福建省计算机专升本
算法设计部分参考代码一

有些题目只按程序填空题的题型来命题。

第二章 线性表

```
typedef struct alist *List;
typedef struct alist{
        int maxsize;            //表的最大长度
         int n;                  //表长
        ListItem *table;         //表元素数组
} AList;
List ListInit(int size)
     /*表结构初始化* /
  {
    List L=malloc(sizeof(*L));
    L->table=malloc(size*sizeof(ListItem));
    L->maxsize=size;
    L->n=0;
    return L;
  }
ListItem ListRetrieve(int k，List L);
    /*返回表 L 位置 k 处的元素* /
{
    if(k<1||k>L->n)
                    Error(“Out of bounds”);
     return L->table[k-1];
}
int ListLocate(ListItem x，List L);
    /*元素 x 在表 L 中的位置* /
{
```

```
    int i;
    for(i=0;i<L->n;i++)
        if(L->table[i]==x)return ++i;
    return 0;
}
```

■ 表项的插入

```
 void ListInsert(int k，ListItem x，List L)
/*在表 L 的位置 k 之后插入元素 x * /  {
      if (k<0||k >L->n) Error(“Out of bounds");
      if (L->n==maxsize) Error(“Out of bounds");
      //参数不合理或表已满，插入不成功
  for (int i =L->n-1; i >= k; i--)
            L->table[i+1]=L->table[i];
        L->table[k]=x;
        L->n++;
          //插入成功
  }
```

■ 表项的删除

```
ListItem ListDelete(int k，List L)
  /*从表 L 中删除位置 k 处的元素* /
{
   int i; ListItem   x;
   if(k<1|| k>L->n)Error(“Out of bounds");
   else if(L->n==0) Error(“List empty”);
   else{
      x=L->table[k-1];
   for(i=k；i<L->n；i++)
       L->table[i-1]=L->table[i];
   L->n--；}
```

```
        return x;
    }
typedef struct node * link;
typedef struct node  {          //链表结点
      ListItem element;                 //结点数据域
      link next;          //结点链域
} Node;
typedef struct    LinkList *List;     //链头指针
typedef struct    LinkList {link first;} Llist
       //其中 first 是链表头指针
List ListInit()
{
       List L=malloc(sizeof(*L));
       L->first=0;
       return L;
}
int ListLength (List L ) {
   int len=0;
   link p  ;
   p=L->first; //搜索指针  p  指向第  1  个结点
   while (p){
   len++; p=p->next; //p 搜索到最后一个结点并记录下其位置
   }
   return len;
}
ListItem ListRetrieve (int k, List L) {
//返回表中第  k  个元素
      int i;
    link p  ;
```

```
    if (k<1) Error("Out bounds");     // k 值不合理
    p=L->first; i=1 ;     //搜索指针 p 指向表头
    while(p&&i<k)
        {p=p->next; i++;}          //p 搜索到第 k 个结点
    return p->element;          //返回第 k 个元素
}
int ListLocate (ListItem x, List L) {
//返回表中值为 x 的元素的位置
    int i=1 ;
    link p  ;
    p=L->first; //搜索指针 p 指向表头
    while(p&&p->element!=x)
        {
      p=p->next; i++; //p 搜索到元素 x 并记录下其位置
    return p ? i : 0;
}
void ListInsert (int k, ListItem x, List L) {
//在链表中位置 k(>=0)后面插入新元素 x
      link p, newnode
      int i;
      if(k<0||(k!=0&&L->first==0))Error("Out of bounds");
      p=L->first;   //搜索指针 p 指向表首
      for(i=1;i<k&&p;i++)p=p->next;   //p 搜索到第 k 个节点
      newnode=NewNode();
      newnode->element=x;

if (k)
{
 newnode->next=p->next;//第 k+1 个节点挂到新节点之后
```

```
  p->next=newnode; //新节点挂到第 k 个节点之后
        }
  else              /*在表首插入*/
     {
        newnode->next=L->first; //新节点插到表首节点之前
        L->first=newnode;    //新节点插入后为表首节点
       }
}
```

