉树至多有 2^k-1 个结点($k\geq 1$)。

$$\sum_{i=1}^{k}(\text{第 } i \text{ 层上的最大结点数}) = \sum_{i=1}^{k} 2^{i-1} = 2^k-1$$

性质3：对任何一棵二叉树 T， 如果其终端/叶结点数为 n_0，度为 2 的结点数为 n_2， 则 $n_0 = n_2 + 1$ 。

证明：(1) 已知，终端结点数为 n_0，度为 2 的结点数为 n_2，

设度为 1 的结点数为 n_1，

由于二叉树中的所有结点的度只能为 0、1、2，

故二叉树的结点总数为 $n = n_0 + n_1 + n_2$；

(2) 除根结点外，其它结点都有一个分支进入，

设 B 为分支总数，故 $n = B + 1$，

由于这些分支均是由度为 1 或 2 的结点引出的，

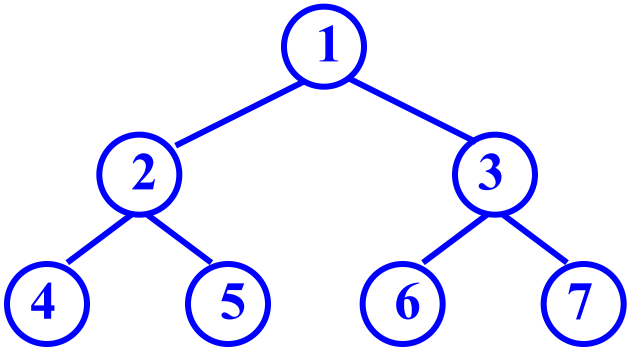
所以有 $B = n_1 + 2n_2$， 故 $n = n_1 + 2n_2 + 1$，

由 (1) 和 (2)， 可得 $n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$，

故有 $n_0 = n_2 + 1$ 。

两种特殊形态二叉树：满二叉树、完全二叉树。

一棵深度为 k 且有 2^k-1 个结点的二叉树称为满二叉树。



满二叉树

特点：(1) 每一层的结点数都达到最大结点数。

(2) 叶子结点在最大层。

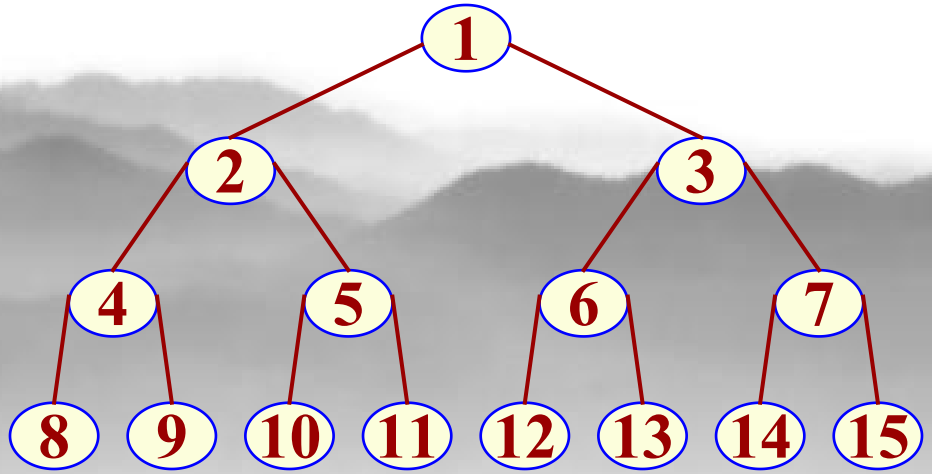
(3) 任一结点，其左、右分支下的子孙的最大层次相等。

性质5: 如果对一棵有n个结点的完全二叉树的结点按层序编号(从第 1 层到第 $\lfloor \log_2 n \rfloor + 1$ 层，每层从左到右)，则对任一结点 i($1 \leqslant i \leqslant n$)，有：

(1) 若 *i=1*，则该结点是二叉树的根，无双亲；如果 *i>1*，编号为 $\lfloor i/2 \rfloor$ 的结点为其**双亲**结点；

(2) 若 *2i>n*，则该结点无左孩子(结点为叶结点)，否则其**左孩子**是结点 *2i* ；

(3) 若 *2i+1>n*，则该结点无右孩子，否则其**右孩子**为结点*2i+1*。

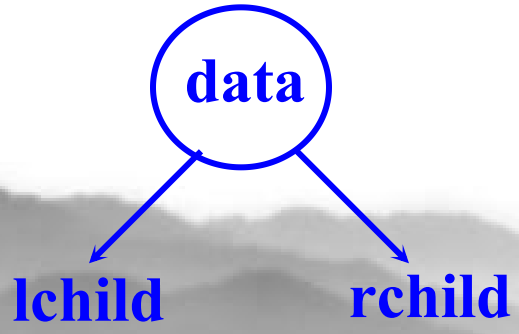


2. 链式存储结构

$D = (root, D_L, D_R)$。

链表结点包含 3 个域： 数据域、左指针域、右指针域

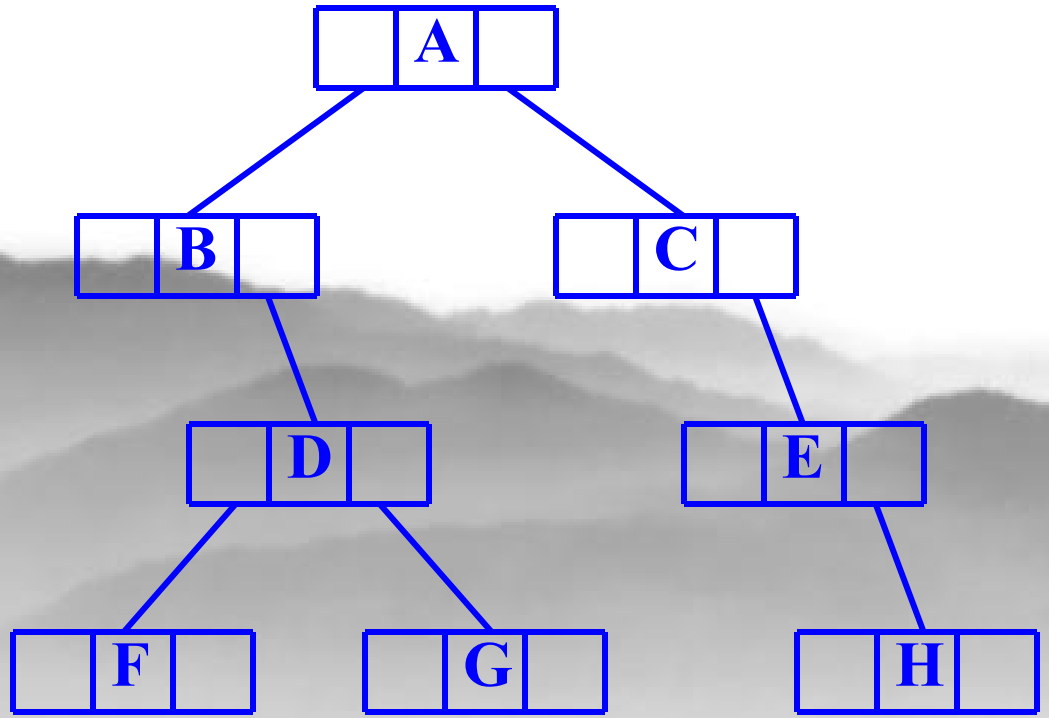
左指针域　数据域　右指针域



由这种结点结构得到的二叉树存储结构称为二叉链表。

二叉链表存储表示

```
struct   BitNode  {
    TElemType          data ;
    struct   BitNode     * lchild ,   * rchild ;
}
```



$D = (root, D_L, D_R)$。

如果能依次遍历这三部分，就可以遍历整个二叉树；

设以 D、L、R 分别表示访问根结点、遍历左子树、遍历右子树，则可以存在 6 种遍历方案: DLR、DRL、LDR、LRD、RDL、RLD；

若限定先左后右的原则，则只有 3 种情况：

先(根)序遍历

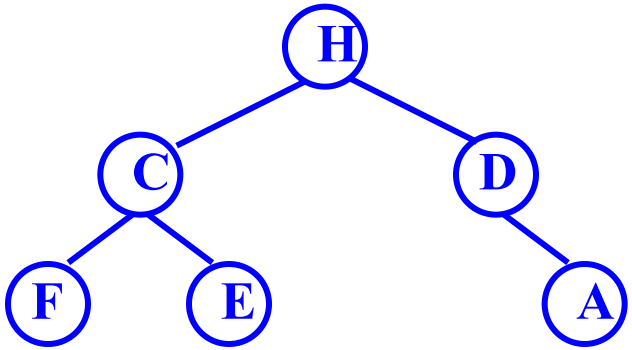
中(根)序遍历

后(根)序遍历。

算法 6.1　先序遍历递归算法

```
Status PreOrderTraverse ( BiTree T , printf(e) )
{    // visit(e) 函数可以看作 printf (e)
     If  (T)  {
          printf(T->data) ;
          PreOrderTraverse ( T->lchild, printf ) ;
          PreOrderTraverse ( T->rchild, printf ) ;
     }
     else   return OK ;
}
```

