

计算机专业基础综合考试

算法附加题

1. 已知线性表 $(a_1, a_2, a_3, \dots, a_n)$ 存放在一维数组 A 中。试设计一个在时间和空间两方面都尽可能高效的算法, 将所有奇数号元素移到所有偶数号元素前, 并且不得改变奇数号 (或偶数号) 元素之间的相对顺序, 要求:

- (1) 给出算法的基本设计思想。
- (2) 根据设计思想, 采用 C 或 C++ 或 Java 语言描述算法, 关键之处给出注释。
- (3) 说明你所设计算法的时间复杂度和空间复杂度。

2. 已知长度为 n ($n > 1$) 的单链表, 表头指针为 L , 结点结构由 `data` 和 `next` 两个域构成, 其中 `data` 域为字符型。试设计一个在时间和空间两方面都尽可能高效的算法, 判断该单链表是否中心对称 (例如 `xyx`、`xyyyx` 都是中心对称的), 要求:

- (1) 给出算法的基本设计思想。
- (2) 根据设计思想, 采用 C 或 C++ 或 Java 语言描述算法, 关键之处给出注释。
- (3) 说明你所设计算法的时间复杂度和空间复杂度。

3. (12 分) 设 $m+n$ 个元素顺序存放在数组 $A[1..m+n]$ 中, 前 m 个元素递增有序, 后 n 个元素递增有序, 试设计一个在时间和空间两方面都尽可能高效的算法, 使得整个顺序表递增有序, 要求:

- (1) 给出算法的基本设计思想。
- (2) 根据设计思想, 采用 C 或 C++ 或 Java 语言描述算法, 关键之处给出注释。
- (3) 说明你所设计算法的时间复杂度和空间复杂度。

1. 解析:

(1) 算法的基本设计思想:

①在数组尾部从后往前, 找到第一个奇数号元素, 将此元素与其前面的偶数号元素交换。这样, 就形成了两个前后相连且相对顺序不变的奇数号元素“块”。

②暂存①中“块”前面的偶数号元素, 将“块”内奇数号结点依次前移, 然后将暂存的偶数号结点复制到空出来的数组单元中。就形成了三个连续的奇数号元素“块”。

③暂存②中“块”前面的偶数号元素, 将“块”内奇数号结点依次前移, 然后将暂存的偶数号结点复制到空出来的数组单元中。就形成了四个连续的奇数号元素

“块”。

④如此继续, 直到前一步的“块”前没有元素为止。

(2) 算法的设计如下:

```
void Bubble_Swap(ElemType A[], int n){
    int i=n, v=1;           //i 为工作指针, 初始假设 n 为奇数, v 为
    “块”的大小
    ElemType temp;          //辅助变量
    if(n%2==0) i=n-1;       //若 n 为偶数, 则令 i 为 n-1
    while(i>1){              //假设数组从 1 开始存放。当 i=1 时, 气
    泡浮出水面
        temp=A[i-1];        //将“块”前的偶数号元素暂存
        for(int j=0; j<v; j++) //将大小为 v 的“块”整体向前平移
            A[i-1+j]=A[i+j]   //从前往后依次向前平移
        A[i+v-1]=temp;       //暂存的奇数号元素复制到平移后空出的
    位置
        i=i-2; v++;          //指针向前, 块大小增 1
    }
}
```

(3) 一共进行了 $n/2$ 次交换, 每次交换的元素个数从 $1 \sim n/2$, 因此时间复杂度为 $O(n^2)$ 。虽然时间复杂度为 $O(n^2)$, 但因 n^2 前的系数很小, 实际达到的效率是很高的。算法的空间复杂度为 $O(1)$ 。

2. 解析:

思路 1 (借助栈, 空间复杂度高): 将表的前半部分依次进栈, 依次访问后半部分时, 从栈中弹出一个元素, 进行比较。思路 2 (类似折纸的思想, 算法复杂): 找到中间位置的元素, 将后半部分的链表就地逆置, 然后前半部分从前往后、后半部分从后往前比较, 比较结束后再恢复 (题中没有说不能改变链, 故可不恢复)。

为了让算法更简单, 这里采用思路 1, 思路 2 中的方法留给有兴趣的读者。

(1) 算法的基本设计思想:

①借助辅助栈, 将链表的前一半元素依次进栈。注意 n 为奇数时要特殊处理。

②在处理链表的后一半元素时, 当访问到链表的一个元素后, 就从栈中弹出一个元素, 两元素比较, 若相等, 则将链表中下一元素与栈中再弹出元素比较, 直至链表到尾。

