

《数据结构》上机报告

2018 年 9 月 25 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：_____

实验题目	顺序表	
问题描述	顺序表是指采用顺序存储结构的线性表，它利用内存中的一片连续存储区域存放表中的所有元素。可以根据需要对表中的所有数据进行访问，元素的插入和删除可以在表中的任何位置进行。	
基本要求	1. (p1)实现顺序表的基本操作，包括顺序表的初始化、第 i 个元素前插入一个新的元素、删除第 i 个元素、查找某元素、顺序表的销毁。 2. (p2)实现对有序表（非递减）插入元素、删除元素、查找元素的功能 3. (p3)实现两个有序（非递减）表合并生成新的有序表的功能 4. (p4)顺序查找顺序表，删除元素 e（删除所有值为 e 的元素）。 5. (p5)删除顺序表中所有多余的元素（即没有相同的元素）。	
	已完成基本内容（序号）：	1, 2, 3, 4, 5
选做要求		
	已完成选做内容（序号）	
数据结构设计	<pre>typedef struct { ElemType* elem; int length; int listsize; } SqList;</pre>	

功能(函数)说明	<p>1、顺序表的初始化</p> <p>分配 LIST_INIT_SIZE 的空间，失败返回-1，成功返回 0.</p> <pre> Status InitList_Sq(SqList &L) { L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType)); if (!L.elem) exit(OVERFLOWED); L.listsize = LIST_INIT_SIZE; L.length = 0; return OK; } </pre> <p>2、基本的插入操作</p> <p>输入插入元素的位置和运算的值，实现数据的插入。如果顺序表的长度大于分配的空间，就再次分配一次空间，成功返回 1，失败返回-1。 找到元素位置后，将该位置之后的所有元素后移，再次进行插入，表的长度增加 1。</p> <pre> Status ListInsert_Sq(SqList &L, int pos, ElemType e) { ElemType *newbase; if (pos >= 1 && pos <= L.length + 1) { if (L.length >= L.listsize) { newbase = (ElemType *)realloc(L.elem, (L.listsize + LISTINCREMENT) * sizeof(ElemType)); if (!newbase) { return ERROR; } L.elem = newbase; L.listsize += LISTINCREMENT; } ElemType *p, *q; p = &(L.elem[pos - 1]); for (q = &(L.elem[L.length - 1]); q >= p; --q) { *(q + 1) = *q; } *p = e; ++L.length; return OK; } else { return OVERFLOWED; } } </pre> <p>3、基本的删除操作</p>
----------	---

遍历顺序表，如果没找到该元素，返回-1。如果找到该元素，将该元素后的所有元素前移，表的长度减一。

```
Status ListDelete_Sq(SqList &L, int pos, ElemType &e)
{
    if (pos >= 1 && pos <= L.length) {
        ElemType *q;
        e = L.elem[pos - 1];
        for (q = &(L.elem[pos - 1]); q <= &(L.elem[L.length - 1]); ++q) {
            *q = *(q + 1);
        }
        L.length = L.length - 1;
        return OK;
    }
    else {
        return OVERFLOWED;
    }
}
```

4、查找指定元素位置

遍历顺序表，如果找到该元素，返回位置，如果未找到，返回-1。

```
int ListLocate_Sq(SqList L, ElemType e)
{
    int a = -1;
    for (int i = 0; i <= L.length - 1; i++) {
        if (L.elem[i] == e) {
            a = i;
            break;
        }
    }
    if (a >= 0 && a <= L.length - 1) {
        return a + 1;
    }
    else {
        return ERROR;
    }
}
```