队列实现层次遍历



出队

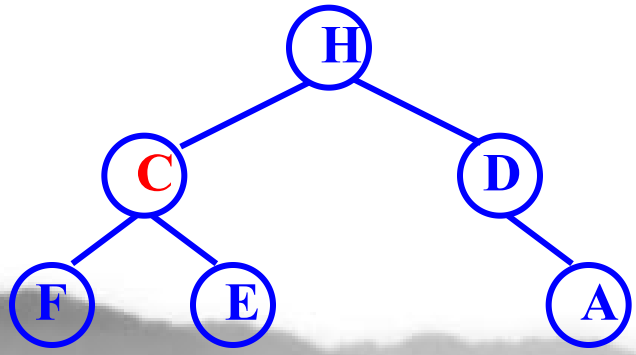
入队

H C D F E A

6.3.2 线索二叉树

二叉树的遍历实现了对一个非线性结构进行线性化的操作，从而使每个结点在线性序列中有且仅有一个直接前驱和直接后继。 => 不完全双向链

但以二叉链表作为存储结构时，



如何直接找到任意结点的前驱和后继结点？

性质：含有 n 个结点的二叉链表中有 n+1 个空链域。

证明：(1) 设，终端结点数为 n_0，

度为 1 的结点数为 n_1，

度为 2 的结点数为 n_2，

故二叉树的结点总数为 $n = n_0 + n_1 + n_2$；

(2) 空链域个数为 $2n_0 + n_1$，

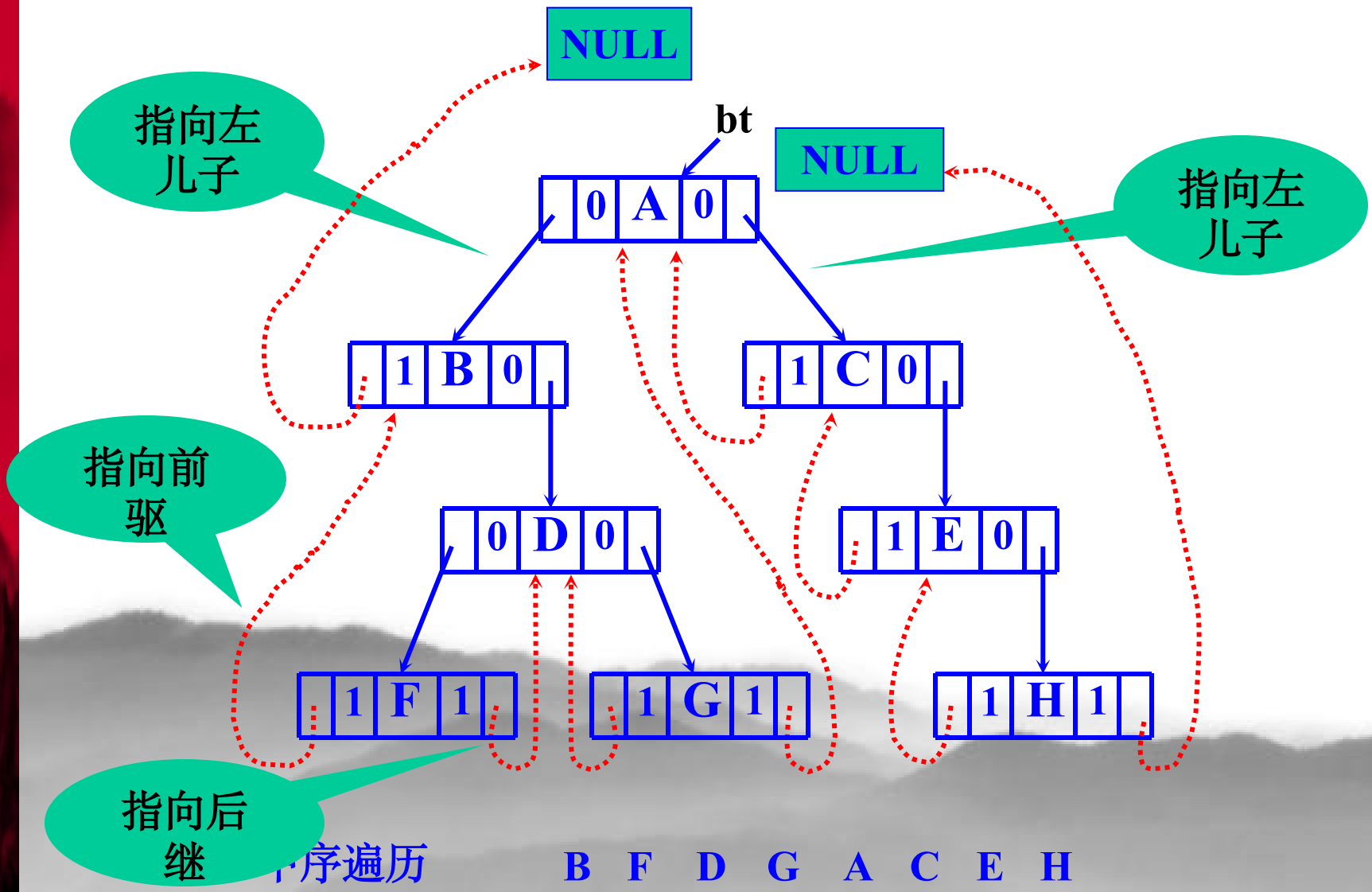
已知，$n_0 = n_2 + 1$，

故，$2n_0 + n_1$

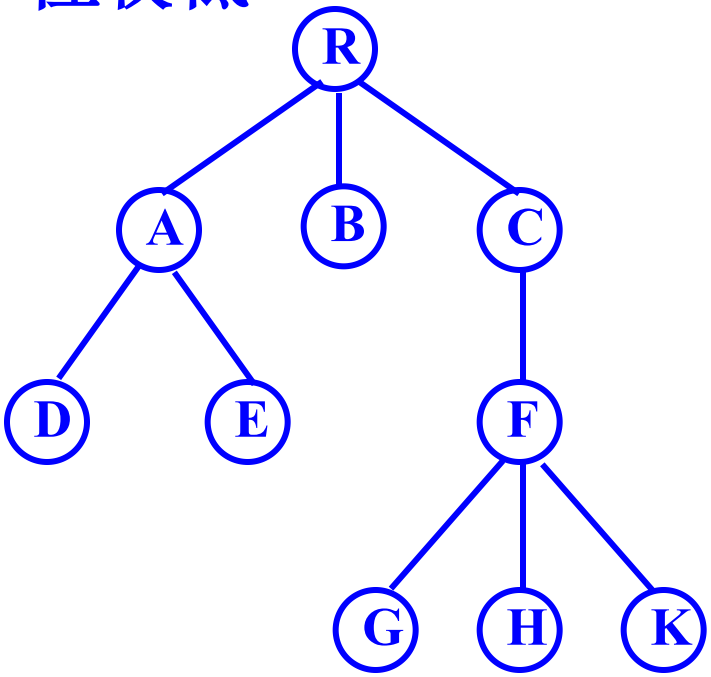
$= n_0 + n_1 + n_2 + 1$

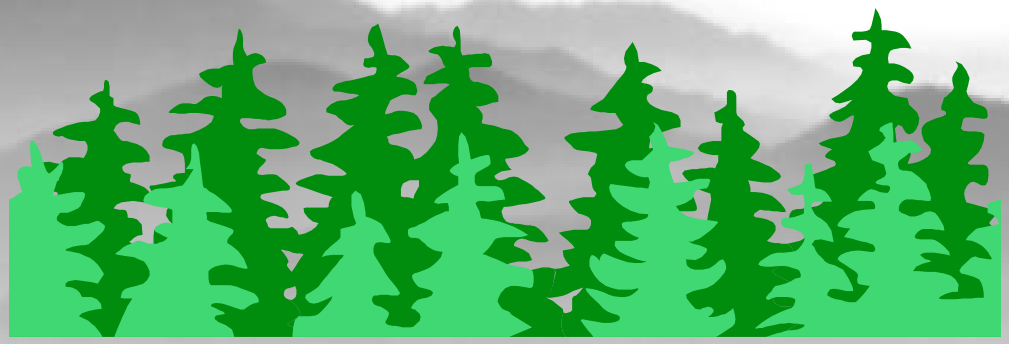
$= n + 1$

例，中序线索链表


NULL
bt
NULL
指向左儿子
指向左儿子
指向前驱
指向后继
0 A 0
1 B 0
1 C 0
0 D 0
1 E 0
1 F 1
1 G 1
1 H 1
中序遍历 B F D G A C E H

6.4 树和森林 – 重要性较低





6.6 树的应用

1. 赫夫曼树(最优二叉树)
2. 赫夫曼编码
3. 二叉分类树(二叉排序树)
4. 判定树

假设有n个权值 $\{w_1,w_2,\cdots w_n\}$，试构造一棵有n个叶子结点的二叉树，每个叶子结点带权为 w_i，则其中带权路径长度WPL最小的二叉树称做最优二叉树或赫夫曼树。

如何构造赫夫曼树?

(1) 根据给定的 n 个权值 $\{w_1,w_2,\cdots w_n\}$ 构成 n 棵二叉树的集合 $F=\{T_1,T_2,\cdots T_n\}$，其中每棵二叉树 T_i 中只有一个权值为 w_i 的根结点。

(2) 在 F 中选取两棵根结点权值最小的树作为左、右子树构造一棵新的二叉树，且置新二叉树的根结点的权值为其左、右子树根结点的权值之和。

(3) 在 F 中删除这两棵树，同时将新得到的二叉树加入集合 F 中。

(4) 重复 (2) 和 (3)，直到 F 中只含一棵树为止。

如何得到最短的二进制前缀编码?

3. 赫夫曼编码

设每种字符在电文中出现的概率 w_i 为，则依此 n 个字符出现的概率做权，可以设计一棵赫夫曼树，使

$$WPL = \sum_{i=1}^{n} w_i \ l_i \quad \text{最小}$$

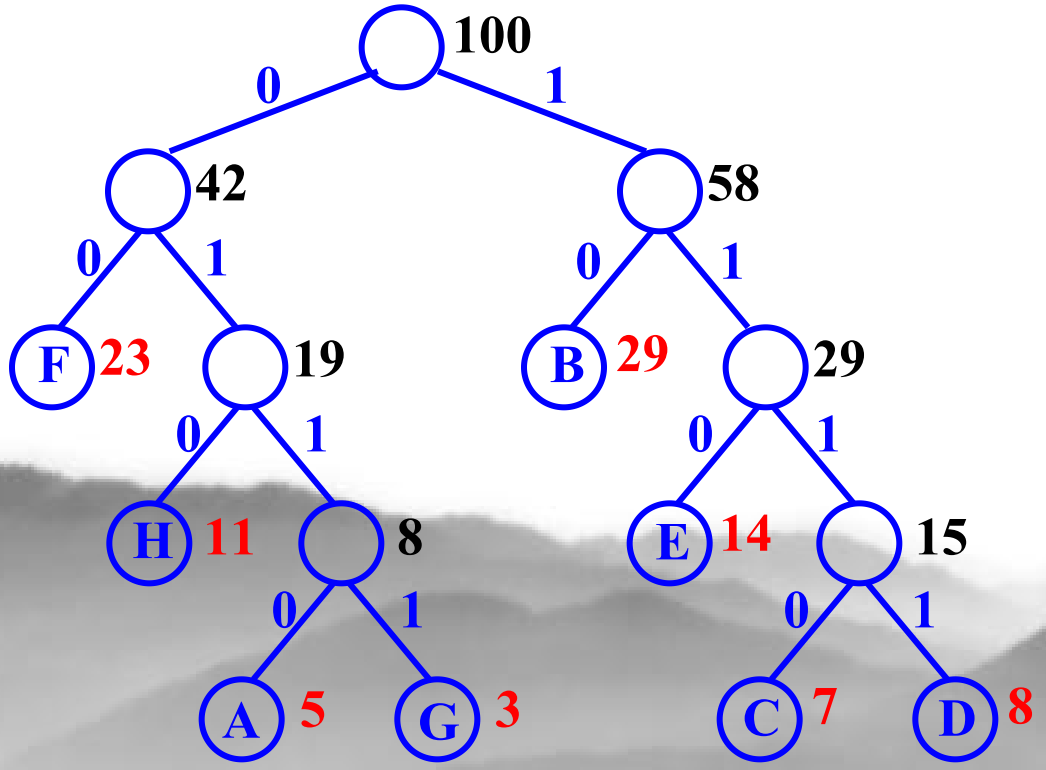
w_i 为叶子结点的出现概率 (权)

l_i 为根结点到叶子结点的路径长度

例，某通信可能出现 A B C D E F G H 8 个字符，其概率分别为 0.05 , 0.29 , 0.07 , 0.08 , 0.14 , 0.23 , 0.03 , 0.11 ， 试设计赫夫曼编码

