**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：201700140056 | 姓名：李港 | | 班级：17.4 |
| 上机学时：4 | | 日期：2020.04.11 | |
| 课程设计题目：外排序 | | | |
| 软件环境：VS2019 | | | |
| 报告内容：  **一、需求描述**  **1.1 问题描述**  应用竞赛树（输者树）结构模拟实现外排序。  **1.2 基本要求**   1. 设计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. 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. 随机创建一个较长的文件；设置归并路数以及缓冲区的大小；获得外排序的访问磁盘的次数并进行分析。可采用小文件来模拟磁盘块。   **1.3 输入说明**   1. 输入设计：采用文件输入的形式，文件内仅包含无序的数组。 2. 输入数据样例：采用文件输出的形式，文件内包含排好序的数组。另外temp文件夹下包含分割后的小文件。       **1.4 输出说明**  输出界面设计    输出数据样例      **二、分析与设计**  **2.1 问题分析**  需要设计两个大部分，输者树与外排序。  输者树即为一种特殊的二叉树。  外排序需要进行文件的分割与合并操作，分割与合并过程都要使用输者树来进行。  **2.2 主程序设计**  采用面向对象方法，主程序仅调用外排序对象并传入参数，不进行其他操作。  **2.3 设计思路**  输者树采用数组实现，一数组保存外节点信息，另一数组以索引形式保存输者树的逻辑结构。  外排序类仅保存元素数量，文件路径，读盘次数等辅助内容  **2.4 数据及数据类(型)定义**    **2.5.算法设计及分析**  输者树的重排算法  //更新结构，thePlayer指向的元素已经被替换  void replay (int thePlayer) {  int n = \_numOfPlayer;  //判断player范围  if (thePlayer <= 0 || thePlayer > n) {  cout << "\_players index is illgal" << endl;  return;  }  //获取顺串新元素在树中的新位置  int fatherPoint;//左右选手的父亲节点  int left, right;//左右选手  if (thePlayer <= lowExt) {//如果变化的节点在最底层  fatherPoint = (thePlayer + offset) / 2;//获取父节点  left = 2 \* fatherPoint - offset; // unify to the left \_players  right = left + 1;  }  else { //the \_players is on the last but one  fatherPoint = (thePlayer - lowExt + n - 1) / 2;  //theplayer的左兄弟是最后一个内部节点  if (2 \* fatherPoint == n - 1) {  left = winners[2 \* fatherPoint];  right = thePlayer;  }  else {  left = 2 \* fatherPoint - (n - 1) + lowExt;  right = left + 1;  }  }  //重新比赛  if (thePlayer == tree[0]) { //重新比赛的选手在之前胜者的位置  for (; fatherPoint >= 1; fatherPoint /= 2) { //上次比赛的输者已经记录在tree[]中  int loserTemp = \_whoLose (tree[fatherPoint], thePlayer);//只需跟败者判断  winners[fatherPoint] = \_whoWin (tree[fatherPoint], thePlayer);  tree[fatherPoint] = loserTemp;//父节点放置新的败者  thePlayer = winners[fatherPoint];//赢者继续比赛  }  }else {//否则，无法判断新节点与老输者的关系，从该节点到根的路径需重新比赛  tree[fatherPoint] = \_whoLose (left, right);  winners[fatherPoint] = \_whoWin (left, right);  //向上一级  if (fatherPoint == n - 1 && n % 2 == 1) {  fatherPoint /= 2;  tree[fatherPoint] = \_whoLose (winners[n - 1], lowExt + 1);  winners[fatherPoint] = \_whoWin (winners[n - 1], lowExt + 1);  }  fatherPoint /= 2;  for (; fatherPoint >= 1; fatherPoint /= 2) {  tree[fatherPoint] = \_whoLose (winners[2 \* fatherPoint], winners[2 \* fatherPoint + 1]);  winners[fatherPoint] = \_whoWin (winners[2 \* fatherPoint], winners[2 \* fatherPoint + 1]);  }  }  //保存最终胜者  tree[0] = winners[1];  }  外排序思路：   1. 将乱序输入采用输者树分块存储为多个排好序的小文件。 2. 将所有小文件采用输者树进行多路归并。   **三、测试**   1. 排序过程输出 2. 排序前后数据对比  可见，原始乱序数据成为了有序数据。 3. 绘制输者树   **四、分析与探讨**   1. 大批量数据算法是面试常考的知识点，外排序只是其中一类。但这些算法都有统一的思想“分治”。 2. 