山东大学 计算机科学与技术 学院

数据结构与算法课程设计 课程实验报告

学号：202200130039	姓名：张向荣	班级： 22.3
上机学时：4 小时	实验日期： 2024.4.8	

课程设计题目：
外排序

软件开发环境：
Dev-c++

报告内容：

1. 需求描述

1.1 问题描述

引用输者树结构模拟实现外排序。

1.2 基本要求

1. 设计并实现**最小输者树**结构 ADT，ADT 中应包括初始化、返回赢者，重构等基本操作。

2. 应用最小输者树设计实现外排序，外部排序中的生成最初归并串以及 K 路归并都应用最小输者树结构实现；

3. 验证你所实现的外排序的正确性。（1）随机创建一个较长的文件作为外排序的初始数据，设置最小输者树中选手的个数，验证生成最初归并串的正确性。获得最初归并串的个数及最初归并串文件，每一最初归并串使用一个文件。（2）使用以上生成的归并串，设置归并路数，验证 K 路归并的正确性。获得 K 路归并中各趟的结果，每一趟的结果使用一个文件。

*4. 获得外排序的访问磁盘次数，并分析其影响因素。

1.3 输入说明

data.in 文件为外排序的初始数据，其中第 1 行为元素个数，第 2 行开始为元素数值， properties.txt 文件中给出最小输者树的大小(树中选手的个数)和归并路数。

输入界面设计

输入样例

1.4 输出说明

output.txt 为最终排序结果，Segments 文件夹内为排序过程中产生的文件，如 Seg0-1.txt 为第 1 个初始顺串，Seg1-1.txt 为第 1 趟归并排序中产生的第一个结果文件。

输出界面设计

输出样例

2. 分析与设计

2.1 问题分析

题目中要求设计并实现最小输者树结构 ADT，首先要理解输者树的定义和原理。

输者树是一种完全二叉树，是树形选择排序的一种变型。每个叶子结点相当于一个选

手，每个中间结点相当于一场比赛，每一层相当于一轮比赛，用父结点记录其左右子结点进行比赛的败者而让胜者参加下一轮的比赛。输者树的中间结点记录的是败者的标号，需要加一个结点来记录整个比赛的胜利者。

外部排序是指针对大规模数据进行排序，其中无法将整个数据集一次性加载到内存中。因此需要将数据划分为适当大小的块，然后对每个块进行排序，在此之后，将这些排好序的块合并成更大的块，直到最终得到一个已排序的数据集。

K 路归并则是将大文件纷呈 K 个子文件并对每个子文件进行排序，然后合并成一个大文件。

外部排序的开销：总时间开销=内部排序所需时间+内部归并所需时间+外部读写所需时间。增大归并路数 k，或减少初始归并段个数 r，都能减少归并趟数 s，进而减少读写磁盘的次数，达到提高外部排序速度的目的。

2.2 主程序设计

2.3 设计思路

K 路合并：

从 K 个顺串的前面不断地移出值最小的元素，该元素被移至输出顺串中。当所有的元素从 k 个输入顺串移至输出顺串中时，便完成了合并过程。只要有足够内存保存 k 个元素值，就可以合并任意长度的 k 个顺串。

2.4 数据及数据类型定义

2.5.算法设计及分析

首先定义一个 losertree.h 文件，定义一个树节点结构体，定义成员顺串号和元素值，并重载运算符,首先比较顺串号再比较元素的值。定义一个 losertree 类,在 losertree.cpp 中实现相关功能。在构造函数中，传入的数组就是外部节点，player 数组表示选手，其孩子节点表示外部节点，定义一个 winer 数组，先记录每个孩子节点的下标。然后再从孩子节点的父节点开始找到每轮的输者，存在 loser 数组中，而 player 数组存储下轮的比赛选手，也就是赢者的值。Winner 数组记录赢者的下标，最后 loser[0]中存放最后赢者的值，也就是最小值。在重构函数中，将最后的赢者输出出去，这个空缺的位置传入一个新的元素值，将这个值放在原来的孩子节点处，并一层层遍历更新父结点与最后的赢者。

