SHANGHAI JIAO TONG UNIVERSITY

多关键字排序

吕艺

517021910745

December 14, 2018

1 需求分析

1. 本演示文件的主要需求为多关键字排序。具体背景为对于学生的高考成绩排序，除了对于总分排序之外，不同专业对于单科分数的要求不同，故除了排序总分外，还需实现总分相同时按用户提出的单科分数排序顺序排出考生的次序。首先先生成随机数 num, 代表学生个数。根据每组随机数个数不超过 10000 个的要求，本演示文件中选择了每组随机数的个数在 0 1000 个。之后再分别生成每个学生的六门成绩（成绩在 0～100 之间）。

2. 演示文件在用户输入对于六科成绩的排序之后，自动将排序后的总分和单科成绩按低到高打印在屏幕上。

3. 程序执行的命令包括:

1. 根据系统时间生成学生名数。

2. 根据系统时间和事先生成的学生名数生成他们的六科成绩

3. 根据总分和用户定义的单科成绩顺序对学生成绩进行排序

4. 将排序后的总分和单科成绩按低到高打印在屏幕上

4. 测试数据

本演示文件中用到的测试数据均为随机生成的成绩

2 概要设计

本演示文件中采用的排序方法为快速排序，在分数相同时依次按用户自定义的顺序比较，最后完成排序。本演示文件中主要使用了 *list* 类来保存学生的成绩信息。

成绩记录类 *list*

数据对象：int num = 0;　　int scores[6];

　　student * records;　int priority[6];

私有结构体：

```
struct student {
        int score[6];
        int total = 0;
        student() {
        for(int i = 0; i < 6; i++)
         score[i] = rand() % 101;
        total += score[i]; }
};
```

基本操作：list();

操作条件：*list* 还未被初始化

操作结果：随机生成数初始化 *list* 类

~ list();

初始条件：*list* 类已被初始化

操作结果：输出总分排序

void quickSort();

初始条件：*list* 类已经被初始化

操作结果：将 *list* 类中的成绩按要求排序

void quickSort(int low, int high);

操作条件：*list* 类已被初始化

操作结果：递归实现成绩的排序

bool Compare(student A, student B);

操作条件：*list* 类已经被初始化

操作结果：返回学生 A 和学生 B 的成绩比较结果

bool Compare(student A, student B,int level);

操作条件：*list* 类已经被初始化

操作结果：递归比较学生 A 和学生 B 的成绩比较结果

int divide(int low, int high);

操作条件：*list* 类已经被初始化

操作结果：使数组 low 位置的值大于其左边所有的数，小于右边所有的数

2. 本程序包括两个模块：

1) int main() {
 list scoreList;
 scoreList.quickSort();
 return 0;}

2) *list* 单元模块--实现学生成绩的初始化和排序

3 详细设计

1）*list* 单元模块

```
class list{
    struct student{                    //私有结构体
        int score[6];                  //六科成绩，分别对应课程0，课程1...
        int total = 0;                 //六科成绩总分

        student() {                    //构造函数，随机生成六科成绩并计算出总分
            for(int i = 0; i < 6; i++)
            { score[i] = rand() % 101;
            total += score[i]; }
        }
    };

    int num = 0;                       //学生名数
    student * records;                 //学生成绩汇总矩阵
```

```
    int priority[6];                    //单科成绩优先级，数组中序号最小的学科优先级最高

public:
    list();                             //构造函数
    ~list();                            //析构函数

public:
    void quickSort();                                   //快速排序外部接口
    void quickSort(int low, int high);                  //快速排序内部接口（递归实现）
    bool Compare(student A, student B);                 //两学生总分比较
    bool Compare(student A, student B,int level);       //两学生总分相同时成绩比较（递归实现）
    int divide(int low, int high);                      //根据数组中low位置对应的值排序
};


list::list()
{
    srand(unsigned((time)(NULL)));                                  //生成学生名数
    num = rand() % 100;
    cout<<"Total␣number␣of␣records:"<<num<<endl;
    records = new student[num];
    cout<<"Please␣input␣the␣subject␣place␣by␣their␣priority"<<endl;
    for(int i = 0; i < 6; i++)                                      //用户自定义输入单科成绩的优先级
        cin>>priority[i];
}


list::~list()
{   cout<<"Sorted␣total␣score!"<<endl;                             //输出排序后的总分和单科成绩
    for(int i = 0; i < num; i++)
    {cout<<records[i].total<<"␣";
    for(int j = 0;j < 6;j++)
        cout<<records[i].score[j]<<"␣";
    cout<<endl;}
}


bool list::Compare(student A, student B, int level)
{                                                                   //总分及各门成绩相同
    if (level == 5 && A.score[priority[level]] == B.score[priority[level]])
        return true;
    if (A.score[priority[level]] > B.score[priority[level]])        //A单科高，返回值为true
        return true;
    if (A.score[priority[level]] < B.score[priority[level]])        //B单科高，返回值为false
        return false;
    return Compare(A, B, level + 1);
}
```

```
bool list::Compare(student A, student B)
{
    if (A.total > B.total)                  //A总分高，返回值为true
        return true;
    if (A.total < B.total)                  //A总分低，返回值为false
        return false;
    else                                    //总分相同，比较单科成绩
       return Compare(A, B, 0);
}


int list:: divide(int low, int high)
{
    student tmp = records[low];                              //将分数按tmp的值排序

    while(low != high) {
        while(low < high && Compare(records[high],tmp) )
            high --;
        if(low < high)
            records[low] = records[high];

        while(low < high && Compare(tmp, records[low]))
            low ++;
        if(low < high)
            records[high] = records[low];
    }

    records[low] = tmp;
    return low;
}


void list::quickSort(int low, int high)          //快速排序递归实现
{
    if(low >= high) return;
    int mid = divide(low, high);
    quickSort(low, mid - 1);
    quickSort(mid + 1, high);
}


void list::quickSort()                           //快速排序外部接口
{
    quickSort(0, num - 1);
}
```

