华图网校
V.HUATU.COM
SINCE 2001
乐享品质　高效学习

数据结构与算法

刘佩贤

华图网校

目录

数据结构与算法

第 1 章 绪 论

主要内容：

1.相关术语

2.数据之间的逻辑关系（逻辑结构）

3.数据在计算机中的存储方式（存储结构）

4.实现数据操作的算法（数据的运算）

5.算法

- 数据结构是一门研究非数值计算的程序设计问题中的操作对象，以及它们之间的关系和操作等相关问题的学科。
- 数据结构概念包括三个方面（三要素）

  ① 数据之间的逻辑关系（逻辑结构）

  ② 数据在计算机中的存储方式（存储结构）

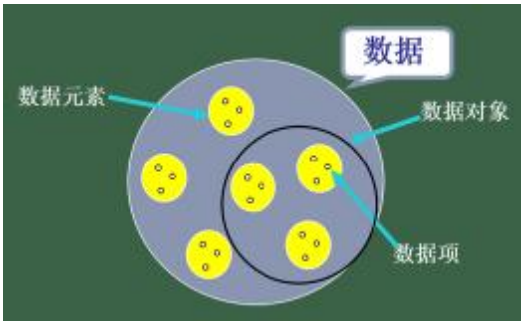
  ③ 实现数据操作的算法（数据的运算）

1.相关术语

① 数据：能输入计算机并能让计算机程序处理的符号的总称。

② 数据元素：数据的基本单位。可以进一步细分为若干数据项。数据项是不可分割的最小单位。

③ 数据对象：具有相同性质的数据元素的集合，是数据的一个子集。



2.数据之间的逻辑关系（逻辑结构）

逻辑结构：描述数据元素间的逻辑关系，与数据的存储无关。

数据逻辑结构可以用二元组（D，R）描述
D：数据元素定义域集合
R：数据元素上的关系 ri 的集合

举例：一种数据结构 G＝（D，R），其中

D=｛小李，老李，小张，老张｝

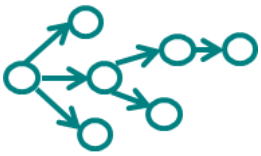
R=｛r｝

且 r=｛<老李,小李>，<老张,小张>｝

逻辑结构种类：

① 线性结构（一对一）： 

② 树形结构（一对多）：

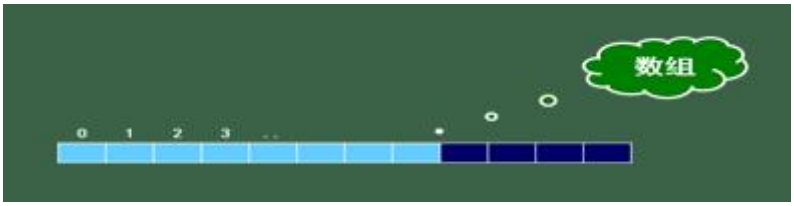


③ 网状、图结构（多对多）：

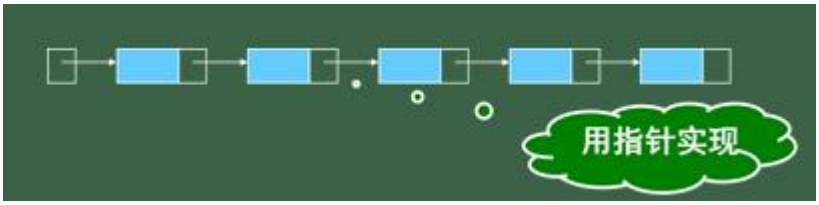
④ 集合（无顺序）：

3.数据在计算机中的存储方式（存储结构）

① 顺序存储结构：把逻辑上相邻的元素存储在物理位置相邻的存储单元中。



② 链式存储结构：对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示。



③ 索引存储结构：在存储结点信息的同时，还建立附加的索引表。

④ 散列存储结构：以数据元素的关键字的值为自变量，通过某个函数（散列函数）计算出该元素的存储位置。

4.实现数据操作的算法（数据的运算）

数据运算定义在数据的逻辑结构上，即施加于数据的操作。

算法的实现与数据的存储结构密切相关，即：

- Ø 逻辑结构决定了算法的设计。
- Ø 物理结构决定了算法的实现。

5.算法

定义：是对特定问题求解步骤的一种描述，是指令的有限序列。

特点：有穷性、确定性、可行性、零个或多个输入、一个或多个输出

判断题：算法必须有输出，但可以没有输入。

- 算法效率的度量——时间复杂度：大 O 表示法
  - Ø 例：$f(n)=n^2$,$f(n)=1/2(n(n-1))$,

$f(n)=(n-1)(n+2)$ 均表示为：$O(n^2)$

  - Ø 加法规则：若 $T1(n)=O(f1(n))$, $T2(n)=O(f2(n))$则两段程序连在一起的时间代价为:

$T(n) =T1(n)+T2(n)=O(max(f1(n),f2(n))$

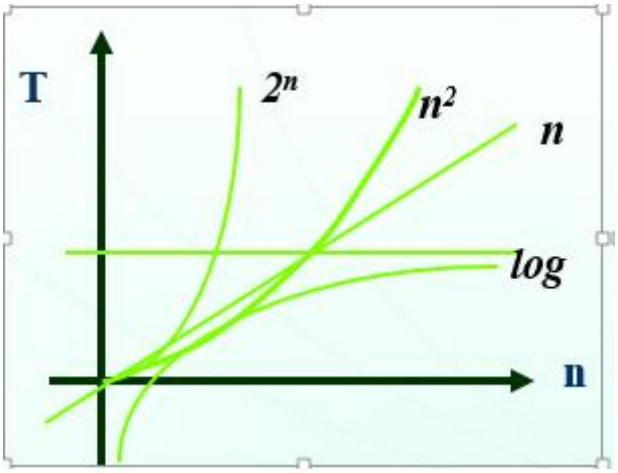
例：语句段	频度 f(n)	时间复杂度 T(n)
x=x+1	1	O(1)
for(j=1;j<=3n+5;j++) x=x+1	3n+5	O(n)
for(i=1;i<=3n;i++) for(j=1;j<=n;j++)	$3n^2$	$O(n^2)$

```
x=x+1;
```

常见的渐进时间复杂度按数量级递增排列为：

$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<$

$O(n^2)<O(n^3)<O(2^n)$



习题解析

1.从逻辑上可以把数据结构分为（　　）两大类。

A. 动态结构、静态结构

B. 顺序结构、链式结构

C. 线性结构、非线性结构

D. 初等结构、构造型结构

2.线性表的逻辑顺序与存储顺序总是一致的，这种说法（　　）。

A.正确　　B.不正确

3.线性表若采用链式存储结构思想，要求内存中可用存储单元的地址（　　）

A.必须是连续的　　B.部分地址必须是连续的

C.一定不是连续的　D.连续或不连续都可以

4. 设某数据结构的二元组形式表示为A=(D,R),D={01,02,03,04,05,06,07,08,09},R={r},r={<01,02>,<01,03>,<01,04>,<02,05>,<02,06>,<03,07>,<03,08>,<03,09>},则数据结构A是（　）。

A.线性结构　　B.树型结构

C.物理结构　　D.图型结构

第 2 章 线性表

主要内容

1.线性表的逻辑结构——线性结构

2.线性表的顺序存储结构及运算实现

3.线性表的链式存储结构及运算实现

1.线性表的逻辑结构——线性结构

线性表是 n 个数据元素的有限序列

当 n=0 时，称线性表为空表

当 n>0 时，线性表中各元素都有确定序号

线性结构的基本特征:

线性结构是一个数据元素的有序（次序）集

① 集合中必存在唯一的一个“第一元素”;

② 集合中必存在唯一的一个“最后元素”;

③ 除最后元素之外，均有唯一的后继;

④ 除第一元素之外，均有唯一的前驱。

⑤ 表中数据元素的类型是相同的。

- 数据结构的运算是定义在逻辑结构层次上的，而运算的具体实现则是建立在存储结构上的。

① 初始化线性表 L：InitList(L)

② 求线性表 L 的长度：GetLength(L)

③ 按序号取元素：GetNode（L，i）

④ 按值查找：LocateList（L，e）

⑤ 插入新元素：InsertList（L，i，e）

⑥ 在线性表中删除元素：DeleteList（L，i）

⑦ 把已有线性表置为空表：ClearList（L）

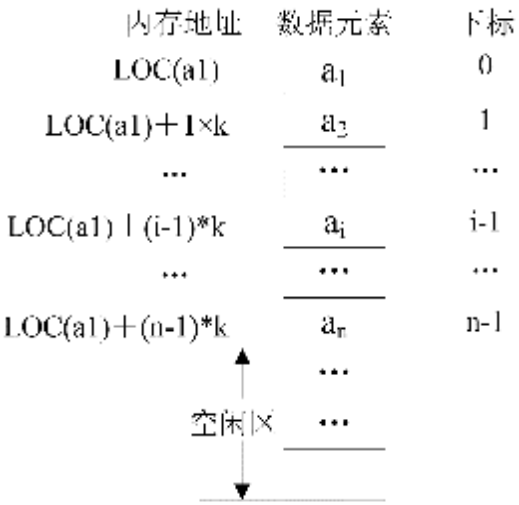
2.线性表的顺序存储结构及运算实现

（1）存储结构

用一组地址连续的存储单元依次存储线性表的元素。
相邻元素存储位置关系：

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

k 为一个数据元素所占存储量



- 在 C 语言中，可用下述类型定义来描述顺序表：

```
#define MaxLen 100
typedef int DataType;
typedef struct{
      DataType data[MaxLen];
      int length;
        }SeqList;
SeqList L;
```

（2）运算实现

C 语言中的数组下标从“0”开始，因此，若 L 是 SeqList 类型的顺序表，表中第 i 个元素是 L.data[i-1]

① 初始化顺序表 InitList(L)的实现

```
void InitList(SeqList *L)
    {
                L->length=0;
        }
```

算法的时间复杂度为 O(1)

② 求线性表长度 GetLength(L)的实现

```
int GetLength(SeqList *L)
    {
                return L->length;
        }
```

该算法的时间复杂度为 O(1)

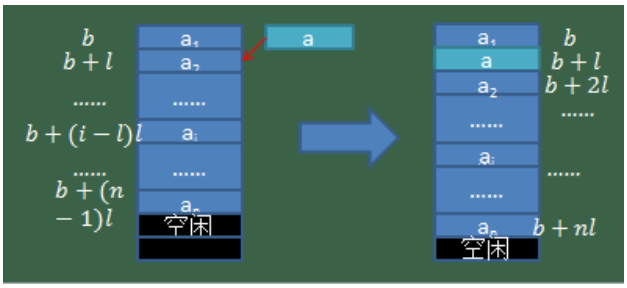
③ 按序号取元素 GetNode(L,i)的实现：时间复杂度 O(1)

```
DataType GetNode(SeqList *L,int i)
 {
       if(i<1 || i>L ->length)
    {
       printf("error!");
       exit(1);
    }
       return L->data[i-1];
 }
```