其主要时间在于寻找正确的插入位置,故其计算时间为 O(k).

```
ListItem ListDelete (int k, List L) {
//在链表中删除第  k  个结点
       link p, q;
       ListItem x;
       int i;
       if(k<1||!L->first) Error("Out of bounds");
       p=L->first; //搜索指针 p 指向表首节点
       if(k==1)                    //删除表首元素
          {L->first=p->next; }
       else {                                  //删除表中或表尾元素
         q=L->first;   //搜索指针 q 指向表首
       for(i=1;i<k-1&&q; i++)q=q->next;
              //q 搜索到第  k-1 个结点
       if(q==0)Error("Out of bounds");
       p=q->next;              //p 指向第 k 个结点
       q->next=p->next;    //第 k+1 个结点挂到第 k-1 个结点之后
       }
       x=p->element; free( p);         //释放第 k 个结点
       return x;
}
```

其主要时间在于寻找待删除元素的位置,故其计算时间为 O(k).

```
void Push (StackItem x, Stack S) {
//若栈满返回 0, 否则新元素 x 进栈并返回 1
    if ( StackFull (S) ) Error("Stack is full");
                        //栈满溢出处理

    S->data[++S->top] = x;          //加入新元素
}
```

1．在一个递增有序的线性表中，有数值相同的元素存在。若存贮方式为单链表，设计一个算法，去掉数值相同的元素，使表中不再有重复的元素。

例如 （7，10，10，21，30，42，42，42，51，70）

将变作（7，10，21，30，42，51，70）

```
    void delete(link  h)
    {
        link q, p;
        p=h->next;
        while(p!=NULL)
        {  q=p;          p=p->next;
           while(p->data==q->data)
                {   q->next=p->next; free(p);   p=q->next;  }
        }
}
```

2. 已知线性表的元素是无序的，且以带头结点的单链表作为存储结构。设计一个

删除表中所有值小于 max 但大于 min 的元素的算法。

算法描述如下：

```
delete(LinkList *head, int max, int min)
{   LinkList *p,*q;
    q=head;
    p=head->next;
    while (p!=NULL)
        if((p->data<=min) || (p->data>=max))
        {   q=p;      p=p->next;          }
        else
        {   q->next=p->next;   ree(p);   p=q->next;   }
}
```

/////////线性表的元素是有序的呢？或者删除的是大于 max 和小于 min 的元素呢？

3. 已知表（K1，K2…，Kn），其中 Ki 为正整数，设计一个算法，能在 O（n）的时间内将线性表划分成两部分，其左半部分的每个关键字均小于 K1，右半部分的关键字均大于等于 K1。

```
void partition(RecType R[ ],int n) /*对 R[0]至 R[n-1]的元素进行划分*/
{
    int i=0,j=n-1;    RecType temp;
    temp=R[0];                    /*用区间的第 1 个记录作为基准*/
        while (i!=j)      /*从区间两端交替向中间扫描,直至 i=j 为止*/
        {
            while (j>i && R[j].key>=temp.key)     j--;/*从右向左扫描,找第 1 个关键字小于 temp.key 的 R[j] */
            if (i<j)              /*表示找到这样的 R[j],R[i]、R[j]交换*/
            {
                R[i]=R[j];   i++;
            }
            while (i<j && R[i].key<temp.key)     i++;    /*从左向右扫描,找第 1
```

个关键字大于等于 temp.key 的记录 R[i] */

```
            if (i<j)            /*表示找到这样的 R[i],R[i]、R[j]交换*/
            {
                R[j]=R[i];   j--;
            }
            R[i]=temp;
    }
}
```

//////////////////////////////////////快速排序的划分？//

4. 试设计将一个循环链表逆置的算法。顺序表如何逆转？

设单循环链表的头指针为 head，类型为 LinkList。逆置时需将每一个结点的指针域作以修改，使其原前趋结点成为后继。如要更改 q 结点的指针域时，设 s 指向其原前趋结点，p 指向其原后继结点，则只需进行 q->next=s;操作即可，算法描述如下：

```
        void  invert(LinkList    *head)
        {   //逆置 head 指针所指向的单循环链表
            linklist *p, *q, *s;
            q=head;
            p=head->next;
            while (p!=head)   //当表不为空时，逐个结点逆置
            {   s=q;
                q=p;
                p=p->next;
                q->next=s;
             }
             p->next=q;
   }
   Reverse(T data[],int length)
{ for(i=0;i<=length/2;i++)
 {   temp=data[i];
```

```
    data[i]=data[length-i-1];
   data[length-i-1]=temp;
  }
}
```

