

# 数据结构 全部作业

计算3班 10200102 石若珏

2023 年 6 月 10 日

## 目录

<b>1</b>	<b>第二章</b>	<b>1</b>
1.1	习题2.6 . . . . .	1
1.2	习题2.7 . . . . .	3
<b>2</b>	<b>第三章</b>	<b>4</b>
2.1	习题3.8 . . . . .	4
<b>3</b>	<b>第四章</b>	<b>5</b>
3.1	习题4.7 . . . . .	5
3.2	习题4.9 . . . . .	5
<b>4</b>	<b>第五章</b>	<b>7</b>
4.1	习题5.11 . . . . .	7
<b>5</b>	<b>第六章</b>	<b>10</b>
5.1	习题6.7 . . . . .	10
5.2	习题6.9 . . . . .	11
5.3	习题6.11 . . . . .	11

## 1 第二章

### 1.1 习题2.6

试写一算法，实现顺序表的就地逆置，即利用原表的存储空间将线性表  $(a_1, a_2, \dots, a_n)$  逆置为  $(a_n, \dots, a_1)$

**invert(SqList &L)**

```
10200102 石若珏 2.6
初始的线性表La的元素依次为:
a b c d e
逆置的线性表La的元素依次为:
e d c b a

-----
Process exited after 0.01221 seconds with return value 0
请按任意键继续. . .
```

图 1: 2.6

```
1  #include<iostream>
2  using namespace std;
3
4  const int LIST_INIT_SIZE=100;
5  const int LISTINCREMENT=10;
6  const bool TRUE=1;
7  const bool FALSE=0;
8
9  typedef struct {
10     char *elem;
11     int length;
12     int listsize;
13     int increamentsize;
14 }SqList;
15
16 void ClearList(SqList &L)
17 {
18     L.length=0;
19 }
20
21 void InitList_Sq(SqList &L,int maxsize=LIST_INIT_SIZE,int increasize=
LISTINCREMENT)
22 {
23     L.elem=new char[ maxsize ];
24     L.length=0;
25     L.listsize=maxsize;
26     L.increamentsize=increasize;
```

```

27     }
28
29     bool ListEmpty_Sq(SqList L)
30     {
31         if (L.length==0)
32             return TRUE;
33         else
34             return FALSE;
35     }
36
37     void invert(SqList &L)
38     {
39         int i,j;
40         char e;
41         for (i=0,j=L.length-1;i<j;i++,j--)
42         {
43             e=L.elem[i];
44             L.elem[i]=L.elem[j];
45             L.elem[j]=e;
46         }
47     }
48
49     void ListTraverse_Sq(SqList L)
50     {
51         for (int i=0;i<=L.length-1;i++)
52         {
53             cout<<L.elem[i]<<" ";
54         }
55         cout<<endl;
56     }
57
58     int main()
59     {
60         SqList La;
61         InitList_Sq(La,100,10);
62         La.elem[0]='a';
63         La.elem[1]='b';
64         La.elem[2]='c';
65         La.elem[3]='d';
66         La.elem[4]='e';
67         La.length=5;
68         cout<<"10200102 石若珏 } 2.6"<<endl;
69         cout<<"初始的线性表的元素依次为: La"<<endl;
70         ListTraverse_Sq(La);
71         invert(La);
72         cout<<"逆置的线性表的元素依次为: La"<<endl;
73         ListTraverse_Sq(La);

```

72  
73  
74

```
}
```

## 1.2 习题2.7

已知指针  $ha$  和  $hb$  分别指向两个单链表的头结点, 并且已知两个链表的长度分别为  $m$  和  $n$ 。试写一算法将两个链表连接在一起(即令其中一个表的首元结点连在另一个表的最后一个节点之后)。假设指针  $hc$  指向链接后的链表的头结点, 并要求算法以尽可能短的时间完成连接运算。试分析时间复杂度。

注: 因在实验报告中已经有具体程序了, 故在此只标注子函数和程序运行结果。

```
新创建的链表ha:
a b c x y z k l
新创建的链表hb:
q w e r
hc为:
a b c x y z k l q w e r

-----
Process exited after 0.01081 seconds with return value 0
请按任意键继续. . .
```

