**暨南大学本科实验报告专用纸**

课程名称 算法分析与设计实验 成绩评定

实验项目名称 k个最接近中位数的数 指导教师 李展 实验地点 N116

实验项目编号 003 实验项目类型 综合型

学生姓名 张瑞鹏 学号 2020101124

学院 信息科学技术学院 系 计算机 专业 计科

实验时间 2022 年 月 日 午～ 月 日 下 午

**（一）问题描述**

给定由n个互不相同的数组成的集合S以及正整数k<=n，试设计一个O(n)时间算法找出S中最接近S的中位数的k个数。

备注：此处“最接近”指数值大小接近，而不是排序位置接近。

（提示：注意时间复杂度要求O(n)，不要排序，排序就超了O(n)，可以利用书上P26页例2.6 线性时间选择例子）

**（二）算法思路**（用文字简单说明）

**整体思路：想尽办法避免排序**

**首先套用书上的线性时间选择的算法，这个是完成此算法的前提。**

**分奇数偶数讨论算出调用线性时间算法select算出找出中位数，并且此时中位数左边的元素的值都是小于中位数的，右边的值都是大于中位数的。这一步的时间复杂度是O（n）**

**需要选取数值上最靠近中位数的数，有可能全部来自左边，也可能全部来自右边，所以肯定不能直接取左边与右边k/2个数字。我通过构造一个新的数组，定义一种struct{int num,int val}；Num表示的是该数字的大小，val是该数字与中位数的距离（差值）。**

**把中位数左边的k个数字放进辅助数组当中，把中位数的右边的k个数组也全部放进辅助数组当中。这一步的时间复杂度是O（2k）,2k<=2n**

**再对新的数组的调用一次select算法，于是就找出最靠近中位数的k个元素。显然这一步的时间复杂度是O（k）**

**算法的时间复杂度 =** O(n+k+k+2k+2k) = O(n)

**细节：**

**1.对于最后一次调用select函数，不在是对整型变量的排序，是对结构体调用，在此使用了函数重载来实现这个步骤。对于所有涉及select调用的函数也要重载。**

**2.对于左边取k个，右边取k个这种做法，是防止离中位数最近的数字全部都在一边，因此这样做。但是考虑到有可能数组越界（k取得比较大），万一越界直接把边界用数组起始和结尾代替。**

**（三）算法实施步骤和流程**（伪代码/流程图等方式描述）

线性时间选择的算法：

int patition(int\* a, int p, int r, int x) {//x是一个位置

if (p == r)return r;

int temp = a[x];//把中枢放在temp中

a[x] = a[p];//相当于调换位置

while (p < r) {

while (p < r && a[r] >= temp) r--;

a[p] = a[r];

while (p < r && a[p] <= temp) p++;

a[r] = a[p];

}

a[r] = temp;

return r;

}

void QSort(int a[], int p, int r, int x) {

int min = patition(a, p, r, x);

if (min - 1 > p) QSort(a, p, min - 1, p);

if (min + 1 < r) QSort(a, min + 1, r, min + 1);

}

int select(int a[], int p, int r, int k)//k不是index，表示的是第几的意思

{

if (r - p <= 10) {

QSort(a, p, r, p);

return a[p + k - 1];//递归的出口

}

for (int i = 0; i <= (p - r - 4) / 5; i++)//余3以下的全都不要了，余4还要

{

QSort(a, p + i \* 5, p + i \* 5 + 4, p + i \* 5);

int temp = a[p + i];//用于交换

a[p + i] = a[p + i \* 5 + 2];

a[p + i \* 5 + 2] = temp;//交换完成

}

select(a, p, p + (p - r - 4) / 5, (p - r + 6) / 10);

int i = patition(a, p, r, (p - r + 6) / 10); int j = i - p + 1;

if (j == k)return a[i];

else if (k < j) return select(a, p, i - 1, k);

else return select(a, i + 1, r, k - j);

}

寻找第k小的算法：  
int\* findkclose(int\* arry,int start,int end, int k)//传递过来数组的起点，终点，k的个数

{

int len = end - start + 1;//计算出传递数组的长度

if(k>=len) return arry;

int mid,left ,right,i,j;;//place表示当前应该填写第几个位置

help\* result = new help[2\*k];//返回值数组

left = len / 2; right = len / 2+1;

if (len % 2 == 0) {//条件分支，表示偶数

int leftval = select(arry, start, end, left);//左值右值已经完成

int rightval = select(arry, start + right-1, end, 1);

mid = (leftval+rightval)/2;

