**《数据结构》实验报告**

**姓名： 杨孟衡 学号： 8002118240**

**班级： 软件工程1809班 专业： 软件工程**

**报告日期： 2019 年 9 月 26 日**

**实验二 线性表的分离与合并**

**一、问题描述与分析**

1. 设某线性数据元素类型为整型，以线性顺序表为存储结构。试编程实现：

1. 编写程序，对由键盘输入的有n个元素的线性表，按照奇数存放数组一，偶数存放数组二的方式，输出其分离前和分离后的所有元素
2. 已知线性表L1和L2中数据均由小到大排序，请用尽可能快的方法将L1和L2中的数据合并到L3中，使数据在L3中按升序排列。
3. 编写程序，对由键盘输入两个线性表，输出其合并后新队列的所有元素
4. 用静态顺序结构实现

2.分析

先创建一个线性空表，提示输入数据元素，然后对实验数据进行分离与合并处理，最后提示输出数据元素。

**二、数据结构与算法设计**

1.数据结构：

静态线性表采用基本数据类型数组作为基本元素构成，搭配一个整型变量int作为数据长度（size）。代码如下：

#define MAXSIZE 200

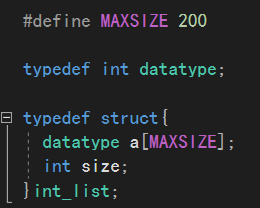
typedef int datatype;

typedef struct{

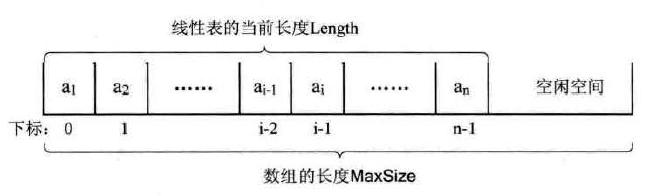
datatype a[MAXSIZE];

int size;

}int\_list;



线性表图示如下：



2.算法

⑴ 快速排序

算法思想：将数据划分区间处理，采取分治的思想，首先根据入参数据左右边界确认中间元素的下标，

采用中间元素作为标杆元素,先从数据左边界开始比较，若找到或没有找到（与数据本身比较）用循环变量i标记下标，再从数据右边界开始比较，若找到或没有找到（与数据本身比较）用循坏变量j标记下标。

若i < j，说明标杆元素左侧有大于等于它的元素，右侧有小于等于它的元素，所以交换i，j两者下标的元素。

若i >= j，说明标杆元素左侧已经没有大于等于它的元素了，右侧也没有小于等于它的元素，所以这时break退出循环，退出循环后，根据i，j值，划分区间，再进行后面的排序，直到最终区间里只有一个元素，这样就排好了。

关于快速排序是升序还是降序，就看循环找标杆元素左右侧的元素的判断条件，若是左侧找大于等于，右侧小于等于，那么是升序，反之降序

**三、算法复杂度分析**

⑴ 线性表的合并（快速排序）

算法复杂度为O（nlogn），这是一种不稳定的算法。

(2)线性表的分离

算法复杂度为O（n）

**四、测试计划**

1.编写目的

加深对数据结构线性表的顺序表存储结构和部分排序算法思想的理解。

2.开发及运行环境

Visual Studio 2017

3.代码流程

（1）线性表的分离：

提示用户输入分利用的数组元素，以0结束，读入临时数组存放；

判断是奇数还是偶数，存放对应的分数组；

打印分离后的两个两个数组线性表；

（2）线性表的合并：

在测试函数里输入线性表的数据，存放在数组里作为合并函数的入参；

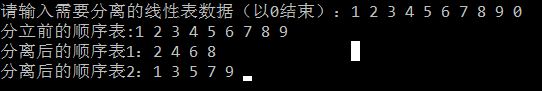
初始化线性表，先判断是否是两种特殊情况，第一个数组的最后一个元素小于第二个数组的第一个元素，以及第二个数组的最后一个元素小于第一个数组的第一个元素，若是这两种特殊情况那么直接拼接数组即可；