不妨设 w = { 5 , 29 , 7 , 8 , 14 , 23 , 3 , 11 }

排序后 w = { 100 }



A (0110)

B (10)

C (1110)

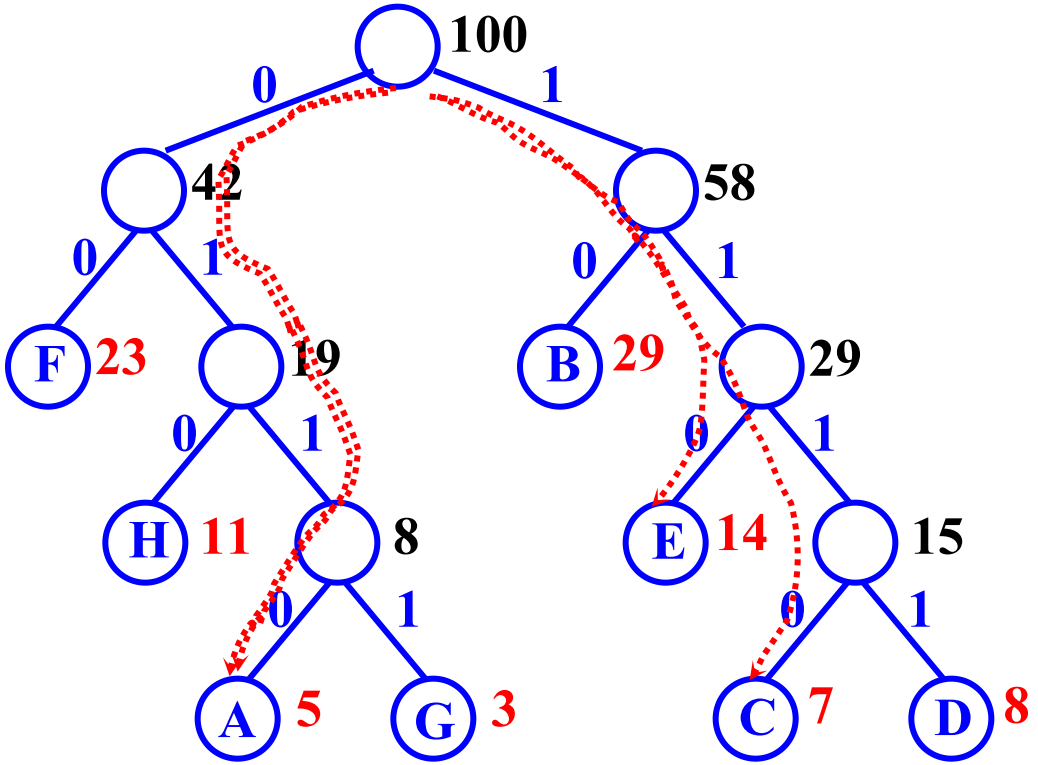
D (1111)

E (110)

F (00)

G (0111)

H (010)



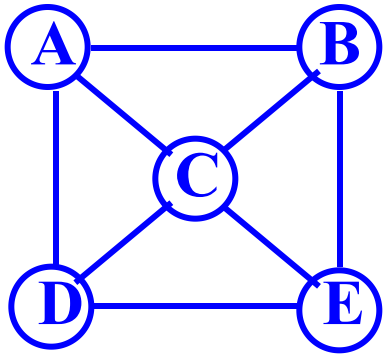
A (0110)
B (10)
C (1110)
D (1111)
E (110)
F (00)
G (0111)
H (010)

ACEA 编码为 0110 1110 110 0110

如何译码？ A C E A

1. 从根结点出发，从左至右扫描编码，
2. 若为 ‘0’ 则走左分支，若为 ‘1’ 则走右分支，直至叶结点为止，
3. 取叶结点字符为译码结果，返回重复执行 1,2,3 直至全部译完为止

第七章　图



图是一种较线性表和树更为复杂的数据结构。

线性表：线性结构

树：层次结构

图：结点之间的关系可以是任意的，即图中任意两个数据元素之间都可能相关。

7.2 图的存储结构

顺序存储　　如何表达顶点之间存在的联系？

邻接矩阵

链式存储　　多重链表，如何设计结点结构？

邻接表

十字链表

邻接多重表

7.2.1 数组表示法(邻接矩阵)

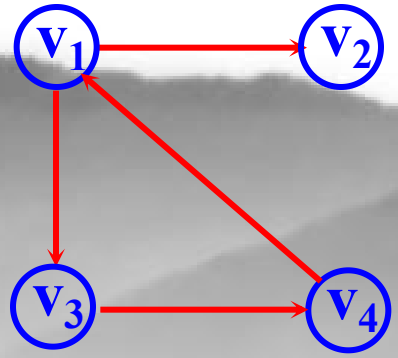
设图 G = (V , E) 具有 n(n≥1) 个顶点 v_1 , v_2 , …; v_n 和 m 条边或弧 e_1 , e_2 , …, e_m , 则 G 的邻接矩阵是 n × n 阶矩阵, 记为 A(G) 。

邻接矩阵存放 n 个顶点信息和 n^2 条边或弧信息。

其每一个元素 a_{ij} 定义为:

$$a_{ij} = \begin{cases} 0 & \text{顶点 } v_i \text{ 与 } v_j \text{ 不相邻接} \\ 1 & \text{顶点 } v_i \text{ 与 } v_j \text{ 相邻接} \end{cases}$$

例有向图 G



$$A(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

```
typedef struct ArcCell { // 弧的定义
    VRType  adj;   // VRType是顶点关系类型。对无权图，用1或0
                   // 表示相邻否；对带权图，则为权值类型。
    InfoType  *info; // 该弧相关信息的指针
} ArcCell,  AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];

typedef struct { // 图的定义
    VertexType   vexs[MAX_VERTEX_NUM];     // 顶点向量
    AdjMatrix    arcs;          // 邻接矩阵
    int  vexnum, arcnum;  // 顶点数，弧数
    GraphKind   kind;       // 图的种类标志{DG / DN / UDG / UDN}
} MGraph;
```

7.2.2 邻接表

对图中每一个顶点建立一个单链表，指示与该顶点关联的边或出弧。

头结点

vexinfo ： 顶点的信息

firstarc ： 第一条关联边结点

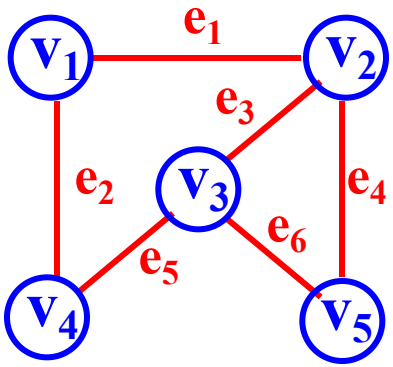
表结点

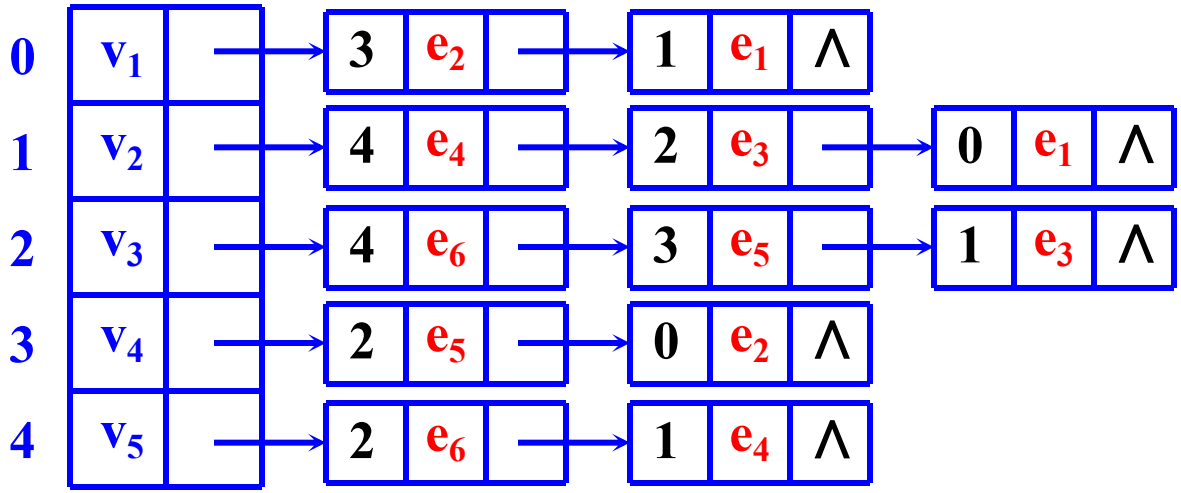
adjvex ： 邻接顶点位置

arcinfo ： 边的信息

nextarc ： 下一条关联边结点

例无向图 G





如何获取顶点的度?