④ 查找运算 Locate(L,x)的实现，时间复杂度为 O(n)。（n=L->length）

```
int Locate(SeqList *L,DataType x)
{ int i;
  i=0;
  while(i<L->length && L->data[i]!=x)
   i++;
  if(i<L->length)  return  i ;
  else  return  -1;
}
```

⑤ 插入一个元素，从插入位置开始，顺序表中的元素依次向后移，将要插入的元素放在移出的空位中。



- 实现需注意的问题：

Ø 插入之前需先检查表空间是否足够；

- 插入位置：1<=i<=n+1；
- 注意数据的移动方向；
- 插入之后线性表的长度增加 1。

时间复杂度为 O(n)

```
void InsertList(SeqList *L, int i,Datatype x)
{ if (i<1 || i>L->length+1)
    { printf("Error!") ;   //插入位置出错
      exit(1) ; }
  if(L->length==MaxLen)
    { printf("overflow!") ;      //表已满
      exit(1);
    }
  for(j=L->length-1;j>=i-1;j--)
    L->data[j-1]=L->data[j];  //数据元素后移
  L->data[i-1]=x;             //插入x
  L->length++;                //表长度加1
}
```

⑥ 顺序表的删除运算 DeleteList(L,i)的实现

- 删除之前需先检查表是否为空；
- 删除位置：1<=i<=n；
- 若删除的元素还有用，则在删除之前应先取出；
- 删除之后线性表的长度减 1。

时间复杂度为 O(n)

```
void DeleteList(SeqList *L, int i)
{  if(i<1 || i>L->length)
       { printf ("position error");
       exit(1); }
   for(j=i;j<=L->length-1;j++)
    L->data[j-1]=L->data[j];
    //向前移动元素
   L->length--;
}
```

习题解析

1．数据在计算机存储器内表示时，物理地址与逻辑地址相同并且是连续的，称之为(　　)：

A.存储结构　　B.逻辑结构

C.顺序存储结构　　D.链式存储结构

2.一个向量第一个元素的存储地址是 100，每个元素的长度为 2，则第 5 个元素的地址是(　　)

A.110　　B.108　　C.100　　D.120

3.向一个有 127 个元素的顺序表中插入一个新元素并保持原来顺序不变，平均要移动（　）个元素　A.8　　B.63.5　　C.63　　D.7

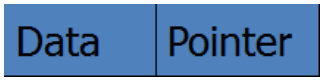
4.若长度为 n 的线性表采用顺序存储结构，在其第 i 个位置插入一个新元素的算法时间复杂度为（　）(1≤i≤n+1)。

A.O(0)　　B.O(1)　　C.O(n)　　D.O(n^2)

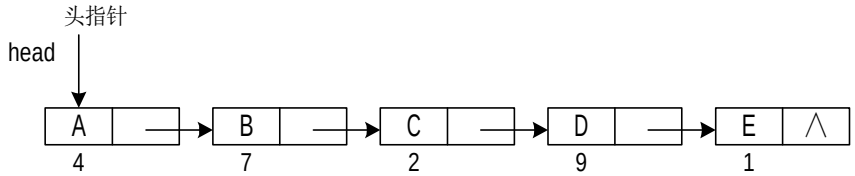
3.线性表的链式存储结构及运算实现

① 单链表

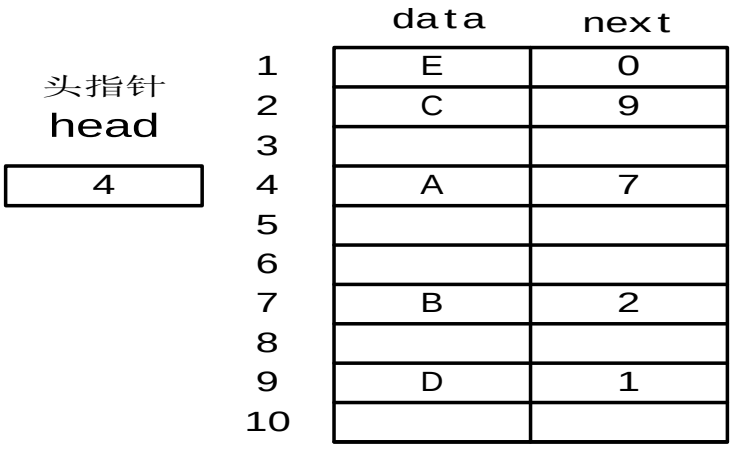
存储空间上一个结点对应线性表上一个元素，结点分为两个字段（或两个域）。一个字段存放元素的数据值；另一个字段存放指针，指向后继元素。



- 头指针：指示链表中第一个结点。
- 头结点：在第一个结点之前附设的结点，其指针域指向链表中第一个结点。



(b)线性表的逻辑状态



(a)线性链表的物理状态

- C 语言采用结构体描述结点和单链表如下：

```
typedef struct node{
  DataType data;      /*结点值*/
  struct node *next;
  /*存储下一个结点的地址*/
}ListNode,*LinkList;
ListNode *p;
LinkList head;
```

同一数据类型的两个变量

- 在 C 语言中通常采用以下函数进行相关操作：

Ø 为结点变量分配存储空间的标准函数

p=(ListNode *)malloc(sizeof(ListNode));

函数 malloc 分配一个类型为 ListNode 的结点变量的空间,并将其首地址放入指针变量 p 中。

Ø 释放结点变量空间的标准函数　　free(p);

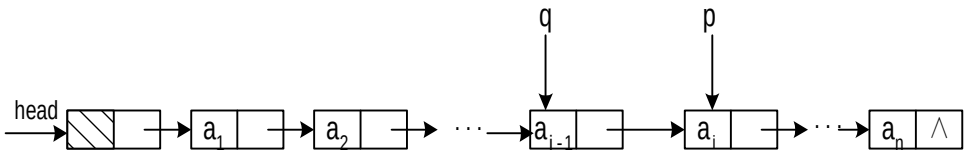
- 初始化链表 InitList(L)的实现：建立一个空的带头结点的单链表。

```
void InitList(LinkList L)
{
        L=(ListNode*)malloc(sizeof(ListNode));
        L->next=NULL;
}
```

- 求链表长度 GetLength(L)的实现，时间复杂度为 O(n)。

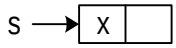
```
int GetLength(LinkList L)
{
        int num=0;
      ListNode *p;
      p=L->next;
      while(p!=NULL)
      {  num++;
         p=p->next;
       }
       return(num);
}
```

- 链表的插入

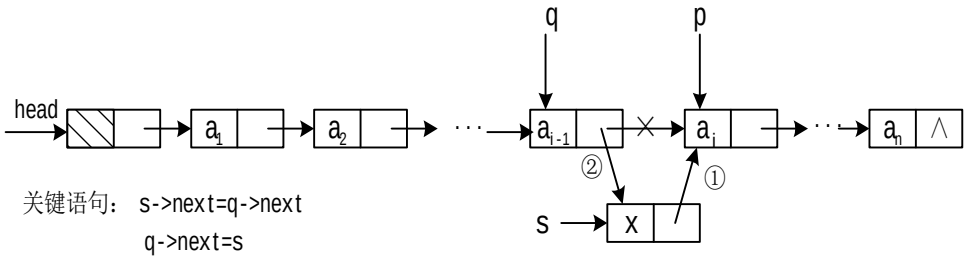


(a)找到插入位置

```
s=(LNode *)malloc(sizeof(LNode));
s->data=x;
```



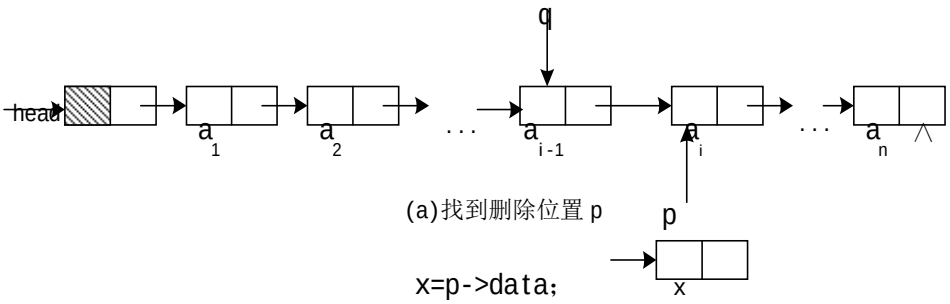
(b)申请新结点s，数据域置x



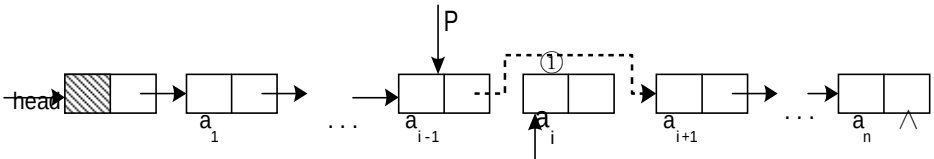
(c)修改指针域，将新结点s插入

```
void InsertList(LinkList head,DataType x,int i)
{       ListNode *p,*q,*s;
      int j=0;  p=head;
      if(i<1||i>GetLength(head)+1) exit(1);
      s=(ListNode *)malloc(sizeof(ListNode));
      s->data=x;
      while(j<i)
      {   q=p;p=p->next;
          j++;
      }    /*找到插入位置*/
      s->next=q->next;
      q->next=s;
}
```

- 链表的删除



(b)返回被删除结点数据 x



关键语句：p->next=q->next;
free(q)

(c)修改指针域，将结点 p 删除

优点：可迅速找到结点前趋。

缺点：增加存储空间。（每个结点增加了一个指针域）

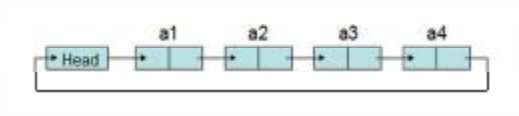
- 双向链表结点的定义如下：

```
typedef struct dlistnode{
    DataType data;
```

```
void DeleteList(LinkList head,int i)
{  ListNode *p,*q;
   int j=1;  p=head;
   if(i<1 || i>GetLength(head)) exit(1);
   while(j<i)
   {    p=p->next;
        j++;   }
   q=p->next; // 使q指向被删除的结点ai
   p->next=q->next;//将ai移除
   free(q);
}
```

②循环链表

- 特点：尾结点的 next 指针指向头结点，可以设头结点和尾结点指针。
- 优点：可迅速找到头、尾结点。



③ 双向链表和双向循环链表

特点：在结点中增加一个指向前趋的指针域。

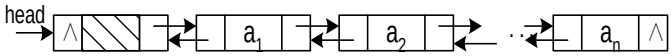
优点：可迅速找到结点前趋。

缺点：增加存储空间。（每个结点增加了一个指针域）

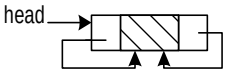
- 双向链表结点的定义如下：

```
typedef struct dlistnode{
    DataType data;
    struct dlistnode *prior,*next;
  }DListNode;
```

