线性表编程作业

姓名：寇一笑　　学号：18020024016　　姓名：安皓源　　学号：18020022001

2020 年 3 月 24 日

Contents

1 第一题

1.1 实验目的和内容

1.1.1 题目描述

已知两个非降序的双向链表序列M1和M2[1]，请构造出它们合并后的非降序双向链表，并分析算法的时间复杂度。

1.2 输入和输出说明

1.2.1 输入

第一行是以空格为分隔的非降序数字序列，结尾是-1表示结束　　第二行是以空格为分隔的非降序数字序列，结尾是-1表示结束

1.2.2 输出

输出一行以空格为分隔的非降序数字序列

1.2.3 样例

- 输入

  2 3 4 -1

  2 3 4 5 -1

- 输出

  2 2 3 3 4 4 5

- 输入

  2 3 3 -1

  2 2 2 -1

- 输出

  2 2 2 2 3 3

- 输入

  -1

  -1

- 输出

  exit(1)

[1]我们定义双向链表输入结尾是-1，这在题目中没有定义

1.2.4 问题分析

根据题目，我们需要解决的问题有：

1. 如何建立双向链表
2. 如何输出双向链表
3. 如何将他们整合

1.3 解题思路

本道题主程序非常简单，主要是需要实现三个函数的功能。首先是创建一个双链表，这可以通过单链表加上一个prior指针来实现。首先建立一个头结点，判断输入的第一个输入是否为-1，如果是则返回exit(1)。然后每次输入一个数字判断是否为-1，如果不是则新建一个结点并给它赋值并且加上双向的链子。

这里最重要的函数是merge函数，为了节省空间，我们在其中一个链表La进行原地操作。先设置第三个链表Lc的头结点为La的，然后判断la和lb的大小，将小的插入到双向链表Lc中，如果相等，则先插入La的，再插入Lb的，实现非降序排列。

最后是输出函数，我们对加上了对于空链表输出NULL，当指针遍历不为*NULL*时，依次输出数据域的值。

1.4 实验代码及注释

1.4.1 双向链表版

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

typedef int ElemType;
typedef struct node
{                       //链表结点
    ElemType data;      //结点数据域
    struct node *next, *prior; //结点指针域
} Listnode, *Linklist; //节点类型，节点指针

Linklist create_list(); //此处用了双向链表
Linklist merge_list(Linklist, Linklist); //声明合并的函数
void output_list(Linklist); //空链表将输出NULL

void main()
{
    Linklist L1, L2, L;
    printf("please input sequence1, end with -1:\t"); //本来想换行结尾，但是这样输入就是了char
    L1 = create_list();
    printf("please input sequence2, end with -1:\t");
    L2 = create_list();
    L = merge_list(L1, L2);
    printf("the result is:");
    output_list(L);
}

Linklist create_list() //创建链表并赋值
```

```
{
    int x;
    Linklist head, pa, pb; //头结点和两个指针,用来开辟空间，用来连接pbpapb
    scanf("%d", &x);
    if(x==1) exit(1); //如果第一个输入为，异常退出-1
    head = (Linklist)malloc(sizeof(Listnode)); //定义头结点
    pa = head;
    while (x != -1)
    {
        pb = (Linklist)malloc(sizeof(Listnode));
        pb->data = x;
        pa->next = pb;
        pb->prior = pa;
        pa = pb;
        scanf("%d", &x);
    }
    pb->next = NULL; //指定最后一个为空，作为结束判断标志
    //创建双向链表
    return head; //返回头结点
}

Linklist merge_list(Linklist La, Linklist Lb)
{
    Linklist Lc, pa, pb, pc;
    Lc = La; //链表的头结点，直接在链表就地操作，降低空间复杂度ca
    pc = La;
    pa = La->next;
    pb = Lb->next;    //pa,是待插入结点的指针，是链表待插入结点的前一个指针pbpcc
    while (pa && pb) //while (pa!=NULL && pb!=NULL) La,Lb 链表不为空
    {
        if (pa->data < pb->data) //为了实现非降序排列需要进行分类讨论
        {
            pc->next = pa;
            pa->prior = pc;
            pc = pa;
            pa = pa->next;
        }
        if (pa->data > pb->data)
        {
            pc->next = pb;
            pb->prior = pc;
            pc = pb;
            pb = pb->next;
        }
        else //相等的话，就重复上述两个操作
        {
            pc->next = pa;
            pa->prior = pc;
            pc = pa;
            pa = pa->next;

            pc->next = pb;
            pb->prior = pc;
            pc = pb;
            pb = pb->next;
        }
    }
    if (pa != NULL)
```