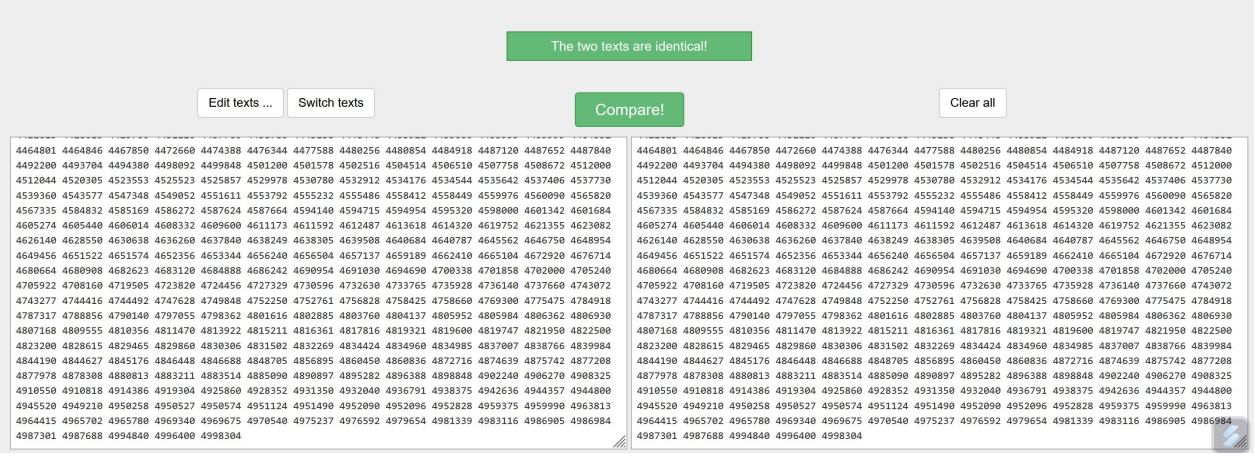
在 merge.h 中定义一个 Merge 类，在 merge.cpp 中定义相关函数。在构造函数中初始化输者树大小以及几路归并。

在 divide（）函数中，首先初始输者树中所有元素的初始顺串为 1，找到每次的赢者，重复下面操作：将每次的赢者放入它的顺串号所对应的顺串中，使得每个顺串中的元素值按从小到大排列。如果集合中有下一个输入元素，则将元素的值与赢者的元素值比较，如果该元素值大于等于赢者的元素值，则该元素的顺串号等于赢者的顺串号，否则该元素的顺串号等于赢者的顺串号+1。然后该元素代替原来的赢者重构赢者树。直到集合中没有元素，要继续将输者树中剩余的数输出，则每次重构时，传入最大值，且顺串号为 MAX_INT，当最后赢者是 MAX_INT 时，说明剩余元素已经输出完，停止循环。每次将元素输入到对应文件中，寻盘次数+1。

在 combine（）函数中，进行归并，直到最后文件数为 1。首先打开所有的归并段文件，每打开一个文件，寻盘次数+1。然后将所有文件进行 K 路归并，在每次归并前判断文件数量，若小于等于 k 则说明是最后一次归并打开文件 output.txt，存储最终的归并结果，否则建立新的归并段。定义一个 k 长度的数组，用来存放每个归并段中的最小值，每组归并 k 个，如果最后剩余不足 k 个归并段，则将数组中剩下的地方存入 MAX_INT，将数组使用最小输者树进行排列，当赢者不是 MAX_INT 时，将每组赢者存入新的归并段中，并找到最小值所在数组中的位置，进而找到原来所在的归并段，向最小输者树种输入下一个元素，如果归并段中没有元素，则输入 MAX_INT，并对树进行重构，直到所有元素输出到新的归并段。将原来所有归并段对应的文件关闭，更新文件数量为，即除以 k 并向上取整，进行新

一轮的归并直到文件数量为 1。

3. 测试



4. 分析与探讨

最小输者树的初始化时间复杂度是 O(n)，先将叶子结点赋值，然后沿着从叶子到根的方向，在内部节点进行 n-1 场比赛。而将赢者输出重新传入一个选手时，需要修改的比赛场次介于 1~$\log_2 n$ 之间，因此重构的时间复杂度是 O(logn)。