//双向循环链表呢？//////////////////////////////////////

5．单循环链表修改成双向循环链表。单向循环链表的结点有 3 个域：data，pre 和 next，pre 为指针域，它的值为空指针。试将此表改成双向循环链表。

第三章　栈和队列递归

历年题目：

*** 练习：**

3．1 假定用一个单循环链表来表示队列（也称为循环队列），该队列只设一个队尾指针，不设队首指针，试编写下列各种运算的算法：

（1）向循环链队列插入一个元素值为 x 的结点；

（2）从循环链队列中删除一个结点。

对一个循环链队列做插入和删除运算，假设不需要保留被删结点的值和不需要回收结点，算法描述如下：

（1）插入（即入队）算法：

```
insert(LinkList *rear, elemtype x)
{   //设循环链队列的队尾指针为 rear,x 为待插入的元素
    LinkList *p;
    p=(LinkList *)malloc(sizeof(LinkList));
    if(rear= =NULL)   //如为空队，建立循环链队列的第一个结点
    {   rear=p;
        rear->next=p;     //链接成循环链表
    }
    else   //否则在队尾插入 p 结点
    {   p->next=rear->next;
        rear->next=p;
        rear=p;
```

```
        }
    }
```

（2）删除（即出队）算法：

```
    delete(LinkList   *rear)
    {   //设循环链队列的队尾指针为 rear
        if (rear= =NULL)   //空队
            printf("underflow\n");
        if(rear->next= =rear)   //队中只有一个结点
            rear=NULL;
        else
            rear->next=rear->next->next;
                //rear->next 指向的结点为循环链队列的队头结点
}
```

3．2 顺序栈各种基本运算

```
 #define MaxSize 100
typedef char ElemType;
typedef struct
{
    ElemType elem[MaxSize];
    int top;      /*栈指针*/
} SqStack;
void InitStack(SqStack *&s)
{
    s=(SqStack *)malloc(sizeof(SqStack));
    s->top=-1;
}
int Push(SqStack *&s,ElemType e)
{
    if (s->top==MaxSize-1)        return 0;
    s->top++;
```

```
    s->elem[s->top]=e;
    return 1;
}
int Pop(SqStack *&s,ElemType &e)
{
    if (s->top==-1)        return 0;
    e=s->elem[s->top];
    s->top--;
    return 1;
}
```

链栈各种基本运算

```
typedef char ElemType;
typedef struct linknode
{
    ElemType data;              /*数据域*/
    struct linknode *next;      /*指针域*/
} LiStack;
void InitStack(LiStack *&s)
{
    s=(LiStack *)malloc(sizeof(LiStack));
    s->next=NULL;
}
void Push(LiStack *&s,ElemType e)
{
    LiStack *p;
    p=(LiStack *)malloc(sizeof(LiStack));
    p->data=e;
    p->next=s->next;        /*插入*p 结点作为第一个数据结点*/
    s->next=p;
}
```

```
int Pop(LiStack *&s,ElemType &e)
{
    LiStack *p;
    if (s->next==NULL)      /*栈空的情况*/
        return 0;
    p=s->next;              /*p 指向第一个数据结点*/
    e=p->data;
    s->next=p->next;
    free(p);
    return 1;
}
```

3．3 顺序队列各种基本运算

```
#define MaxSize 5
typedef char ElemType;
typedef struct
{
    ElemType elem[MaxSize];
    int front,rear;         /*队首和队尾指针*/
} SqQueue;
void InitQueue(SqQueue *&q)
{
    q=(SqQueue *)malloc (sizeof(SqQueue));
    q->front=q->rear=0;
}
int QueueLength(SqQueue *q)
{
    return (q->rear-q->front+MaxSize)%MaxSize;
}
int enQueue(SqQueue *&q,ElemType e)
{
```

```
    if ((q->rear+1)%MaxSize==q->front)     return 0;     /*队满*/
    q->rear=(q->rear+1)%MaxSize;
    q->elem[q->rear]=e;
    return 1;
}
int deQueue(SqQueue *&q,ElemType &e)
{
    if (q->front==q->rear)   /*队空*/          return 0;
    q->front=(q->front+1)%MaxSize;
    e=q->elem[q->front];
    return 1;
}
```

