2015.10.12, 数据结构与算法作业 1

王贺，2014060102018

2015 年 10 月 7 日

1 Question 1

将两个递增的有序链表合并为一个递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其他的存储空间。表中不允许有重复的数据。

1.1 语言描述

ListA，ListB 合并到 ListC

1）选取 ListA 和 ListB 第一个数据较小的添加到 ListC，将较小的那个指向后继后删除路过的结点

2）比较 ListA 和 ListB 当前数据，将较小的添加到 ListC 中，如果出现相同的则删除其一，直至链表尾

3）将当前不为空的后继所有结点依次添加到 ListC 中

1.2 伪代码描述

MERGELIST(A)

```
1  la ← ListA
2  lb ← ListB
3  lc ← ListC
4  if la.data < lb.data
5      then
6           lc ← la
7           la ← del(la)
8      else
9           lc ← lb
10          lb ← del(lb)
11 while la.next ≠ null  and  lb.next ≠ null
12       do
13          if la.data = lb.data
14            then
15                 append(lc, la, True)
16            else
17                 if la.data < lb.data
18                   then
19                        append(lc, la, False)
20                   else
21                        append(lc, lb, False)
22 if la.next = NULL
23     then
24          lc.next ← lb
25     else
26          lc.nect ← la
```

1.3 代码描述

```
1  typedef struct
2  {
3      int data;
```

```
4       struct node* next;
5   }*LinkNode, Linklist;
6
7
8   LinkNode del(LinkNode  l)
9   {
10      LinkNode temp = l;
11      l = l ->next;
12      //delete temp;
13      return l;
14  }
15
16  LinkNode append(LinkNode front, LinkNode next, erase = Flase)
17  {
18
19      LinkNode temp = front -> next;          //save front's next
20      front -> next = next;           //insert
21      front -> next -> next = temp;   //update the insert
22      next = next -> next;            //update next
23      if (erase)                       //delete
24      {
25          del(next);
26      }
27  }
28
29  void mergeList(LinkList ListA, LinkList ListB, LinkList &ListC)
30  {
31      LinkNode la,lb,lc;
32      la = ListA;
33      lb = ListB;
34      lc = ListC;
35      la -> data < lb -> data ? {lc = la; la = del(la);} : {lc = lb; lb = del(lb);};
36
37      while (la -> next != NULL && lb -> next != NULL)
38      {
```

```
39          if (la -> data == lb -> data)
40          {
41              append(lc, la, True);
42          }
43          else
44          {
45              la -> data < lb -> data ? append(lc, la, False) : append(lc, lb, False);
46          }
47      }
48      la -> next == NULL ? lc -> next = lb : lc -> next = la;
49  }
```

2 Question 2

将两个非递减的有序链表合并为一个非递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其他的存储空间。表中允许有重复的数据。

2.1 语言描述

1）以 ListA，ListB 小的元素结点为跟踪标识，并定义扫描标识

2）扫描标识数据位比较，取小的，并将取走后的标识位后移（删除无用的头结点）

3）更新上一步骤取走的结点的后继为跟踪标识，更新跟踪标识为这个结点

4）回到步骤 2 直至一链表为空，则重复 3 填充跟踪标识

2.2 伪代码描述

```
mergeList(B)
 1  if ListA.data <= ListB.data
 2     then
 3          li ← ListA
 4          scannerA ← ListA.next
 5          scannerB ← ListB
 6          ▷ if delete
 7          deleteListA
 8     else
 9          li ← ListB
10          scannerB ← ListB.next
11          scannerA ← ListA
12          ▷ if delete
13          deleteListB
14  while scannerA ≠ NULL  and  scannerB ≠ NULL
15        do
16          if scannerA.data <= scannerB.data
17            then
18                 li ← li.next ← scannerA
19                 scannerA ← scannerA.next
20            else
21                 li ← li.next ← scannerA
22                 scannerB ← scannerB.next
23  if scannerA = NULL
24     then
25          scanner ← scannerB
26     else
27          scanner ← scannerA
28  while scanner ≠ NULL
29        do
30          li ← li.next ← scanner
31          scanner ← scanner.next
```

2.3 代码描述

```
1  void mergeList(LinkList &ListA, LinkList &ListB)
2  {
3      LinkNode scannerA,scannerB,li;
4      scannerA = scannerB = li = NULL;
5
6      if (ListA -> data <= ListB -> data)
7      {
8          li = ListA;
9          scannerA = ListA -> next;
10         scannerB = ListB;
11         //delete ListA;
12     }
13     else
14     {
15         li = ListB;
16         scannerB = ListB -> next;
17         scannerA = ListA;
18         //delete ListB;
19     }
20
21     while (scannerA != NULL && scannerB != NULL)
22     {
23         if (scannerA -> data <= scannerB -> data)
24         {
25             li = li -> next = scannerA;
26             scannerA = scannerA -> next;
27         }
28         else
29         {
30             li = li -> next = scannerB;
31             scannerB = scannerB -> next;
32         }
33     }
```

```
34      LinkNode scanner = NULL;
35      scanner = scannerA == NULL ? scannerB : scannerA;
36      while (scanner != NULL)
37      {
38          li = li -> next = scanner;
39          scanner = scanner -> next;
40      }
41  }
```

