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：202300130207 | 姓名：李臣思 | | 班级：23.5 |
| 实验题目：外排序 | | | |
| 实验学时：10 | | 实验日期：25.4.9 | |
| 实验目的：  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。  掌握外排序的结构与构建过程。 | | | |
| 软件开发工具：  Visual Studio Code | | | |
| 1. 实验内容   问题描述：  实现一个外部排序算法，通过败者树管理缓冲区数据，生成顺串并多路归并，最终输出有序结果。  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路归并都应用竞赛树结构实现；  （3）随机创建一个较长的文件作为外排序的初始数据；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. 数据结构与算法描述（整体思路描述，所需要的数据结构与算法）  * 败者树结构   player结构体：每个参与败者树的选手都包含一个id（顺串编号）和element（元  素值），重载<=运算符用于比较。   * minlosertree类：包含成员变量tree（内部节点数组，存放败者索引）、leave（外   部节点数组，存储胜者索引）、players（参赛元素数组）、k（选手数量）、low1（最底层外部节点数量）、offset(父节点索引偏移量)。  **核心算法：私有函数winner、loser:**因为是输者树，根据外排序的实现所以较小的乙方  为赢者，反而较大的为输者，所以winner、loser分别返回输入的x、y索引对应的选手较小和较大的一方。  **函数play(int p, int leftchild, int rightchild)：**比较leftchild和rightchild，将较小的一方索引放入tree[p]和leave[p]中。然后继续向上找到p对应的父节点，如果原本的p是右侧的孩子的话那么就比较p-1和p对应的成员，循环往复，直到达到根节点或者p对应是左孩子时停止。  **构造函数minlosertree(T\* theplayer=NULL,int thenumberofplayers=0):**初始化tree数组和leave数组为空，调用initialize函数。  **初始化函数initialize(T\* theplayer, int number)：**初始化将输入的theplayer和number分别放入成员players和k中。构造数组tree和leave。for (s = 1; 2 \* s <= n - 1; s += s)找到最底层最左端的内部节点。Low1=2\*(n-s),offset=2\*s-1分别得到最底层外部节点的数量，以及计算父节点的索引偏移量。例如在15个选手时，那么最底层最左端的内部节点应该是8，最底层外部节点的数量是14个，父节点索引偏移量是15。  那么先将14个节点进行比较，调用7次play函数。再判断如果是奇数个节点的话，更新temp=low1+3，并且与最底层最后一个已经比较过的父节点比较，然后更新各个父节点；如果有偶数个节点，那么temp=low1+2。最后处理剩余的外部节点。最后将leave[1]赋给tree[0].时间复杂度为O(nlog n)，需要操作全局所有顶点。  **更新函数replay(int theplayer, T newvalue):**首先计算将要比赛的场次和左儿子右儿子，分为比赛节点在最下层和在次下层分别更新。分情况，如果改变的是最后的赢家，需要从底层一直回溯到根节点，逐层比较新旧胜者；如果修改的是普通节点，那么仅需调整当前路径的父节点。相对于initialize函数时间复杂度更低为O(log n)。   * 外排序external\_sort():初始化输入各个数字模仿磁盘中的数据。根据选择的缓冲区构   造一个败者树tree1和选手数组playerss，将数据输入缓冲区，并且初次id初始化都为1。然后进行败者树初始化，利用败者树输出最小的数据，再从磁盘块中读取下一个数据进入，当读入的数据比这个输出的最小的数据大时沿用当前的顺串编号id，否则新输入的数据id号加一，说明是下一顺串的。依此类推，生成各个顺串，生成的顺串数就是实际归并路数。  再实现多路归并，还是利用败者树tree2，依次从各个归并路中读取数据，访存次数+1，输出最小的数据，最终输出排序好的序列，访存次数+1。如果归并路数为1的话，那么将该归并路直接输出到结果即可访存次数每次+2。   1. 测试结果（测试输入，测试输出）   测试结果如图1：    图 1   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   结果中访存次数出错，循环次数设置错误或者无效的读写，正确的访存次数应为84+70=154次。   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   struct player {  int id;//编号  int element;//元素  bool operator<=(const player& p) const {  if (id != p.id) {  return id < p.id;  } else {  return element <= p.element;  }  }  };  template<class T>  class losertree {  public:  virtual ~losertree(){}  virtual void initialize(T\* theplayer, int number) = 0;  virtual int getthewinner() const = 0;  virtual void replay(int theplayer, T newvalue) = 0;  };  template<class T >  class minlosertree:public losertree<T> {  private:  int\* tree;//内部节点，tree[0]是最终的赢者下标  int\* leave;//外节点，记录晋级的成员  T\* players;//参加比赛的元素  int k;//选手个数  int file\_count;//文件数目  int low1;//最底层外部节点的个数，2\*（n-s）  int offset;//2\*s-1  void play(int p, int leftchild, int rightchild);  int loser(int x, int y) {//分数小的获胜，返回分数小的  return players[x] <= players[y] ? y : x;  }  int winner(int x, int y) {  return players[x] <= players[y] ? x : y;  }  public:  minlosertree(T\* theplayer=NULL,int thenumberofplayers=0){  tree = NULL;  leave = NULL;  initialize(theplayer, thenumberofplayers);  }  ~minlosertree() {  delete[] tree;  