输者树可以降低数据变动对数据结构的影响   **五、附录：实现源代码**  #include<iostream>  #include<vector>  #include<map>  #pragma warning(disable:4996)  using namespace std;  template<class T>  class LoserTree {  public:  LoserTree () { tree = nullptr; };  LoserTree (T\* thePlayers, int n) { initTree (thePlayers, n); } //创建输者树  ~LoserTree () { delete[] tree; delete[] winners; } //释放资源  void initTree (T\* thePlayers, int n) {  auto \_play = [this](int p, int left, int right) ->void {  //根据player[l],\_players[r]的值，设置tree[p]的值，即在左右子节点l，r之间举办比赛  //之后如果p是右孩子节点，则继续举行上一层的比赛  tree[p] = \_whoLose (left, right);  winners[p] = \_whoWin (left, right);  //内节点p是右孩子且非根  while (p % 2 == 1 && p > 1) {  tree[p / 2] = \_whoLose (winners[p - 1], winners[p]);  winners[p / 2] = \_whoWin (winners[p - 1], winners[p]);  p /= 2;  }  };  tree = nullptr, winners = nullptr;  if (n < 2) { return; }  \_players = thePlayers;  \_numOfPlayer = n;  tree = new int[n];  winners = new int[n];  int i, s;  //计算s, lowExt, offset  for (s = 1; 2 \* s <= n - 1; s \*= 2);  lowExt = 2 \* (n - s);  offset = 2 \* s - 1;  //对所有底层外部节点进行play()  for (i = 2; i <= lowExt; i += 2)  \_play ((i + offset) / 2, i - 1, i);  //n为奇数时，会出现一个内节点其孩子分别是内节点和外节点  //先对这个内节点和外节点进行play()  if (n % 2 == 1) {  \_play (n / 2, winners[n - 1], lowExt + 1);  i = lowExt + 3;  }  else {  i = lowExt + 2;  }  //再对剩余的外部节点play()  for (; i <= n; i += 2)  \_play ((i - lowExt + n - 1) / 2, i - 1, i);  //将最终的赢者记录在tree[0]  tree[0] = winners[1];  }  //更新结构，thePlayer指向的元素已经被替换  void replay (int thePlayer) {  int n = \_numOfPlayer;  //判断player范围  if (thePlayer <= 0 || thePlayer > n) {  cout << "\_players index is illgal" << endl;  return;  }  //获取顺串新元素在树中的新位置  int fatherPoint;//左右选手的父亲节点  int left, right;//左右选手  if (thePlayer <= lowExt) {//如果变化的节点在最底层  fatherPoint = (thePlayer + offset) / 2;//获取父节点  left = 2 \* fatherPoint - offset; // unify to the left \_players  right = left + 1;  }  else { //the \_players is on the last but one  fatherPoint = (thePlayer - lowExt + n - 1) / 2;  //theplayer的左兄弟是最后一个内部节点  if (2 \* fatherPoint == n - 1) {  left = winners[2 \* fatherPoint];  right = thePlayer;  }  else {  left = 2 \* fatherPoint - (n - 1) + lowExt;  right = left + 1;  }  }  //重新比赛  if (thePlayer == tree[0]) { //重新比赛的选手在之前胜者的位置  for (; fatherPoint >= 1; fatherPoint /= 2) { //上次比赛的输者已经记录在tree[]中  int loserTemp = \_whoLose (tree[fatherPoint], thePlayer);//只需跟败者判断  winners[fatherPoint] = \_whoWin (tree[fatherPoint], thePlayer);  tree[fatherPoint] = loserTemp;//父节点放置新的败者  thePlayer = winners[fatherPoint];//赢者继续比赛  }  }else {//否则，无法判断新节点与老输者的关系，从该节点到根的路径需重新比赛  tree[fatherPoint] = \_whoLose (left, right);  winners[fatherPoint] = \_whoWin (left, right);  //向上一级  if (fatherPoint == n - 1 && n % 2 == 1) {  fatherPoint /= 2;  tree[fatherPoint] = \_whoLose (winners[n - 1], lowExt + 1);  winners[fatherPoint] = \_whoWin (winners[n - 1], lowExt + 1);  }  fatherPoint /= 2;  for (; fatherPoint >= 1; fatherPoint /= 2) {  tree[fatherPoint] = \_whoLose (winners[2 \* fatherPoint], winners[2 \* fatherPoint + 1]);  winners[fatherPoint] = \_whoWin (winners[2 \* fatherPoint], winners[2 \* fatherPoint + 1]);  }  }  //保存最终胜者  tree[0] = winners[1];  }  void output () const {  for (int i = 0; i < \_numOfPlayer; i++) {  cout << \_players[tree[i]] << endl;  }  }  int\* theTree () { return tree; } //返回输者树  int winner () { return winners[1]; }  private:  int\* tree; //输者树内部节点  int\* winners; //存放赢者  T\* \_players; //输者树外部节点  int \_numOfPlayer;//选手的个数  int lowExt; //2\*(n-s),最底层外部节点的个数,s=2^(log2(n-1))  //s为最底层最左端的内部节点的位置  int offset; //2\*s-1  int \_whoWin (int x, int y) { return \_players[x] <= \_players[y] ? x : y; } //返回赢者  int \_whoLose (int x, int y) { return \_players[x] <= \_players[y] ? y : x; } //返回输者  };  #include"loserTree.h"  #include<limits.h>  #include<fstream>  #include<cstdio>  using namespace std;  //输者树的元素  struct node {  int value; //元素值  int index;//顺串号  operator int () { return value; }//将类转化为int类型  bool operator<=(node b) {  if (index < b.index)  return true;  else if (index == b.index)  return value <= b.value;  else  return false;  }  };  class externalsort {  public:  string \_filepath;  externalsort (int buf\_size,string filepath) :\_bufferLength(buf\_size),\_filepath(filepath){}  ~externalsort () {}  //进行文件分割  void split () {  \_clearfile (300);  cout << ">>>分割文件开始" << endl;  //打开文件  ifstream in (\_filepath);  if (in.fail ()) {  cout << "no such file!";  exit (0);  }  //新建p+1个选手  node\* players = new node[\_bufferLength + 1];  //  //读取文件，初始化所有选手  int value;  int q = 0;  while (q < \_bufferLength && in >> value) {  \_countReadDiskOnce ();  q++;  players[q].value = value;  players[q].index = 1;  cout <<"排序输者树："<< players[q].value << endl;  }  q++;  if (\_bufferLength > q)  \_bufferLength = q;  //使用这些选手初始化输者树  LoserTree<node> \_spliter;  \_spliter.initTree (players, \_bufferLength);  \_fileNum = 1;  //分割文件  cout << "开始分割文件：" << endl;  \_totalSize = \_bufferLength;  int temp\_in\_num;  //将剩余数据添加到各小文件中  while (in >> temp\_in\_num) {  \_totalSize++;  \_countReadDiskOnce ();  node newNode;  //输入的数字比原有的小  if (temp\_in\_num < players[\_spliter.winner ()].value) {  newNode.index = players[\_spliter.winner ()].index + 1;  if (newNode.index > \_fileNum)  \_fileNum = newNode.index;  } else  newNode.index = players[\_spliter.winner ()].index;  newNode.value = temp\_in\_num;  char temp\_file\_name[50];  sprintf (temp\_file\_name, "winners/%d.txt", players[\_spliter.winner ()].index);  cout << players[\_spliter.winner ()].value << "添加到文件 " << temp\_file\_name << endl;  ofstream out;  out.open (temp\_file\_name, ios::app);//追加模式写  out << players[\_spliter.winner ()].value << " ";  out.close ();  \_countReadDiskOnce ();  int i = \_spliter.winner ();  players[i] = newNode;  \_spliter.replay (i);  }  //将所有数据清空  for (int i = 0; i < \_bufferLength; i++) {  \_countReadDiskOnce ();  char temp\_file\_name[50];  sprintf (temp\_file\_name, "winners/%d.txt", players[\_spliter.winner ()].index);  cout << players[\_spliter.winner ()].value << "添加剩余数据到文件"<< temp\_file\_name << endl;  ofstream