3．4 链队各种基本运算

```
typedef char ElemType;
typedef struct qnode
{
    ElemType data;
    struct qnode *next;
} QNode;
typedef struct
{
    QNode *front;
    QNode *rear;
} LiQueue;
void enQueue(LiQueue *&q,ElemType e)
{
    QNode *s;
     s=(QNode *)malloc(sizeof(QNode));
    s->data=e;
    s->next=NULL;
```

```
    if (q->rear==NULL)          /*若链队为空,则新结点是队首结点又是队尾结点
*/
        q->front=q->rear=s;
    else
    {
        q->rear->next=s;    /*将*s 结点链到队尾,rear 指向它*/
        q->rear=s;
    }
}
int deQueue(LiQueue *&q,ElemType &e)
{
    QNode *t;
    if (q->rear==NULL)      /*队列为空*/
        return 0;
    if (q->front==q->rear)    /*队列中只有一个结点时*/
    {
        t=q->front;
        q->front=q->rear=NULL;
    }
    else                    /*队列中有多个结点时*/
    {
        t=q->front;
        q->front=q->front->next;
    }
    e=t->data;
     free(t);
    return 1;
}
void InitQueue(LiQueue *&q)
{
```

```
    q=(LiQueue *)malloc(sizeof(LiQueue));
    q->front=q->rear=NULL;
}
void ClearQueue(LiQueue *&q)
{
    QNode *p=q->front,*r;
    if (p!=NULL)          /*释放数据结点占用空间*/
    {
        r=p->next;
        while (r!=NULL)
        {
            free(p);
            p=r;
           r=p->next;
        }
    }
    free(q);              /*释放头结点占用空间*/
}
int QueueLength(LiQueue *q)
{
    int n=0;
    QNode *p=q->front;
    while (p!=NULL)
    {
        n++;
        p=p->next;
    }
    return(n);
}
```

第四章　树

练习：

4.1. 试编写算法，对一棵二叉树根结点不变，将左、右子树进行交换，树中每个结点的左、右子树进行交换。

分析：

子树交换就是将原来的二叉树中每个结点的左、右子树分别交换生成一棵新的二叉树，该二叉树与原来二叉树成对称形状。先将二叉树根结点的左、右子树交换，然后在将左（右）子树的左、右子树交换直到子树为空树。

```
void Exchange(BiTree T)
{
   BiTNode p;
   if   (T)
   {
      p = T->lchild;
      T->lchild = T->rchild;
      T->rchild = p;
   Exchange(T->lchild);
   Exchange(T->rchild);
   }
}
```

4.2　typedef char ElemType;

```
typedef struct node
{
    ElemType data;              /*数据元素*/
    struct node *lchild;       /*指向左孩子*/
    struct node *rchild;       /*指向右孩子*/
} BTNode;
```

4.3 判断二叉树 b1 和 b2 是否相似

```
int  like (BTNode  b1,  BTNode  b2)

/*判断二叉树 b1 和 b2 是否相似，若相似，返回 1，否则返回 0*/
{   int like1, like2;
    if (b1==NULL && b2==NULL)       return(1);       /*b1 和 b2 均为空树，相似，返回 1*/
    else if (b1==NULL || b2==NULL) return(0); /*b1 和 b2 其中之一为空树，不相似，返回 0*/
      else
        {
          like1=like (b1->lchild, b2->lchild);       /*判断 b1 和 b2 的左子树是否相似*/
          like2=like (b1->rchild, b2->rchild);     /*判断 b1 和 b2 的右子树是否相似*/
          return (like1 && like2);
        }
  }
```

4.4 已知二叉树中的结点类型 BinTreeNode 定义为：struct BinTreeNode { ElemType data; BinTreeNode *left, *right;}; 其中 data 为结点值域,left 和 right 分别为指向左,右子女结点的指针域.下面函数的功能是返回二叉树 BT 中值为 X 的结点所在的层号,请在划有横线的地方填写合适内容.

```
int NodeLevel(BinTreeNode * BT, ElemType X)
{
if (   1   )   return –1; //空树的层号为-1
else if (   2   )   return 0; //根结点的层号为 0
//向子树中查找 X 结点
else {
int c1=NodeLevel(BT->left,X);
if(c1>=0) _____(3)_____;
int c2=_____(4)_____;
if (c2>=0)_____(5)_____;
//若树中不存在 X 结点则返回-1
else return -1;
}
```

}

(1) BT==NULL (2) BT->data==X (3) return c1+1

(4)NodeLevel(BT ->right , X) (5) return c2+1

4.5 前序、中序、后序遍历**递归算法:**

```
void InOrder(BTNode *b)              /*中序遍历的递归算法*/
{
    if (b!=NULL)
    {
        InOrder(b->lchild);      /*递归访问左子树*/
        printf("%c ",b->data);  /*访问根结点*/
        InOrder(b->rchild);      /*递归访问右子树*/
    }
}
```