若不是特殊情况，那么先将两个数组拼接成一个数组，然后利用快速排序算法对数据进行快速排序；

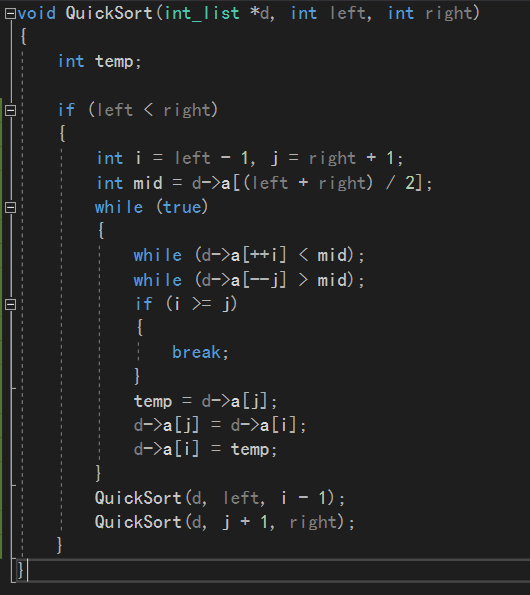
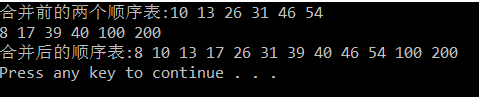
打印合并后的数组

4.测试截图

（1）线性表的分离



（2）线性表的合并（快速排序）



**五、源程序**

**<int\_list.h>:**

#include <stdio.h>

#include <stdlib.h>

#define MAXSIZE 200

typedef int datatype;

typedef struct{

datatype a[MAXSIZE];

int size;

}int\_list;

void Initiate(int\_list \*d);

int Empty(int\_list \*d);

void Display(int\_list \*d);

int Find(int\_list d, datatype D);

void Insert(int\_list \*d, datatype x, int position);

void Delete(int\_list \*d, int position);

int\_list\* Assemble(int\_list \*d);

void Seperate(int\_list \*d);

//初始化函数，用于初始化列表

void Initiate(int\_list \*d)

{

d->size = 0;

}

//顺序表尾部插入元素，比如数组增加size，尾部插入新的元素

void Input(int\_list \*d, int D)

{

if(d->size == MAXSIZE)

{

printf("顺序表已满！");

exit(-2);

}

d->size++;

d->a[d->size - 1] = D;

}

//判断顺序表是否为空，若为空返回-1，否则返回当前顺序表的size值

int Empty(int\_list \*d)

{

return (d->size == 0?-1:d->size);

}

//显示函数，用于显示列表所有元素值

void Display(int\_list \*d)

{

if(!d->size)

{

printf("顺序表为空！");

}

else

{

for(int i = 0 ; i < d->size; i++)

{

printf("%d ", d->a[i]);

}

}

printf("\n");

}

//查找函数，查找顺序表中的某个元素,若没有找到返回-3，若找到返回下标值

int Find(int\_list d, datatype D)

{

int i = 0;

while(i < d.size && d.a[i] != D)i++;

return (i < d.size?i:-1);

}

//线性表的插入操作，在顺序表的某个position插入某个元素

void Insert(int\_list \*d, datatype x, int position)

{

if (d->size == MAXSIZE)

{

printf("\n顺序表已满！无法插入！");

exit(-3);

}

if (position < 0 || position > d->size)

{

printf("\n指定的插入位置不存在！");

exit(-3);

}

for (int i = d->size; i > position; i--)

{

d->a[i] = d->a[i - 1];

}

d->a[position] = x;

d->size++;

}

//线性表的删除操作,在顺序表的某个position删除某个元素

void Delete(int\_list \*d, int position)

{

if (d->size == 0)

{

printf("\n顺序表为空！无法删除！");

exit(-4);

}

if (position < 0 || position > d->size)

{

printf("\n指定的删除位置不存在！");

exit(-4);

}

for (int i = position; i < d->size - 1; i++)

{

d->a[i] = d->a[i + 1];

}

d->size--;

}

//线性表的分离操作，将参数顺序表划分为两个独立的顺序表（奇数，偶数），再合并为一个顺序表

void Seperate(int\_list \*d)

{

int\* temp[2];

int i = 0;

int j = 0;

int k = 0;

Initiate(d);

printf("请输入需要分离的线性表数据（以0结束）：");

while (1)

{

scanf("%d", &d->a[i]);

if (d->a[i] == 0)

{

break;