顶点 v_i 的度为第 i 条链表中的结点数。

需要多少存储空间?

n + 2e

表结点

adjvex	nextarc	info

头结点

data	firstarc

```
typedef struct ArcNode {
  int                adjvex;     // 该弧所指向的顶点的位置
  struct ArcNode  *nextarc;   // 指向下一条弧的指针
  InfoType           *info;      // 该弧相关信息的指针
} ArcNode;
typedef struct VNode {
  VertexType  data;      // 顶点信息
  ArcNode     *firstarc; // 指向第一条依附该顶点的弧
} VNode, AdjList[MAX_VERTEX_NUM];
typedef struct {
    AdjList  vertices;
    int        vexnum, arcnum;
    int        kind;        // 图的种类标志
  } ALGraph;
```

7.3 图的遍历

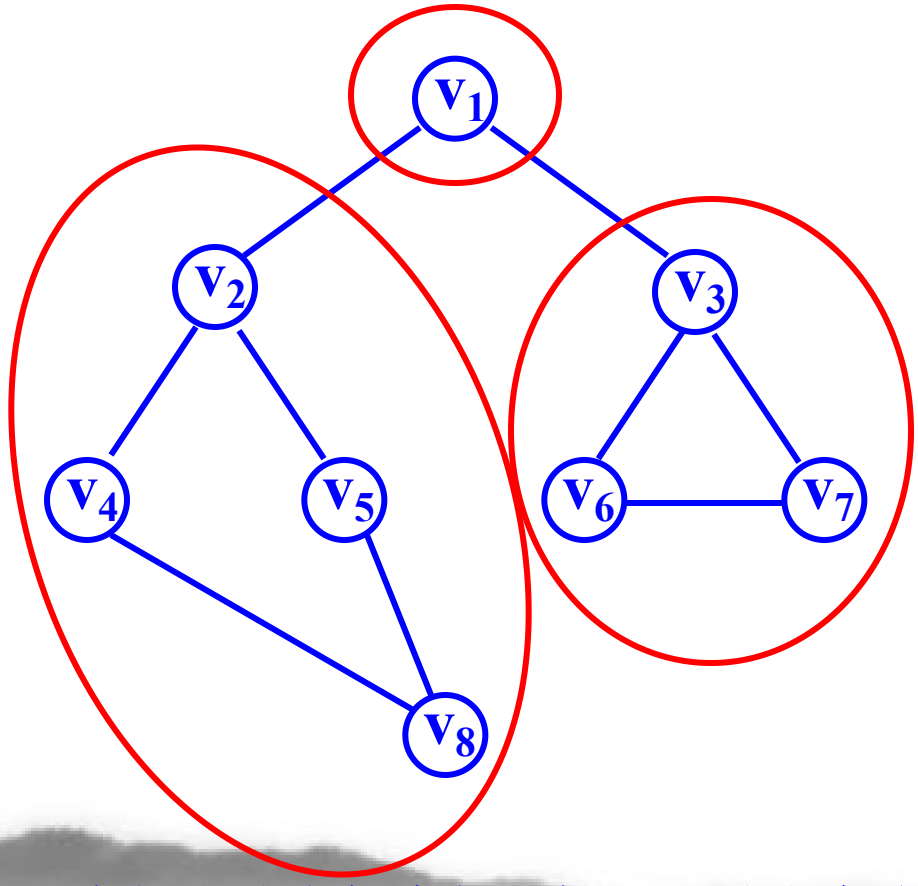
与树的遍历类似，如果从图中某一顶点出发访遍图中所有顶点，且使每一个顶点仅被访问一次，这一过程称为图的遍历。

图的遍历算法是求解图的连通性问题、拓扑排序和求关键路径等算法的基础。

通常有两条遍历图的路径：深度优先搜索、广度优先搜索。

图的遍历相对复杂，为了避免同一个顶点被访问多次，增设一个辅助的布尔数组 visited[0 . . n−1] 指示顶点是否已被访问过。

7.3.1 深度优先搜索



图可分为三部分:

基结点

第一个邻接结点导出的子图

其它邻接顶点导出的子图

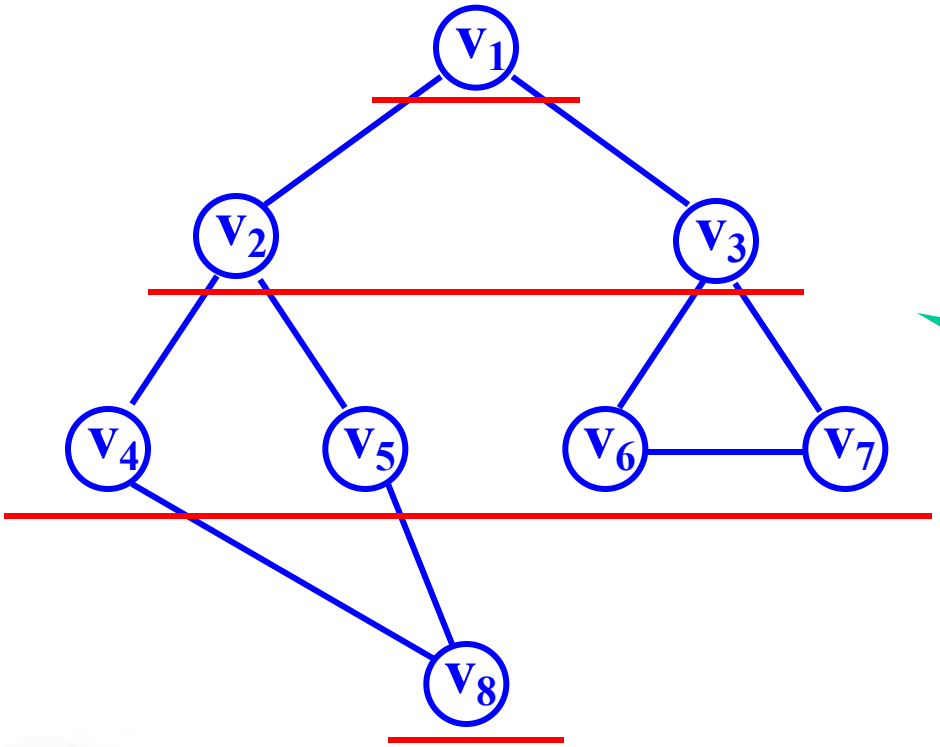
深度优先搜索是类似于树的一种先序遍历

深度优先搜索顺序: v_1 v_2 v_4 v_8 v_5 v_3 v_6 v_7

```
Boolean visited[MAX];
Status (*VisitFunc)(int v);

void DFSTraverse(Graph G,  Status (*Visit)(int v)) {  // 图 G 深度优先遍历
  VisitFunc = Visit;
  for (v=0; v<G.vexnum; ++v)
     visited[v] = FALSE; // 访问标志数组初始化
  for (v=0; v<G.vexnum; ++v)
     if (!visited[v])  DFS(G, v);  // 对尚未访问的顶点调用DFS
}
void DFS(Graph G, int v) {
   // 从顶点v出发，深度优先搜索遍历连通图 G
     visited[v] = TRUE;   VisitFunc(v);
     for(w=FirstAdjVex(G, v); w>=0; w=NextAdjVex(G,v,w))
        if (!visited[w])  DFS(G, w);
              // 对v的尚未访问的邻接顶点w，递归调用DFS
} // DFS
```

7.3.2 广度优先搜索



把图人为的分层，

按层遍历。

只有父辈结点
被访问后才会
访问子孙结点!

深度优先搜索类似于树的层次遍历，

广度优先搜索顺序：　v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

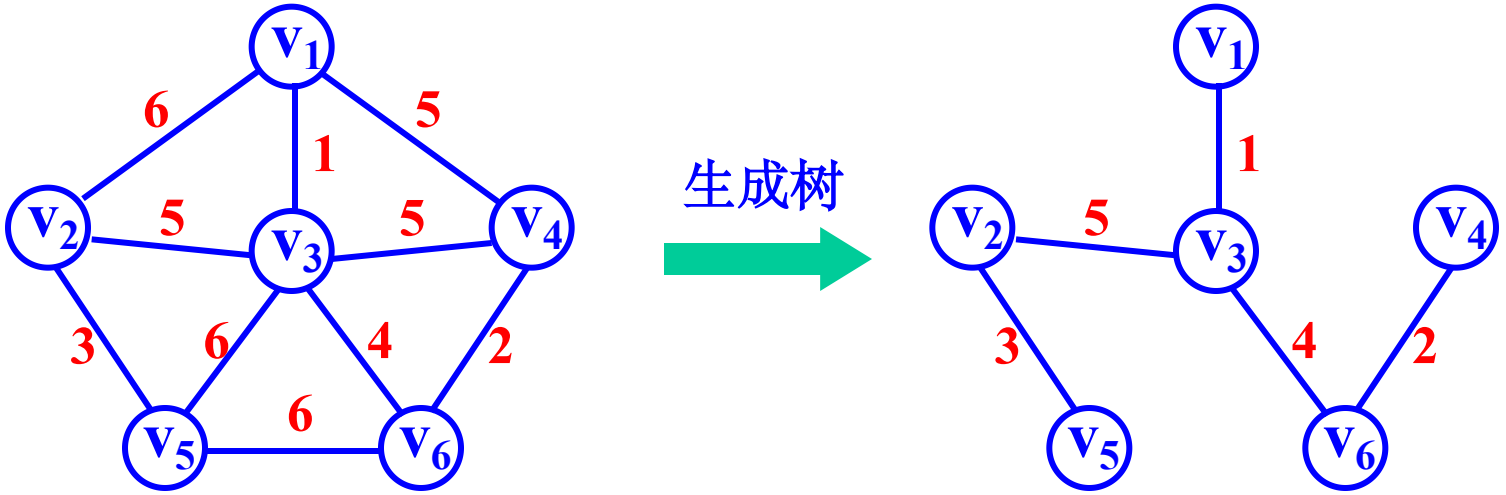
队列实现广度优先搜索算法

```
void BFSTraverse(Graph G, Status (*Visit)(int v))
{
  for (v=0; v<G.vexnum; ++v)   visited[v] = FALSE;  //初始化访问标志
  InitQueue(Q);       // 置空的辅助队列Q
  for ( v=0;  v<G.vexnum;  ++v )
    if (!visited[v]) {          // v 尚未访问
          visited[v] = TRUE;  Visit(v);    // 访问v
          EnQueue(Q, v);              // v入队列
          while (!QueueEmpty(Q))  {
            DeQueue(Q, u);          // 队头元素出队并置为u
            for(w=FirstAdjVex(G, u); w>=0;  w=NextAdjVex(G,u,w))
              if (!visited[w])  {
                visited[w]=TRUE;  Visit(w);
                EnQueue(Q, w); // 访问的顶点w入队列
              } // if
          } // while
  } // if
} // BFSTraverse
```

7.4.3 最小生成树(Minimum Cost Spanning Tree, MST)