```
87      {
88          pc->next = pa;
89      }
90      if (pb != NULL)
91      {
92          pc->next = pb;
93      }
94      // 或者为空的话，直接将后续的链表加上去papb
95      return Lc; //返回头结点
96  }
97
98  void output_list(Linklist s)
99  {
100     s = s->next;
101     if (s == NULL)
102     {
103         printf("NULL"); //空链表输出NULL
104         return;          //直接结束，不执行下面代码
105     }
106     while (s != NULL)
107     {
108         printf("%d ", s->data);
109         s = s->next;
110     }
111     printf("\n");
112     return;
113 }
114
```

Listing 1: 双向链表版本

1.4.2 双向循环链表版（未通过测试）

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <malloc.h>
4
5   typedef int Elemtype;
6   typedef struct node
7   {                       //链表结点
8       Elemtype data;      //结点数据域
9       struct node *next; //结点指针域
10  } Listnode, *Linklist; //节点类型，节点指针
11
12  typedef struct dnode
13  {
14      Elemtype data;              //结点数据域
15      struct dnode *prior, *next; //结点指针域
16  } DblNode, *DblList;            //节点类型，节点指针
17
18  DblList createDblList(); //此处用了双向循环链表
19  DblList mergeDblList(DblList, DblList);
20  void outputDblList(DblList); //空链表将输出NULL
21
22  void main()
23  {
```

```
    DblList L1, L2, L;
    printf("please input sequence1, end with -1:\t"); //本来想换行结尾，但是这样输入就是了char
    L1 = createDblList();
    printf("please input sequence2, end with -1:\t");
    L2 = createDblList();
    L = mergeDblList(L1, L2);
    printf("the result is:");
    outputDblList(L);
}

DblList createDblList() //建立双向循环链表
{
    DblList head, pa, pb;
    int x = 0;
    scanf("%d", &x);
    if (x == -1)
        exit(1);
    head = (DblList)malloc(sizeof(DblNode));
    head->prior = head->next = head; //表头结点的链指针指向自己
    pa = head;
    while (x != -1)
    {
        pb = (DblList)malloc(sizeof(DblNode));
        pb->data = x;
        pb->prior = pa;
        pb->next = pa->next;
        pa->next->prior = pb;
        pa->next = pb;
        pa = pb;
        scanf("%d", &x);
    }
    return head; //返回头指针
}

DblList mergeDblList(DblList La, DblList Lb)
{
    DblList Lc, pa, pb, pc;
    // Lc = (DblList)malloc(sizeof(DblNode));
    Lc = La; //链表的头结点，直接在链表头结点就地操作，降低空间复杂度ca
    pc = La; // 是用于插入的前一个结点pc
    pa = La->next;
    pb = Lb->next;                  //，是待插入的结点papb
    while (pa != La && pb != Lb) //while (pa!=NULL && pb!=NULL) //La,链表不为空Lb
    {
        if (pa->data < pb->data) //为了实现非降序排列需要进行分类讨论
        {
            pa->prior = pc;
            pa->next = pc->next;
            pc->next = pa;
            pc->next->prior = pa; //插入结点，并加入连接
            pc = pa;
            pa = pa->next; //后移，后移pcpa
            // DblList ptr = pa->next;
            // pa->prior = pc;
            // pa->next = pc->next;
            // pc->next->prior = pa;
            // pc->next = pa;
            // pc = pa;
```

```
            // pa = ptr;
        }
        if (pa->data > pb->data)
        {
            // DblList ptr = pb->next;
            // pb->prior = pc;
            // pb->next = pc->next;
            // pc->next->prior = pb;
            // pc->next = pb;
            // pc = pb;
            // pb = ptr;
            pb->prior = pc;
            pb->next = pc->next;
            pc->next = pb;
            pc->next->prior = pb;
            pc = pb;
            pb = pb->next; //同上
        }
        else //相等的话，就重复上述两个操作
        {
            pa->prior = pc;
            pa->next = pc->next;
            pc->next = pa;
            pc->next->prior = pa;
            pc = pa;
            pa = pa->next;

            pb->prior = pc;
            pb->next = pc->next;
            pc->next = pb;
            pc->next->prior = pb;
            pc = pb;
            pb = pb->next;
        }
    }
    if (pa != La)
    {
        while (pa != La)
        {
            pa->prior = pc;
            pa->next = pc->next;
            pc->next = pa;
            pc->next->prior = pa;
            pc = pa;
            pa = pa->next; //重复对操作a
        }
    }
    if (pb != Lb)
    {
        while (pb != Lb)
        {
            pb->prior = pc;
            pb->next = pc->next;
            pc->next = pb;
            pc->next->prior = pb;
            pc = pb;
            pb = pb->next; //重复对操作b
        }
```

```
    }
    return Lc; //返回头结点
}

void outputDblList(DblList s)
{
    DblList head = s;
    s = s->next;
    if (s == head)
    {
        printf("NULL"); //空链表输出NULL
        return;          //直接结束，不执行下面代码
    }
    while (s != head)
    {
        printf("%d ", s->data);
        s = s->next;
    }
    printf("\n");
    return;
}