5、有序表的插入

有序表内部是有顺序的，所以可以使用二分法进行检索，以减少时间复杂度。在插入时，使用二分法确定元素的位置，找到位置后，将该位置后的所有元素后移，表的长度+1。

使用两个函数。首先确定要插入的元素的位置，在进行插入。分开功能便于代码的管理。

```
Status ListFind_sq(Ordered_list L, int e)
{
```

```

int low = 0, high = L.length - 1;
if (L.elem[high]<e) {
    return(high + 2);
}
if (L.elem[high] == e) {
    return(high + 1);
}
if (L.elem[low] >= e) {
    return(low + 1);
}

int mid = (low + high) / 2;
while (low != high) {
    if (L.elem[mid]>e) {
        high = mid - 1;
    }
    else {
        low = mid;
    }
    mid = (low + high) / 2;
    if (low == high - 1) {
        break;
    }
}

int temp = high;
if (L.elem[high] == e) {
    return(temp + 1);
}
if (L.elem[high]<e) {
    return(temp + 2);
}
if (L.elem[0] == e) {
    return(1);
}
while (L.elem[temp] >= e) {
    temp--;
}
return (temp + 2);
}

Status ListInsert_sq(Ordered_list &L, int i, int e)
{
    if (i<1 || i>L.length + 1) {
        return ERROR;
    }
    if (L.length >= L.listsize) {

```

```

        int *newbase;
        newbase = (int*)realloc(L.elem, (L.listsize + LISTINCREMENT) * sizeof(int));
        if (!newbase) exit(OVERFLOW);
        L.elem = newbase;
        L.listsize += LISTINCREMENT;
    }

    int *q, *p;
    q = &(L.elem[i - 1]);
    for (p = &(L.elem[L.length - 1]); p >= q; --p) {
        *(p + 1) = *p;
    }

    *q = e;
    ++L.length;
    return OK;
}

```

6、有序表的删除

遍历找到该元素的位置，若未找到，返回-1。若找到该元素，将该元素后的每一元素前移，表的长度减一。

同样使用两个函数进行编写，首先找到位置，再进行删除。

```

Status ListLocate_sq(Ordered_list L, int y)
{
    int i = 1;
    while (i <= L.length && L.elem[i - 1] != y) {
        i++;
    }

    if (L.elem[i - 1] == y && i - 1 < L.length) {
        return i;
    }

    else {
        return -1;
    }
}

Status ListDelete_sq(Ordered_list &L, int i, int &e)
{
    if (i < 1 || i > L.length) {
        return ERROR;
    }

    int *p, *q;
    p = &(L.elem[i - 1]);
    e = *p;
    q = L.elem + L.length - 1;
    for (++p; p <= q; ++p) {
        *(p - 1) = *p;
    }
}

```

```
--L.length;
return OK;
}
```

7、合并有序表

同时遍历表 1 和表 2，每次选取一个元素作比较，将小的数加入到新的表中。

```
Status Merge_Sq(Ordered_list &L1, Ordered_list &L2, Ordered_list &L3)
```

```
{
    int i=0, j=0;
    while (i != L1.length&& j != L2.length) {
        if (L1.elem[i] < L2.elem[j]) {
            L3.elem[i + j] = L1.elem[i];
            L3.length++;
            i++;
        }
        else {
            L3.elem[i + j] = L2.elem[j];
            j++;
            L3.length++;
        }
    }
    if (i == L1.length&& j == L2.length) {
        return 0;
    }
    if (i == L1.length) {
        for (; j < L2.length; j++) {
            L3.elem[i + j] = L2.elem[j];
            L3.length++;
        }
    }
    else {
        for (; i < L1.length; i++) {
            L3.elem[i + j] = L1.elem[i];
            L3.length++;
        }
    }
    return 0;
}
```

8、删除元素 e。

使用原地删除的方式可大大缩短时间复杂度。在一次循环中就可以完成删除。每次将当前的数值与之前的数值作比较，若无相等的，则直接进行前移。

```
Status ListDelete_Sq(SqList &L1, ElemType &e)
```

```
{
    int i = 0, j = 0;
    for (i = 0; i < L1.length; i++) {
        if (L1.elem[i] != e) {
```

	<pre> L1.elem[j] = L1.elem[i]; j++; } } L1.length -= (i - j); if (i == j) { return 0; } return 1; }</pre> <p>9、去重操作</p> <p>遍历顺序表，将当前元素与前面的所有元素进行比较，如果相等，跳过，如果不相等，将该元素原地前移。</p> <pre>Status ListDelete_Sq(SqList &L1) { int i = 0, j = 0, k=1; bool flag = true; for (i = 1; i < L1.length; i++) { flag = true; for (j = 0; j < k; j++) { if (L1.elem[i] == L1.elem[j]) { flag = false; break; } } if (flag == false) { continue; } else { L1.elem[k] = L1.elem[i]; k++; } } L1.length -= (i - k); return 0; }</pre>
开发环境	Visual studio 2017

