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报告

学号：202000130143	姓名： 郑凯饶	班级： 20.1
上机学时： 6		日期：2022-4-5
课程设计题目：外排序		
软件环境：Windows 10 家庭中文版 64 位（10.0，版本 18363） Microsoft VS Code cmake version 3.23.0-rc3		

报告内容：

1.需求描述

1.1 问题描述

应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 中应包含初始化、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，设置最小输者树中的选手的个数，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

输入界面设计

无（使用文件流读取标准输入）

输入样例

1.4 输出说明

输出界面设计

无（使用文件流输出至按规则指定文件）

输出样例

2.分析与设计

2.1 问题分析

1.最小输者树和赢者树对于父子节点、内外部关系的处理类同。当外部节点的个数为 n 时，内部节点的个数为 n-1,。最底层最左端的内部节点设为 s，满足

$$s = 2\text{^}\lfloor \log_2(n-1) \rfloor$$

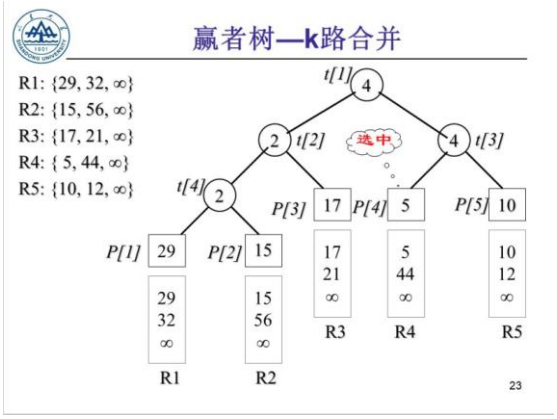
因此，最底层内部节点的个数是 n-s，最底层外部节点的个数 LowExt 是这个数的 2 倍。令$\text{offset} = 2 * s + 1$，对于任何一个外部节点 player[i]，其父节点 tree[p]由以下公式给出：

$$p = \begin{cases} \dfrac{i + offset}{2}, i \le lowExt \\ \dfrac{i - lowExt + n - 1}{2}, i > lowExt \end{cases}$$

建树过程，两者几乎相同。输者树关注对赢者的改变，重构过程相对简单不需要考虑特殊情况。

2. 利用输者树进行求顺串。使用老师讲授的方法。

3. 利用输者树进行 K 路归并。



4. 获得外排序的访问磁盘次数。设计一个磁盘控制器对象（DiskControler），用于记录模拟的磁盘访问信息。

2.2 主程序设计

```
> struct Player ···

  int n;  // 元素总数
  int K;  // 归并路数
  Player players[200010];
  DiskControler mIN, mOUT;    // 磁盘读写控制

> void ConstructRuns() {···

> void KRoadMerge(int mRound) {···

> void testForCorrection(int testPoint) {···

> int main() {···
```

ConstructRuns()：利用输者树生成顺串。

KRoadMerge()：利用输者树进行 K 路归并。

TestForCorrection()：进行磁盘控制器对象初始化，调用 ConstructRuns()和 KRoadMerge()实现外排序操作。

2.3 设计思路

主要是利用输者树不断决出赢者（最小值）。

生成顺串使用顺串号和值进行比赛，因为新的赢者可能小于之前没有和它比较的赢者，实际是由于树的大小，比较具有局部性，所以用顺串号将它们分成多个顺串。

K 路归并也是不断找出 K 路之中的赢者，再用赢者所在顺串中的下一位进行重构。

磁盘模拟我考虑的是假设内存只能存下输者树，也没有输入输出缓冲区。初始化时由于所有数据都在一起，仅 1 次读。而之后每次决出赢者，重构，都需要 1 次读，1 次写。

2.4 数据及数据类(型)定义

```
template<class T>
class LoserTree {
    public:
        LoserTree(T *thePlayer, int theNumberOfPlayers) {
            tree = NULL;
            Winners = NULL;
            initialize(thePlayer, theNumberOfPlayers);
        }
        ~LoserTree() { delete [] tree; }
        void initialize(T*, int);
        int winner() const { return tree[0]; }
        int loser(int i) const {
            return (i < numberOfPlayers && i > 0) ? tree[i] : 0;
        }
        void replay(int, T);
        void output() const;

    private:
        int Winner;     // 比赛赢者
        int *Winners;   // 初始化时暂存赢者
        int lowExt;     // 最底层外部节点个数
        int offset;
        int *tree;      // 内部节点[1, n - 1]
        int numberOfPlayers;
        T *player;
        void play(int, int, int);
};
```