delete[] leave;  }  void initialize(T\* theplayer, int number);  int getthewinner() const {  return tree[0];  }  void replay(int theplayer, T newvalue);    };  template<class T>  void minlosertree<T>::play(int p, int leftchild, int rightchild) {  tree[p] = loser(leftchild, rightchild);  leave[p] = winner(leftchild, rightchild);  while (p % 2 == 1 && p > 1) {  tree[p / 2] = loser(leave[p - 1], leave[p]);  leave[p / 2] = winner(leave[p - 1], leave[p]);  p = p / 2;  }  }  template<class T>  void minlosertree<T>::initialize(T\* theplayer, int number) {  int n = number;  if (n < 2) {  cout << "error!" << endl;  return;  }  players = theplayer;//需要比较的元素  k = n;  delete[] tree;  delete[] leave;  tree = new int[n + 1];  leave = new int[n + 1];  int s;  for (s = 1; 2 \* s <= n - 1; s += s);//找到最底层最左端的内部节点  low1 = 2 \* (n - s);//最底层外部节点的数量(偶数个）  offset = 2 \* s - 1;//计算父节点的索引  for (int i = 2; i <= low1; i += 2) //比较n-s次  play((i + offset) / 2, i - 1, i);//全部比较完了  int temp = 0;  if (n % 2 == 1) {//如果有奇数个节点  play(n / 2, leave[n - 1], low1 + 1);//比较更新最后的父节点  temp = low1 + 3;  }  else  temp = low1 + 2;  for (int i = temp; i <= n; i += 2) {//处理剩余的外部节点  play((i - low1 + n - 1) / 2, i - 1, i);  }  tree[0] = leave[1];  };  template<class T>  void minlosertree<T>::replay(int theplayer, T newvalue) {  int n = k;  if (theplayer <= 0 || theplayer > n) {  cout << "error!!" << endl;  return;  }  players[theplayer] = newvalue;  int matchs, leftchild, rightchild;//将要比赛的场次，左儿子、右儿子  if (theplayer <= low1) {//如果要比赛的结点在最下层  matchs = (offset + theplayer) / 2;  leftchild = 2 \* matchs - offset;  rightchild =leftchild+1;  }  else {//要比赛的结点在次下层  matchs = (theplayer - low1 + n - 1) / 2;  if (2 \* matchs == n - 1) {//特殊情况，其中一方是晋级后的人  leftchild = leave[2 \* matchs];  rightchild = theplayer;  }  else {  leftchild = 2 \* matchs - n + 1 + low1;  rightchild = leftchild + 1;  }  }  //确定要比赛的场次以及比赛的选手  if (theplayer == tree[0]) {//如果改变的是最后的赢家  for (; matchs >= 1; matchs /= 2) {  int oldlosernode = tree[matchs];  tree[matchs] = loser(oldlosernode, theplayer);  leave[matchs] = winner(oldlosernode, theplayer);//比较旧败者和新胜者  theplayer = leave[matchs];//向上传递，继续处理更高层父节点  }  }  else {  //更新当前父节点的败者和胜者  tree[matchs] = loser(leftchild, rightchild);  leave[matchs] = winner(leftchild, rightchild);  //处理奇数节点的情况  if (matchs == n - 1 && n % 2 == 1) {  matchs /= 2;  tree[matchs] = loser(leave[n - 1], low1 + 1);  leave[matchs] = winner(leave[n - 1], low1 + 1);  }  //向上更新所有祖先节点  matchs /= 2;  for (; matchs >= 1; matchs /= 2) {  tree[matchs] = loser(leave[matchs \* 2], leave[matchs \* 2 + 1]);  leave[matchs] = winner(leave[matchs \* 2], leave[matchs \* 2 + 1]);  }  }  tree[0] = leave[1];  }  void external\_sort() {  string path;  string infile;  string outfile;  int buffer=0;//缓冲区大小  int numbers = 0;//个数  int count = 0;//访问磁盘次数  //string filename = "input.txt";  cout << "请输入想要模拟的磁盘文件路径(例如C:/Users/hit/Desktop/,若选择默认测试样例文件路径请输入0):";  cin >> path;  cout << "请输入缓冲区大小：";  cin >> buffer;  if (path == "0")  path = "";  infile = path + "input.txt";  outfile = path + "output.txt";  ofstream file(infile);  if (!file.is\_open()) {  cerr << "无法打开txt!!" << endl;  return;  }  cout << "随机创建文本请输入0,手动输入请输入1:";  int a = -1;  cin >> a;  if (a == 0) {  cout << "请输入想要在磁盘中初始化数字的个数：";  cin >> numbers;  cout << "磁盘初始内容为:";  for (int i = 1; i <= numbers; i++) {  int x = gen() % 1000 + 1;  file << x<<' ';  cout << x << ' ';  }  }  else if(a==1){  cout << "请输入想要在磁盘中初始化数字的个数：";  cin >> numbers;  cout << "请输入：";  for (int i = 1; i <= numbers; i++) {  int x;  cin >> x;  file << x << ' ';  }  }  file.close();  ifstream fin(infile);//打开文件  player\* playerss=new player[numbers+1];  //int runnum = (numbers + buffer - 1) / buffer;//总共的run文件数量  for (int i = 1; i <= buffer&&fin>>playerss[i].element; i++) {  playerss[i].id = 1;  count++;  }//依次读取磁盘中的数字进入缓冲区  minlosertree<player> tree1(playerss, buffer);  int num = 0;  int merger = 0;  for (int i = 0; i < numbers; i++) {  if (!(fin >> num)) {  num = INT\_MAX; // 使用 INT\_MAX 作为文件结束标志  } else {  count++;  }  player tempwinner;  tempwinner.element = playerss[tree1.getthewinner()].element;  tempwinner.id = playerss[tree1.getthewinner()].id;  player tempnum;  tempnum.element = num;    if (num != INT\_MAX && num >= tempwinner.element) {  tempnum.id = tempwinner.id; // 沿用当前顺串编号  } else {  tempnum.id = tempwinner.id + 1; // 生成新顺串  }    tree1.replay(tree1.getthewinner(), tempnum);  string test1 = path + "test" + to\_string(tempwinner.id) + ".txt";  merger = max(merger, tempwinner.id); // 更新归并路数  ofstream fout(test1, ios::app);  if (num != INT\_MAX) {  fout << tempwinner.element << ' '; // 写入实际数据    }  fout.close();  count++;  }  fin.close();  cout<<endl;  cout << "顺串生成完毕,一共生成"<<merger<<"个顺串如下：" << endl;  for (int i = 1; i <= merger; i++) {  string test= path + "test" + to\_string(i) + ".txt";  ifstream fin(test);  int temp = 0;  cout << "test" + to\_string(i) + ".txt:";  while (fin >> temp) {  cout << temp << ' ';  }  cout << endl;  fin.close();  }  ofstream fout(outfile);  if (merger == 1) {  string test = path + "test" + to\_string(1) + ".txt";  ifstream fin(test);  int tempnumber;  cout << "最终排序结果如下：";  while (fin >> tempnumber) {  fout << tempnumber << ' ';  cout << tempnumber << ' ';  count += 2;  }  fout.close();  fin.close();  cout<<"访存次数为："<<count<<endl;  return;  }  else {  //cout << "将这些文件进行" << merger << "路归并，结果如下：" << endl;  int\* dplayer = new int[merger + 10];  int\* pointer = new int[merger + 10];  for (int i = 1; i <= merger; i++) {//分别从各个顺串中读取数据进行排序  string test = path + "test" + to\_string(i) + ".txt";  ifstream fin(test);  fin >> dplayer[i];  pointer[i] = fin.tellg();  fin.close();  count++;  }  minlosertree<int> tree2(dplayer, merger);  int cnt = 0;  while (cnt < numbers) {  cnt++;  int temp = tree2.getthewinner();//得到胜者后  int tempwinner = dplayer[temp];  if(tempwinner!=INT\_MAX){fout << tempwinner << ' ';}  //cout << tempwinner << ' ';  count++;  string test = path + "test" + to\_string(temp) + ".txt";//继续读入新的数据进行排序  ifstream fin(test);  fin.clear();  fin.seekg(pointer[temp] + 1);  int tempnum;  if (pointer[temp] + 1 == 0)  tempnum = INT\_MAX;  else {  fin >> tempnum;  pointer[temp] = fin.tellg();//保存上次在顺串中读到的位置  if (pointer[temp] + 1 == 0)  tempnum = INT\_MAX;  }  tree2.replay(temp, tempnum);//重新得出胜者  fin.close();  count++;  if(tempwinner==INT\_MAX){  break;  }    }  cout << "将这些文件进行" << merger << "路归并，结果如下：" << endl;  int t;  fout.close();  ifstream fout(outfile);  while(fout>>t){cout << t << ' ';}  fout.close();  cout<<"访存次数为："<<count<<endl;  }  } | | | |