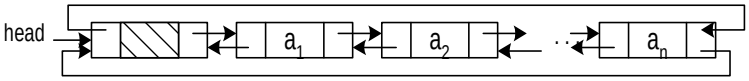
(a)空双向链表



(b)非空双向链表

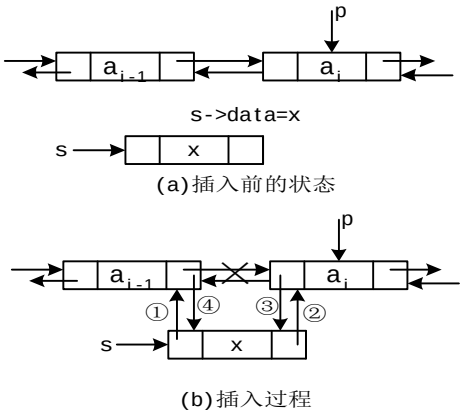


(a)空双向循环链表



(b)非空双向循环链表

- 插入：



关键语句：

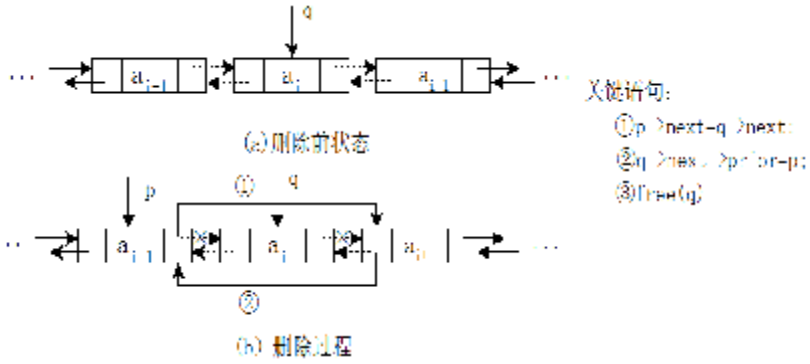
①s->prior=p->prior;

②s->next=p;

③p->prior=s;

④s->prior->next=s;

- 删除



关键语句：

①p->next=q->next;

②q->next->prior=p;

③free(q);

- 顺序实现与链接实现的比较，时间性能比较
  - Ø 定位运算：顺序表和单链表,均为 O(n)
  - Ø 读表元：顺序表-O(1) (随机存取);

    单链表-O(n)
  - Ø 插入/删除：顺序表-O(n);

    单链表-O(1) （插入、删除方便）

习题解析

1.带头结点的单链表的头指针为 head,则单链表为空的判定条件是（ ）

2.非空循环单链表的头指针为 head,则其尾指针 P 满足（ ）

A.P->next==NULL B.P==NULL

C.P->next==head D.P==head

3.线性表采用链式存储时，其元素地址（ ）

A.必须是连续的 B.一定是不连续的

C.部分地址是连续的　　D.连续与否均可

4.线性表L在（　　）情况下适用于使用链式结构实现。

A.需经常修改L中的结点值

B.需不断对L进行删除插入

C.L中含有大量的结点

D.L中结点结构复杂

第3章 栈和队列

- 栈与队列

Ø 队列和栈是两种重要的数据结构。从数据结构的角度看，栈和队列也是一种线性表。其特殊性在于栈与队列的基本操作。

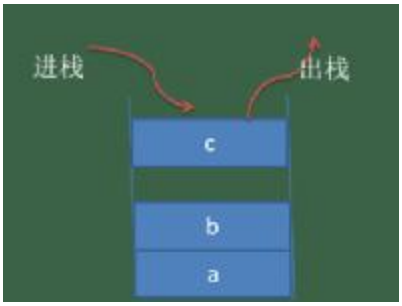
Ø 队列和栈可以是顺序表，也可以是链表，它强调的是数据的插入删除，而不是底层的实现。

1.栈

① 栈的类型定义

插入、删除只在表尾进行的线性表被称为栈。

② 栈的特点：后进先出（LIFO）



③ 栈的表示和实现

Ø 顺序栈：开辟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 top 指针指示栈顶元素的当前位置。

```
#define MaxSize 100
typedef char SElemType;
typedef struct {
                    SElemType data[MaxSize];
                    int top;
}STACK;
```

- 初始化栈 InitStack(S)，习惯上 top==-1 为空栈。

```
int InitStack(STACK *S)
{
        S->top=-1;
        return 1;
}
```

- 压栈 Push(S,x)

```
int Push(STACK *S,SElemType x)
{
        if(S->top==MaxSize-1) // top==MaxSize-1为满栈
        {
          printf("\n Stack is full!");
          return 0;
        }
        S->top++;
        S->data[S->top]=x;
        return 1;
}
```

- 判栈空 EmptyStack(S)

```
int EmptyStack(STACK *S)
{
        return (S->top==-1?1:0);
}
```

- 出栈 Pop(S,x)

```
int Pop(STACK *S,SElemType *x)
{
        if(EmptyStack(S))
        {
        printf("\n Stack is free!");
        return 0;
  }
  *x=S->data[S->top];//记住要删除的元素值
        S->top--;
        return 1;
}
```

- 取栈顶元素 GetTopStack(S)

```
SElemType GetTop(STACK *S)
{
        SElemType x;
        if(EmptyStack(S))
        {
        printf("\n Stack is free!");
        exit(1);
        }
        x=S->data[S->top];
        return x;
}
```

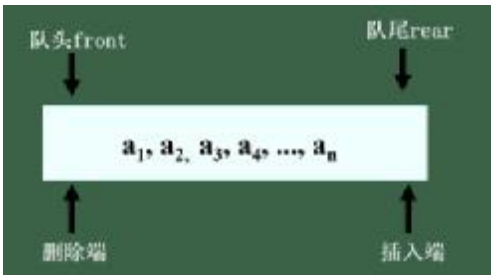
链栈：采用链式存储结构的栈，并由其栈顶指针惟一确定。它是运算受限的单链表，插入和删除操作仅限制在表头位置上进行。top 的后继指针指向栈顶元素。

```
typedef struct Snode { //定义链栈结点类型
  SElemType data;
  struct Snode  *next;
}LinkSTACK;
LinkSTACK *top;
```

2.队列

① 队列定义：若线性表的插入操作在一端进行，删除操作在另一端进行，则称此线性表为队列。

② 队列特点：先进先出（FIFO）



③ 队列的存储表示

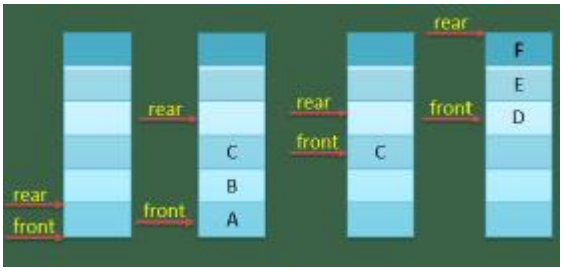
顺序队列

```
#define MaxSize <队列的最大容量>
typedef struct{
  QElemType data[MaxSize];
  int front,rear;
}QUEUE;
```

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。设队头指针为 front，队尾指针为 rear。

约定：当 front=rear=0，表示空队列；

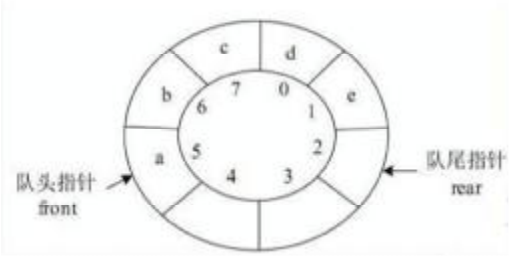
front 指向队头元素；rear 指向队尾元素的下一个位置。



- 队列的假溢出

系统作为队列用的存储区还没有满，但队列却发生了溢出，我们把这种现象称为"假溢出"。

循环队列：将队列的头、尾相连形成一个环，构成循环队列。

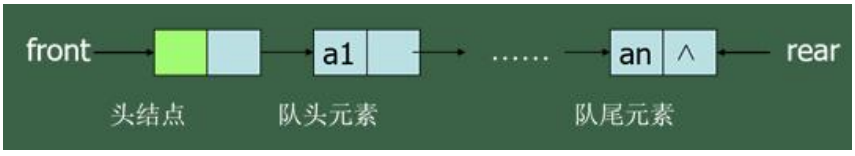


循环队列中，由于入队时尾指针向前追赶头指针；出队时头指针向前追赶尾指针，造成队空和队满时头尾指针均相等。因此，无法通过条件 front==rear 来判别队列是"空"还是"满"。

解决方法：少用一个存储空间

链队列：用链表表示的队列简称为链队列。

为了链队列的操作方便，增加一个头结点，front 指向头结点，rear 指向队尾元素。如图所示：



习题解析

1.栈和队列都是（　　）。

A.顺序存储的线性结构

B.限制存取点的线性结构

C.链接存储的线性结构

D.限制存取点的非线性结构

2.在一个具有 n 个单元的顺序栈中，假定以地址低端作为栈底，以 top 作为栈顶指针，则当出栈时，top 变化为（　　）。

A. top 不变　　B. top＝-n

C. top＝top-1　　D. top=top+1

3.若进栈序列为 1,2,3,4,进栈过程中可以出栈，则（　　）不可能是一个出栈序列。

A. 3,4,2,1　　B. 2,4,3,1

C. 1,4,2,3　　D. 3,2,1,4

4.若已知一个栈的入栈序列是 1,2,3,…,n，其输出序列为 p1,p2,p3,…,pn，若 p1=n，则 pi 为(　)。

A.i　　B.n-i+1

C.n-i　　D.不确定

5.若在一个大小为6的数组上实现循环队列，且当前rear和front的值分别为0和3，当从队列中删除一个元素，再加入两个元素后，rear和front的值分别为（　）。

A.1和5 B.2和4

C.4和2　　D.5和1

6.进队列顺序为abcdefg，出队顺序为（　）

A　abedcfg　　B cdefgab

C　abcdefg　　D aegfdbc

队列先进先出的性质，进队顺序与出队顺序一致。

第4章 串

主要内容：

1.串定义

2.串的存储结构

3.串的模式匹配算法

1 串定义

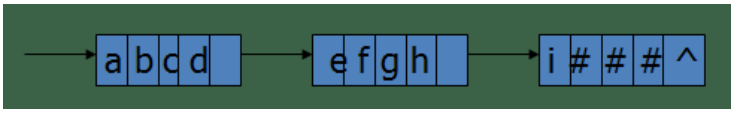
有零个或n个字符构成的有限序列。

- 记为S=‘a1a2…an’(n>=0)；其中S是串名，a1a2..an是串值，ai(i<=n 且 i>=1)可以是字母、字符、数字，n是串长度，当n=0时为空串。