3 Question 3

设计一个算法，通过一趟遍历在单链表中确定值最大的结点。

3.1 语言描述

1) 跟踪标识为头数据结点，扫描标识为下一个结点

2）比较两标识数据，如果跟踪标识小，则移动跟踪标识；否则扫描标识后移

3）当扫描标识为空，跟踪标识为最大

3.2 伪代码描述

findMax

```
1  maxNode ← list
2  cursor ← list.next
3  while cursor ≠ NULL
4        do
5           if maxNode.data < cursor.data
6             then
7                  maxNode ← cursor
8           cursor ← cursor.next
```

3.3 代码描述

```
1  void findMax(LinkList list)
2  {
3      LinkNode maxNode = list;
4      LinkNode cursor = list -> next;
5
6      while(cursor != NULL)
7      {
8          if (maxNode -> data < cursor -> data)
9          {
10             maxNode = cursor;
11         }
12         cursor = cursor -> next;
13     }
14 }
```

4 Question 4

设计一个算法，通过一趟遍历，将链表中所有结点的链接方向逆转，且仍利用原表的存储空间。

4.1 语言描述

1) 跟踪结点为头结点，保存跟踪结点的后继为扫描结点，将头结点后继指向 null，跟踪结点更新到扫描结点

2）保存跟踪结点的后继为扫描结点，将跟踪结点后继指向跟踪结点本身，跟踪结点更新到扫描结点

4.2 伪代码描述

REVERSE

```
1  tracker ← list
2  cursor ← list.next
3  list.next ← NULL
4  while cursor ≠ NULL
5        do
6           cursor ← tracker.next
7           tracker.next ← tracker
8           tracker ← cursor
9  return tracker
```

4.3 代码描述

```
1  LinkNode reverse(LinkList& list)
2  {
3      LinkNode tracker = list;
4      LinkNode cursor = list -> next;
5      list -> next = NULL;
6      while (cursor != NULL)
7      {
8          cursor = tracker -> next;
9          tracker -> next = tracker;
10         tracker = cursor;
11     }
12     return tracker;
13 }
```

5 Question 5

将编号为 0 和 1 的两个栈存放于一个数组空间 V[m] 中，栈底分别处于数组的两端。当第 0 号栈的栈顶指针 top[0] 等于-1 时该栈为空；当第 1 号栈的栈顶指针 top[1] 等于 m 时，该栈为空。两个栈均从两端向中间增长。试编写双栈初始化，判断栈空、栈满、进栈和出栈等算法的函数。双栈数据结构的定义如下：

```
typedef struct{
    int top[2], bot[2];  //栈顶和栈底指针
    SElemType *V;        //栈数组
    int m;               //栈最大可容纳元素个数
}DblStack;
```

5.1 代码描述

```
typedef struct
{
    int top[2], bot[2];  //栈顶和栈底指针
    SElemType *V;        //栈数组
    int m;              //栈最大可容纳元素个数
}DblStack;

void initDblStack(DblStack& s, SElemType size)
{
    s.m = size;
    s.V = new SElemType(size);
    s.top[0] = s.bot[0] = -1;
    s.top[1] = s.bot[1] = size -1;
}



int IsEmpty(DblStack s, int i)
{
    if (i == 0)
    {
        return s.top[0] == -1 ? 1 : 0;
    }
    else if (i == 1)
    {
```

```
26          return s.top[1] == s.m ? 1 : 0;
27      }
28      else
29      {
30          return -1;
31      }
32  }
33
34  int IsFull(DblStack s)
35  {
36      return s.top[1]-s.top[0] > 0 ? 0 : 1;
37  }
38
39  void Dblpush(DblStack &s,SElemType x,int i)
40  {
41      if (i == 0 && s.top[0] + 1 < s.top[1])
42      {
43         s.V[++s.top[0]] = x;
44      }
45      else if (i == 1 && s.top[1] - 1 > s.top[0])
46      {
47          s.V[--s.top[1]] = x;
48      }
49
50  }
51
52  int Dblpop(DblStack &s,int i,SElemType &x)
53  {
54      if (i == 0 && !IsEmpty(s, i))
55      {
56          x = s.V[top[0]--];
57      }
58      else if (i == 1 && !IsEmpty(s, i))
59      {
60          x = s.V[top[1]++];
```

```
61          }
62          else
63          {
64              return 0;
65          }
66          return 1;
67      }
```