一个无向图可以对应多个生成树。

一个带权无向图(无向网)同样可以对应多个生成树。

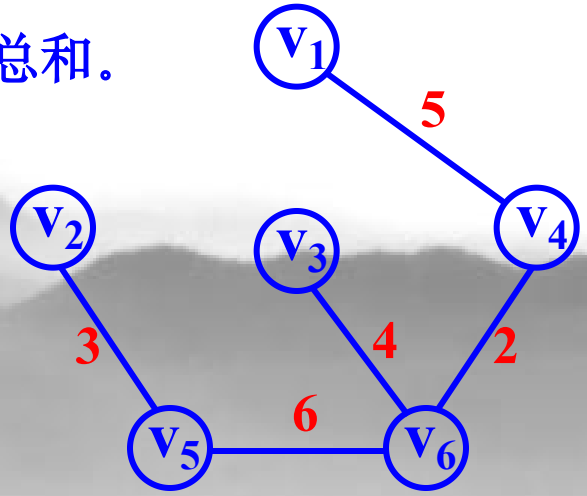


一棵生成树的代价定义为树上各边的权之总和。

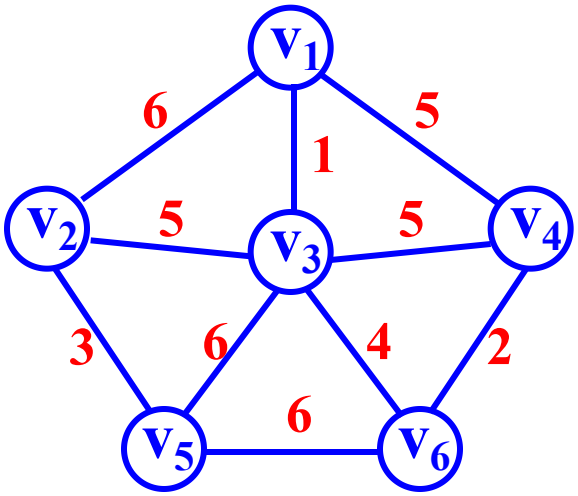
代价最小的生成树称为最小生成树。

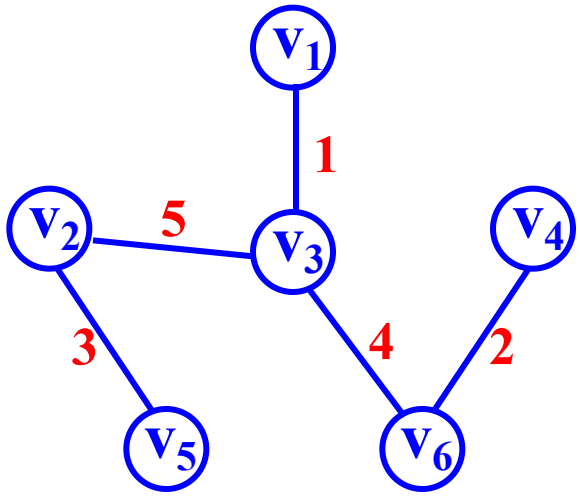
实际价值?

如何求取最小生成树?



例，





初始：$U=\{v_1\}$，$V-U=\{v_2,v_3,v_4,v_5,v_6\}$　　$TE=\{\}$

$U=\{v_1,v_3\}$，$V-U=\{v_2,v_4,v_5,v_6\}$　　$<v_1,v_3>$

$U=\{v_1,v_3,v_6\}$，$V-U=\{v_2,v_4,v_5\}$　　$<v_3,v_6>$

$U=\{v_1,v_3,v_4,v_6\}$，$V-U=\{v_2,v_5\}$　　$<v_6,v_4>$

$U=\{v_1,v_2,v_3,v_4,v_6\}$，$V-U=\{v_5\}$　　$<v_3,v_2>$

$U=\{v_1,v_2,v_3,v_4,v_5,v_6\}$，$V-U=\{\}$　　$<v_2,v_5>$

重点：边一定存在于 U 与 V-U 之间。

```
void MiniSpanTree_PRIM(MGraph G, VertexType u) {
  //用普里姆算法从顶点u出发构造网G的最小生成树
  //struct {
  //    VertexType  adjvex; // U集中的顶点序号
  //    VRType     lowcost; // 边的权值
  //} closedge[MAX_VERTEX_NUM]; //  顶点到U所形成的子树的距离/最短边
  k = LocateVex ( G, u );
  for ( j=0; j<G.vexnum; ++j ) // 辅助数组初始化
    if (j!=k)
      closedge[j] = { u, G.arcs[k][j].adj }; // 图存为邻接矩阵
  closedge[k].lowcost = 0;     // 初始，U＝{u}
  for (i=0; i<G.vexnum; ++i) {
      k = minimum(closedge);  // 求出加入生成树的下一个顶点k,
                              //  closedge[k].lowcost>0
      printf(closedge[k].adjvex, G.vexs[k]); // 输出生成树上一条边
      closedge[k].lowcost = 0;          // 第k顶点并入U集，距离为0
      for (j=0; j<G.vexnum; ++j)        //修改其它顶点的最小边
        if (G.arcs[k][j].adj < closedge[j].lowcost)
          closedge[j] = { G.vexs[k], G.arcs[k][j].adj };
}
```

Kruskal 算法

思想:

N = (V , E) 是 n 顶点的连通网，设 E 是连通网中边的集合；

构造最小生成树 N' = (V , TE)，TE 是最小生成树中边的集合，初始 TE = {} ；

重复执行:

选取 E 中权值最小的边 (u , v) ,

判断边 (u , v) 与 TE 中的边是否构成回路 ？

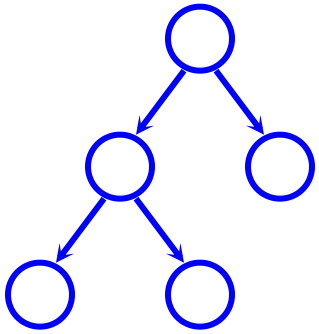
否，将边 (u , v) 纳入 TE 中，并从 E 中删除边 (u , v) ；

u 和 v 一定不在同一个连通分量中

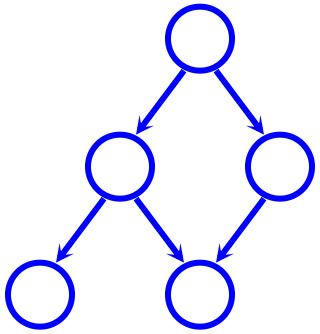
直至 E 为空 ；

7.5 有向无环图及应用

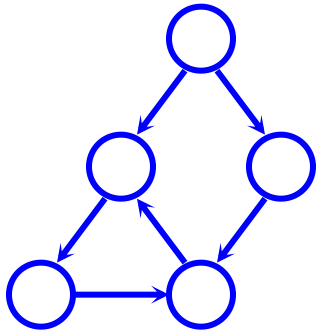
一个无环的有向图称为有向无环图，简称 DAG 图。



有向树



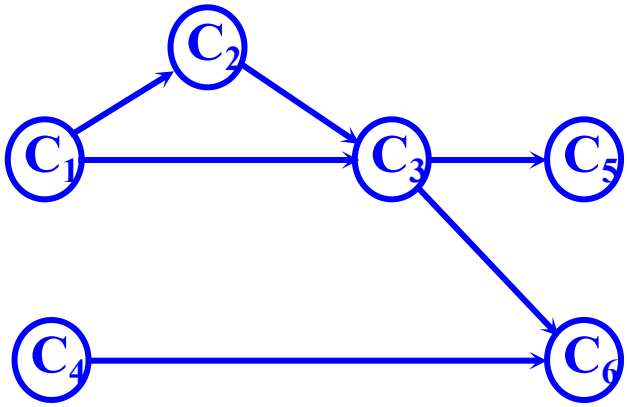
DAG 图



有向图

方法2 —— 拓扑排序法

课程编号	课程名称	先决条件
C_1	程序语言基础	
C_2	离散数学	C_1
C_3	数据结构	C_1 , C_2
C_4	微机原理	
C_5	编译原理	C_3
C_6	操作系统	C_3 , C_4



表示课程间优先关系的有向图

这种用顶点表示活动，用弧表示活动之间的优先关系的有向图称为顶点表示活动的网 —— AOV网(Activity On Vertex Network) 。

在 AOV 网中不应该出现有向环，否则将存在某项活动以自己为先决条件。

性质：若AOV 网中的所有顶点都在它的拓扑排序中，则该 AOV 网中必定不存在环；否则必存在环。

拓扑排序算法证明

算法思想：

1. 在有向图中选取一个没有前驱的顶点并输出；

2. 从图中删除该顶点及所有以此顶点为尾的弧；

3. 重复上述两步，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。

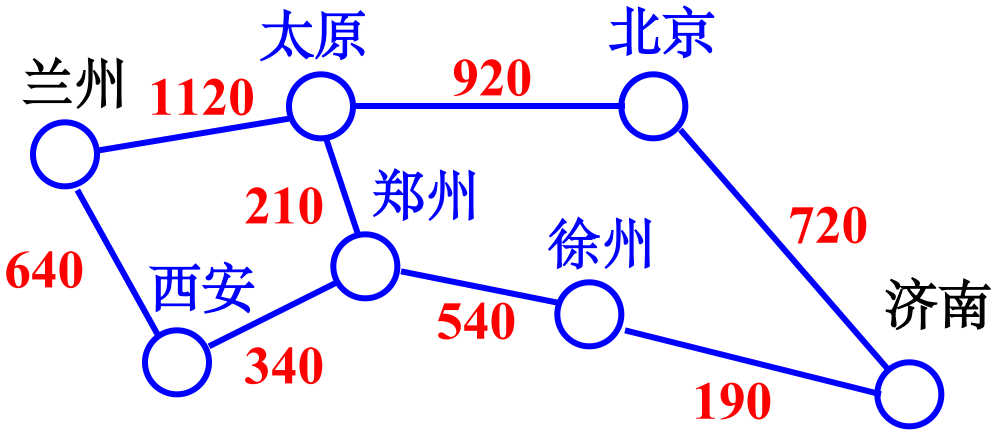
得到一个拓扑排序

存在环

例，

拓扑排序　v_1 v_6 v_4 v_3 v_2 v_5

7.6 最短路径



旅客希望停靠站越少越好，则应选择

济南——北京——太原——兰州

旅客考虑的是旅程越短越好，

济南——徐州——郑州——西安——兰州

Dijkstra 算法描述

利用已得到的顶点的最短路径来修改得到其它顶点的更短路径。

假设用带权的邻接矩阵 arcs[i][j] 来表示带权有向图。

初始，D[i] 存放 v_0 到 v_i 各顶点的弧的权值，D[i]=arcs[0][i]，S={};