2.串的存储结构

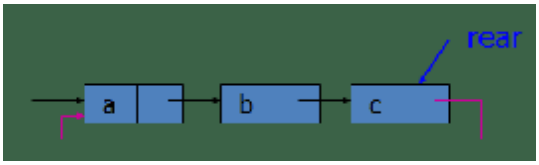
① 顺序存储结构：用一组地址连续的存储单元存储串的字符序列。

② 动态存储结构：用一组任意的存储单元（可以连续，可以不连续）存放串的字符序列。

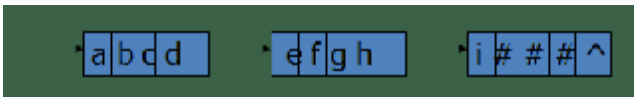


- 链式存储结构

  Ø 每个结点存放一个字符。



Ø 每个结点存放多个字符——块链结构。



3.串的模式匹配算法

- 子串的定位操作通常称为串的模式匹配。

Index(S,T,pos)

Ø 其中 S 为主串，T 被称为模式串。

Ø 若 T 为 S 子串，则返回 T 在 S 中第 pos 个字符后首次出现的位置。

Ø 若 T 不为 S 子串，则返回 0。

习题解析

1.下面关于串的的叙述中，哪一个是不正确的（　　）

A.串是字符的有限序列

B.空串是由空格构成的串

C.模式匹配是串的一种重要运算

D.串既可以采用顺序存储　也可以采用链式存储

2.串是一种特殊的线性表，下面哪个叙述体现了这种特殊性？（　　）

A. 数据元素是一个字符

B. 可以顺序存储

C. 数据元素可以是多个字符

D. 可以链接存储

3.设有两个串 p 和 q，其中 q 是 p 的子串，求 q 在 p 中首次出现的位置的算法称为(　　)

A.求子串　　B.联接

C.模式匹配　　D.求串长

第 5 章 数组和广义表

主要内容：

1.数组定义（n 维数组）

2.数组的顺序存储表示

3.矩阵的存储及运算

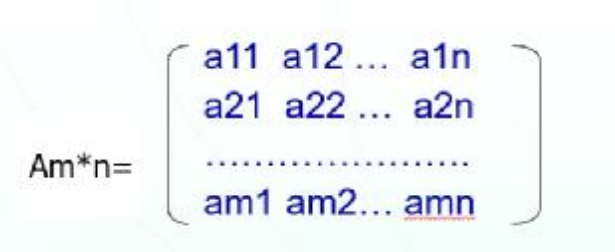
4.广义表

1.数组定义（n 维数组）

当 n=1 时，称为一维数组。

当 n>2 时，称为多维数组。一个 n 维数组是由若干个 n-1 维数组构成的。

举例：二维数组 m*n



可以看成由若干个一维数组组成，这些一维数组或按行或按列向量构成

2.数组的顺序存储表示

问题：存储单元是一维的结构，而数组是多维结构，如何用一组连续存储空间存放数组的数据元素呢？

解决方法：

行优先存储；

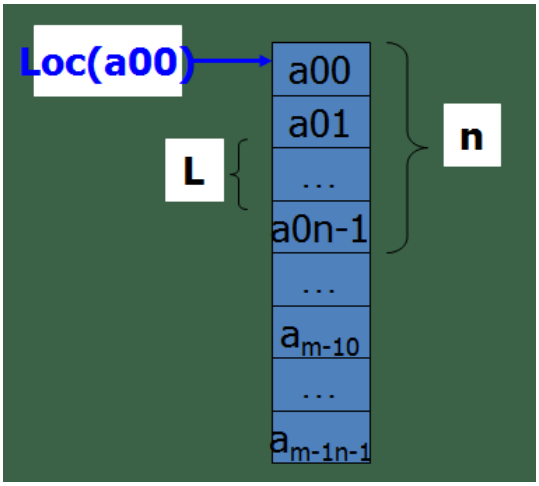
列优先存储。

例：数组元素满足 1<=i<=m,1<=j<=n

设每个数据元素占 L 个存储单元，第一个元素 a00 存储地址是 Loc(a00)，问 aij 在空间上的存储地址是什么？

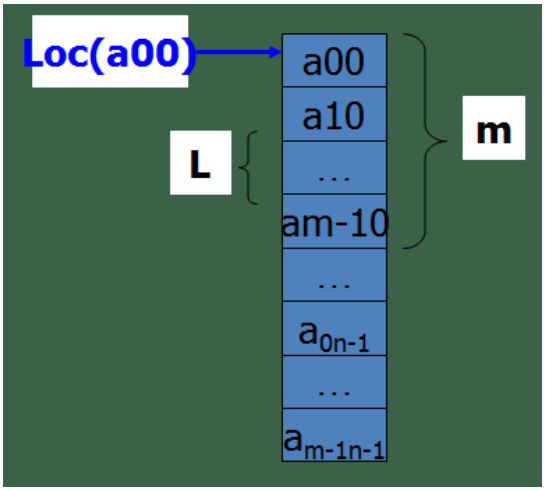
① 行优先存储

假设：行、列的下界为 0



Loc(aij)=Loc(a00)+(i*n+j)*L

② 列优先存储



Loc(aij)=Loc(a00)+(j*m+i)*L

3.矩阵的存储及运算

- 一般矩阵的存储：通常用二维数组存储矩阵。

特点：矩阵中的每个元素占用二维数组中的相应位置。

- 特殊矩阵：矩阵中相当一部分元素取值相同或都为 0 或元素在矩阵中分布有一定规律。

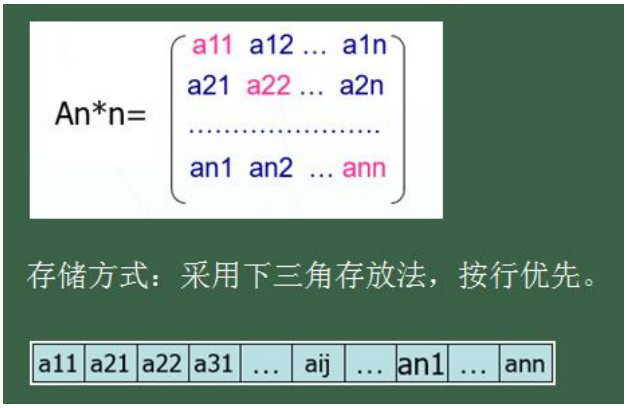
特殊矩阵的存储方式：通常采用压缩存储。为多个值相同的矩阵元只分配一个存储空间；对 0 元不分配空间。

① 对称矩阵

定义：对于一个 n*n 方阵，且 aij=aji（1<=i<=n,1<=j<=n）,则称为对称矩阵。

特点：矩阵中元素关于对角线元素对称。

存储方式：采用三角存放法。即指存储对角线及对角线以下或对角线以下的元素。对于 n 阶对称矩阵，仅用 n(n+1)/2 个存储单元。



② 三角矩阵

特征：n 阶矩阵，对角线下（或上）的元素均为零或各元素相等于某一个常数 C。

存储方式：采用三角存放法。即指存储对角线及对角线以下的元素。除此之外，再增加一个存储常数 C 的存储单元。对于 n 阶三角矩阵，仅用 n(n+1)/2+1 个存储单元。

③ 对角线阵（对角矩阵）

特征：n*n 方阵，若他的全部非 0 元都集中在以主对角线为中心的带状区域内。

存储方式：压缩存储。

④ 稀疏矩阵的存储及运算

特征：矩阵中的许多元素为 0，并且 0 元素的个数远远大于非 0 元的个数。

存储方式：三元组表

三元组表，是顺序存储结构，稀疏矩阵中的一个非 0 元用一个三元组(i,j,value)表示；其中：

- i 表示该非 0 元在矩阵中的行号；
- j 表示该非 0 元在矩阵中的列号；

Ø Value 表示该非 0 元的值。

整个矩阵中每个非 0 元的三元组共同构成该稀疏矩阵的三元组表。

4.广义表

定义：设广义表 LS=(α1，α2,…，αn)，n 是表长。

(1)当 n=0 时，称广义表 LS 是空表；

(2)当 n>0 时， α1，α2,…，αn 是满足线性有序关系的，且αi 可以是单个元素，也可以是一个表结构。前者称为**原子**，后者称为**子表**。

举例

广义表	表长
LS1=()	n=0(空表)
LS2=(a)	n=1(只有原子)
LS3=(a,(b))	n=2
LS4=(a,b)	n=2
LS5=(a,(b,(c)))	n=2
F=(a,F)	n=2(递归表)

- 广义表的术语：

当广义表非空时：

(1)广义表的表头：表中第一个元素；

(2)广义表的表尾：除了表头之外，表中其余元素构成的子表。

特点：表头可以是原子或子表，而表尾一定是子表。

举例：

LS=((a,b),c)

Head:(a,b) Tail:(c)

LS=(a)

Head: a Tail: () 空表

LS=((),(e),(a,(b,c,d)))

Head: () Tail: ((e),(a,(b,c,d)))

广义表运算：设广义表 LS= (α1，α2,…，αn)

取广义表表头 Head(LS)= α1

取广义表表尾 Tail(LS)= (α2,…，αn)

习题解析

1.设一个一维数组第一个元素的存储单元地址为 100，每个元素的长度为 6，则它的第 5 个元素的地址是（　　）。

A.130　　B.105　　C.106　　D.124

2.三元组表不包括（　　）

A.行数　　B.列数　　C.元素值　　D.元素总数

3.设 n 阶方阵是一个上三角矩阵，则需要存储的元素个数是（　　）

A.$n^2/2$　　B.n(n+1)/2　　C.n　　D.n^2

4.广义表 L=（a，（b，c)），进行 Tail（L）操作后的结果为（　　）。

A. c　B. b，c　C.（b，c）　D.（(b，c)）

5.对一些特殊矩阵采用压缩存储的目的主要是为（　　）。

A.表达变得简单

B.减少不必要的存储空间的开销

C.去掉矩阵中的多余元素

D.对矩阵元素的存取变得简单

第 6 章 树和二叉树

主要内容：

1.树

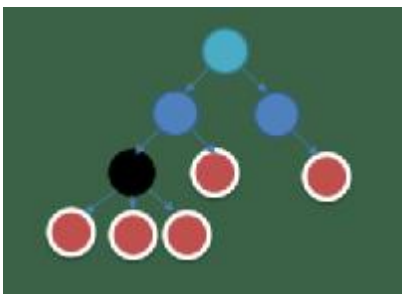
2.二叉树

3.二叉树的遍历

4.最优二叉树（哈夫曼树）

1.树

树型结构是一类重要的非线性数据结构。其中以二叉树最为常用。直观来看，树是以分支关系定义的的层次结构。



- 树的术语

Ø 树的最顶层元素为根，一棵树只有一个根。

Ø 树的结点包含一个数据元素及若干指向其子树的分支，结点拥有的子树数称为结点的度。

Ø 度为 0 的结点成为叶子。

Ø 树的度为结点最大的度。

Ø 结点子树的根称为孩子，该结点称为孩子的双亲，或者父结点。

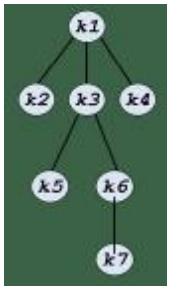
Ø 有相同父结点的孩子成为兄弟结点。

Ø 结点的层次从根开始为第一层，树中的最大层次称为树的深度。

对比树形结构和线性结构的结构特点

线性结构	树形结构
第一个数据元素 （无前驱）	根结点 （无前驱）
最后一个数据元素 （无后继）	多个叶子结点 （无后继）
其它数据元素 （一个前驱、 一个后继）	其它数据元素 （一个前驱、 多个后继）