}

else if (len % 2 == 1) {//表示奇数

mid = select(arry, start, end, len/2+1);

right = len / 2 + 2;

}//计算出中位数

printf("算出来的中位数是%d\n",mid);

int lboundary= 1;int rboundary= len;//定义左边界和右边界

if(left-k+1>1){

lboundary = left-k+1;

select(arry,start,start + left-1,left-k+1);

}

if(right+k-1<len){

rboundary = right+k-1;

select(arry, start + right - 1, end, k);

}

int resultlen = 0 ;

for(i = 0,j = start+lboundary-1;j <=start + left-1;i++,j++){

result[i].num = arry[j];

result[i].val = mid-arry[j];

resultlen +=1;

}

for(j = start+right-1;j<=start + rboundary-1;i++,j++){

result[i].num = arry[j];

result[i].val = arry[j] - mid;

resultlen +=1;

}

if(len % 2 == 1) {

result[i].num = mid;

result[i].val = 0;

resultlen+=1;

}

select(result,0,resultlen-1,k);

int\* outcome = new int(k);

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

outcome[i] = result[i].num;

}

return outcome;

}

**（四）源代码**（通过了编译运行的正确程序）

//由于涉及到函数重载，代码量显得比较多

#include<stdio.h>

#include<iostream>

using namespace std;

struct help{

int val;

int num;

};

int patition(int\* a, int p, int r, int x) {//x是一个位置

if (p == r)return r;

int temp = a[x];//把中枢放在temp中

a[x] = a[p];//相当于调换位置

while (p < r) {

while (p < r && a[r] >= temp) r--;

a[p] = a[r];

while (p < r && a[p] <= temp) p++;

a[r] = a[p];

}

a[r] = temp;

return r;

}

void QSort(int a[], int p, int r, int x) {

int min = patition(a, p, r, x);

if (min - 1 > p) QSort(a, p, min - 1, p);

if (min + 1 < r) QSort(a, min + 1, r, min + 1);

}

int select(int a[], int p, int r, int k)//k不是index，表示的是第几的意思

{

if (r - p <= 10) {

QSort(a, p, r, p);

return a[p + k - 1];//递归的出口

}

for (int i = 0; i <= (p - r - 4) / 5; i++)//余3以下的全都不要了，余4还要

{

QSort(a, p + i \* 5, p + i \* 5 + 4, p + i \* 5);

int temp = a[p + i];//用于交换

a[p + i] = a[p + i \* 5 + 2];

a[p + i \* 5 + 2] = temp;//交换完成

}

select(a, p, p + (p - r - 4) / 5, (p - r + 6) / 10);

int i = patition(a, p, r, (p - r + 6) / 10); int j = i - p + 1;

if (j == k)return a[i];

else if (k < j) return select(a, p, i - 1, k);

else return select(a, i + 1, r, k - j);

}

//函数重载

int patition(help\* a, int p, int r, int x) {

if (p == r)return r;

help temp;

temp = a[x];//把中枢放在temp中

a[x] = a[p];//相当于调换位置

while (p < r) {

while (p < r && a[r].val >= temp.val) r--;

a[p] = a[r];

while (p < r && a[p].val <= temp.val) p++;

a[r] = a[p];

}

a[r] = temp;

return r;

}

void QSort(help a[], int p, int r, int x) {

int min = patition(a, p, r, x);

if (min - 1 > p) QSort(a, p, min - 1, p);

if (min + 1 < r) QSort(a, min + 1, r, min + 1);

}

int select(help a[], int p, int r, int k)//k不是index，表示的是第几的意思

{

if (r - p <= 10) {

QSort(a, p, r, p);

return a[p + k - 1].val;//递归的出口

}

for (int i = 0; i <= (p - r - 4) / 5; i++)//余3以下的全都不要了，余4还要

{

QSort(a, p + i \* 5, p + i \* 5 + 4, p + i \* 5);

help temp = a[p + i];//用于交换

a[p + i] = a[p + i \* 5 + 2];

a[p + i \* 5 + 2] = temp;//交换完成

}

select(a, p, p + (p - r - 4) / 5, (p - r + 6) / 10);

int i = patition(a, p, r, (p - r + 6) / 10); int j = i - p + 1;

if (j == k)return a[i].val;

else if (k < j) return select(a, p, i - 1, k);

else return select(a, i + 1, r, k - j);