③若栈是空栈, 则得出链表中心对称的结论; 否则, 当链表中一元素与栈中弹出元素不等时, 结论为链表非中心对称。

(2) 算法的实现如下:

```
typedef struct LNode{           //链表结点的结构
    char data;                  //结点数据
    struct LNode *next;         //结点链接指针
} *LinkList;
```

```

int Str_Sym(LinkList L,int n){
//本算法判断带头结点的单链表是否是中心对称
    Stack s;initstack(s);                //初始化栈
    LNode *q,*p=L->next;                //q 指向出栈元素,
p 工作指针
    for(int i=1;i<=n/2;i++){              //前半段结点入栈
        push(p);
        p=p->next;
    }
特殊处理
    if(n%2==1) p=p->next;                //若 n 为奇数,需要
    while(p!=null){                      //后半表依次和
前一半表比较
        q=pop(s);                        //出栈一个结点
        if(q->data==p->data) p=p->next;    //相等则继续比较
下一个结点
        else break;                      //不等则跳出循环
    }
    if(empty(s)) return 1;                //栈空,则说明对称
    else return 0;                        //否则不对称
}

```

(3) 算法的时间复杂度为 $O(n)$, 空间复杂度为 $O(n)$ 。

思考: 若当长度未知时, 该如何操作比较方便?

这里给出两种参考方法:

①先用遍历一遍链表数出元素个数再按参考答案操作。

②同时设立一个栈和一个队列, 直接遍历一边链表把每个元素的值都入栈、入队列, 然后再一一出栈、出队列比较元素的值是否相同。

3. 解析:

(1) 算法基本设计思想:

①把数组的前 m 个元素看成一个归并段, 后 n 个元素看成一个归并段, 增加一个临时数组 $B[1..m+n]$ 存储临时归并结果。分别设置两个指针 $k1, k2$, 指向两个归并段首元素, 再设置一个指针 $k3$ 指向临时数组下一个结果位置。

②如果 $1 \leq k1 \leq m$ 而且 $m+1 \leq k2 \leq m+n$ 执行③; 否则执行④。

③比较两个归并段指针所指元素的大小。如果 $A[k1] \leq A[k2]$, 那么 $B[k3++] = A[k1++]$; 否则 $B[k3++] = A[k2++]$ 。执行②。

④如果 $k1 > m$, 则第二个归并段的元素还未比较完, 把第二个归并段的剩余元素复制到数组 B 。如果 $k2 > m+n$, 则第一个归并段的元素还未比较完, 把第一个归并段的剩余元素复制到数组 B 。最后把数组 B 复制到数组 A 。

(2) 算法的实现如下:

```

void Merge(int[] A) {                    //实现数组 1-m 和 m+1-m+n 两个归

```

并段归并。

```

    int B[m+n+1];                        //临时辅助数组 B[1...m+n]
    int k1,k2,k3;
    k1=1;k2=m+1;k3=1;                    //3 个指针
    while(k1<=m&& k2<=m+n){              //如果两个归并段都没有比较完
        if(A[k1]<=A[k2]) B[k3++]=A[k1++]; //第一个归并段指针指向
元素较小
        else B[k3++]=A[k2++];              //第二个归并段指针指
向元素较小
    }
    if(k1>m)                              //把没有比较完的归并段中的剩余元
素复制到数组 B
        while(k2<=m+n) B[k3++]=A[k2++];
    else
        while(k1<=m) B[k3++]=A[k1++];
    for(int i=1;i<=m+n;i++)              //把临时数组 B 复制到 A
        A[i]=B[i];
}

```

(3) 总共遍历了 A 数组两遍, 第一遍合并, 第二遍复制结果, 时间复杂度为 $O(m+n)$ 。临时数组 B 空间大小 $m+n$, 所以空间复杂度为 $O(m+n)$ 。

【解析 2】(1) 算法的基本设计思想:

将数组 A 看成是两个长度分别为 m 和 n 的有序表 $L1$ 和 $L2$, 只需要将 $L2$ 中的每个元素依次向前插入到前面有序数组部分中的合适位置即可。插入过程如下:

①取表 $L2$ 中的第一个元素 $A[m+1]$, 暂存在 $temp$ 中, 让 $temp$ 前插到合适的位置。

②重复过程①, 继续插入 $A[m+2], A[m+3], \dots, A[m+n]$, 直到数组 A 整体有序。

(2) 算法的实现如下:

```

void InsertSort(int A[],int m,int n){
    int temp;                            //辅助变量, 暂存待插入元素
    for(int i=m+1;i<=m+n;i++){            //将 L2 中的元素依次插入到前面有
序部分
        temp=A[i];
        for(int j=i-1;j>=1&&temp<A[j];j--) //向前查找待插入位置
            A[j+1]=A[j];                  //向后挪位
        A[j+1]=temp;                      //复制到插入位置
    }
}

```

(3) 本算法的时间复杂度由 m 和 n 共同决定, 最内层循环的 $A[j+1]=A[j]$ 为基本语句。在最坏情况下, 即 $L2$ 中的所有元素均小于 $L1$ 中的最小元素, 则对于 $L2$ 中的每个元素, 为了找到其插入位置都需要做 m 次移动, 故时间复杂度为 $O(mn)$ 。空间复杂度为 $O(1)$ 。

博瑞祥隆图书专营店