例：有棵树如下图所示，请回答下面的问题：



① 这棵树的根结点是（　　）

② 结点 K3 的度是（　　）

③ 这棵树的度为（　　）

④ 这棵树的深度为（　　）

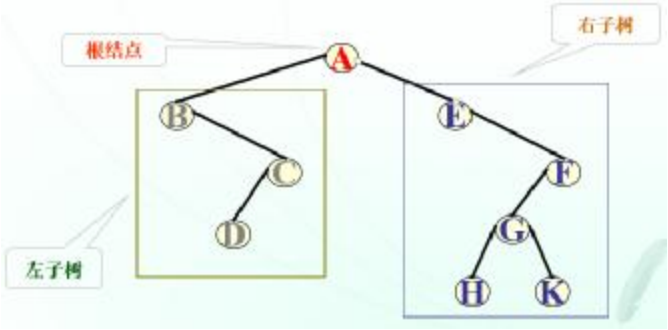
⑤ 结点 K3 的孩子是（　　）

⑥ 结点 K3 的父结点是（　　）

⑦ 这棵树的叶子结点是（　　）

2.二叉树

概念：二叉树或为空树；或是由一个根结点加上两棵分别称为左子树和右子树的互不交的二叉树组成。



- 二叉树的重要特性

Ø 性质 1：第 i 层上最多有 2^{i-1} 个结点。

Ø 性质 2：深度为 K 的二叉树最多有 2^K-1 个结点。

Ø 性质 3：设叶子（度为 0）数为 n_0，度为 2 的结点数为 n_2，则 $n_0=n_2+1$。

- 两类特殊的二叉树

① 满二叉树：指的是深度为 k 且含有 2^k-1 个结点的二叉树。（对满二叉树中的结点约定编号：自根起，从上到下，从左到右）

② 完全二叉树：树中所含的 n 个结点和满二叉树中编号为 1 至 n 的结点一一对应。（仅允许最下层右侧分支不满）

Ø 性质 4：具有 n 个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor+1$

Ø 性质 5：对含 n 个结点的完全二叉树从上到下且从左至右进行 1 至 n 的编号，则对完全二叉树中任意一个编号为 i 的结点：

① 若 i=1，则该结点是二叉树的根，无双亲，否则，编号为 $\lfloor i/2 \rfloor$ 的结点为其双亲结点；

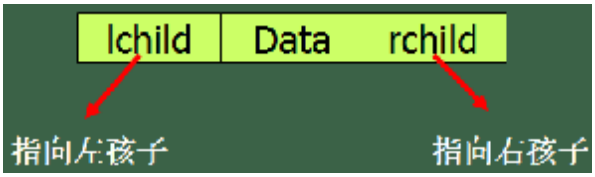
② 若 2i>n 则该结点无左孩子，否则，编号为 2i 的结点为其左孩子结点；

③ 若 2i+1>n，则该结点无右孩子结点，否则，编号为 2i+1 的结点为其右孩子结点。

- 二叉树存储结构

① 二叉树的顺序存储表示：适用于完全二叉树或满二叉树。

② 二叉树的链式存储表示：二叉链表，含有三个域：数据域、左指针域和右指针域。



- 二叉链表的 C 语言类型描述如下：

```
typedef char TElemType;
typedef struct Node {
   TElemType    data;
   struct Node  *lchild, *rchild;
} BiTNode, *BiTree;  //将BiTree定义为指向二叉链表结点结构的指针类型
```

习题解析

1.二叉树中叶子结点数为 25，仅有一个孩子的结点数为 30，则总结点数为（　　）

2.满二叉树一定是完全二叉树，完全二叉树不一定是满二叉树。（　　）

3.深度为 5 的二叉树至多有（　　）个结点，至少（　　）个结点。

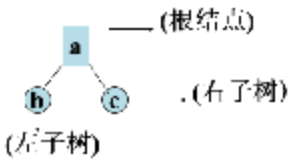
4.设某棵二叉树中有 2000 个结点，则该二叉树的最小高度为（　　）。

A.9　　B.10　　C.11　　D.12

3.二叉树的遍历

顺着某一条搜索路径巡访二叉树中的结点，使得每个结点均被访问一次，而且仅被访问一次。

由二叉树的递归定义，二叉树的三个基本组成单元是：根结点、左子树和右子树。



- 遍历算法：

① 先序：先访问根结点，然后访问左子树，再访问右子树。

```
void PreOrderTraverse(BiTree bt)
{ /*最简单的Visit函数：Visit(TElemType e){printf(e);}*/
 if (bt)
   {     Visit(bt->data);
         PreOrderTraverse (bt->lchild);
         PreOrderTraverse (bt->rchild);
   }
}
```

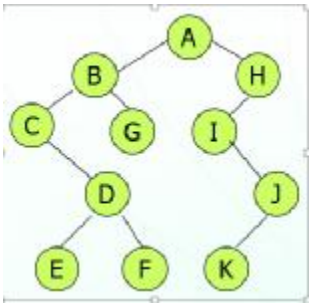
② 中序：先访问左子树，然后访问根结点，再访问右子树。

```
void InOrderTraverse(BiTree bt)
{   if (bt)
    {   InOrderTraverse(bt->lchild);
        Visit(bt->data);
        InOrderTraverse(bt->rchild);
    }
}
```

③ 后序：先访问左子树，然后访问右子树，再访问根结点。

```
void PostOrderTraverse(BiTree bt)
{  if (bt)
   {        PostOrderTraverse(bt->lchild);
            PostOrderTraverse(bt->rchild);
            Visit(bt->data);
   }
}
```

例：



先序：ABCDEFGHIJK

中序：CEDFBGAIKJH

后序：EFDCGBKJIHA

习题解析

1.任何一棵二叉树的叶结点在先序、中序和后序遍历序列中的相对次序（　　）

A、不发生改变　　B、发生改变

C、不能确定　　D、以上都不对

2.在一非空二叉树的中序遍历序列中，根结点的右边（　　）

A、只有右子树上的所有结点；

B、只有右子树上的部分结点；

C、只有左子树上的部分结点；

D、只有左子树上的所有结点。

3.二叉树的前序遍历序列中，任意一个结点均处在其子女结点的前面。（　　）

4.某二叉树的前序遍历结点访问顺序是 abdehcfe,中序遍历的结点访问顺序是 dbheafec,则其后序遍历的结点访问顺序是（　　）

5.已知某二叉树的后序遍历序列是 dabec,中序遍历序列是 debac,它的前序遍历序列是（　　）

4.最优二叉树（哈夫曼树）

- 路径长度：从树中的一个结点到另外一个结点之间的分支构成两个结点的之间的路径的长度
- 树的带权路径长度：所有点到树根之间的路径长度与结点上权的乘积之和。

。

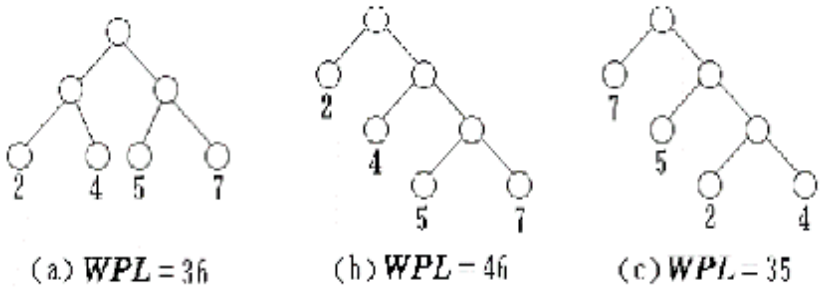
- 具有最小的带权路径长的树为最优二叉树，或者哈夫曼树。

例：具有不同带权路径长度的二叉树

(a) WPL=2*2+4*2+5*2+7*2=36

(b) WPL=2*1+4*2+5*3+7*3=46

(c) WPL=7*1+5*2+2*3+4*3=35



注意：完全二叉树不一定是最优二叉树！

- 哈夫曼树的构造

（1）根据给定的个权值构成的二叉树的集合，其中每棵二叉树中只有一个带权的根结点，其左右子树均空。

（2）在选取两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置新的二叉树的根结点的权值为其左右子树上根结点的权值之和

（3）在集合中删除这两棵树，同时将新的二叉树加入

（4）重复（2）和（3），直到只含一棵树为止

习题解析

1.以数据集{4，5，6，7，10，12，18}为结点权值所构造的哈夫曼树为（　　），其带权路径长度为（　　）。

2.　设一组权值集合 W={2，3，4，5，6}，则由该权值集合构造的哈夫曼树中带权路径长度之和为（　　）。

A.20　　B.30　　C.40　　D.45

第 7 章 图

主要内容：

1.图的术语　　2.图的存储结构

3.图的遍历　　4.最小生成树

5.拓扑排序　　6.关键路径

7.最短路径

- 图

图是一种更为复杂的非线性结构，它不像树一样具有明显的层次。在图型结构中，任意两个结点之间的联系都是可能的。

1.图的术语

- 顶点：图中的元素
- 边：顶点与顶点之间的链路
- 有向图：边是有方向的图
- 无向图：边是无方向的图
- 完全无向图：图中任意两个不同的顶点间都有一条无向边，具有 n(n-1)/2 条边。
- 完全有向图：图中任意两个不同的顶点间都有一条弧。具有 n(n-1)条边。

- 权：与图的边和弧相关的数。权可以表示从一个顶点到另一个顶点的距离或耗费。
- 无向图中，两个顶点之间有路径，称这两点连通，任意两个顶点连通，该图为连通图。
- 稀疏图：有很少边或弧的图，反之为稠密图
- 生成树：极小连通子图为该图的生成树，含有图中全部顶点，但只有足以构成一棵树的 n-1 条边。

2.图的存储结构

① 邻接矩阵表示法（数组表示法）

用一维数组存储图中顶点的信息；

用矩阵（二维数组）表示图中各顶点之间的相邻关系。

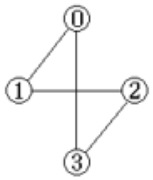
适合稠密图

图 G=(V,E),n 个顶点，则 G 的邻接矩阵是具有如下性质的 n 阶方阵。

$$A_{ij}=\begin{cases}1, & \text{如果}<i,j>\in E \text{ 或者 } (i,j)\in E\\ 0, & \text{否则}\end{cases}$$