图 2: 2.7

```
1 void ListConcat(LinkList &ha, LinkList &hb, LinkList &hc)
2 {
3     Lnode *pa, *pb;
4     pa=ha->next; pb=hb->next;
5     int m=ListLength_L(ha);
6     int n=ListLength_L(hb);
7     if(m<=n) {
8         while(pa) pa=pa->next;
9         hc=ha; pa->next=pb;
10        delete hb;
11    }
12    else {
13        while(pb) pb=pb->next;
14        hc=hb; pb->next=pa;
15        delete ha;
16    } // if
17 } // ListConcat O(Min(m,n))
18
```

## 2 第三章

### 2.1 习题3.8

编写一个双向起泡的排序算法，即相邻两边分别向相反方向起泡。

```
初始排序为
8 1 9 7 2 4 5 6 10 3 25 21 45 66 73 15 28 90 44 100
双向起泡排序后
1 0 2 3 4 5 6 7 8 9 10 15 21 25 28 44 45 66 73 90

-----
Process exited after 0.01342 seconds with return value 0
请按任意键继续. . .
```

图 3: 3.8

```
1 void BubbleSort2(sqlist &L)
2 { int change=1,low,high,i;
3   low=1;
4   high=L.length;
5   while(low<high && change)
6   { change=0;
7     for(i=low;i<high;i++)
8       if(L.r[i].key>L.r[i+1].key)
9       { L.r[0]=L.r[i]; L.r[i]=L.r[i+1];
10        L.r[i+1]=L.r[0]; change=1;
11      }
12    high--;
13    for(i=high;i>low;i--)
14      if(L.r[i].key<L.r[i-1].key)
15      { L.r[0]=L.r[i]; L.r[i]=L.r[i-1];
16       L.r[i-1]=L.r[0]; change=1;
17      }
18    low++;
19  }
20 }
21
```

### 3 第四章

#### 3.1 习题4.7

仿照图4.5画出下列表达式转换成后缀式的过程。

$$(a + b) * (c / (d - e) + f) + ab * c$$

序号	表达式	运算符栈	后缀式
1	.	#(	
2	.	#(	a
3	.	#(+	a
4	.	#(+	ab
5	.	#	ab+
6	.	#	ab+
7	.	#	ab+
8	.	#x	ab+
9	.	#xc	ab+c
10	.	#xc/	ab+c
11	.	#xc/c	ab+c
12	.	#xc/c	ab+c d
13	.	#xc/c-	ab+c d
14	.	#xc/c-	ab+c d
15	.	#xc/c-	ab+c d
16	.	#xc/c-	ab+c d
17	.	#xc/c-	ab+c d
18	.	#xc/c-	ab+c d
19	.	#xc/c-	ab+c d
20	.	#xc/c-	ab+c d
21	.	#xc/c-	ab+c d
22	.	#xc/c-	ab+c d
23	.	#xc/c-	ab+c d
24	.	#xc/c-	ab+c d
25	.	#xc/c-	ab+c d
26	.	#xc/c-	ab+c d
27	.	#xc/c-	ab+c d
28	.	#xc/c-	ab+c d

图 4: 4.7

#### 3.2 习题4.9

假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点(注意不设头指针)，编写相应的队列初始化，入队列和出队列算法。

```

新创建的队列为:
a b c x y z k l o t
入队列一个元素为:
a b c x y z k l o t c
出队列一个元素为:
b c x y z k l o t c
-----
Process exited after 0.01173 seconds with return value
请按任意键继续. . .