2）主函数模块

```
int main() {
    list scoreList;              //随机生成学生成绩并由用户自定义单科成绩重要性
    scoreList.quickSort();       //对学生成绩进行排序并输出
    return 0;
}
```

4 调试分析

1. 一开始本演示文件仅设计了 *student* 类，之后生成数组来存放学生，这导致程序的可移植性较低，为了更好地凸显面向对象的优势，程序中又设计了 *list* 类，将 *student* 类作为 *list* 类的私有类，从而使主函数更加简洁。

2. 本演示文件一开始采用插入排序法，但由于数据是随机生成的，比较次数和移动次数过多，最后选择采用快速排序的方法。

3. 算法的复杂度分析

1）时间复杂度由于采用数组的形式存储学生的成绩，各种操作的算法复杂度比较合理。quickSort 函数的时间复杂度是 $O(n\log n)$, 函数 Compare,student 类的初始化和析构函数的时间复杂度均为 $O(1)$，而 divide 函数的时间复杂度是 $O(n)$

quickSort 函数通过递归分别使左边右边有序，因此这个函数的时间复杂度为 $O(n\log n)$。

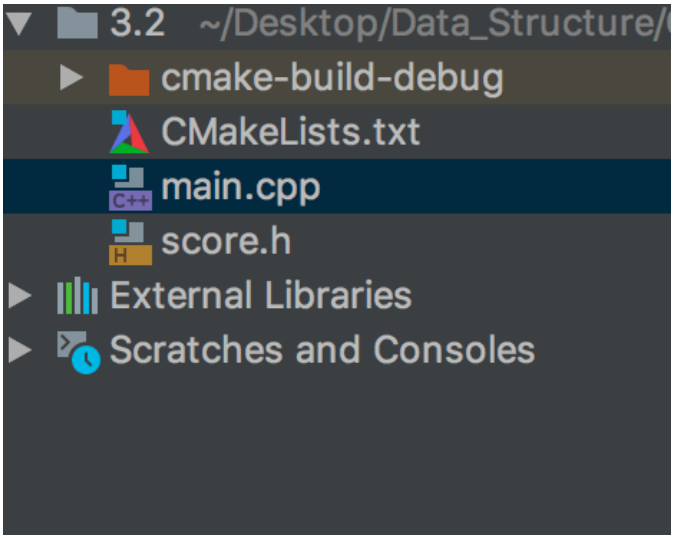
Compare,student 类的初始化和析构函数的时间复杂度都是线性的，故为 $O(1)$。divde 函数使区间内的数值根据数组第一个元素值排序，因此时间复杂度与元素个数成正比，因此时间复杂度为 $O(n)$。

2）空间复杂度

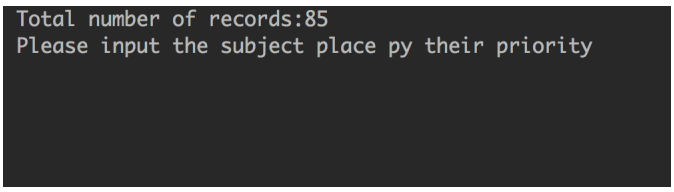
list 类模块的空间复杂度与叶节点的个数成正比，即 $O(n)$。主函数模块的复杂度取决于定义主函数作用域中的 *list* 类，故空间复杂度也为 $O(n)$。

5 用户手册

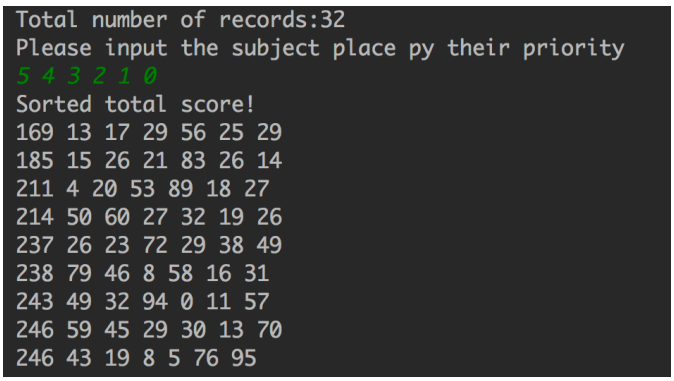
1. 本程序以 Jetbrains Clion 2018.2.5, 采用 C++ 11 标准，程序以项目方式组织 (project)，如图 1 所示：



2. 依次点击菜单"Run"->build, 再点击"Run", 显示文本方式的用户界面, 如图 2 所示:



3. 键入单科成绩排序后按“回车键”(例如：5 4 3 2 1 0, 代表最重要的学科是学科五，最不重要的学科是学科零)，程序就执行排序命令，并在屏幕上打印排序后的总分和单科成绩。



6 测试结果

程序开始后先自动生成随机个数的学生和他们相对应的六科成绩，待用户输入单科成绩排序后，在屏幕上打印排序后的总分和六科成绩。

7 附录

源程序文件名清单

main.cpp //主函数 score.h //*list* 类单位模块