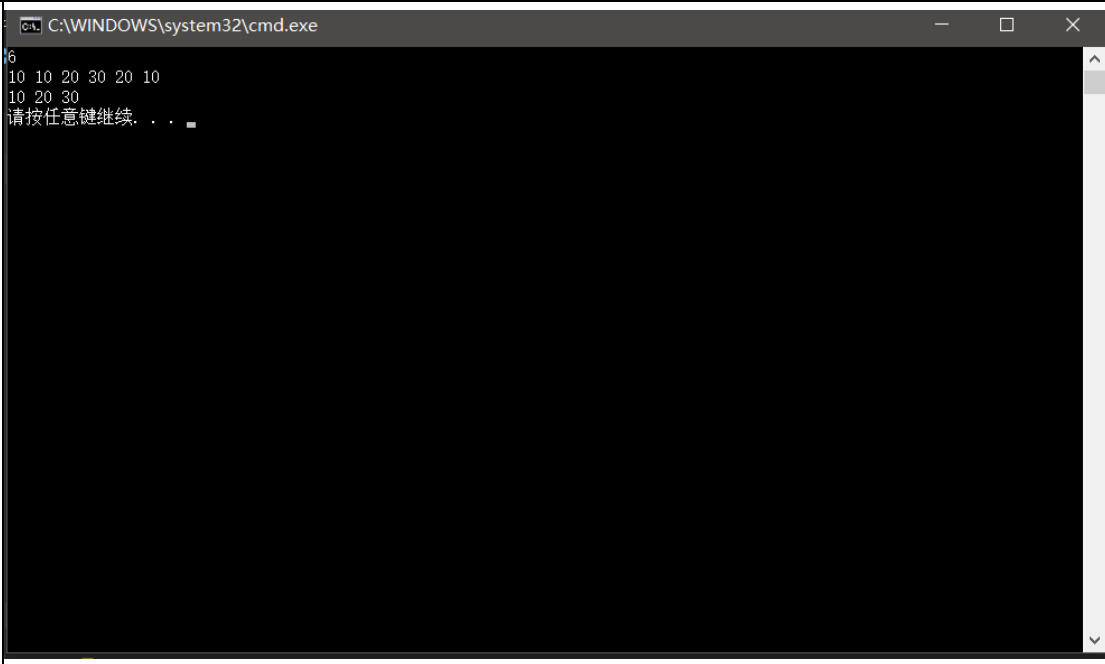
调试分析

The image displays two sequential screenshots of a Windows command prompt window, titled "C:\WINDOWS\system32\cmd.exe". The window has a dark background and a light gray title bar with standard Windows window controls (minimize, maximize, close). The command prompt shows the output of a debugger, with lines of text representing the execution flow of a program. The first screenshot shows the following text: 4, 10 20 40 30, 10 20 40 30, 1 25, 25 10 20 40 30, 3, 25 10 40 30, 10, 2, and 请按任意键继续. . . . The second screenshot shows the following text: 10 20 40 30 0, 25, 3, 20, 2, 10 25 30 40, and 请按任意键继续. . . . The text is white and appears to be a mix of hexadecimal and decimal values, likely representing memory addresses or register values. The command prompt is positioned on the right side of the image, with a vertical white bar on the left containing the text "调试分析".

```
C:\WINDOWS\system32\cmd.exe
4
10 20 40 30
10 20 40 30
1 25
25 10 20 40 30
3
25 10 40 30
10
2
请按任意键继续. . .

C:\WINDOWS\system32\cmd.exe
10 20 40 30 0
25
3
20
2
10 25 30 40
请按任意键继续. . .
```




	
心得体会	<ol style="list-style-type: none"> 1、二分法减少有序表的时间复杂度 当进行有序表的操作时，可利用表内自身元素有顺序的特点，使用二分法进行检索和查找。可大大减少时间复杂度，从遍历的 $O(n)$ 减少到 $O(\log n)$。 2、使用 typedef 增强兼容性 在定义数据结构时，可通过定义大量的类型名和数据，来增强可改动的特性，便于代码的管理。 3、在做第四题时，会遇到 time limit exceed 的报错。应在最开始时对线性表的内存分配进行扩容，会大大减少之后分配空间的时间。 4、将不同功能的函数尽量分开编写，便于代码的管理与 debug。