```

图 5: 4.9

```

1  typedef struct QNode
2  {   Qchar      data;
3      struct QNode  *next;
4  } LNode, *QueuePtr;
5  typedef struct
6  {   QueuePtr  rear;
7  } CLinkQueue;
8  CLinkQueue  Q ;
9
10 void InitCQueue (CLinkQueue &Q) )
11 {   Q.rear = new LNode;
12     Q.rear->next=Q.rear;
13 }
14
15 void EnCQueue(CLinkQueue &Q, char e)
16 {   QueuePtr  p=new LNode;
17     p->data = e;
18     p->next = Q.rear->next;
19     Q.rear->next = p;
20     Q.rear=p;
21 }
22
23 bool DeCQueue (CLinkQueue &Q , char &e)
24 {
25     if (Q.rear->next == Q.rear)
26         return false;
27     p = Q.rear->next->next;
28     e = p->data;
29     Q.rear->next->next = p->next;
30     if (Q.rear==p)
31         Q.rear=Q.rear->next;
32     delete p;
33     return true;

```



34  
35

}

## 4 第五章

### 4.1 习题5.11

假设稀疏矩阵  $A$  和  $B$  均以三元组顺序表作为存储结构。试写出矩阵相加的算法，另设三元组表  $C$  存放结果矩阵。注：第一行输入为  $A$  和  $B$  非零元个数，之后几行以三元组的方式输入  $A$  和  $B$ 。

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 2 \end{pmatrix} C = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 4 \\ 0 & 0 & 3 \end{pmatrix}$$

```
3 3
1 1 1
2 2 1
3 3 1
1 2 1
2 3 4
3 3 2
C:
1 1 1
1 2 1
2 2 1
2 3 4
3 3 3
```

```
-----
Process exited after 25.39 seconds with return value 0
请按任意键继续. . .
```

图 6: 5.11

```
1  #include<iostream>
2  #include<stdio.h>
3  using namespace std;
4  #include<stdlib.h>
5  #define MAXSIZE 1000
6  typedef struct Triple
7  {
8      int e;
```

```

9      int row,col;
10     }Triple;
11
12     typedef struct TSMatrix
13     {
14         Triple data[MAXSIZE+1];
15         int m,n,len;
16     }TSMatrix;
17     TSMatrix t1;
18     TSMatrix t2;
19     void InputMatrix(TSMatrix *t1,TSMatrix *t2)
20     {
21
22         scanf("%d%d",&t1->len,&t2->len);
23         int i;
24         for(i=1;i<=t1->len;i++)
25         {
26             scanf("%d%d%d",&t1->data[i].row,&t1->data[i].col,&t1->data[i].e);
27         }
28
29         for(i=1;i<=t2->len;i++)
30         {
31             scanf("%d%d%d",&t2->data[i].row,&t2->data[i].col,&t2->data[i].e);
32         }
33     }
34
35     void Output(TSMatrix t)
36     {
37
38         int i;
39         for(i=1;i<=t.len;i++)
40         {
41             printf("%d %d %d\n",t.data[i].row,t.data[i].col,t.data[i].e);
42         }
43     }
44
45
46
47     void AddMastrix(TSMatrix a,TSMatrix b,TSMatrix *c)
48     {
49         int i=1,j=1,k=1;
50         c->m=a.m; c->n=a.n;
51         while (i<=a.len && j<= b.len)
52         {
53             if (a.data[i].row < b.data[j].row)

```

```

54     {
55         while (j <= b.len && i<= a.len && a.data[i].row < b.data[j].row)
56         {
57
58             c->data[k] = a.data[i];
59             i++;
60             k++;
61         }
62     }
63     else if (a.data[i].row > b.data[j].row)
64     {
65         while (j <= b.len && i <= a.len && a.data[i].row > b.data[j].row)
66         {
67
68             c->data[k] = b.data[j];
69             j++;
70             k++;
71         }
72     }
73     else if (a.data[i].row == b.data[j].row)
74     {
75         if (a.data[i].col < b.data[j].col)
76         {
77             c->data[k] = a.data[i];
78             i++;
79             k++;
80         }
81         else if (a.data[i].col > b.data[j].col)
82         {
83             c->data[k] = b.data[j];
84             j++;
85             k++;
86         }
87         else if (a.data[i].col == b.data[j].col)
88         {
89             if (a.data[i].e + b.data[j].e != 0)
90             {
91                 c->data[k].row=a.data[i].row;
92                 c->data[k].col=a.data[i].col;
93                 c->data[k].e=a.data[i].e+b.data[j].e;
94                 k++;
95             }
96             j++;
97             i++;
98         }

