**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：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 上机学时：4 | | 日期： 2019.4.3 | |
| 课程设计题目：基本数据结构及其应用之外排序 | | | |
| 软件开发环境：  Microsoft Visual Studio 2017 | | | |
| 报告内容：  1.需求描述  1.1 问题描述  应用竞赛树结构模拟实现外排序。  1.2 基本要求  （1） 设计实现最小竞赛树结构。  （2） 设计实现外排序，外部排序中的生成最初归并串 以及K路归并都应用竞赛树结构实现。  （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  无  1.4 输出说明  无  2.设计  2.1 系统结构设计  使用文件读入，文件输出，  使用竞赛树，竞赛树为输者树。  2.2 设计思路  利用竞赛树处理读入的数据，将其分成诺干个有序的归并串存放在文件中，最后进行K路归并将多个文件合并成一个有序的文件。  2.3 数据及数据类(型)定义  竞赛树的类型为一个结构体，结构体中有两个元素v和id分别表示读入元素的权值以及它属于哪个文件。该结构体排序时以id为第一关键字，v为第二关键字。  2.4.算法设计及分析  （各模块算法及类内函数的算法伪码表示）  读入元素的处理：  由于内存中只能存在ROOM个元素，首先读入ROOM个元素，然后利用竞赛树取得最小值，并将其放入id对应的文件中，在读入下一个元素。如果读入的下一个元素比刚才写出的元素大，则将其归为同一文件即id相同，否则将其归为下一文件即id加1，如此循环直至文件中的元素全部读入。此时竞赛树中还有ROOM个元素，在按照相同的方法将其放入对应的文件夹中即可获得所有的归并串。    K路归并：  首先统计文件个数maxID以及每个文件中的元素个数，然后分配每个文件会有多少个元素出现在竞赛树中，方法为平均分配，多余的再依次分配。归并时每次弹出最小值将其写入目的文件中，再从相同的文件中读入一个元素。不断循环直至竞赛树为空。由于归并串是有序的，最终的结果也是有序的。    3. 测试结果  输入文件内容    输出文件内容    完成了排序任务，且内部空间占用较小。  4. 分析与探讨  5. 附录：实现源代码  /\*minWinnerTree.h\*/  #ifndef \_minWinnerTree  #define \_minWinnerTree  #include <stdexcept>  #include <iostream>  using namespace std;  template<typename T>  class minWinnerTree {  public:  minWinnerTree(T\* thePlayer, int n) {  tree = nullptr;  build(thePlayer, n);  };  ~minWinnerTree() {  delete[] tree;  };  void build(T\*, int);  void replay(int);  int winner(int i = 1) const {  return i < n ? tree[i] : 0;  };  private:  int lowExt;  int offset;  int \*tree;  int n;  T \*player;  void play(int, int, int);  };  template<typename T>  void minWinnerTree<T>::build(T\* thePlayer, int n) {  if (n < 2) throw std::logic\_error("must have at least 2 players");  player = thePlayer;  this->n = n;  delete[] tree;  tree = new int[n];  int i, s;  for (s = 1; 2 \* s <= n - 1; s += s);  lowExt = 2 \* (n - s);  offset = 2 \* s - 1;  for (i = 2; i <= lowExt; i += 2) play((offset + i) / 2, i - 1, i);  if (n & 1) {  play(n / 2, tree[n - 1], lowExt + 1);  i = lowExt + 3;  }  else i = lowExt + 2;  for (; i <= n; i += 2) play((i - lowExt + n - 1) / 2, i - 1, i);  }  template<typename T>  void minWinnerTree<T>::play(int p, int l, int r) {  tree[p] = player[l] <= player[r] ? l : r;  while ((p & 1) && p > 1) {  tree[p >> 1] = player[tree[p - 1]] <= player[tree[p]] ? tree[p - 1] : tree[p];  p >>= 1;  }  }  template<typename T>  void minWinnerTree<T>::replay(int k) {  if (k <= 0 || k > n) return;  int cur, l, r;  if (k <= lowExt) {  cur = (offset + k) / 2;  l = 2 \* cur - offset;  r = l + 1;  }  else {  cur = (k - lowExt + n - 1) / 2;  if (2 \* cur == n - 1) {  l = tree[2 \* cur];  r = k;  }  else {  l = 2 \* cur - n + 1 + lowExt;  r = l + 1;  }  }  tree[cur] = player[l] <= player[r] ? l : r;  if (cur == n - 1 && (n & 1)) {  cur >>= 1;  tree[cur] = player[tree[n - 1]] <= player[lowExt + 1] ? tree[n - 1] : lowExt + 1;  }  cur >>= 1;  for (; cur >= 1; cur >>= 1) tree[cur] = player[tree[cur << 1]] <= player[tree[cur << 1 | 1]] ? tree[cur << 1] : tree[cur << 1 | 1];  }  #endif  /\*playerNode.h\*/  #ifndef \_playerNode  #define \_playerNode  template<typename T>  struct playerNode {  int id;  T key;  bool operator <= (const playerNode& A) const {  if (id != A.id) return id <= A.id;  return key <= A.key;  }  };  #endif  /\*externalSorter.h\*/  #ifndef \_externalSorter  #define \_externalSOrter  #include <fstream>  #include <algorithm>  #include <stdexcept>  #include "minWinnerTree.h"  #include "playerNode.h"  #define ROOM 10  template<typename T>  class