无向图的邻接矩阵是对称的，有向图的邻接矩阵不一定是对称的。

例：



G8

G3

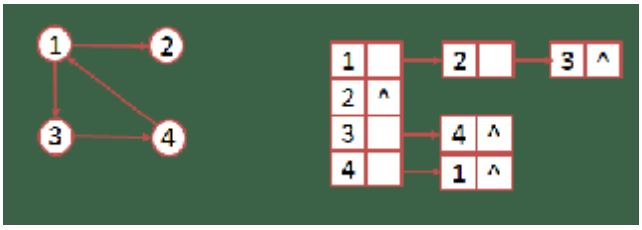
$$A.Edge=\begin{array}{c}\begin{matrix}⓪&①&②&③\end{matrix}\\ \begin{pmatrix}0&1&0&1\\1&0&1&0\\0&1&0&1\\1&0&1&0\end{pmatrix}\end{array}\begin{matrix}⓪\\①\\②\\③\end{matrix}\qquad A.Edge=\begin{array}{c}\begin{matrix}⓪&①&②\end{matrix}\\ \begin{pmatrix}0&1&0\\1&0&1\\0&0&0\end{pmatrix}\end{array}\begin{matrix}⓪\\①\\②\end{matrix}$$

- 图的邻接矩阵存储

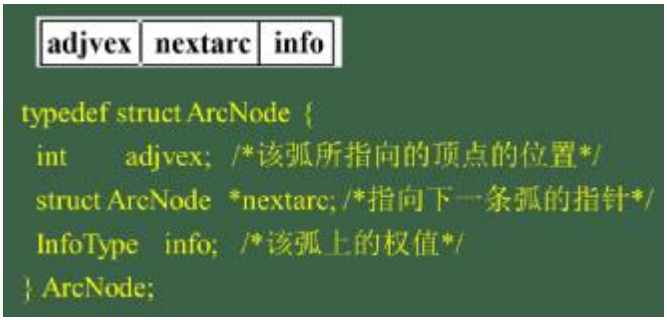
```
#define  MAX_VERTEX_NUM 100
typedef int VertexType;
typedef struct{
VertexType v[MAX_VERTEX_NUM]; //顶点表
int A[MAX_VERTEX_NUM][MAX_VERTEX_NUM];//邻接矩阵
int vexnum,arcnum;//图的顶点数和弧数
}MGraph;
```

② 图的邻接表（图的一种链式存储结构）

- 为每个顶点建立一个单链表，
- 第 i 个单链表中包含顶点 Vi 的所有邻接顶点。

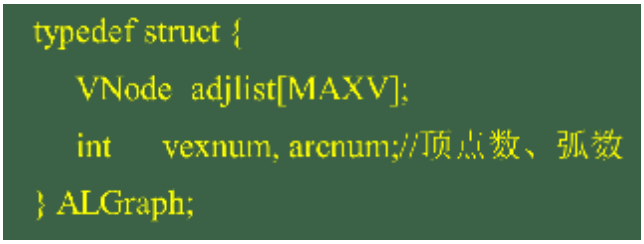


- 边结点的结构



- 顶点的结点结构

- 图的邻接表存储结构定义



习题解析

1.在一个具有 n 个顶点的无向图中，要连通全部顶点至少需要（　　）条边。

2.对于一个具有 100 个顶点，3 条边的无向图，若采用邻接矩阵表示，则该矩阵的大小是（　　），非零元素的个数为（　）个。

3.对于一个具有 5 个顶点和 7 条边的无向图，若采用邻接表表示，则表头数组的大小为(　)；所有邻接表中的结点总数是（　）

4.在无向图的邻接矩阵 A 中，若 A[i][j]等于 1，则 A[j][i]等于（　）

5.对于一个有向图，若一个顶点的入度为 k1,、出度为 k2，则对应邻接表中该顶点单链表中的结点数为（　　）。

A.k1　　B.k2　　C.k1-k2　　D.k1+k2

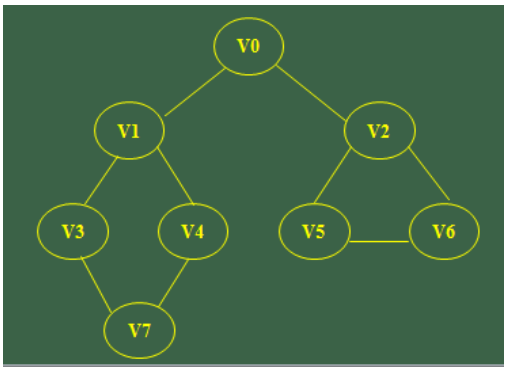
3.图的遍历

图的遍历(Traversing Graph)是指从图中的给定一顶点 v0 出发，按照某种遍历方法对图中的所有顶点访问一次，且每个顶点只能访问一次。

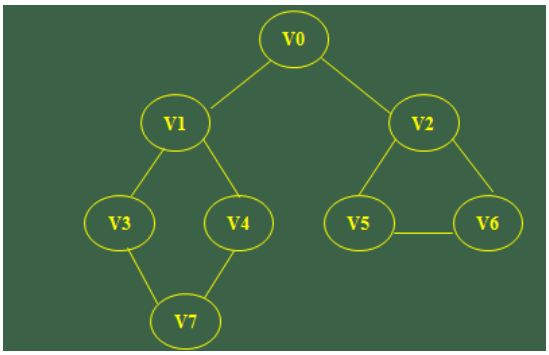
深度优先遍历(DFS)

广度优先遍历(BFS)

- 深度优先搜索(DFS)
  - Ø 访问当前顶点 vi，寻找与 vi 相邻且未被访问顶点 vj，若存在，将其作为当前点，访问，并重复上述搜寻过程。
  - Ø 否则（所有邻接点都被访问过），退回一步，寻找与前一个顶点相邻的未被访问顶点，访问，并重复上述过程。
  - Ø 若上述过程无法访问图中的所有顶点，则另选一个新顶点重新开始。
- 广度优先搜索（BFS）
  - Ø 访问出发点 v0，依次访问 v0 的各个未曾访问过的邻接点，然后分别从这些邻接点出发重复上述过程。
  - Ø 若上述过程无法访问图中的所有顶点，则另选一个新顶点重新开始。

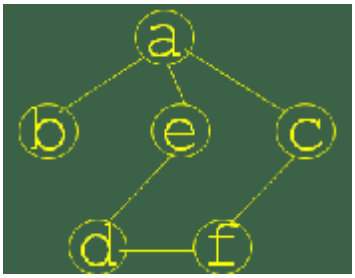


- 深度优先搜索：V0⇒ V1 ⇒V3⇒ V7⇒V4⇒V2 ⇒V5 ⇒V6
- 广度优先搜索序列：V0⇒ V1 ⇒V2⇒ V3 ⇒V4⇒V5⇒V6⇒V7



习题解析

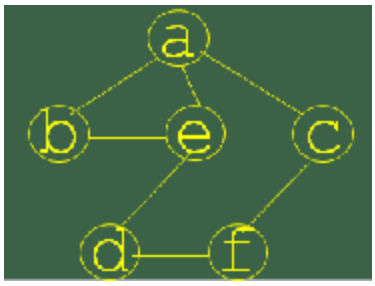
1.下图中，若从顶点 a 出发按深度优先搜索进行遍历，可能得到的一种顶点序列为（　　）。



A.a,b,e,c,d,f　　B.a,c,f,e,b,d

C.a,e,b,c,f,d　　D.a,e,d,f,c,b

2.若从顶点 a 出发按广度搜索法对下图进行遍历，可能得到的一种顶点序列为（　　）。



A.a,b,c,e,d,f　　B.a,b,c,e,f,d

C.a,e,b,c,f,d　　D.a,c,f,d,e,b

4.最小生成树

问题的提出：假设要在 n 个城市之间建立通讯联络网，则连通 n 个城市只需要修建 n-1 条线路，如何在最节省经费的前提下建立这个通讯网？

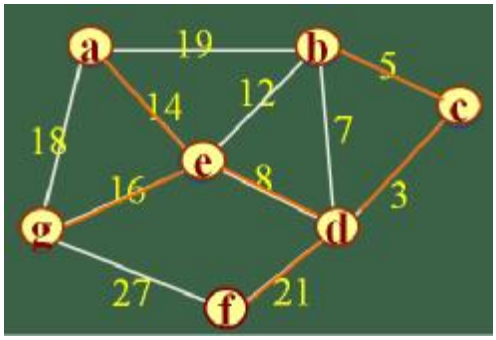
该问题等价于：构造网的一棵最小生成树，即：在 e 条带权的边中选取 n-1 条边（不构成回路），使“权值之和”为最小。

① 普里姆 Prim 算法

指定图中某个顶点 v 作为生成树的根，随后往生成树上添加新的顶点 w。

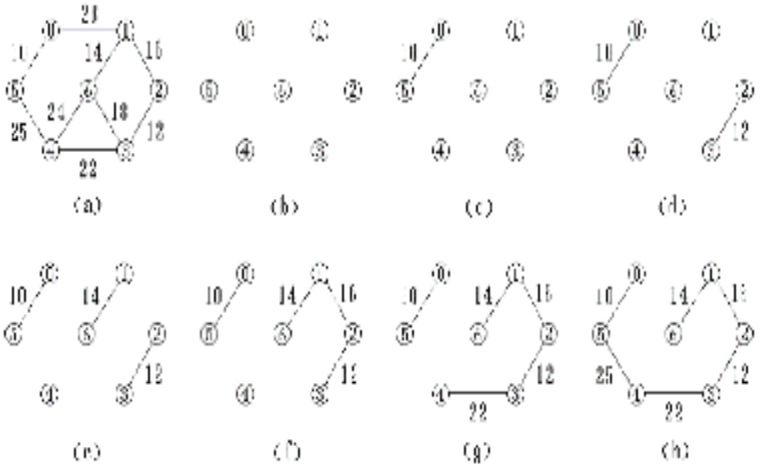
在添加的顶点 w 和已经在生成树上的顶点 v 之间必定存在一条边，并且该边的权值在所有连通顶点 v 和 w 之间的边中取值最小。之后继续往生成树上添加顶点，直至生成树上含有 n 个顶点为止。

所得生成树权值和=14+8+3+5+16+21=67



② 克鲁斯卡尔(Kruskal)算法

设有一个有 n 个顶点的连通网络 N={V,E},最初先构造一个只有 n 个顶点，没有边的非连通图 T={V,Æ},图中每个顶点自成一个连通分量。当在 E 中选到一条具有最小权值的边时，若该边的两个顶点落在不同的连通分量上，则将此边加入到 T 中；否则将此边舍去，重新选择一条权值最小的边。如此重复下去，直到所有顶点在同一个连通分量上为止。

