

排序

本文档需要用到的编程相关知识点：

C语言：struct（结构体）、typedef、malloc（free）、函数做参数。

本文档的参考教材：

数据结构（C语言版） 严蔚敏

一. 归并排序

归并排序的实现参考教材283页。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define OK 1
5  #define TRUE 1
6  #define FALSE 0
7  #define ERROR 0
8  #define OVERFLOW -1
9  #define LIST_INIT_SIZE 10 // 线性表存储空间初始分配容量
10 #define LISTINCREMENT 10 //线性表存储空间分配增量
11 #define NOTEXIT 0
12 typedef int Status;
13 typedef int ElemType;
14
15 struct SqList {
16     ElemType *elem; // 存储空间基址
17     int length; // 当前长度
18     int listsize; // 当前分配的存储容量
19 };
20
21 /**
22  * 构造一个空的线性表L
23  */
24 Status InitList_Sq(struct SqList &L)
25 {
26     L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(struct SqList));
27     if (!L.elem)
28         return (OVERFLOW); // 存储分配失败
29     L.length = 0; // 空表长度为0
30     L.listsize = LIST_INIT_SIZE; // 初始存储容量
31     return OK;
32 } // InitList_Sq
33
34 /**
35  * 在第i个位置之前插入数据元素e，L的长度加1
36  * 要求线性表存在，1<=i<=ListLength_Sq(L)+1
37  */
38 void ListInsert_Sq(struct SqList &L, int i, ElemType e)
```

```

39 {
40     int j;
41     ElemType *newbase;
42     if (L.length + 2 > L.listsize) { // 因为0号元素未用，所以是加2
43         newbase = (ElemType *)realloc(
44             L.elem, (L.listsize + LISTINCREMENT) * sizeof(ElemType));
45         if (!newbase)
46             exit(OVERFLOW); // 存储分配失败
47         L.elem = newbase; // 新基址
48         L.listsize += LISTINCREMENT; // 增加存储容量
49     }
50     for (j = L.length + 1; j >= i; j--) {
51         *(L.elem + j) = *(L.elem + j - 1);
52     }
53     *(L.elem + i - 1) = e;
54     L.length++; // 表长加1
55 } // ListInsert_Sq
56
57 /**
58  * 依次对L的每个元素调用函数visit(),
59  * 一旦visit()失败, 则操作失败, 返回FALSE, 否则返回TRUE 要求线性表存在
60  */
61 Status ListTraverse_Sq(struct SqList L, Status (*visit)(ElemType))
62 {
63     int i;
64     for (i = 1; i <= L.length; i++) {
65         if (!visit(*(L.elem + i)))
66             return FALSE;
67     }
68     return TRUE;
69 } // ListTraverse_Sq
70
71 Status visit_display_Sq(ElemType e)
72 {
73     printf("%d ", e);
74     return TRUE;
75 } // visit_display_Sq
76
77 /**
78  * 算法10.12, 将有序的SR[i...m]和SR[m+1...n]归并为有序的TR[i...n]
79  */
80 void Merge(int SR[], int TR[], int i, int m, int n)
81 {
82     int j, k;
83     for (j = m + 1, k = i; i <= m && j <= n; k++) {
84         if (SR[i] <= SR[j]) {
85             TR[k] = SR[i];
86             i++;
87         } else {
88             TR[k] = SR[j];
89             j++;
90         }
91     }
92     if (i <= m) {
93         for (j = i; j <= m; j++) {
94             TR[k] = SR[j];
95             k++;
96         }

```

```

97     }
98     if (j <= n) {
99         for (i = j; i <= n; i++) {
100             TR[k] = SR[i];
101             k++;
102         }
103     }
104 }
105
106 /**
107  * 算法10.13, 将SR[s...t]归并排序为TR1[s...t]
108  */
109 void MSort(int SR[], int TR1[], int s, int t)
110 {
111     int m;
112     int TR2[100];
113     if (s == t) {
114         TR1[s] = SR[s];
115     } else {
116         m = (s + t) / 2;
117         MSort(SR, TR2, s, m);
118         MSort(SR, TR2, m + 1, t);
119         Merge(TR2, TR1, s, m, t);
120     }
121 }
122
123 /**
124  * 算法10.14
125  */
126 void MergeSort(SqList &L) { MSort(L.elem, L.elem, 1, L.length); }

```

归并排序测试程序：（测试程序代码放在实现代码下方即可）