4.6 返回 data 域为 x 的结点指针

```
BTNode *FindNode(BTNode *b,ElemType x)
/*返回 data 域为 x 的结点指针*/
{
    BTNode *p;
    if (b==NULL)
        return NULL;
    else if (b->data==x)
        return b;
    else
    {
        p=FindNode(b->lchild,x);
        if (p!=NULL)
            return p;
        else
```

```
            return FindNode(b->rchild,x);
        }
}
```

若先序序列与后序序列相同，则或为空树，或为只有根结点的二叉树。

若中序序列与后序序列相同，则或为空树，或为任一结点至多只有左子树的二叉树。

若先序序列与中序序列相同，则或为空树，或为任一结点至多只有右子树的二叉树。

历年题目：

1．二叉树以二叉表为存储结构，结点结构的定义如下，请写出一个求二叉树中叶子结点个数的算法。

```
typedef   struct   btnode  *btlink;
struct btnode{
  TreeItem element;
  btlink lchild;
  btlink rchild;
}Btnode;
int LeafNodes(Btnode *b)    /*求二叉树 b 的叶子结点个数*/
{
      int num1,num2;
      if (b==NULL)
           return 0;
      else if (b->lchild==NULL && b->rchild==NULL)
           return 1;
      else
      {
            num1=LeafNodes(b->lchild);
            num2=LeafNodes(b->rchild);
            return (num1+num2);
      }
```

}

注意：如何将这一个题目变形？

2．二叉树以二叉链表为存储结构，类型声明如下，请写出一个求二叉树中结点个数的算法。

```
int f(Btnode *b) /*求二叉树 b 的结点个数*/
{
    int num1,num2;
    if (b==NULL)
        return 0;
    else if (b->lchild==NULL && b->rchild==NULL)
        return 1;
    else
    {
        num1=f(b->lchild);
        num2=f(b->rchild);
        return (num1+num2+1);
    }
}
int Depth(BTNode *b)   /*求二叉树 b 的深度*/
{
    int ldep,rdep;
    if (b==NULL)
        return(0);                          /*空树的高度为 0*/
    else
    {
        ldep=Depth(b->lchild);  /*求左子树的高度为 ldep*/
        rdep=Depth(b->rchild); /*求右子树的高度为 rdep*/
        return (ldep>rdep)? (ldep+1):(rdep+1);
    }
```

}

第五章　图

历年题目：无

练习：

第六章　查　找

练习：

6．1 编写算法，返回二叉排序树中的关键字最大的结点地址。

提示：关键字最大的结点是“最右下”的结点。

```
pointer search(bitree t)
        {
         pointer p;
           if(t==NULL)
              return NULL;  //空树
              p=t;
           while(p->rchild!==NULL)
              p=p->rchild; //向右下搜索
        return p;
        }
```

注意：返回二叉排序树中的关键字最小的结点地址。关键字最小的结点是“最左下”的结点。

6．2　/*二分查找算法*/

```
typedef int KeyType;
typedef char InfoType[10];
typedef struct
{
    KeyType  key;                  /*KeyType 为关键字的数据类型*/
    InfoType  data;                /*其他数据*/
} NodeType;
typedef NodeType SeqList[MAXL];       /*顺序表类型*/
```

```
int BinSearch(SeqList R,int n,KeyType k)   /*二分查找算法*/
{
    int low=0,high=n-1,mid,count=0;
    while (low<=high)
    {
        mid=(low+high)/2;
        if (R[mid].key==k)        /*查找成功返回*/
            return mid;
        if (R[mid].key>k)         /*继续在 R[low..mid-1]中查找*/
            high=mid-1;
        else
            low=mid+1;            /*继续在 R[mid+1..high]中查找*/
    }
    return -1;
}
6．3   /*顺序查找算法*/
#define MAXL 100                  /*定义表中最多记录个数*/
typedef int KeyType;
typedef char InfoType[10];
typedef struct
{
    KeyType  key;                 /*KeyType 为关键字的数据类型*/
     InfoType  data;              /*其他数据*/
} NodeType;
typedef NodeType SeqList[MAXL];           /*顺序表类型*/

int SeqSearch(SeqList R,int n,KeyType k) /*顺序查找算法*/

{
```

```
    int i=0;
    while (i<n && R[i].key!=k)
    {
        printf("%d ",R[i].key);
        i++;                          /*从表头往后找*/
    }
    if (i>=n)     return -1;
    else
    {
        printf("%d",R[i].key);
        return i;
    }
}
```