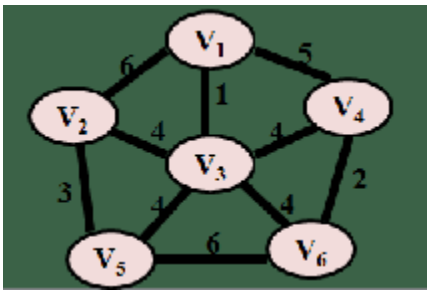


- 普里姆(Prim)算法与克鲁斯卡尔 (Kruskal)算法的比较

  - 普里姆：时间复杂度：O（n^2），n 为图中结点个数。与边无关，适合于求边稠密的网的最小生成树。
  - 克鲁斯卡尔 ：时间复杂度：O（e），e 为图中边的条数。适合于求边稀疏的网的最小生成树。

习题解析

任何一个无向连通图的最小生成树(　　)



A.有一棵或多棵　B.只有一棵

C.一定有多棵　　D.可能不存在

5.拓扑排序

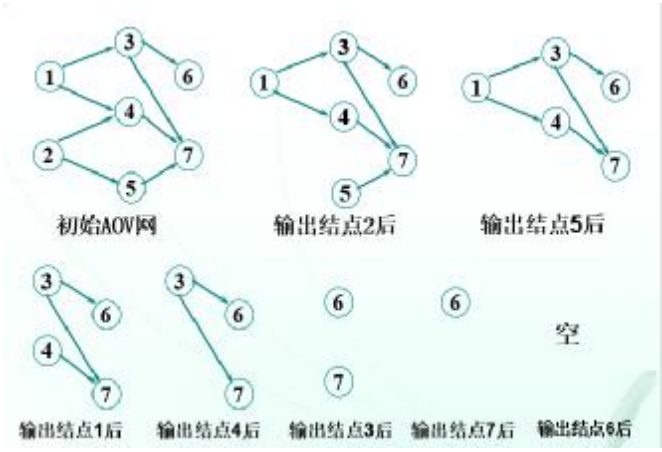
AOV 网：顶点表示活动，弧表示活动间的优先关系的有向图。

在无回路的 AOV 网中，所有的活动可以排成一个线性序列，使得每个活动的所有前驱活动都排在该活动的前边，称此序列为拓扑序列，由 AOV 网构成拓扑序列的过程称为拓扑排序。

- 拓扑排序算法思想：

选择有向图中入度为 0 的顶点输出，并从图中删去该点相邻弧；

重复上述操作，直到图中全部顶点均被输出或图中已不存在入度为0的顶点(有回路)。



习题解析

1.一个有向无环图的拓扑排序序列（　　）是唯一的。

A.一定　　B.不一定

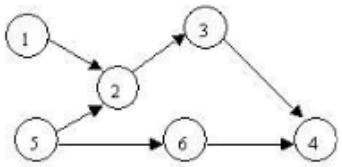
2.在有向图G的拓扑序列中，若顶点Vi在顶点Vj之前，则下列情形不可能出现的是（　　）。

A.G中有弧<Vi,Vj>　　B. G中没有弧<Vi,Vj>

C.G中有一条从Vi到Vj的路径

D.G中有一条从Vj到Vi的路径

3.在下图所示的拓朴排列的结果序列为（　　）。

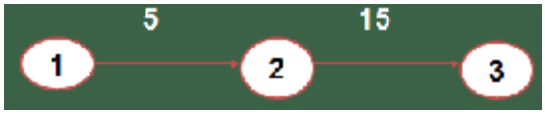


A.125634　　B.516234

C.123456　　D.521634

6.关键路径（在AOE网中实现）

AOE网：定义结点表示事件(时刻)，弧表示活动，弧的权值表示活动进行所需要的时间。



源点：表示整个工程的开始点，也称起点。

汇点：表示整个工程的结束点，也称终点。

AOE 网中的源点与汇点是唯一的！

事件结点（顶点）：表示的是时刻。

活动（有向边）：它的权值定义为活动进行所需要的时间。方向表示起始结点事件先发生，而终止结点事件才能发生。

利用 AOE 网进行工程管理时可解决下列问题：

(1) 完成整个工程至少需要多少时间?

(2) 哪些活动是影响工程进度的关键？

从源点到汇点具有最大路径长度的路径称为关键路径，关键路径长度就是整个工程所需的最短工期。

7.最短路径

最短路径问题：从图中某一顶点(称为源点)到达另一顶点(称为终点)的路径可能不止一条，如何找到一条路径使得沿此路径上各边上的权值总和达到最小，例：公交查询系统。

问题解法：

- Ø 求从某个源点到其余各点的最短路径—迪杰斯特拉算法（Dijkstra 算法）
- Ø 每一对顶点之间的最短路径—弗洛伊德算法（Floyd 算法）

习题解析

关键路径是事件结点网络中（　　）。

A.从源点到汇点的最长路径

B.从源点到汇点的最短路径

C.最长的回路

D.最短的回路

第 8 章 查找

主要内容：

1.基本概念

2.静态查找表

3.动态查找表

4.散列表也叫哈希表（Hash）

1.基本概念

- 查找表的常见操作

① 查询某个“特定的”数据元素是否在查找表中；

② 检索某个“特定的”数据元素的各种属性；

③ 在查找表中插入一个数据元素；

④ 从查找表中删去某个数据元素。

- 查找表可分为两类：

① 静态查找：仅作查询和检索操作。查找前后查找表未发生变化。

② 动态查找：在查询之后，将 “不在查找表中”的数据元素插入到查找表中；或者，从查找表中删除“在查找表中”的数据元素。

- 平均查找长度——衡量查找算法效率的标准

对于含有 n 个记录的查找表，查找成功时的平均查找长度为：

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

n 是查找表中记录的个数

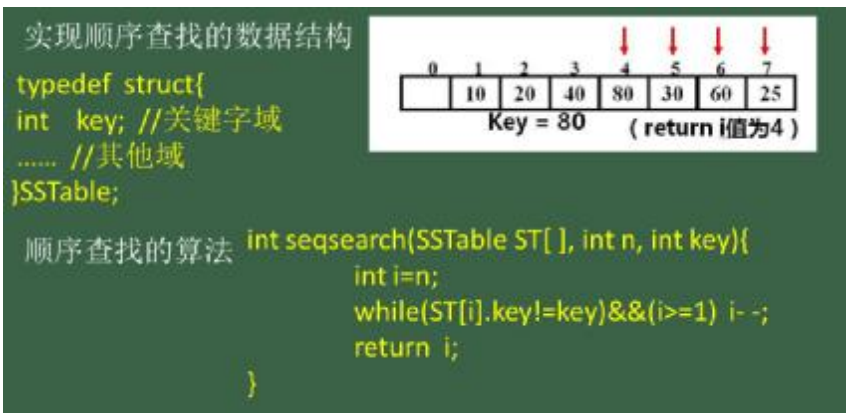
Pi 是查找第 i 个记录的概率

Ci 是找到第 i 个记录所需进行的比较次数。

2.静态查找表

① 顺序表的查找

从顺序表的一端开始，用给定的 key（关键字）逐个与各记录的关键字比较。如某个记录的关键字与给定值比较相等，则查找成功；反之，若直至顺序表的另一端，其关键字与给定值比较都不等，则表明表中没有所查记录，则查找不成功。



- 实现顺序查找的数据结构
- 顺序查找的算法
- 查找成功时，顺序查找的平均查找长度为：

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

② 有序表查找——折半查找或二分法查找

前提：静态查找表必须按关键字有序

折半查找的时间复杂度为 $O(\log_2 n)$。

思想：

- Ø 在有序表中，取中间元素作为比较对象，若给定值与中间元素的关键字相等，则查找成功；
- Ø 若给定值小于中间元素的关键字，则在中间元素的左半区继续查找；
- Ø 若给定值大于中间元素的关键字，则在中间元素的右半区继续查找。
- Ø 不断重复上述查找过程，直到查找成功，或所查找的区域无数据元素，查找失败。

- 二分查找的算法

```
• 二分查找的算法
int Search_Bin( SSTable ST[ ], int n, int  key)
{  int low, high, mid;
   low=1; high=n;
  while(low<=high)
   {   mid=(low+high)/2;
      if(ST[mid].key= = key)  return mid;
    else
        if( key< ST[mid].key)  high=mid-1;
        else  low=mid+1;
   }
 return 0; }
```

习题解析

1.顺序查找法适合于存储结构为（　　）的线性表。 A．散列存储　　B．顺序存储或链接存储

C．压缩存储　　D．索引存储

2.采用顺序查找方法查找长度为 n 的线性表时，每个元素的平均查找长度为（　　）。

A．n　　B．n/2　　C．(n+1)/2　　D．(n-1)/2

3.采用二分查找方法查找长度为 n 的线性表时，每个元素的平均查找长度为（　　）。

A. O（n^2） B. O($n\log_2 n$) C. O(n) D. O($\log_2 n$)

4. 有一个有序表为{1，3，9，12，32，41，45，62，75，77，82，95，100}，当二分查找值为 82 的结点时，（ ）次比较后查找成功。

A. 1 B. 2 C. 4 D. 8

3.动态查找表

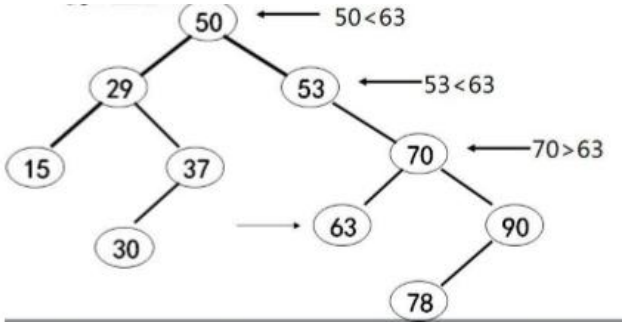
对于给定值 Key，若表中存在其关键字等于 Key 的记录，则查找成功，否则插入该记录。称此类查找表为动态查找表。

- 二叉排序树（BST）

定义：如果一棵二叉树中，每个结点的值都大于它的左子树上所有结点的值，而小于它的右子树上所有结点的值，则称此树为二叉排序树。

重要性质：中序遍历一棵二叉树所得的结点访问序列是键值的递增序列。

- 例：查找 63



- 二叉排序树的数据类型描述

```
struct BtreeNode{
ElemType key; //关键字
BtreeNode *lchild,*rchild;//左、右孩子
}BtreeNode, *Bitree;
```

- 二叉排序树的查找算法

```
BiTree SearchBST (BiTree T, KeyType key)
{
  if( (T==NULL)||(key==T->key)) return (T); //查找结束
  else
      if (key < T->key)
            return(SearchBST(T->lchild, key)); //在左子树中继续查找
      else
            return(SearchBST(T->rchild, key)); // 在右子树中继续找
}//SearchBST
```

习题解析

1.二叉排序树中，最小值结点的（　　）。

A.左指针一定为空

B.右指针一定为空

C.左、右指针均为空

D.左、右指针均不为空

第 9 章 排序

所谓排序是指将一组“无序”的记录序列调整为“有序”的记录序列的过程。

例如：下列是一组记录对应的关键字序列

(52，49，80，36，14，58，61，23，97，75)

调整为

(14，23，36，49，52，58，61 ,75，80，97)