```

```

99     }
100   }
101   while (i<=a.len)
102   {
103     c->data[k]=a.data[i];
104     k++;
105     i++;
106   }
107   while (j<= b.len)
108   {
109     c->data[k]=b.data[j];
110     k++;
111     j++;
112   }
113   c->len=k-1;
114 }
115 int main()
116 {
117   TSMatrix a,b,c;
118   InputMatrix(&a,&b);
119   AddMastrix(a,b,&c);
120   cout<<"C:"<<endl;
121   Output(c);
122   return 0;
123 }
124

```

## 5 第六章

### 5.1 习题6.7

试找出所有满足下列条件的二叉树：

- (a) 它们在先序遍历和中序遍历时，得到的结点访问序列相同；
- (b) 它们在后序遍历和中序遍历时，得到的结点访问序列相同；
- (c) 它们在先序遍历和后序遍历时，得到的结点访问序列相同；

答：现已知先序遍历二叉树的顺序是“根—左子树—右子树”，中序遍历“左子树—根—右子树”，后序遍历顺序是：“左子树—右子树—根”。故：

- (a) 空树或任一结点至多只有右子树的二叉树；
- (b) 空树或任一结点至多只有左子树的二叉树；
- (c) 空树或只有根结点的二叉树；

## 5.2 习题6.9

编写递归算法，计算二叉树中叶子结点的数目。

```
abc##d##ef##g##
a b c d e f g
叶子结点个数为: 4

-----
Process exited after 30.12 seconds with return value 0
请按任意键继续. . .
```

图 7: 6.9

```
1  int LeafCount(BiTree T)
2  {   if(!T)   return 0;
3      else if(!T->lchild&&!T->rchild)
4      return 1;
5      else return Leaf_Count(T->lchild)+Leaf_Count(T->rchild);
6  }
7
```

## 5.3 习题6.11

编写递归算法：求二叉树中以元素值为  $x$  的结点为根的子树的深度。

```
1
2
3
0
0
2
3
1
1
0
0
0
0
0
0
0
请输入x的值: 3
以元素值为x的结点为根的子树的深度为: 3

-----
Process exited after 17.15 seconds with return value 0
请按任意键继续. . .
```

图 8: 6.11

```

1  #include <stdio.h>
2  #include <malloc.h>
3
4  typedef struct BTNode {
5      int data;
6      struct BTNode *left , *right;
7  }BTNode;
8
9  BTNode *creat_bt () { //按扩展前序建二叉树;
10     BTNode *t; int x;
11     scanf ("%d",&x);
12     if (x==0) t=NULL;
13     else
14     { t=(BTNode *) malloc (sizeof(BTNode));
15       t->data=x;
16       t->left=creat_bt ();
17       t->right=creat_bt ();
18     }
19     return t;
20 }
21
22
23 //先以先序遍历递归找到元素值为的结点，然后再递归求以为根结点的子树的深度xx
24
25
26 BTNode *preorder_x (BTNode *bt , int x)
27 {
28     if (bt != NULL){
29         if (bt->data==x)
30             return bt;
31         BTNode *t=NULL;
32         t=preorder_x (bt->left , x);
33         if(t) return t;
34         t=preorder_x (bt->right , x);
35         if(t) return t;
36     }
37     return NULL;
38 }
39
40 int depth (BTNode *bt)
41 {
42     int h1=0,h2=0;
43     if (bt == NULL) return 0;
44     else if (bt->left == NULL && bt->right == NULL) return 1;
45     else{

```

```

46     h1 = depth(bt->left);
47     h2 = depth(bt->right);
48     return (h1>h2?h1:h2)+1;
49 }
50 }
51
52
53 int main()
54 {
55
56     BTNode *bt = creat_bt();
57     int x;
58     printf("请输入的值: x");
59     scanf("%d", &x);
60
61     BTNode *p = preorder_x(bt, x);
62     int d=depth(p);
63     printf("以元素值为的结点为根的子树的深度为: x%d", d);
64
65 }
66
67

```