历年题目：

1. 二叉排序（搜索）树 t 以二叉表为存储结构，

请编写算法实现在该树上查找值为 x 的节点。

```
typedef   int   TreeItem;
typedef   struct   btnode  *btlink;
typedef   struct   btnode{
  TreeItem data;
  Btlink   lchild, rchild; /*左右孩子指针*/
}BiTNode
```

算法函数原型 BiTNode Locate(BiTNode *t, TreeItem x)

解答：

```
BiTNode   Locate(BiTNode   *t, TreeItem  x){
if(t = =NULL)   return NULL:
if( x = = t-> data ) return *t;     /*  注意*，因为返回类型是 BiTNode*/
else if( x <   t-> data ) return Locate(t->lchild,  x)   /*左子树*/
else   return  Locate(t->rchild, x); /*右子树*/
}
```

2. 二叉搜索树 T 用二叉链表存储结构表示，

其中各元素的值均不相同。编写算法，

按递减顺序打印 T 中各元素的值。结点结构定义如下：

```
typedef   int   TreeItem;
typedef   struct   btnode   *btlink;
typedef   struct   btnode{
 TreeItem   data;
 btlink   left, right;
}BTNODE;
```

解答:

```
void f(btlink t) {   // 或 void f(BTNODE *t)
 if(t){
  f(t->right);
  printf("%d ",t->data);
  f(t->left);
 }
}
```

第七章　排　序

练习：

7．1．利用一维数组 a 可以对 n 个整数进行排序，其中一种排序算法的处理思想是：将 n 个整数分别作为数组 a 的 n 个元素的值，每次（即第 i 次）从元素 a[i]到 a[n]中挑出最小的一个元素 a[k](i≤k≤n)，然后将 a[k]与 a[i]换位。这样反复 n-1 次完成排序。编写实现上述算法的函数: void SelectSort(SqList &L)。

```
void SelectSort(SqList &L)
{//对顺序表 L 做简单选择排序。
 int i,j,t;
 for(i=1;i<L.count;++i) //选择第 i 小的记录,并交换到位
  {
```

```
    j=SelectMinKey(L,i); //在 L.elem[i..L.count]中选择关键字最小的记录
    if(i!=j)                    //与第 i 个记录交换
       {
        t=L.elem[i];
        L.elem[i]=L.elem[j];
        L.elem[j]=t;
       }
   }
}
int SelectMinKey(SqList L,int low)
{
//在 L.elem[low..L.count]中寻找关键字最小的记录
 int i,j,t;
 t=L.elem[low];
 j=low;
 for(i=low+1;i<=L.count;i++)
  if(L.elem[i]<t)
       {
        t=L.elem[i];
        j=i;
       }
 return j;
}
7．2      /*冒泡排序算法*/
 typedef int KeyType;
typedef char InfoType[10];
typedef  struct              /*记录类型*/
{
     KeyType  key;              /*关键字项*/
```

```
    InfoType data;          /*其他数据项,类型为 InfoType*/
} RecType;
void BubbleSort(RecType R[    ],int n)    /*冒泡排序算法*/
{
    int i,j,k;
    RecType temp;
    for (i=0;i<n-1;i++)
    {
        for (j=n-1;j>i;j--)    /*比较,找出本趟最小关键字的记录*/
            if (R[j].key<R[j-1].key)
            {
                temp=R[j];   /*R[j]与 R[j-1]进行交换,将最小关键字记录前移*/
                R[j]=R[j-1];
                R[j-1]=temp;
            }
/*输出每一趟的排序结果，以下代码考试时可以不写*/
        printf("          i=%d    ",i);   /*输出每一趟的排序结果*/
        for (k=0;k<n;k++)
            printf("%2d",R[k].key);
        printf("\n");
    }
```

7．3 快速排序的划分算法

```
 typedef  struct                //定义排序表的结构
{   int    elem[MAXSIZE];  /数据元素关键字
    int  count;                //表中当前元素的个数
}SqList;
int Partition(SqList &L,int low,int high)
{//交换顺序表 L 中子表 r[low..high]的记录，参考记录到位，并返回其所在位置，
```