K 路排序的时间：初始顺串的长度，能影响扫描遍数和外存读写次数。归并顺序的安排能影响排序时间。

将每一个元素合并到输出顺串中需 O(K)时间，因此产生大小为 n 的顺串所需要的内部时间为 O(kn)。

对同一个文件而言，进行外排序所需读写外存的次数与归并趟数有关。假设有 m 个初始顺串，每次对 k 个顺串进行归并，归并趟数为 $\log_k m$。

为了减少归并趟数，可以减少初始顺串的个数 m，增加归并的顺串数量 k（会增加内部归并的时间），减少 K 路合并时间。

5. 附录：实现源代码

```
template<class T>//打开文件一次，向其中存一个数字，寻盘次数+1
void Merge<T>::divide(string name)                    //初始归并段的生成
{
    ifstream myfile(name,ios::in);                //打开名为 name 的文件
    int number;
     myfile>>number;          //从文件中读取数据的个数

    times++;

    Node* ls=new Node[Size];     //建立一个大小为输者树大小的数组
    for (int i=0;i<Size;i++)
    {
        myfile>>ls[i].number;
        ls[i].id = 1;            //初始元素顺串号均为 1
    }

    losetree<Node> tree(ls,Size);
```

```
    int num=0;
    while (1)
    {
        myfile>>num;//集合中下一元素
        if (myfile.eof())                    //判断文件末位，文件到末尾则退出循环
            break;

        Node p1=tree.win_min();//最小值

        Node p;
        p.number=num;
        if (num>=p1.number)                  //如果 num 比最小输者树的最小值大，则优先级相同，反之则优先级+1
        {
            p.id=p1.id;
        }
        else
        {
            p.id=p1.id+1;//顺串号+1
            NumberOfFile=max(NumberOfFile, p.id);
           //得到产生的初始归并段的个数，为最大顺串号
        }

         tree.reinit(p);                     //重构
        string filename=s+"0-"+to_string(p1.id)+s1;      //文件名,初始顺串
        ofstream ifile(filename,ios::app);          //ios::app 在文件末位接着输出，ios::out 重新在文件中输出
        ifile<<p1.number<<' ';
        ifile.close();

        times++;            //由于打开了 s3 文件因此，寻盘次数加一
    }

    while (1)   //将输者树中剩余的数输出
    {
        if(tree.win_min().number==MAX_INT)
            break;

        Node p1=tree.win_min();

        Node p;             //p 的优先级得是最低，并且值是最大
        p.id=MAX_INT;
        p.number=MAX_INT;

        tree.reinit(p);
        string filename=s+"0-"+to_string(p1.id)+s1;
```

```
            ofstream ifile(filename, ios::app);       //以追加的方式打开文件
            ifile<<p1.number<<' ';
            ifile.close();

            times++;
        }
}

template<class T>
void Merge<T>::combine()                              //归并段的合并
{
    for (int t=0;NumberOfFile!=1;t++)//第 t+1 趟归并
    {
        int* R=new int[k];
        ifstream* myfile=new ifstream[NumberOfFile+1]; //将所有的归并段都放入缓冲区
        for (int i=1;i<=NumberOfFile;i++)
        {
            //segh-i.txt ，第 h 趟归并排序中产生的第 i 个结果
            myfile[i].open(s+to_string(t)+"-"+to_string(i)+s1,ios::in);
            times++;//打开一次，寻盘次数+1
        }

        int j=0;
        for (;j<ceil((double)NumberOfFile/(double)k);j++)//将所有的文件进行 k 路归并,j 表示
该趟的第 j+1 组
        {
            string out;
            if(NumberOfFile<=k) //判断产生的文件是否是最终的归并结果，对输出的文件
名进行定义
            {//最后一次归并
                out=answer;//output.txt
            }
            else
            {       //segh+1-j+1.txt
                out=s+to_string(t+1)+"-"+to_string(j+1)+s1;
            }

            ofstream outfile(out,ios::out);

            times++;
            for (int i=1;i<=k;i++)
            {
                if(j*k+i>=NumberOfFile+1)//判断剩余的文件个数是否小于归并路数，
                //如果小于则要将这个数组中的差值用最大值进行补充
                {
                    R[i-1]=MAX_INT;//从 0 开始
```

```
                }
                else
                    myfile[j*k+i]>>R[i-1];
            }

            losetree<int> L(R,k);

            while (L.win_min()!=MAX_INT)
            {
                outfile<<L.win_min()<<' ';
                for(int f=0;f<k;f++)
                {       ///f+k 中的 f 表示在传入数组中的位置，而 k 表示内部节点的个数
                    if(L.win_player()==f+k&&j*k+f+1<=NumberOfFile)
                    //得到最小输者树的最小值下标是具体的那个文件，此时还要注意有可能存在剩余的文件个数小于归并路数的情况
                    {
                        int u=0;
                        myfile[j*k+f+1]>>u;
                        if(myfile[j*k+f+1].eof())//如果到文件末位则将最大值输入到输者树中，然后进行重排即可
                            u=MAX_INT;
                        L.reinit(u);
                        break;
                    }
                }
            }
            outfile.close();
        }

        for(int i=1;i<=NumberOfFile;i++)
            myfile[i].close();
        NumberOfFile=j;//numberfile/k;
    }
}
```