和赢者树类同。增加：

Winner：一场比赛的赢者，用于重构；

*Winner：初始化时暂存赢者，*tree 仅仅存储输者，无法向上建树。

```
struct DiskControler
{
    // 默认输入输出缓冲区大小为1
    int numOfDiskAccess;  // 访问次数
    int memorySize;       // 内存大小
    int numOfiles;        // 操作文件数量
    DiskControler():numOfDiskAccess(0), memorySize(0), numOfiles(0) {}
    string Path;
    string fileName;
    string filePath;
};
```

磁盘控制器对象：

NumOFDiskAccess:磁盘访问次数；

MemorySize:内存大小；

NumOfFiles:操作文件的数量；

Path:读写文件目录；

Filename：读写文件名；

filePath：读写文件路径；

```
struct Player
{
    int serialNum, val;
    bool operator >= (const Player &a) const {
        if (serialNum == a.serialNum) return val >= a.val;
        return serialNum > a.serialNum;
    }
};
```

选手对象，用于生成顺串，serialNum 为顺串号，val 为值。

2.5. 算法设计及分析

1. 顺串生成：

```
LoserTree<Player> tree(players, mIN.memorySize);

int num;
for (int i = 1; i <= n; i++) {
    if (!(f >> num))
        num = INT_MIN;
    else mIN.numOfDiskAccess++;

    Player mWinner = players[tree.winner()];
    Player mPlayer;
    mPlayer.val = num;

    if (num >= mWinner.val) {
        mPlayer.serialNum = mWinner.serialNum;
    }
    else if (num == INT32_MIN) {
        mPlayer.serialNum = INT32_MAX;
    }
    else mPlayer.serialNum = mWinner.serialNum + 1;

    tree.replay(tree.winner(), mPlayer);

    string seg0File = mOUT.Path + MYSEG0 + to_string(mWinner.serialNum) + TXT;
    mOUT.numOfiles = max(mOUT.numOfiles, mWinner.serialNum);
    ofstream fout(seg0File, ios::app);   // 追加
    fout << mWinner.val << " ";
    fout.close();
    mOUT.numOfDiskAccess++;
}
```

用 INT32_MAX 加入输者树，决出 N 个赢者。

2. K 路归并

```
int mRound = 1;
do {
    KRoadMerge(mRound);
    mIN.numOfiles = (mIN.numOfiles + K - 1) / K;
    mRound++;
} while (mIN.numOfiles > 1);
```

总共进行$\lceil \log_K n \rceil$轮的 K 路归并。因此可以通过减少 n（顺串的个数）或者增加归并路数 K 来提高外排序时间。

```
// 初始化 && 不足K路
int overCnt = 0;        // 已结束顺串
for (int j = 1; j <= K; j++) {
    if ((i + j - 1) > mIN.numOfiles) {
        pplayers[j] = INT32_MAX;
        overCnt++;
        continue;
    }
    string seg0File = mIN.Path + MYSEG + to_string(mRound - 1) + "-" + to_string(i + j - 1) + TXT;
    ifstream f(seg0File);
    f >> pplayers[j];
    if (DEBUG) cout << "pplayers " << j << " : " << pplayers[j] << '\n';

    mIN.numOfDiskAccess++;
    FP[j] = f.tellg();
    f.close();
}
```

初始化，以及针对不足 K 路的情况用 INT32_MAX 替代，一旦所有 K 路均为 INT32_MAX，代表归并完成。同时利用*FP 保存文件指针。

4. 分析与探讨

堆、竞赛树比较相似，树的形态是完全二叉树，但后者有内外部节点之分，要处理一些内外部节点之间的关系。功能上，他们都可以 O(1)查看最值，并且 O(logn)重构。由于没有维护原始数据对象，堆难以进行外排序，而竞赛树提供了解决之道。

5. 附录：实现源代码

（另附）