重复执行 n−1 遍，每遍求出一条新的最短路径

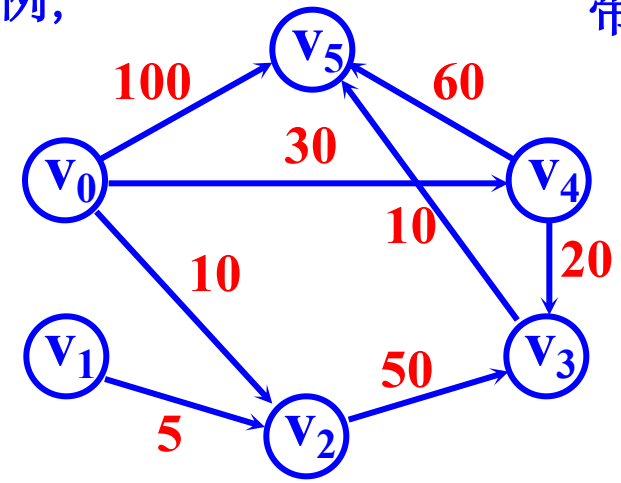
利用公式 $D[j] = Min\{ D[i] \mid v_i \in V-S \}$ 得到一条新的从 v_0 出发的最短路径及新的终点 v_j，令 $S = S + \{ v_j \}$；

利用 v_j 修改从 v_0 出发到集合 V−S 中任一顶点 v_k 可达的路径的长度；

D[j] + arcs[j][k]　　vs.　　D[k]

算法时间复杂度：$O(n^2)$

例，



带权邻接矩阵

	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

顶点 \ S		$\{v_2\}$	$\{v_2, v_4\}$	$\{v_2, v_4, v_3\}$	
v_1	∞	∞	∞	∞	
v_2	10				
v_3	∞	60	50		
v_4	30	30			
v_5	100	100	90	60	
最短路径	v_0v_2	v_0v_4	$v_0v_4v_3$	$v_0v_4v_3v_5$	
新顶点	v_2	v_4	v_3	v_5	v_1
路径长度	10	30	50	60	∞

每次修改都用的是最新加入集合 S 的顶点

第九章　　查找

查找和排序是数据处理系统中最重要的两个操作；

其次是插入、删除操作；

讨论查找、排序，不可避免要涉及文件、记录、关键字等概念。

文件——查找表，是由同一类型的数据元素(记录)构成的集合

记录——构成文件的数据元素，是文件中可存取的数据的基本单位

字段——数据项，数据的最小单位

关键字——某个可以用来标识记录的数据项

主关键字——某个可以用来唯一标识记录的数据项

次关键字——可以用来识别若干记录的数据项

顺序查找的性能分析

定义：平均查找长度(Average Search Length)

为确定记录在查找表中的位置，需和给定值进行比较的关键字个数的期望值。

对于含有n个记录的表，查找成功时的平均查找长度为

$$ASL = \sum_{i=1}^{n} P_i C_i$$

其中：P_i 为查找表中第i个记录的概率，且 $\sum_{i=1}^{n} P_i = 1$

C_i为找到该记录时，曾和给定值比较过的关键字个数。

对顺序表而言，$C_i = n-i+1$

$$ASL = nP_1 + (n-1)P_2 + \ldots + 2P_{n-1} + P_n$$

在等概率查找的情况下，$P_i = \frac{1}{n}$

顺序表查找的平均查找长度为：

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

9.1.2 有序表的查找

表中数据元素按照主关键字顺序排列。

5，13，19，21，37，56，64，75，80，88，92

- 折半查找
- 斐波那契查找
- 插值查找

没讲的内容不考

9.2 动态查找表

静态查找只是对表中元素进行检索，返回成功或不成功；

动态查找不但对表中元素进行检索，还可以通过检索过程来实现表的更新；

- 检索到，则返回成功；
- 检索不到，则将新元素插入到表中适当位置。

1．二叉排序树的查找算法：

若二叉排序树为空，则查找不成功；
否则，

1）若给定值等于根结点的关键字，则查找成功；

2）若给定值小于根结点的关键字，则继续在左子树上进行查找；

3）若给定值大于根结点的关键字，则继续在右子树上进行查找。

9.3 哈希表

前面介绍的内容中，记录在文件中的存储地址是随机的。

	学号	姓名	年龄	
01	200305	张四	19	
02	200304	李四	20	
03	200302	张五	19	

查找某一条记录需要进行一系列的“比较”。

查找的效率依赖于比较的次数。

能否在记录的关键字和存储地址之间构造这样一种关系 f，使得关键字和存储地址一一对应？

此对应关系 f 称为哈希函数。

2. 哈希冲突

对于不同的关键字可能得到同一哈希地址，即 key1≠ key2 ， 而 *f*(key1) = *f*(key2) ， 这种现象称为哈希冲突。

造成原因:

A. 主观设计不当

例，数字分析法中

8	1	3	4	6	5	3	7
7	1	3	7	2	2	4	7
8	1	3	8	7	4	2	2
8	2	3	0	1	3	6	7
8	1	4	2	2	8	1	7
8	1	3	3	8	9	6	7

冲突!

B. 客观存在

如何设计都不可能完全避免冲突的出现 ？

哈希地址是有限的，而记录是无限的。

解决方法:

(1) 开放定址法

(2) 再哈希法

*(3) 链地址法

(4) 建立一个公共溢出区

(1) 开放定址法

$H(key) = (key) \bmod m$

$H(key) = (H(key) + d_i) \bmod m$

$H(key)$ 哈希函数

m 哈希表长

d_i 增量序列

在 key mod m 的基础上，若发现冲突，则使用增量 d_i 进行新的探测，直至无冲突出现为止。

关键如何设计 d_i

线性探测法

二次探测法

随机探测法

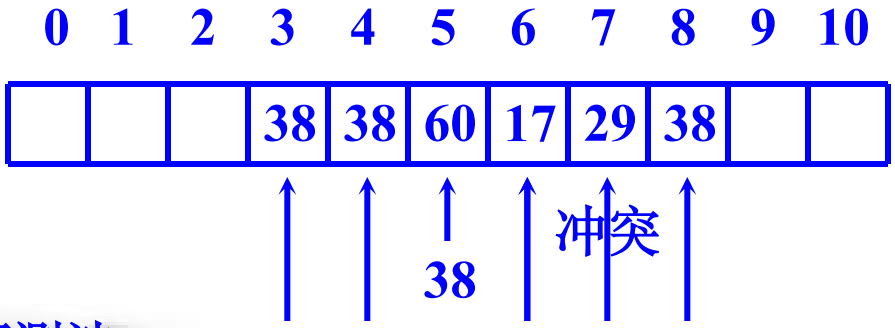
线性探测法　$d_i = 1, 2, 3, \ldots, m-1$

二次探测法　$d_i = 1^2, -1^2, 2^2, -2^2, 3^2, \ldots, \pm k^2$

伪随机探测法　d_i = 伪随机数

例，关键字为 (17 , 60 , 29 , 38)，哈希表长 11，H(key)=key%11

初始,



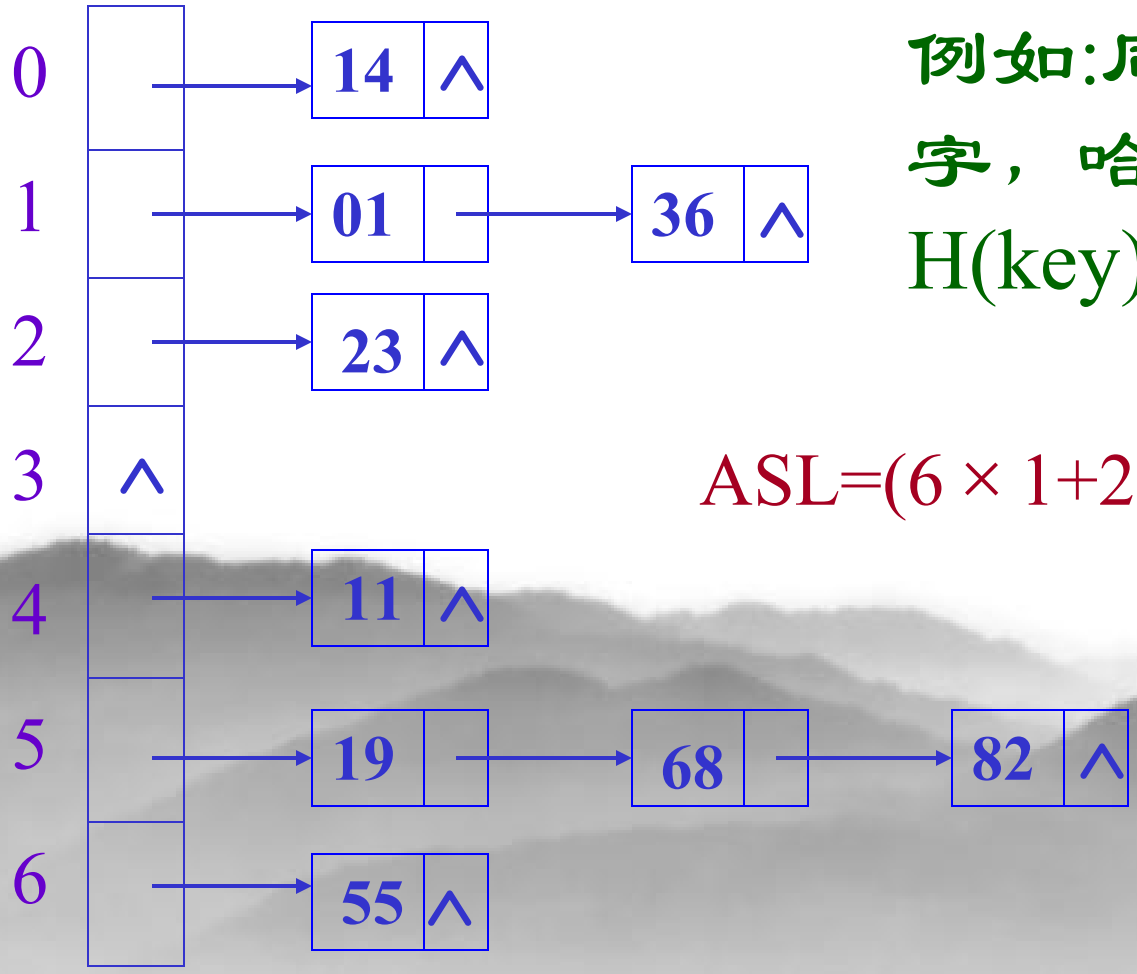
线性探测法　d = 0 无冲突　d = 1 无冲突　d = 2 无冲突

二次探测法

伪随机探测法　　不妨设第一次伪随机数为 9

(3) 链地址法

将所有哈希地址相同的记录都链接在同一链表中。



例如:同前例的关键字，哈希函数为

H(key)=key MOD 7

ASL=(6 × 1+2 × 2+3)/9=13/9

第十章　内部排序

10.1　排序

3　　10　　5　　78　　36

3　　5　　10　　36　　78

排序:

设 n 个记录的序列为　　$\{ R_1 , R_2 , R_3 , \ldots , R_n \}$

其相应的关键字序列为　$\{ K_1 , K_2 , K_3 , \ldots , K_n \}$

若规定 1 , 2 , 3 , . . . , n 的一个排列 $p_1 , p_2 , p_3 , \ldots , p_n$, 使得相应的关键字满足如下非递减关系:

$$K_{p1} \le K_{p2} \le K_{p3} \le \ldots \le K_{pn}$$

则原序列变为一个按关键字有序的序列:

$$\{R_{p1} , R_{p2} , R_{p3} , \ldots , R_{pn}\}$$

此操作过程称为排序。

10.7 各种排序方法的综合比较(pp289)

	平均时间	最差	最佳	辅助空间	稳定性
直接插入	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
起泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
希尔排序	$O(n^{1.5})$			$O(1)$	不稳定
快速排序	$O(n\log n)$	$O(n^2)$	同平均	$O(\log n)$	不稳定
堆排序	$O(n\log n)$	同平均	同平均	$O(1)$	不稳定
归并排序	$O(n\log n)$	同平均	同平均	$O(n)$	稳定
基数排序	$O(d(n+r))$	同平均	同平均	$O(n+r)$	稳定

2.单选题 (2分)

最后修改：2023-12-19 22:34

设F是一个森林,B是由F变换得的二叉树。若F中有n个非终端结点,则B中右指针域为空的结点有()个。

A n-1

B n

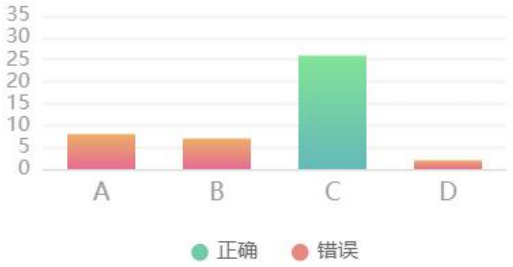
C n+1

D n(n-1)

正确答案：C

答题统计 (统计数据中的人数，为已交卷人数)

43/66	65.15%	60.47%
完成人数	完成率	正确率



答题分布

C ✓	共26人，占比39.39%	>
A	共8人，占比12.12%	>
B	共7人，占比10.61%	>
D	共2人，占比3.03%	>
未作答	共2人，占比3.03%	>

3.单选题 (2分)

最后修改：2023-12-19 22:34

一棵有124个叶结点的完全二叉树,最多有( )个结点。

A 247

B 248

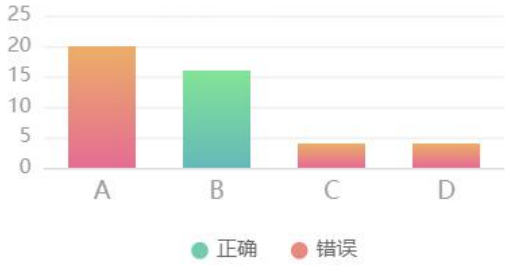
C 249

D 250

正确答案：B

答题统计 (统计数据中的人数，为已交卷人数)

44/66	66.67%	36.36%
完成人数	完成率	正确率



答题分布

B ✓	共16人，占比24.24%
A	共20人，占比30.30%
C	共4人，占比6.06%
D	共4人，占比6.06%
未作答	共1人，占比1.52%

5.单选题 (2分)

最后修改：2023-12-19 22:34

快速排序在输入序列基本有序时时间复杂度是（ ）

A O(n)

B O(n^2)

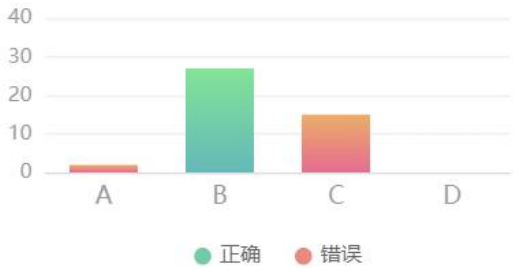
C O(n*logn)

D O(logn)

正确答案：B

答题统计 (统计数据中的人数，为已交卷人数)

44/66	66.67%	61.36%
完成人数	完成率	正确率



答题分布

B	✓	共27人，占比40.91%
A		共2人，占比3.03%
C		共15人，占比22.73%
D		共0人，占比0.00%

6.单选题 (2分)

最后修改：2023-12-19 22:34

假定对线性表(38,25,74,52,48)进行哈希存储,采用H(K)=K%7作为哈希函数，表长为7，若采用线性探查再散列法处理冲突,则对哈希表进行查找的平均查找长度为()。

A 1.4

B 2

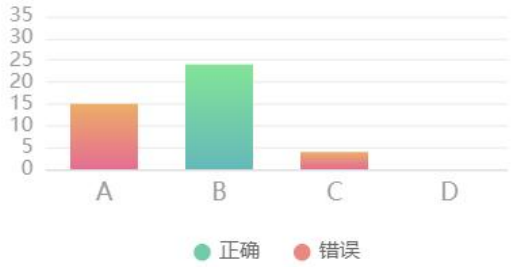
C 2.4

D 3

正确答案：B

答题统计 (统计数据中的人数，为已交卷人数)

43/66	65.15%	55.81%
完成人数	完成率	正确率



答题分布

B ✓	共24人，占比36.36%
A	共15人，占比22.73%
C	共4人，占比6.06%
D	共0人，占比0.00%
未作答	共2人，占比3.03%

8.单选题 (2分)

最后修改：2023-12-19 22:34

在有序表A[0..20]中,按二分查找方法进行查找,查找长度为5的元素个数是( )。

A 2

B 4

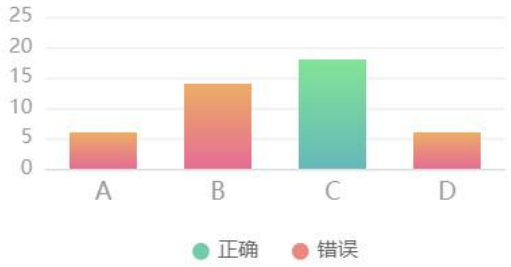
C 6

D 8

正确答案：C

答题统计 (统计数据中的人数，为已交卷人数)

44/66	66.67%	40.91%
完成人数	完成率	正确率



答题分布

C ✓	共18人，占比27.27%
A	共6人，占比9.09%
B	共14人，占比21.21%
D	共6人，占比9.09%
未作答	共1人，占比1.52%

13.填空题 (5分)

最后修改：2023-12-19 22:34

设某棵二叉树中度数为0的结点数为N0,度数为1的结点数为N1,则该二叉树中度数为2的结点数为[填空1];若采用二叉链表作为该二叉树的存储结构,则该二叉树中共有[填空2]个空指针域。

正确答案：待定/N0-1/N_0 - 1/N0 -1/N0 - 1；待定/2*N0+N1/2N0+N1/N1+2*N0;

答题统计 (统计数据中的人数，为已交卷人数)

39/66	59.09%	48.72%
完成人数	完成率	正确率

答题分布

正确	共19人，占比28.79%
错误	共20人，占比30.30%
未作答	共6人，占比9.09%

16.填空题 (5分)

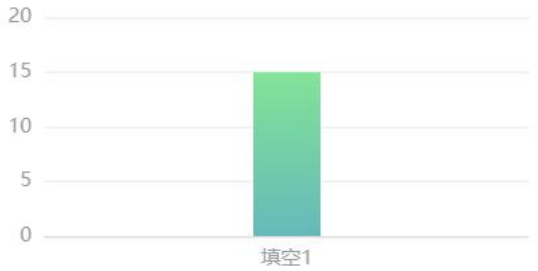
最后修改：2023-12-19 22:34

设一组初始记录关键字为(62,63,61,13,84，6,5),则以记录关键字62为基准的一趟快速排序结果为 [填空1]。

正确答案：待定/(5,6,61,13,62,84,63)/5 6 61 13 62 84 63/5，6，61，13，62，84，63；

答题统计 (统计数据中的人数，为已交卷人数)

39/66	59.09%	38.46%
完成人数	完成率	正确率



答题分布

正确	共15人，占比22.73%
错误	共24人，占比36.36%

17.填空题 (5分)

最后修改：2023-12-19 22:34

设一组初始记录关键字序列为(20，18，22，16，30，19)，则根据这些初始关键字序列建成的初始小顶堆层次遍历序列为[填空1]

正确答案：16，18，19，20，32，22/16，18，19，20，30，22/（16,18,19,20,30,22）/{16,18,19,20,32,22}/（16，18，19，20，32，22）/(16, 18, 19, 20, 32, 22)/16 18 19 20 30 22；

34/66
完成人数

51.52%
完成率

47.06%
正确率

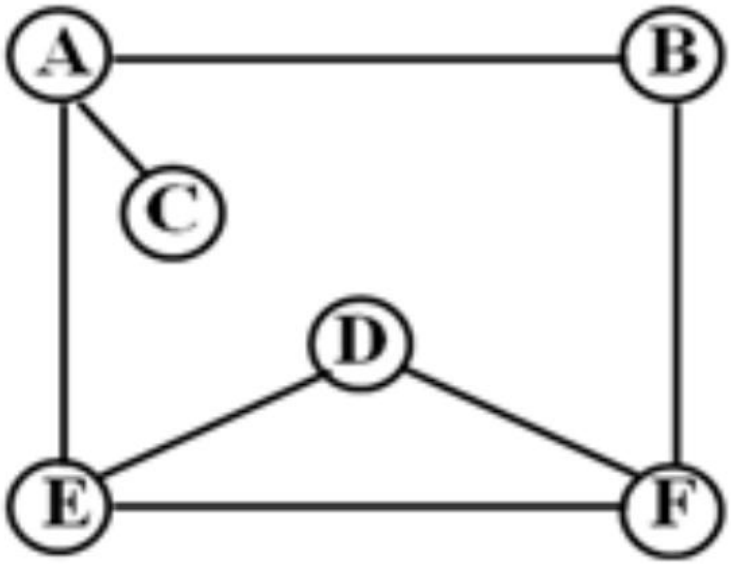
答题分布

正确	共16人，占比24.24%	>
错误	共18人，占比27.27%	>
未作答	共12人，占比18.18%	>

21.填空题 (5分)

