**《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 顺序表的调整**

**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，对由键盘输入线性表，输出调整后的顺序表。
2. 无论输入还是输出，要给出适当的提示信息
3. 用静态顺序结构实现

2.分析

请编写一个算法函数ｐａｒｔｉｏｎ（ｓｅｑｕｅｎｃｅ＿ｌｉｓｔ　＊L），尽可能快地将顺序表\*L中的所有奇数调整到表的左边，所有偶数调整到表的右边，并分析算法的时间复杂度。

先创建一个顺序空表，提示输入数据元素，然后对该顺序表调整处理，最后提示输出该顺序表调整之后的元素。

**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

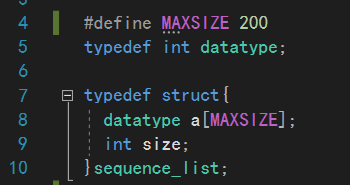
typedef int datatype;

typedef struct{

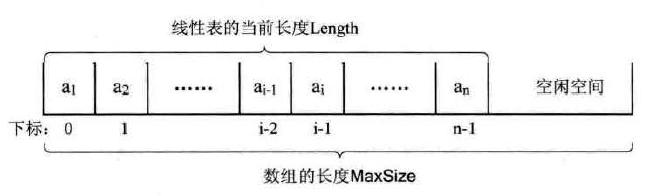
datatype a[MAXSIZE];

int size;

}sequence\_list;



线性表图示如下：



2.算法

⑴ 顺序比较法：

将需要调整处理的顺序表元素一个一个取出，做求余操作，根据求余操作值采取不同的存放方式，若是奇数则放在临时顺序表的左侧，若是偶数则放在临时顺序表临时顺序表额的右侧。

**三、算法复杂度分析**

⑴ 线性表的合并（顺序比较法）

算法复杂度为O（n），这是一种稳定的算法

**四、测试计划**

1.编写目的

加深对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）顺序表的初始化：

提示用户输入顺序表的数组元素，以0结束，读入数组li1存放；

（2）顺序表的顺序比较：

顺序取出顺序表的元素比较处理

（3）打印顺序表的元素：

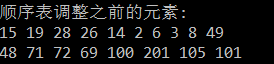
先打印用户输入的顺序表li1，再打印调整处理后的temp

4.测试截图

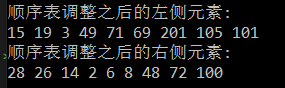
（1）顺序表的初始化



（3）顺序表的调整之前的元素：



（4）调整之后的顺序表元素：



**五、源程序**

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

#include <stdio.h>

#include <stdlib.h>

#define MAXSIZE 200

typedef int datatype;

typedef struct{

datatype a[MAXSIZE];

int size;

}sequence\_list;

void Initiate(sequence\_list \*li);

void Input(sequence\_list \*li);

int deleteAllX(sequence\_list \*li, datatype x);

void Inputfromfile(sequence\_list \*li, char \* f);

void Print(sequence\_list \*li);

int Find(sequence\_list \*li, datatype x);

void Inter(sequence\_list \* li1, sequence\_list \* li2, sequence\_list \* li3);

//初始化顺序列表

void Initiate(sequence\_list \*li)

{

li->size = 0;

}

//列表插入元素

void Input(sequence\_list \*li)

{

datatype x;

Initiate(li);

printf("\n输入插入列表的元素（以0结束）：\n");

scanf("%d", &x);

while (x)

{

li->a[li->size++] = x;

scanf("%d", &x);

}

}

//从文件中读入数据,第一个入参是存放数据的容器，第二个入参是需要打开的文件名

void Inputfromfile(sequence\_list \*li, char \* f)

{

FILE \* fp=fopen(f, "r");

Initiate(li);

if (fp)

{

while (!feof(fp))

{

fscanf(fp, "%d", &li->a[li->size++]);

}

fclose(fp);

}

}

//打印顺序列表中的所有元素值

void Print(sequence\_list \*li)

{

printf("\n");

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

{

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

//每打印10个元素后，输出一个换行符，整理格式

if ((i+1)%10==0) printf("\n");

}

printf("\n");

}

//查找顺序列表中的某个元素

int Find(sequence\_list \*li, datatype x)

{

int i = 0;

while(i < li->size && li->a[i] != x) i++;

return (i < li->size? i: -1);

}

//删除顺序列表中的某个元素

int deleteAllX(sequence\_list \*li, datatype x)

{

int i = 0, j = 0;

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

{

if (li->a[i] != x)

li->a[j++] = li->a[i];

}

li->size = j;

return 0;

}

/\*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(sequence\_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);

}

}

/\*折半查找（二分查找），用给定值k先与中间结点的关键字比较，中间结点把线形表分成两个子表，

若相等则查找成功；若不相等，再根据k与该中间结点关键字的比较结果确定下一步查找哪个子表，

这样循环进行，直到查找到或查找结束发现表中没有这样的结点。

\*/

int BinarySearch(int a[], int value, int n)

{

int low, high, mid;

low = 0;

high = n - 1;

while (low <= high)

{

mid = (low + high) / 2;

if (a[mid] == value)

return mid;

if (a[mid] > value)

high = mid - 1;

if (a[mid] < value)

low = mid + 1;

}

return -1;

}

//顺序表的交集，实现两个顺序列表(li1, li2)之间求交集，之后将交集存放在li3中

void Inter(sequence\_list \* li1, sequence\_list \* li2, sequence\_list \* li3)

{

int flag = 0;

Initiate(li3);

//先利用快速排序将两个顺序列表升序排序

QuickSort(li1, 0, li1->size - 1);

QuickSort(li2, 0, li2->size - 1);

//利用折半查找的方法，对两个顺序列表查找相同的元素,若相同则存储于li3中

for (int i = 0; i < li1->size; i++)

{

flag = BinarySearch(li2->a, li1->a[i], li2->size);

if (flag != -1)

{

li3->a[li3->size++] = li1->a[i];

}

}

//打印求交集前的各顺序列表元素，以及求交集后的顺序列表元素

printf("求交集之前的两个顺序列表元素:");

Print(li1);

Print(li2);

printf("求交集之后的顺序列表元素:");

Print(li3);

}

//顺序表的调整，将顺序表\*li中的所有奇数调整到表的左边，所有偶数调整到表的右边

void Partion(sequence\_list\* li)

{

sequence\_list temp;

int j = 199;

Initiate(&temp);

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

{

switch (li->a[i] % 2)

{

case 1:

{

temp.a[temp.size++] = li->a[i];

break;

}

case 0:

{

temp.a[j--] = li->a[i];

break;

}

default: printf("算法出错！"); exit(-1);

}

}

printf("顺序表调整之前的元素:");

Print(li);

printf("顺序表调整之后的左侧元素:");

Print(&temp);

printf("顺序表调整之后的右侧元素:\n");

for (int i = 199; i > j; i--)

{

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

if ((i - 1) % 10 == 0) printf("\n");

}

printf("\n");

}

**<random.cpp>:**

#include <iostream>

#include"sequlist\_excise.h"

using namespace std;

int main()

{

/\*

bool flag;

char\_stack cs;

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);

flag = IsPlalindrome();

if (!flag)

{

printf("此字符串不是回文字符串！\n");

}

else printf("此字符串是回文字符串！\n");

CalPostFix(&cs);

\*/

sequence\_list li1;

/\*sequence\_list li2;

sequence\_list li3;

Input(&li1);

Input(&li2);

Inter(&li1, &li2, &li3);\*/

Input(&li1);

Partion(&li1);

system("pause");

return 0;

}