- 排序概念

趟（Pass）

实现一个元素的有序性所完成的操作。

稳定性

若在排序前两个记录在序列中的关系为…R_i…R_j…，且 R_i 与 R_j 的关键字相同 $Key_i=Key_j$，排序后若…R_i…R_j…称排序方法稳定，若…R_j…R_i…称排序方法不稳定。

内部排序和外部排序:若整个排序过程不需要访问外存便能完成，则称此类排序问题为内部排序；反之，若参加排序的记录数量很大，整个序列的排序过程不可能在内存中完成，则称此类排序问题为外部排序。

- 排序方法分类

1.插入排序　　直接插入排序

　　希尔排序

2.交换排序　　冒泡排序

　　快速排序

3.选择排序　　简单选择排序

4.归并排序　　堆排序

5.基数排序

- 对待排序记录类型的顺序表描述：

```
#define MAXSIZE 20  //一个顺序表的最大长度
typedef int KeyType; //定义关键字为整数类型
typedef struct{
    KeyType key;     //关键字项
    InfoType otherinfo;   //其他数据项
}RedType; //记录类型
typedef struct{
   RedType r[MAXSIZE+1]; //r[0]闲置或用作哨兵单元
   int length;     //顺序表长度
}SqList; //顺序表类型
```

1. 插入排序

① 直接插入排序

每一趟都在已排好序的有序子表中插入一个元素，使有序子表不断扩大。

平均时间复杂度：$O(n^2)$

稳定性：算法稳定

- 直接插入排序算法描述

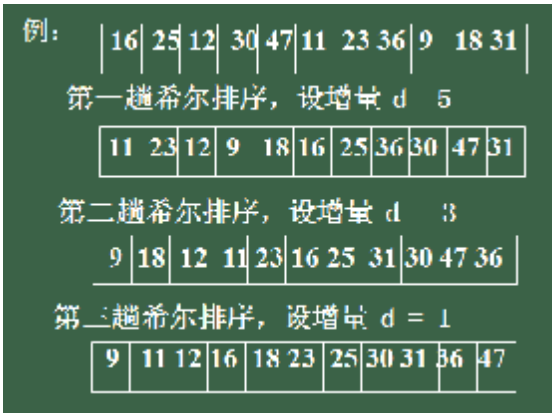
```
void InsertionSort ( SqList &L )
{// 对记录序列 R[1..L.length]作直接插入排序。
  for ( i=2; i<=L.length; ++i )
    { L.r[0] = L.r[i]; // 复制为监视哨
      for(j=i-1;L.r[0].key < L.r[j].key; j-- )
          L.r[j+1] = L.r[j]; // 记录后移
      L.r[j+1] = L.r[0]; // 插入到正确位置
  }
}// InsertSort
```

② 希尔排序（Shell 排序）又称缩小增量排序，是直接插入排序的改进。

基本思想：将待排记录序列按增量 dh 值分成若干子表进行直接插入排序；然后缩小增量值，再分割，再排序，直至整个序列“基本有序”时，再对整个序列做一次直接插入排序。增量序列不同，时间复杂度不同

稳定性：算法不稳定

例：



2. 交换排序

① 冒泡排序

相邻元素依次进行比较，小的前移，大的后移，第一趟结束后，最大元素放在了最后；再进行第二趟，使次大元素放在最大元素之前；继续，直到在一趟排序中没有交换发生，排序结束。

时间复杂度：$O(n^2)$ 稳定性：算法稳定

② 快速排序

每趟让待排表中的第一元素作支点，排序后入终位 k，将原表一分为二，使得 r[1]~r[k-1]中元素小于等于 r[k]，而 r[k+1]~r[n]中元素都大于等于 r[k]，递归进行，直到表长为 1 时，排序结束。

快速排序的时间复杂度为 $O(n\log_2 n)$

稳定性：不稳定

支点放入 x

附设两个指针 i 和 j：初始 i 指向第一元，j 指向第 n 元

从右向左，放过≥x.key 元（比 x 大的元素），j 变化，当 r[j].key<x.key，放入 i 位置

从左向右，放过≤x.key 元（比 x 小的元素），i 变化，当 r[i].key>x.key，放入 j 位置

直到 i=j，一趟结束，i 为 x 终位

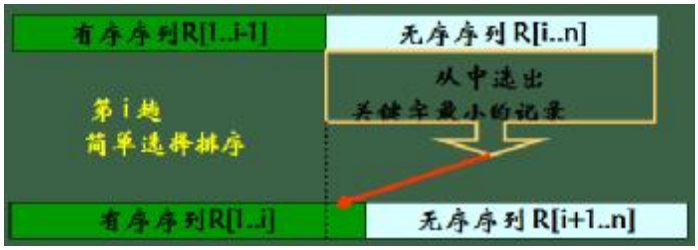
例：

[49	38	65	97	76	13	27	49′]
i↑						j↑	
[27	38	65	97	76	13		49′]
	i↑					j↑	
[27	38	65	97	76	13		49′]
		i↑				j↑	
[27	38		97	76	13	65	49′]
		i↑			j↑		
[27	38	13	97	76		65	49′]
			i↑		j↑		
[27	38	13		76	97	65	49′]
			i↑	j↑			
[27	38	13	49	76	97	65	49′]
			i↑j↑				

3. 选择排序

① 简单选择排序

假设排序过程中，待排记录序列的状态为：




```
void Select_Sort(RecType R[],int n)
{  int i,j,k;
   RecType temp;
   for(i=1;i<n;i++)
   { k=i; //假定起始位置为最小记录的位置
     for(j=i+1;j<=n;j++) //查找最小记录
       if(R[j].key<R[k].key)
         k=j;
       if(i!=k) //如果k不是假定位置，则交换
         {L.R[0]=L.R[k];   /*R[0]作为辅助存储空间*/
          L.R[k]=L.R[i];
          L.R[i]= L.R[0];
         }
     }
}
```

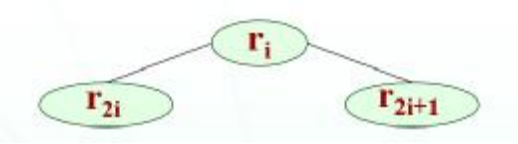
② 堆排序

堆的定义：堆是满足下列性质的数列{r1，r2，…，rn}：

$$\begin{cases} r_i \le r_{2i} \\ r_i \le r_{2i+1} \end{cases} \text{(小顶堆)} \quad \text{或} \quad \begin{cases} r_i \ge r_{2i} \\ r_i \ge r_{2i+1} \end{cases} \text{(大顶堆)}$$

通常将该记录序列看作一棵完全二叉树，

则 r_{2i} 是 r_i 的左孩子； r_{2i+1} 是 r_i 的右孩子。



小顶堆{12，36，27，65，40，34，98，81，73，55，49}

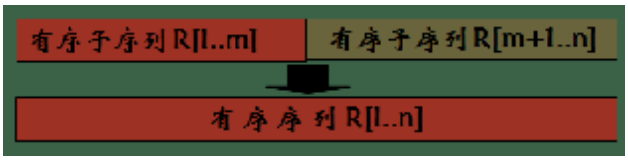
不是堆{12，36，27，65，40，14，98，81，73，55，49}

- 堆排序的过程：对一组待排序记录，首先把它们的关键字按堆定义排列成一个序列（称为初始建堆），堆顶元素为最小关键字的记录，将堆顶元素与序列中的最后一个元素交换位置；然后对剩余的记录再建堆，得到次最小关键字记录；如此反复进行，直到全部记录有序为止，这个过程称为堆排序。

4．归 并 排 序

归并排序的过程基于下列基本思想进行：

将两个或两个以上的有序子序列 “归并” 为一个有序序列。在内部排序中，通常采用的是 2-路归并排序。即：将两个位置相邻的记录有序子序列



例如：对下列关键字序列进行，2 路一归并排序。52， 23， 80， 36， 68， 14

[52， 23]，[80，36]，[68，14]

(23 ,52)，(36 ，80)，(14 ,68)

(23,36,52,80),(14,68)

(14,23,36,52，68,80)

习题解析

1． 从未排序序列中依次取出一个元素与已排序序列中的元素依次进行比较，然后将其放在已排序序列的合适位置，该排序方法称为（ ）排序法。

A.直接插入 B.简单选择 C.希尔 D.二路归并

2.设有一组关键字值（46,79,56,38,40,84），则用堆排序的方法建立的初始堆为

（　　）。

A.79,46,56,38,40,80

B.84,79,56,38,40,46

C.84,79,56,46,40,38

D.84,56,79,40,46,38

3.设有一组关键字值（46,79,56,38,40,84），则用快速排序的方法，以第一个记录为基准得到的一次划分结果为（　　）。

A.38,40,46,56,79,84

B.40,38,46,79,56,84

C.40,38,46,56,79,84

D.40,38,46,84,56,79

4.直接插入排序在最好情况下的时间复杂度为（　　）。

A.O(logn)　B.O(n)　C.O(n*logn)　D.O(n^2)

5.设一组初始记录关键字序列（5,2,6,3,8,），以第一个记录关键字 5 为基准进行一趟快速排序的结果为（　　）。

A.2,3,5,8,6　　B.3,2,5,8,6

C.3,2,5,6,8　　C.2,3,6,5,8

6.设有 5000 个待排序的记录关键字，如果需要用最快的方法选出其中最小的 10 个记录关键字，则用下列（　）方法可以达到目的。

A.快速排序　　B.堆排序

C.归并排序　　D.插入排序

华图网校 V.HUATU.COM
SINCE 2001
乐享品质　高效学习

■ 华图网校介绍

华图网校（V.HUATU.COM）于2007年3月由华图教育投资创立，是华图教育旗下的远程教育高端品牌。她专注于公职培训，　目前拥有遍及全国各地500万注册用户，已成为公职类考生学习提高的专业门户网站。

华图网校是教育部中国远程教育理事单位。她拥有全球最尖端高清录播互动技术和国际领先的网络课程设计思想，融汇华图教育十余年公职辅导模块教学法，凭借强大师资力量与教学资源、利用教育与互联网的完美结合，真正为考生带来“乐享品质”的学习体验，通过“高效学习”成就品质人生。

华图网校课程丰富多元，涵盖公务员、事业单位、招警、法院、检察院、军转干、选调生、村官、政法干警、三支一扶、乡镇公务员、党政公选等热门考试、晋升及选拔。同时，华图网校坚持以人为本的原则，不断吸引清华、北大等高端人才加入经营管理，优化课程学习平台，提升用户体验，探索网络教育新技术和教学思想，力争为考生提供高效、个性、互动、智能的高品质课程和服务。

华图网校将秉承“以教育推动社会进步”的使命，加快网站国际化进程，打造全球一流的网络学习平台。

我们的使命：以教育推动社会进步

我们的愿景：德聚最优秀人才，仁就基业长青的教育机构

我们的价值观：诚信为根、质量为本、知难而进、开拓创新。

■ 咨询电话：400-678-1009
■ 听课网址：v.huatu.com（华图网校）