}

int\* findkclose(int\* arry,int start,int end, int k)//传递过来数组的起点，终点，k的个数

{

int len = end - start + 1;//计算出传递数组的长度

if(k>=len) return arry;

int mid,left ,right,i,j;;//place表示当前应该填写第几个位置

help\* result = new help[2\*k];//返回值数组

left = len / 2; right = len / 2+1;

if (len % 2 == 0) {//条件分支，表示偶数

int leftval = select(arry, start, end, left);//左值右值已经完成

int rightval = select(arry, start + right-1, end, 1);

mid = (leftval+rightval)/2;

}

else if (len % 2 == 1) {//表示奇数

mid = select(arry, start, end, len/2+1);

right = len / 2 + 2;

}//计算出中位数

printf("算出来的中位数是%d\n",mid);

int lboundary= 1;int rboundary= len;//定义左边界和右边界

if(left-k+1>1){

lboundary = left-k+1;

select(arry,start,start + left-1,left-k+1);

}

if(right+k-1<len){

rboundary = right+k-1;

select(arry, start + right - 1, end, k);

}

int resultlen = 0 ;

for(i = 0,j = start+lboundary-1;j <=start + left-1;i++,j++){

result[i].num = arry[j];

result[i].val = mid-arry[j];

resultlen +=1;

}

for(j = start+right-1;j<=start + rboundary-1;i++,j++){

result[i].num = arry[j];

result[i].val = arry[j] - mid;

resultlen +=1;

}

if(len % 2 == 1) {

result[i].num = mid;

result[i].val = 0;

resultlen+=1;

}

select(result,0,resultlen-1,k);

int\* outcome = new int(k);

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

outcome[i] = result[i].num;

}

return outcome;

}

int main(void)

{

int a[12] = { 42,33,12,5,7,234,1,0,9,8,23,-1};

int k = 5;

cout<<"输入序列：";

for (int i = 0; i < 12; i++) {

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

}

printf("\n");

int\* result = findkclose(a,0,11,5);

printf("打印%d个最近接近中位数的数字:",k);

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

printf("%d ",result[i]);

}

printf("\n");

/\*

int result = select(a, 0, 13, 1);printf("the result is:%d\n", result);\*/

QSort(a, 0, 11, 5);

cout<<"\n\n\n\n\n算法之外的步骤，方便对答案\n";

printf("观察序列:");