此时在它之前(后)的记录均不大(小)于它

```
  int  pivotkey;
  pivotkey=L.elem[low]; //用子表的第一个记录作参考记录
  while(low<high)          //从表的两端交替地向中间扫描
     {
       while(low<high&&L.elem[high]>=pivotkey)      --high;
            L.elem[low]=L.elem[high];//将比参考记录小的记录移到低端
       while(low<high&&L.elem[low]<=pivotkey)         ++low;
            L.elem[high]=L.elem[low]; //将比参考记录大的记录移到高端
     }
  L.elem[low]=pivotkey;   //参考记录到位
  return  bw;                 //返回参考记录
}
```

7．4 堆排序

```
  typedef int KeyType;
typedef char InfoType[10];
typedef  struct                /*记录类型*/
{
      KeyType  key;             /*关键字项*/
       InfoType data;           /*其他数据项,类型为 InfoType*/
} RecType;

void Sift(RecType R[],int low,int high)    /*调整堆*/
{
      int  i=low,j=2*i;                          /*R[j]是 R[i]的左孩子*/
      RecType temp=R[i];
      while (j<=high)
      {
```

```
        if (j<high && R[j].key<R[j+1].key)  /*若右孩子较大,把 j 指向右孩子*/
            j++;                            /*变为 2i+1*/
         if (temp.key<R[j].key)
        {
            R[i]=R[j];                      /*将 R[j]调整到双亲结点位置上*/
            i=j;                            /*修改 i 和 j 值,以便继续向下筛选*/
            j=2*i;
        }
         else  break;                       /*筛选结束*/
    }
     R[i]=temp;              /*被筛选结点的值放入最终位置*/
}
void HeapSort(RecType R[],int n)    /*对 R[1]到 R[n]元素实现堆排序*/
{
    int i;
    RecType temp;
    for (i=n/2;i>=1;i--)        /*循环建立初始堆*/
        Sift(R,i,n);
    printf("    初始堆:");
   DispHeap(R,1,n);
    for (i=n;i>=2;i--)      /*进行 n-1 次循环,完成推排序*/
    {
        printf("    交换%d 与%d,输出%d\n",R[i].key,R[1].key,R[1].key);
        temp=R[1];                  /*将第一个元素同当前区间内 R[1]对换*/
         R[1]=R[i];
         R[i]=temp;
         Sift(R,1,i-1);         /*筛选 R[1]结点,得到 i-1 个结点的堆*/
        printf("    筛选调整得到堆:");
        DispHeap(R,1,i-1);
```

```
            printf("\n");     /*输出每一趟的排序结果*/
        }
    }
}
```

7．5 二路归并排序

```
 #define MAXE 20              /*线性表中最多元素个数*/
typedef int KeyType;
typedef char InfoType[10];
typedef  struct               /*记录类型*/
{
    KeyType  key;             /*关键字项*/
     InfoType data;           /*其他数据项,类型为 InfoType*/
} RecType;
void Merge(RecType R[],int low,int mid,int high)
/*将两个有序表 R[low..mid]和 R[mid+1..high]归并为一个有序表 R[low..high]中*/
{
    RecType *R1;
    int i=low,j=mid+1,k=0; /*k 是 R1 的下标,i、j 分别为第 1、2 段的下标*/
    R1=(RecType *)malloc((high-low+1)*sizeof(RecType));  /*动态分配空间*/
    while (i<=mid  &&  j<=high)          /*在第 1 段和第 2 段均未扫描完时循环*/
        if (R[i].key<=R[j].key)        /*将第 1 段中的记录放入 R1 中*/
        {
            R1[k]=R[i];
            i++;   k++;
        }
         else                          /*将第 2 段中的记录放入 R1 中*/
        {
            R1[k]=R[j];
            j++;    k++;
        }
```

```
    while (i<=mid)                    /*将第 1 段余下部分复制到 R1*/
    {
        R1[k]=R[i];
        i++;   k++;
    }
    while (j<=high)                   /*将第 2 段余下部分复制到 R1*/
    {
        R1[k]=R[j];
        j++;   k++;
    }
    for (k=0,i=low;i<=high;k++,i++) /*将 R1 复制回 R 中*/
        R[i]=R1[k];
}
```