```

Listing 2: 双向循环链表版

1.5 运行结果截图

```
PS C:\c_algorithm> cd "c:\c_algorithm\" ; if ($?) { gcc merge_two_list_dbl.c -o merge_two_list_dbl } ; if ($?) { .\merge_two_list_dbl
 }
please input sequence1, end with -1:    2 3 4 -1
please input sequence2, end with -1:    2 3 4 5 -1
the result is:2 2 3 3 4 4 5
```

图 1: 运行结果1

```
PS C:\c_algorithm> cd "c:\c_algorithm\" ; if ($?) { gcc merge_two_list_dbl.c -o merge_two_list_dbl } ; if ($?) { .\merge_two_list_dbl
 }
please input sequence1, end with -1:    2 2 2 -1
please input sequence2, end with -1:    2 3 3 3 -1
the result is:2 2 2 2 3 3 3
```

图 2: 运行结果2

1.6 算法时间复杂度

算法中的三个函数分别进行了一次while循环遍历整个链表，不存在嵌套循环情况，所以时间复杂度为$O(n)$

1.7 总结体会

第一次做这道题时我是用单链表做的，因为一开始没有看清题目的要求是双链表，成功的通过了测试，但不符合题目描述。之后我模仿着ppt使用双向循环链表，但是没有通过测试，debug结果为能够成功创建和

输出双向链表，但是不能进行合并。于是在单链表上改成了双链表。这次因为电脑容量不足以安装*visualstudio*在配置*vscode*的*debug*配置json环境花了很多时间。

2 第二题

2.1 实验目的和内容

祖玛是一款曾经风靡全球的游戏，其玩法是：在一条轨道上初始排列着若干个彩色珠子，其中任意三个相邻的珠子不会完全同色。此后，你可以发射珠子到轨道上并加入原有序列中。一旦有三个或更多同色的珠子变成相邻，它们就会立即消失。这类消除现象可能会连锁式发生，其间你将暂时不能发射珠子。

开发商最近准备为玩家写一个游戏过程的回放工具。他们已经在游戏内完成了过程记录的功能，而回放功能的实现则委托你来完成。

游戏过程的记录中，首先是轨道上初始的珠子序列，然后是玩家接下来所做的一系列操作。你的任务是，在各次操作之后及时计算出新的珠子序列。

2.2 输入和输出说明

2.2.1 输入

第一行是一个由大写字母ßAß ßZß组成的字符串，表示轨道上初始的珠子序列，不同的字母表示不同的颜色。

第二行是一个数字n，表示整个回放过程共有n次操作。

接下来的n行依次对应于各次操作。每次操作由一个数字k和一个大写字母$\sum$描述，以空格分隔。其中，$\sum$为新珠子的颜色。若插入前共有m颗珠子，则$k \in [0, m]$表示新珠子嵌入之后（尚未发生消除之前）在轨道上的位序。