externalSorter {  public:  externalSorter(const char\* inputFile, const char\* outputFile, int n) : inputFile(inputFile), outputFile(outputFile), n(n) { cnt = new int[n + 1]; }  ~externalSorter() { delete[] cnt; }  bool sort() {  try {  run();  KwayMerge();  for (int i = 1; i <= maxID; ++i) remove((to\_string(i) + ".tmp").c\_str());  }  catch (exception) {  return false;  }  return true;  };  protected:  void run();  void KwayMerge();  const char \*inputFile, \*outputFile;  int n;  int maxID;  int \*cnt;  };  template<typename T>  void externalSorter<T>::run() {  playerNode<int> \*player = new playerNode<T>[ROOM + 1];  ifstream fin(inputFile);  int driver = 0;  for (int i = 1; i <= ROOM; ++i) {  fin >> player[i].key;  driver++;  player[i].id = 1;  }  minWinnerTree<playerNode<T>> winnerTree(player, ROOM);  maxID = 0;  for (int i = 1; i <= n; ++i) cnt[i] = 0;  for (int i = ROOM + 1; i <= n; ++i) {  int winner = winnerTree.winner();  int winnerID = player[winner].id;  cnt[winnerID]++;  if (maxID < winnerID) maxID = winnerID;  string fileName = to\_string(winnerID) + ".tmp";  ofstream fout(fileName.c\_str(), ios::app);  if (cnt[winnerID] > 1) fout << ' ';  fout << player[winner].key;  fout.close();  int key;  fin >> key;  if (key < player[winner].key) player[winner].id += 1;  player[winner].key = key;  winnerTree.replay(winner);  driver += 2;  }  for (int i = 1; i <= ROOM; ++i) {  int winner = winnerTree.winner();  int winnerID = player[winner].id;  cnt[winnerID]++;  if (maxID < winnerID) maxID = winnerID;  string fileName = to\_string(winnerID) + ".tmp";  ofstream fout(fileName.c\_str(), ios::app);  if (cnt[winnerID] > 1) fout << ' ';  fout << player[winner].key;  fout.close();  player[winner].key = 0;  player[winner].id = n + 1;  winnerTree.replay(winner);  driver += 1;  }  fin.close();  delete[] player;  cout << "run: " << driver << endl;  }  template<typename T>  void externalSorter<T>::KwayMerge() {  int div = ROOM / maxID;  int mod = ROOM % maxID;  playerNode<int> \*player = new playerNode<int>[ROOM + 1];  ifstream \*\_fin = new ifstream[maxID + 1];  for (int i = 1; i <= maxID; ++i) cnt[i] = min(cnt[i], div + (i <= mod ? 1 : 0));  for (int i = 1; i <= maxID; ++i) \_fin[i] = ifstream(to\_string(i) + ".tmp");  for (int i = 1; i <= ROOM; ++i) player[i].id = 2;  int sum = 0;  int driver = 0;  int \*belong = new int[n + 1];  for (int i = 1; i <= maxID; ++i) {  for (int j = 1; j <= cnt[i]; ++j) {  \_fin[i] >> player[sum + j].key;  driver++;  player[sum + j].id = 1;  belong[sum + j] = i;  }  sum += cnt[i];  }  minWinnerTree<playerNode<int> > winnerTree(player, ROOM);  ofstream fout(outputFile);  for (int i = 1; i <= n; ++i) {  int winner = winnerTree.winner();  int winnerID = belong[winner];  fout << player[winner].key << ' ';  driver++;  if (\_fin[winnerID].peek() != EOF) {  \_fin[winnerID] >> player[winner].key;  driver++;  }  else player[winner].id = 2;  winnerTree.replay(winner);  }  fout.close();  delete[] belong;  delete[] player;  delete[] \_fin;  cout << "k-way merge: " << driver << endl;  }  #endif  /\*main.cpp\*/  #include <iostream>  #include <string>  #include "externalSorter.h"  using namespace std;  int main() {  int n;  string input, output;  cout << "Please input the name of inputFile, the name of outputFile and the number of elements, splited by space:" << endl;  cin >> input >> output >> n;  externalSorter<int> sorter(input.c\_str(), output.c\_str(), n);  if (sorter.sort()) cout << "success" << endl;  else cout << "failed" << endl;  return 0;  } | | | |