for (int i = 0; i < 12; i++) {

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

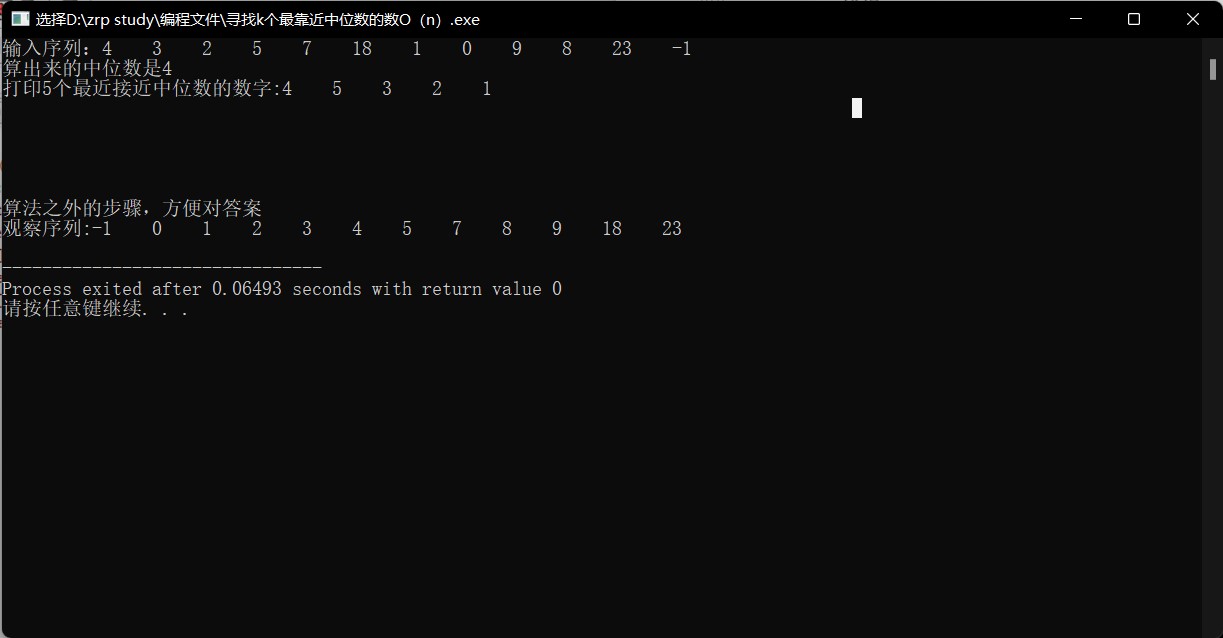
}

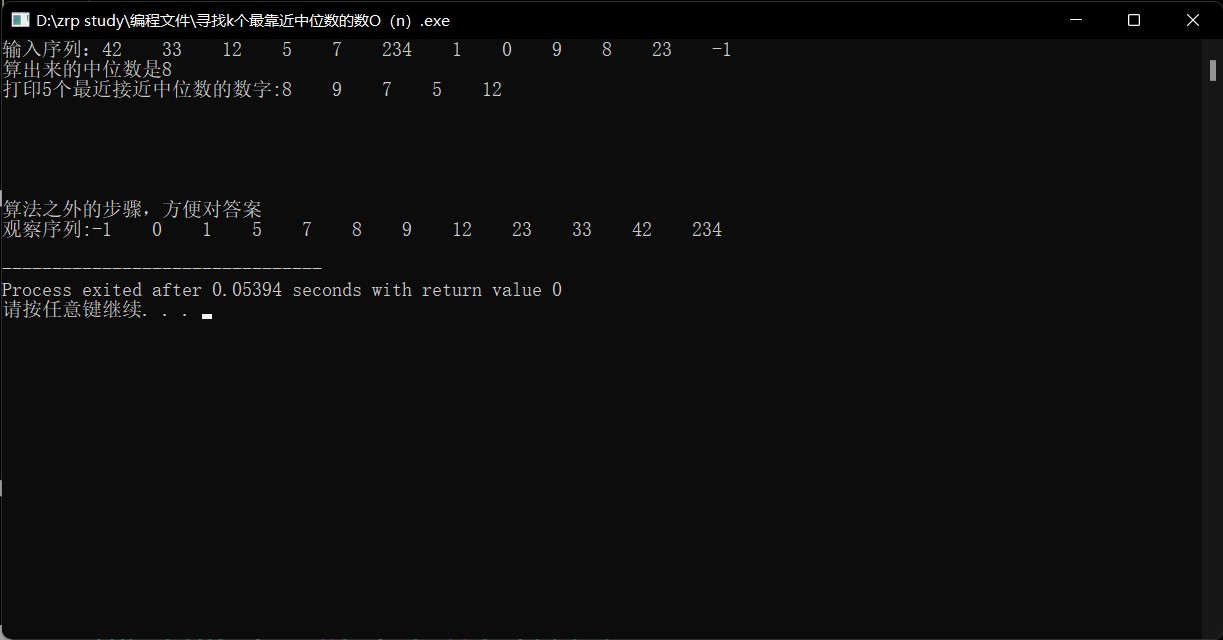
printf("\n");

return 0;

}

**（五）测试结果**（至少有两个以上算例及程序运行结果，截图贴进实验报告）



****

**解释：后面排序输出是方便老师检查还有自己对答案，算法本身不涉及排序**

**（六）实验总结**（至少三句话，可以写复杂度分析、遇到问题、可能的改进措施、心得体会等）

1.本次实验最终成果是通过不断改进最终得到的。一开始是仅完整了线性时间选择但是没有头绪。后来是想到了先调用一次线性时间选择找到中位数，然后再通过左右双指针一个一个比较然后放入数组，但是发现这样会出现k\*O（n）的时间复杂度，万一取值大的话，就不在满足要求。在上述版本已经完成之后才发现有这个漏洞，于是才重新动工完成现在这个算法。

2.总结一下算法的时间复杂度：

第一次找中位数O(n)

然后截取左边k个数O（k），截取右边k个数O（k）

把2k个数字与中位数的差值放进数组

对辅助数组调用线性时间选择函数O(2k)

O(n+k+k+2k+2k) = O(n)

3.用到了重载函数，因此某一部分代码直接翻倍，显得代码量很大，这个暂时还没想到改进方法。

4.算法可能不能一下子立刻有思路，慢慢做可能就会有新发现。

**暨南大学本科实验报告专用纸(附页)**