2.2.2 输出

输出共n行，依次给出各次操作（及可能随即发生的消除现象）之后轨道上的珠子序列。

如果轨道上已没有珠子，则以“−”表示。

2.2.3 样例

- 输入

  ACCBA

  5

  1 B

  0 A

  2 B

  4 C

  0 A

- 输出

ABCCBA

AABCCBA

AABBCCBA

-

A

2.3 解题思路

本题主要用了五个函数。
第一个建立双向链表函数，定义了头指针和尾指针并赋值'–'，并且用数组传参
第二个插入函数，就是找到某一个位置并进行插入
第三个是找到双向链表尾部，这个函数为删除，输出函数铺垫
第四个删除函数，需要注意的有两点：

- 第一是需要找准删除的起点和终点
- 第二是需要讨论三种及以上的球的情况

第五个是输出函数，我们将双向链表赋值给数组并输出

2.4 实验代码及注释

```
#include <stdio.h>
#include "string.h"
#include <stdlib.h>
#include <malloc.h>

#define Len 20000
#define Up (Len * 3 / 4)

typedef char ElemType;
typedef struct node //定义双向链表
{
    ElemType data; //数据域
    struct node *next;
    struct node *front; //指针域
} List, *pList;          //结点类型，结点指针

// pList pHead = (pList)malloc(sizeof(List));
// pList pTail = (pList)malloc(sizeof(List));
//常量表达式中不允许函数调用

char ans[Len + 5];
int forprt = 0;

pList creat(char *a, int n)
{
    /* 为参数个数，n*为传入参数a*/
    int i;
    pList pHead = (pList)malloc(sizeof(List));
    pList pTail = (pList)malloc(sizeof(List)); //创建头指针和尾指针
```

```
30
31      pList pt = pHead; // 指向头指针pt
32
33      pTail->front = pHead;
34      pTail->next = NULL;
35      pHead->next = pTail;
36      pHead->front = NULL;                //创建空的双向链表
37      pHead->data = pTail->data = '-'; //赋值为表示轨道上已没有珠子'-',
38
39      for (i = 0; i < n; i++)
40      {
41          pList pNew = (pList)malloc(sizeof(List));
42          pNew->data = a[i];  // 先通过得到数组储存字符串，然后通过数组来传值gets
43          pNew->front = pt;
44          pNew->next = pt->next;
45          pt->next->front = pNew;
46          pt->next = pNew;
47          pt = pNew;  //移动指针pt
48      }
49      return pHead;
50  }
51
52  pList insert(int m, char ch, pList pHead)
53  {
54      int i = -1; // i方便对位置零操作=-1.
55
56      pList pt = pHead, pNew = (pList)malloc(sizeof(List));
57
58      while (i++ < m)
59          pt = pt->next; //找到带插入位置
60
61      pNew->data = ch;
62      pNew->next = pt;
63      pNew->front = pt->front;
64      pt->front->next = pNew;
65      pt->front = pNew; //注意是插在找到位置的前面
66
67      return pHead;
68  }
69
70  pList find_tail(pList pHead)
71  {
72      pList pt = pHead->next;
73      //while(!(pt->data == '-')) pt = pt ->next;
74      while (pt->next != NULL)
75          pt = pt->next;    //这两种方法都可以找到pTail
76      return pt;
77  }
78
79  pList del(int m, pList pHead)
80  {
81      pList pTail, point_tmp;
82      pTail = find_tail(pHead);
83      //获得ptail
84      pList p1 = NULL, p2 = NULL, p3 = NULL, p4 = NULL, pt = pHead; //空指针最好赋值 NULL
85      pList begin = pHead, end = pTail;
86      int boo = 1; //编译器不支持类型，所以用表示逻辑gccboolint
87      int repeat, i = -1;
```

```
88
89      // find position
90      while (i++ < m - 2)
91          pt = pt->next; // 找到插入位置前两个指针
92
93      //init for 'begin' and 'end'
94      begin = pt;
95      end = pt;
96      i = 0;
97      while (i++ < 4 && end->next != pTail)
98          end = end->next; //找到插入位置后两个指针
99
100     while (boo && pt != pTail)
101     {
102         boo = 0; //判断有没有发生消除
103         repeat = 1; //计数重复的个数
104         while (pt != end) //在位置ptbegin
105         {
106             pt = pt->next;
107
108             if (pt->front->data == pt->data) //后移，如果相同，就计数加一ptdata
109                 repeat++;