}

i++;

d->size++;

}

temp[0] = (int\*)malloc(sizeof(d));

temp[1] = (int\*)malloc(sizeof(d));

for (int i = 0; i < d->size && d->a[i] != 0; i++)

{

if (d->a[i] % 2 == 0)

{

temp[0][j] = d->a[i];

j++;

}

else

{

temp[1][k] = d->a[i];

k++;

}

}

printf("分立前的顺序表:");

Display(d);

printf("分离后的顺序表1：");

for (int i = 0; i < j; i++)

{

printf("%d ", temp[0][i]);

}

printf("\n分离后的顺序表2：");

for (int i = 0; i < k; i++)

{

printf("%d ", temp[1][i]);

}

}

/\*快速排序

算法思想：将数据划分区间处理，采取分治的思想，首先根据入参数据左右边界确认中间元素的下标，

采用中间元素作为标杆元素,先从数据左边界开始比较，若找到或没有找到（与数据本身比较）用循环变量i标记下标，

再从数据右边界开始比较，若找到或没有找到（与数据本身比较）用循坏变量j标记下标。

若i < j，说明标杆元素左侧有大于等于它的元素，右侧有小于等于它的元素，所以交换i，j两者下标的元素。

若i >= j，说明标杆元素左侧已经没有大于等于它的元素了，右侧也没有小于等于它的元素，所以这时break退出循环，

退出循环后，根据i，j值，划分区间，再进行后面的排序，直到最终区间里只有一个元素，这样就排好了。

关于快速排序是升序还是降序，就看循环找标杆元素左右侧的元素的判断条件，

若是左侧找大于等于，右侧小于等于，那么是升序，反之降序

\*/

void QuickSort(int\_list \*d, int left, int right)

{

int temp;

if (left < right)

{

int i = left - 1, j = right + 1;

int mid = d->a[(left + right) / 2];

while (true)

{

while (d->a[++i] < mid);

while (d->a[--j] > mid);

if (i >= j)

{

break;

}

temp = d->a[j];

d->a[j] = d->a[i];

d->a[i] = temp;

}

QuickSort(d, left, i - 1);

QuickSort(d, j + 1, right);

}

}

//线性表的合并操作，将参数顺序表数据中的两个数据按升序排序重新生成一个新数据,将这个新数据快速排序后，返回这个新数据的首地址

int\_list\* Assemble(int\_list \*d)

{

int\_list\* temp = (int\_list\*)malloc(sizeof(int\_list));

int loop = d[0].size + d[1].size;

Initiate(temp);

if (d[0].a[d[0].size - 1] < d[1].a[0])

{

for (int i = 0; i < loop; i++)

{

if (i < d[0].size)

{

Input(temp, d[0].a[i]);

}

else Input(temp, d[1].a[i - d[0].size]);

}

d = temp;

free(temp);

return d;

}

if (d[1].a[d[1].size - 1] < d[0].a[0])

{

for (int i = 0; i < loop; i++)

{

if (i < d[1].size)

{

Input(temp, d[1].a[i]);

}

else Input(temp, d[0].a[i - d[1].size]);

}

d = temp;

free(temp);

return d;

}

for (int i = 0; i < loop; i++)

{

if (i < d[0].size)

{

Input(temp, d[0].a[i]);

}

else Input(temp, d[1].a[i - d[0].size]);

}

d = temp;

QuickSort(d, 0, d->size - 1);

return d;

}

**<random.cpp>:**

#include <iostream>

#include"int\_list.h"

using namespace std;

int main()

{

int\_list dlist[2];

int\_list\* li;

Initiate(&dlist[0]);

Initiate(&dlist[1]);

Input(&dlist[0], 10);

Input(&dlist[0], 13);

Input(&dlist[0], 26);

Input(&dlist[0], 31);

Input(&dlist[0], 46);

Input(&dlist[0], 54);

Input(&dlist[1], 8);

Input(&dlist[1], 17);

Input(&dlist[1], 39);

Input(&dlist[1], 40);

Input(&dlist[1], 100);

Input(&dlist[1], 200);

printf("合并前的两个顺序表:");

Display(&dlist[0]);

Display(&dlist[1]);

li = Assemble(dlist);

printf("合并后的顺序表:");

Display(li);

Seperate(li);

system("pause");

return 0;

}