```

1  int main()
2  {
3      int arr[8] = {33, 12, 75, 0, 49, 67, 8, 999};
4      SqList list1;
5      InitList_Sq(list1);
6      for (int i = 1; i < 9; i++) // 将arr数组中的8个元素插入顺序表中
7          ListInsert_Sq(list1, i + 1, arr[i - 1]);
8
9      printf("初始线性表为: \n");
10     ListTraverse_Sq(list1, visit_display_Sq);
11
12     MergeSort(list1);
13     printf("\n排序后的线性表为: \n");
14     ListTraverse_Sq(list1, visit_display_Sq);
15
16     return 0;
17 }

```

测试程序运行结果为：

初始线性表为：

33 12 75 0 49 67 8 999

排序后的线性表为：

0 8 12 33 49 67 75 999

二. 快速排序

快速排序的实现参考教材274页。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define OK 1
5  #define TRUE 1
6  #define FALSE 0
7  #define ERROR 0
8  #define OVERFLOW -1
9  #define LIST_INIT_SIZE 10 // 线性表存储空间初始分配容量
10 #define LISTINCREMENT 10 //线性表存储空间分配增量
11 #define NOTEXIT 0
12 typedef int Status;
13 typedef int ElemType;
14
15 struct Sqlist {
16     ElemType *elem; // 存储空间基址
17     int length; // 当前长度
18     int listsize; // 当前分配的存储容量
19 };
20
21 /**
22  * 构造一个空的线性表L
23  */
24 Status InitList_Sq(struct Sqlist &L)
25 {
26     L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(struct Sqlist));
27     if (!L.elem)
28         return (OVERFLOW); // 存储分配失败
29     L.length = 0; // 空表长度为0
30     L.listsize = LIST_INIT_SIZE; // 初始存储容量
31     return OK;
32 } // InitList_Sq
33
34 /**
35  * 在第i个位置之前插入数据元素e, L的长度加1
36  * 要求线性表存在, 1<=i<=ListLength_Sq(L)+1
37  */
38 void ListInsert_Sq(struct Sqlist &L, int i, ElemType e)
39 {
40     int j;
41     ElemType *newbase;
42     if (L.length + 2 > L.listsize) { // 因为0号元素未用, 所以是加2
43         newbase = (ElemType *)realloc(
44             L.elem, (L.listsize + LISTINCREMENT) * sizeof(ElemType));
45         if (!newbase)
46             exit(OVERFLOW); // 存储分配失败
47         L.elem = newbase; // 新基址
48         L.listsize += LISTINCREMENT; // 增加存储容量
49     }
50     for (j = L.length + 1; j >= i; j--) {
51         *(L.elem + j) = *(L.elem + j - 1);
52     }
```

```

53     *(L.elem + i - 1) = e;
54     L.length++; // 表长加1
55 } // ListInsert_Sq
56
57 /**
58  * 依次对L的每个元素调用函数visit(),
59  * 一旦visit()失败, 则操作失败, 返回FALSE, 否则返回TRUE 要求线性表存在
60  */
61 Status ListTraverse_Sq(struct SqList L, Status (*visit)(ElemType))
62 {
63     int i;
64     for (i = 1; i <= L.length; i++) {
65         if (!visit(*(L.elem + i)))
66             return FALSE;
67     }
68     return TRUE;
69 } // ListTraverse_Sq
70
71 Status visit_display_Sq(ElemType e)
72 {
73     printf("%d ", e);
74     return TRUE;
75 } // visit_display_Sq
76
77 /**
78  * 交换两个数i, j
79  */
80 void swap(int &i, int &j)
81 {
82     int temp;
83     temp = i;
84     i = j;
85     j = temp;
86 }
87
88 /**
89  * 算法10.6 (a), 交换顺序表L中子表L.r[low...high]的记录, 使枢轴记录到位,
90  * 并返回其所在位置, 此时在它之前(之后)的记录不大(小)于它。
91  */
92 int Partition_a(SqList &L, int low, int high)
93 {
94     int pivotkey;
95     pivotkey = L.elem[low]; // 第一个记录作枢轴记录
96     while (low < high) {
97         while (low < high && L.elem[high] >= pivotkey)
98             --high;
99         swap(L.elem[low], L.elem[high]);
100         while (low < high && L.elem[low] <= pivotkey)
101             ++low;
102         swap(L.elem[low], L.elem[high]);
103     }
104     return low;
105 }
106
107 /**
108  * 算法10.6(b), 对算法10.6(a)的改进算法, 减少了交换的步骤
109  */
110 int Partition(SqList &L, int low, int high)

```

```

111 {
112     int pivotkey;
113     L.elem[0] = L.elem[low];
114     pivotkey = L.elem[low]; //第一个记录作枢轴记录
115     while (low < high) {
116         while (low < high && L.elem[high] >= pivotkey) {
117             --high;
118         }
119         L.elem[low] = L.elem[high];
120         while (low < high && L.elem[low] <= pivotkey) {
121             ++low;
122         }
123         L.elem[high] = L.elem[low];
124     }
125     L.elem[low] = L.elem[0];
126     return low;
127 }
128
129 /**
130  * 算法10.7, 递归法对顺序表L中的子序列L.r[low...high]作快速排序
131  */
132 void QSort(SqList &L, int low, int high)
133 {
134     int pivotloc;
135     if (low < high) {
136         pivotloc = Partition(L, low, high);
137         QSort(L, low, pivotloc - 1);
138         QSort(L, pivotloc + 1, high);
139     }
140 }
141
142 /**
143  * 算法10.8, 快速排序
144  */
145 void QuickSort(SqList &L) { QSort(L, 1, L.length); }

```

快速排序测试程序：（测试程序代码放在实现代码下方即可）

```

1  int main()
2  {
3      int arr[8] = {33, 12, 75, 0, 49, 67, 8, 999};
4      SqList list1;
5
6      InitList_Sq(list1);
7      for (int i = 1; i < 9; i++) // 将arr数组中的8个元素插入顺序表中
8          ListInsert_Sq(list1, i + 1, arr[i - 1]);
9      printf("初始线性表为: \n");
10     ListTraverse_Sq(list1, visit_display_Sq);
11
12     QuickSort(list1);
13     printf("\n排序后的线性表为: \n");
14     ListTraverse_Sq(list1, visit_display_Sq);
15
16     return 0;
17 }

```

测试程序运行结果为：

初始线性表为：

33 12 75 0 49 67 8 999

排序后的线性表为：

0 8 12 33 49 67 75 999

Author: XF

Finished time: 2019/12/8