最后修改：2023-01-16 08:59

已知无向图如下，按字母A-F的顺序输入图中的6个顶点并使用头插法（在邻接表的表头处插入）得到图的邻接表，则从顶点A开始基于此邻接表的图的深度优先遍历结果是[填空1]，广度优先遍历结果是[填空2]。



正确答案：待定/AEFDBC；待定/AECBFD；

答题统计 (统计数据中的人数，为已交卷人数)

39/46	84.78%	12.82%
完成人数	完成率	正确率

答题分布

正确	共5人，占比10.87%	>
错误	共34人，占比73.91%	>
未作答	共7人，占比15.22%	>

22.填空题 (5分)

最后修改：2023-01-16 10:36

设{49，38，65，97，76，13，27}为初始的待排序序列，把它们调整成的初始大顶堆的层次遍历序列是[填空1]，输出一个最大元素后，调整完成后的大顶堆的层次遍历序列是[填空2]。

正确答案：待定/97,76,65,38,49,13,27/97, 76, 65, 38, 49, 13, 27/97 76 65 38 49 13 27/97.76,65,38,49,13,27/（97,76,65,38,49,13,27）；待定/76,49,65,38,13,27/76,49,65,38,27,13/76, 49, 65, 38, 27, 13/76 49 65 38 27 13/（76,49,65,38,27,13,97）/（76，49，65，38，27，13）；

答题统计 (统计数据中的人数，为已交卷人数)

31/46	67.39%	38.71%
完成人数	完成率	正确率

答题分布

正确 共12人，占比26.09%

错误 共19人，占比41.30%

24.填空题 (5分)

最后修改：2023-01-16 08:59

已知字母A、B、C、D、E、F、G出现的概率分别是0.03, 0.04, 0.06, 0.11, 0.12, 0.25, 0.30，以它们为叶子结点构造出相应的哈夫曼树(要求：左子树根结点的权小于等于右子树根结点的权)，请写出每个字母的哈夫曼编码：A：[填空1]，B：[填空2]，C：[填空3]， D：[填空4]， E：[填空5]， F：[填空6]， G：[填空7]。

正确答案：待定/0010；待定/0011；待定/000；待定/010；待定/011；待定/10；待定/11；

36/46	78.26%	27.78%
完成人数	完成率	正确率

答题分布

正确	共10人，占比21.74%
错误	共26人，占比56.52%

- 在完全二叉树中，编号为i和j的两个结点处于同一层的条件是_____[Log_2i]== [log_2^j]。

- 设F是由T1,T2,T3三棵树组成的森林,与F对应的二叉树为B,已知T1,T2,T3的结点数分别为n1,n2和n3则二叉树B的左子树中有_N1-1 _____个结点，右子树中有_ N2+N3 _____个结点。

- 下面哪一方法可以判断出一个有向图是否有环（回路）(B):
- A. 深度优先遍历 B. 拓扑排序 C. 求最短路径 D. 求关键路径

- 在图采用邻接表存储时，求最小生成树的 Prim 算法的时间复杂度为(B)。
- A. O(n) B. O(n+e) C. $O(n^2)$ D. $O(n^3)$

Minimum edge weight data structure	Time complexity (total)
adjacency matrix, searching	$O(\|V\|^2)$
binary heap and adjacency list	$O((\|V\| + \|E\|) \log \|V\|) = O(\|E\| \log \|V\|)$
Fibonacci heap and adjacency list	$O(\|E\| + \|V\| \log \|V\|)$

. Dijkstra最短路径算法从源点到其余各顶点的最短路径的路径长度按_____次序依次产生，该算法弧上的权出现__ ____情况时，不能正确产生最短路径。

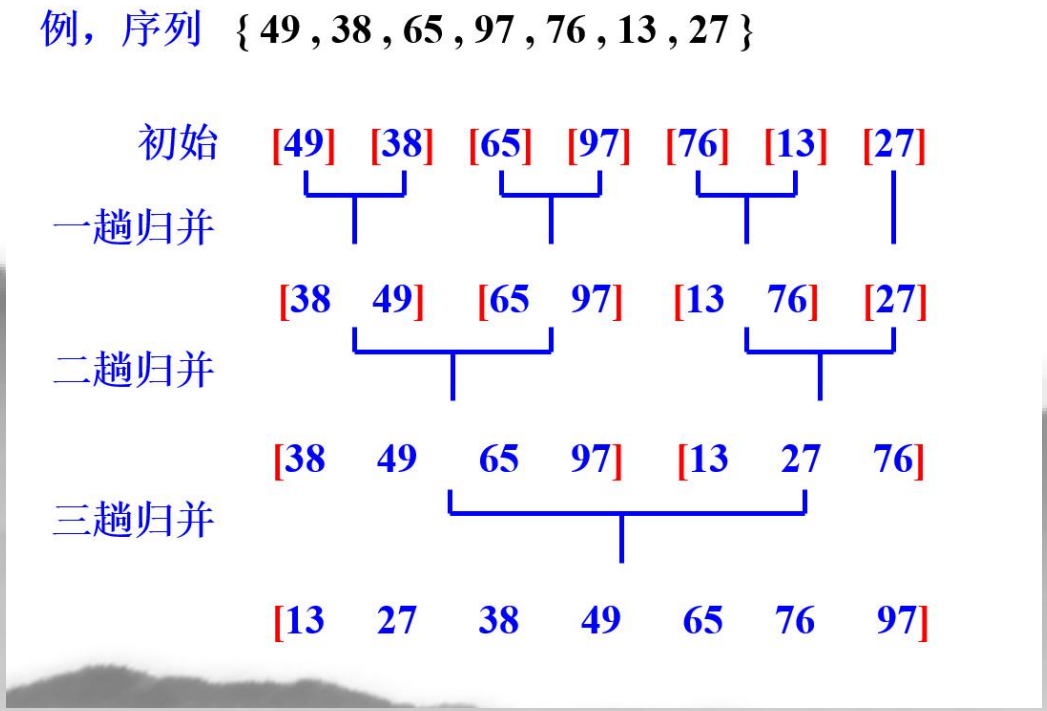
- 3、n个元素在两路归并排序中，归并的趟数是(B)。
- A. O（n） B. O（$\log_2 n$）
- C. O（$n\log_2 n$） D. O（n*n）

- 5、如果只想得到1000个元素组成的序列中第5个最小元素之前的部分排序的序列，用（ D ）方法最快。
- A．起泡排序　B．快速排列 C．Shell排序 D．堆排序 E．简单选择排序

6、具有12个关键字的有序表，折半查找的平均查找长度（ A ）

A. 3.1　　B. 4　　C. 2.5　　D. 5

8、假定一组记录的排序码为(46,79,56,38,40,80),对其进行二路归并排序的过程中，第二趟归并后的结果为__38 46 56 79 40 80_____。



9、假定对线性表(38,25,74,52,48)进行散列存储，采用H(K)=K %7作为散列函数，若分别采用线性探查法和链接法处理冲突，则对各自散列表进行查找的平均查找长度分别为___2____和___1.4_____。

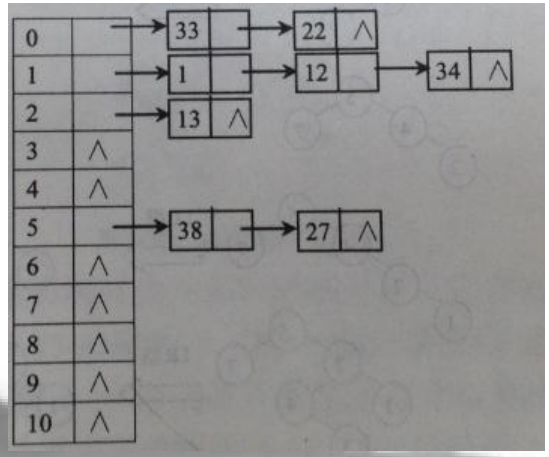
10、由带权为3，9，6，2，5的5个叶子结点构成一棵哈夫曼树，则带权路径长度为_____55_____

- 1、使用散列函数hash f(x)=x mod 11，把一个整数值转换成散列表下标，现要把数据：
- （1,13,12,34,38,33,27,22）插入到散列表中。
- （1）使用线性探查再散列法来构造散列表。
- （2）使用链地址法构造散列表。
- 确定其装填因子
- 针对这两种情况：计算查找成功所需的平均探查次数

散列地址	0	1	2	3	4	5	6	7	8	9	10
关键字	33	1	13	12	34	38	27	22			
成功比较	1	1	1	3	4	1	2	8			

不成功比较 9 8 7 6 5 4 3 2 1 1 1

- $ASL_{succ}=21/8$



ASL_{succ}= (4*1+3*2+3) /8=13/8

- 使用散列函数H(Key) = (key × 3) MOD 7，其**装填因子**为0.7，采用线性探测再散列法处理冲突，将关键字序列（7、8、30、11、18、9、14）插入到散列表中，构造散列表并计算查找成功所需的平均探查次数。

- 有一待排序序列为{46，27，88，42，65，14，68，34，12}，请给出采用快排序方法对其进行排序(以第一个元素作分界值)时，每一次的划分过程。

- 12 27 34 42 14【46】 68 65 88

- 设{42，13，24，91，23，16，05，88}为初始的待排序序列，请画出把它们调整成初始大顶堆的过程：指出开始结点并给出完成每一步调整后的树形结构图。

练习

- 有10个带权结点，其权值分别为30，50，60，20，78，45，190，180，196，125，试以它们为叶子结点生成一棵哈夫曼树，要求左子树根结点的权**小于等于**右子树根结点的权，画出相应的哈夫曼树并计算该哈夫曼树的带权路径长度。

堆排序

- 写出该堆构建的过程以及排序输出时的调整过程
- 2 5 18 9 34 25

- 模式串P=‘abaabcac’的next函数值序列为，nextval函数值序列为_____。

已知一个图的顶点集V和边集G分别为：

V={0，1，2，3，4，5，6，7}；

G={(0,1)3,(0,3)5,(0,5)18,(1,3)7,(1,4)6,(2,4)10,(2,7)20,(3,5)15,(3,6)12, (4,6)8,(4,7)12}；

其中G的元素(vi,vj)w表示顶点vi和vj之间边的权重为w。试按照普里姆算法从顶点2出发得到最小生成树，并写出依次得到的各条边。