历年题目：

2009：r[]为一维数组，其中 r[0]到 r[n-1]为待排序的 n 个元素，排序好的元素仍放在 r[0]到 r[n-1]中。请写出对该数组进行非递减排序的直接插入排序算法， 取名为 InsertSort(elemtype r[ ]，int n)。

```
void InsertSort(RecType R[],int n) /*对 R[0..n-1]按递增有序进行直接插入排序*/
{
    int i,j,k;
    RecType temp;
    for (i=1;i<n;i++)
    {
        temp=R[i];
        j=i-1;      /*从右向左 1 在有序区 R[0..i-1]中找 R[i]的插入位置*/
        while (j>=0 && temp.key<R[j].key)
        {
            R[j+1]=R[j];     /*将关键字大于 R[i].key 的记录后移*/
            j--;
        }
```

```
            R[j+1]=temp;                          /*在 j+1 处插入 R[i]*/
               /*输出每一趟的排序结果，以下代码考试时可以不写*/
            printf("              i=%d    ",i);
           for (k=0;k<n;k++)
               printf("%3d",R[k].key);
           printf("\n");
      }
}

            ////////////////////////////算法设计部分代码二////////////////////////////////
/*  顺序存储类型  */
typedef struct
{
     ElemType data[MAXSIZE]; /*存放线性表的数组*/
  int  length;                    /*  length 是顺序表的长度*/
}SeqList;

/*  从顺序表中查找与给定元素值相同的元素在顺序表中的位置  */
int ListLocate(SeqList L, ElemType x)
{
   int i=0;
  while(i<L.length && L.data[i]!=x)
                  i++;
  if (i<L.length)    return (i);
  else    return 0;
}
/*  向顺序表中插入元素  */
SeqList ListInsert(SeqList L,int i,ElemType x)
{   int j;
  if(L.length==MAXSIZE)
```

```
        printf("表满,不能插入\n");
  else    if(i<0||i>L.length)
        printf("插入位置不正确\n");
            else {
       for(j=L.length-1;j>=i;j--)
            L.data[j+1]=L.data[j];
               L.data[i]=x;
               L.length++;
               }
  return L;
  }
/*  从顺序表中删除元素  */
SeqList ListDelete(SeqList L,int i)
{
int j;ElemType x;
  if (i<0||i>L.length-1)
      printf("删除位置不正确\n");
  else {
            x=L.data[i];
            for(j=i;j<=L.length-1;j++)
            L.data[j]=L.data[j+1];
            L.length--;
            printf("%d 已被删除\n",x);
            }
  return L;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct node
{
```

```
    char  data;       //数据域
  struct node *next;//指向下一个节点的指针
}LNode,*LinkList;


int LocateElem(SqList *L, ElemType e)
{
    int i=0;
    while (i<L->length && L->elem[i]!=e)    i++;
    if (i>=L->length)
        return 0;
    else
        return i+1;
}

int ListInsert(SqList *&L,int i,ElemType e)
{
    int j;
    if (i<1 || i>L->length+1)
        return 0;
    i--;                          /*将顺序表位序转化为 elem 下标*/
    for (j=L->length;j>i;j--)       /*将 elem[i]及后面元素后移一个位置*/
        L->elem[j]=L->elem[j-1];
    L->elem[i]=e;
    L->length++;                        /*顺序表长度增 1*/
    return 1;
}
int ListDelete(SqList *&L,int i,ElemType &e)
{
    int j;
    if (i<1 || i>L->length)
```

```
        return 0;
    i--;                                /*将顺序表位序转化为 elem 下标*/
    e=L->elem[i];
    for (j=i;j<L->length-1;j++)
        L->elem[j]=L->elem[j+1];
    L->length--;
    return 1;
}

void InterSect(LinkList *ha,LinkList *hb,LinkList *&hc)/*求两有序集合的交*/
{
    LinkList *pa=ha->next,*pb,*s,*tc;
    hc=(LinkList *)malloc(sizeof(LinkList));
    tc=hc;
    while (pa!=NULL)
    {
        pb=hb->next;
        while (pb!=NULL && pb->data<pa->data)       pb=pb->next;
        if (pb!=NULL && pb->data==pa->data) /*若 pa 结点值在 B 中*/
        {
            s=(LinkList *)malloc(sizeof(LinkList));   /*复制结点*/
            s->data=pa->data;
            tc->next=s;
            tc=s;
        }
        pa=pa->next;        }
    tc->next=NULL;
}
```