110             else
111                 repeat = 1;
112
113             if (repeat == 3)
114             {
115                 boo = 1;
116                 if (pt->data == pt->next->data) //已经满足消除条件了，再看看能不能继续满足
117                 {
118                     repeat++;
119                     pt = pt->next;
120                 }
121
122                 if (repeat == 3)
123                 {
124                     p3 = pt;
125                     p2 = p3->front;
126                     p1 = p2->front;
127                     p1->front->next = p3->next;
128                     p3->next->front = p1->front;
129                     pt = pt->next; //消除这三个，将后移pt
130                     free(p1);
131                     free(p2);
132                     free(p3);
133                 }
134                 else
135                 {
136                     p4 = pt;
137                     p3 = p4->front;
138                     p2 = p3->front;
139                     p1 = p2->front;
140                     p1->front->next = p4->next;
141                     p4->next->front = p1->front;
142                     pt = pt->next; //消除这个，将4后移pt
143                     free(p1);
144                     free(p2);
145                     free(p3);
```

```
                    free(p4);
                }

                break;
            }
        }

        if (boo && pt != pTail)
        {
            begin = pt;
            i = 0;
            while (i++ < 2 && begin->front != pHead)
                begin = begin->front;
            end = pt;
            i = 0;
            if (i++ < 1 && end->next != pTail)
                end = end->next;
            pt = begin; //将的和指针归位ptbeginend
        }
    }
    return pHead;
}

pList show(int boo, pList pHead) //将双向链表的值存到数组便于输出
{

    pList pt = pHead->next;
    pList pTail = find_tail(pHead);

    if (pt == pTail)
        ans[forprt++] = '-'; //没有数
    else
    {
        while (pt->next != NULL)
        {
            ans[forprt++] = pt->data; //扩容并存data
            pt = pt->next;  //后移pt
        }
    }

    ans[forprt++] = '\n'; //输出需要有换行

    if (forprt >= Up || boo) //如果到尽头了
    {
        ans[forprt] = '\0'; //数组一定要以 \0 结尾
        printf("%s", ans); //输出
        forprt = 0;
    }
    return pHead; //其实不需要返回了
}

int main(void)
{
    char a[10005];
    int n, k;
    pList pHead = NULL;

    //printf请输入初始队列("\n");
```

```
    gets(a); // 初始的珠子序列
    //printf请输入操作数("");
    scanf("%d", &n); // 共有次操作n

    pHead = creat(a, strlen(a)); //创建对应初始的珠子序列的双向链表

    for (k = 0; k < n; k++)
    {
        int m;
        char ch;

        scanf("%d ", &m); //新的珠子位序

        ch = getchar(); // 插入珠子颜色

        // insert ch
        insert(m, ch, pHead);

        // delete all 3-same block, making it the right string
        del(m, pHead);

        // print the string
        show(k == n - 1 ? 1 : 0, pHead);
    }

    return 0;
}
```

2.5 运行结果截图

```
PS C:\c_algorithm> cd "c:\c_algorithm\" ; if ($?) { gcc zuma.c -o zuma } ; if ($?) { .\zuma }
请输入初始队列
ACCBA
请输入操作数5
1 B
0 A
2 B
4 C
0 A
ABCCBA
AABCCBA
AABBCCBA
-
A
```

图 3: 运行结果

2.6 总结体会

主要复习了debug的基本操作，熟悉了双向链表的建立和删除操作。同时对于“实际问题”，需要分类讨论，考虑到多种情况。对于一些重复利用的操作可以写成函数方便调用

A 不用链表的zuma实现

请看另一份报告