out\_file\_stream;  out\_file\_stream.open (temp\_file\_name, ios::app);//追加模式写  out\_file\_stream << players[\_spliter.winner ()].value << " ";  out\_file\_stream.close ();  //替换元素  players[\_spliter.winner ()] = { INT\_MAX ,INT\_MAX };  //重排  \_spliter.replay (\_spliter.winner ());//重排  }  delete[]players;  cout << "分割文件数：" << \_fileNum << "每个文件都已经排序" << endl;  cout << ">>>分割文件完成" << endl;  }  void merges ();  void visitstime () { cout << "访问磁盘次数: " << \_readDiskCount << endl; }  private:  //int \_bufferLength;//初始化顺串时最小竞赛树的规模,初始顺串的平均长度为2p  int \_fileNum;//输入顺串数  int \_totalSize;//待排序元素总数  int \_readDiskCount;//访问磁盘次数  int \_bufferLength;  static int \_bufferUsed;  //清空所有文件  void \_clearfile (int n) {  cout << ">>>正在清空临时文件：" << endl;  char a[100];  for (int i = 1; i <= n; i++) {  sprintf (a, "winners/%d.txt", i);  remove (a);  }  cout << ">>>临时文件清空完成：" << endl;  }  //模拟读硬盘  void \_countReadDiskOnce () {  \_bufferUsed++;  if (\_bufferUsed > \_bufferLength) {  \_readDiskCount++;  \_bufferUsed = 0;  }  }  };  int externalsort::\_bufferUsed = 0;  //排序过程  void externalsort::merges () {  //分割  split ();  //用于文件归并排序  LoserTree<int> \_final\_sorter;  cout << "开始合并：" << endl;  cout << "分割文件个数: " << \_fileNum << endl;  cout << "总数据量: " << \_totalSize << endl;  //打开k个文件输入流  char a[50];  ifstream\* in\_files = new ifstream[\_fileNum + 1];  for (int i = 1; i <= \_fileNum; i++) {  sprintf (a, "winners/%d.txt", i);  in\_files[i].open (a);  if (!in\_files[i].is\_open ()) {  cout << "open winners file error" << endl;  return;  }  }  //读入所有文件的头一个数据  int\* da = new int[\_fileNum + 1];//头数据数组  for (int i = 1; i <= \_fileNum; i++) {  in\_files[i] >> da[i];//数值表示key，下标表示顺串号  }  //初始化合并输着树  \_final\_sorter.initTree (da, \_fileNum);  //遍历各文件中所有数据  {  ofstream out\_file ("result.txt");//出作用域自动关闭文件  for (int i = 0; i < \_totalSize; i++) {  //每次从竞赛树中弹出冠军读取其顺串号，再输出key值到文件  int winner\_file\_id = \_final\_sorter.winner ();//竞赛树中冠军的下标，顺串号  out\_file << da[winner\_file\_id] << " ";  //根据顺串号从相应文件读取下一个数据，若不存在则用INT\_MAX代替，替换冠军，重排  \_countReadDiskOnce ();  //获取下一个数字+顺串空判断  int x;  if (in\_files[winner\_file\_id] >> x)  da[winner\_file\_id] = x;  else  da[winner\_file\_id] = INT\_MAX;  //输者树重新比赛  \_final\_sorter.replay (winner\_file\_id);  }  }  delete[] in\_files;//删除顺串输入流数组  cout << "合并完成" << endl;  }  #pragma warning(disable:4996)  #include"externalSort.h"  #include<string>  #include<iostream>  int main(){  cout << R"( \_ \_\_\_\_\_\_\_  | | |\_\_ \_\_|  | | \_\_\_ \_\_\_ \_\_\_ \_ \_\_| |\_ \_\_ \_\_\_ \_\_\_  | | / \_ \/ \_\_|/ \_ \ '\_\_| | '\_\_/ \_ \/ \_ \  | |\_\_\_| (\_) \\_\_ \ \_\_/ | | | | | \_\_/ \_\_/  |\_\_\_\_\_\_\\_\_\_/|\_\_\_/\\_\_\_|\_| |\_|\_| \\_\_\_|\\_\_\_|  )";  cout << "\*LoserTree external sort"<<endl;  cout << "\* file path:";  string filepath;  cin >> filepath;  cout << "\* buffer size:";  int size;  cin >> size;  externalsort es(size,filepath);  //进行归并  es.merges();  return 0;  } | | | |