**《（课程名称）》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2020级计算机科学与技术01班** | | | **姓名** | **陈鹏宇** |
| **实验题目** | **有理数的堆排序** | | | | | |
| **实验时间** | **2021/12/14** | | **实验地点** | **DS1408** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  1. 掌握最小堆的基本概念，堆的基本运算以及堆排序的流程  2. 掌握有理数类的定义及逻辑运算规则 | | | | | | |
| 二、实验项目内容  1. 完成有理数的类定义以及有理数逻辑运算函数  class Rational{  friend bool operator<(const Rational& r1, const Rational& r2) {}  friend bool operator<=(const Rational& r1, const Rational& r2) {}  friend bool operator>(const Rational& r1, const Rational& r2) {}  friend bool operator>=(const Rational& r1, const Rational& r2) {}  friend bool operator==(const Rational& r1, const Rational& r2) {}  friend bool operator!=(const Rational& r1, const Rational& r2) {}  public:  int N; //分子  int D; //分母， 要求大于0    Rational() {} //default constructor  Rational(int n){} //constructor for integer value  Rational(int n, int d) {} //normal constructor  Rational(const Rational& r){} //copy constructor  Rational& operator=(const Rational& r) {} // assignment override  }；  2. 创建有理数的最小堆，实现siftdown, siftup, insert等功能  3． 实现基于最小堆的堆排序，按从小到大的顺序输出有理数  4． 为在线测评系统检测程序的运行，对程序文档及IO做如下规范：  （1）所有类、函数及主程序都写在一个**单cpp文档**里，不能有其他include用的.h或.cpp文档  （2）程序不能输出任何提示用的字符串  （3）输入： 第一行包含一个整数T (1≤T≤105)；接下来的T行，每一行有两个整数n, d (|n|<103, 0<d<103)，用空格隔开，表示输入的有理数的分子和分母。  （4）输出：第一行输出有理数的最小堆序列，第二行输出从小到大排序后的序列。  （5）输出的每个有理数必须规约，以n/d的形式输出，其中d>0且gcd(n,d)=0；如果d=1或n=0则直接输出n  （例子）  输入：  5 //有理数个数  3 2 //第1个有理数 3/2  1 3 //第2个有理数 1/3  4 2 //第3个有理数 4/2  12 10 //第4个有理数 12/10  4 6 //第5个有理数 4/6  输出：  1/3 2/3 2 6/5 3/2 //第一行 最小堆  1/3 2/3 6/5 3/2 2 //第二行 排序结果 | | | | | | |
| 1. 实验过程或算法（源程序）   #include <iostream>  #include <bits/stdc++.h>  using namespace std;  class Rational{ //有理数类定义  friend bool operator<(const Rational& r1, const Rational& r2)  {  return (result(r1) < result(r2));  }  friend bool operator<=(const Rational& r1, const Rational& r2)  {  return (result(r1) <= result(r2));  }  friend bool operator>(const Rational& r1, const Rational& r2)  {  return (result(r1) > result(r2));  }  friend bool operator>=(const Rational& r1, const Rational& r2)  {  return (result(r1) >= result(r2));  }  friend bool operator==(const Rational& r1, const Rational& r2)  {  return (result(r1) == result(r2));  }  friend bool operator!=(const Rational& r1, const Rational& r2)  {  return (result(r1) != result(r2));  }  public:  int N; //分子  int D; //分母， 要求大于0  void huajian(int &n , int &d)  {  int a = min(n,d);  for(int i = a ; i > 1 ; i--)  {  if((n % i == 0) && (d % i) == 0)  {  n /= i;  d /= i;  }  }  }  Rational() {} //default constructor  Rational(int n) //constructor for integer value  {  N = n ; D = 1;  }  Rational(int n, int d)//normal constructor  {  huajian(n , d);  N = n ; D = d;  }  Rational(const Rational& r)//copy constructor  {  N = r.N ; D = r.D;  }  Rational& operator=(const Rational& r)// assignment override  {  Rational \*t = new Rational(r.N , r.D);  return \*t;  }  double static result(const Rational& r1)  {  return (double)r1.N/r1.D;  }  };  void Assert(bool val, std::string s)  {  if (!val) {  std::cout << "Assertion Failed: " << s << std::endl;  exit(-1);  }  }  template <typename E>  void swap(E\* Heap, int pos1, int pos2)  {  E temp = Heap[pos1];  Heap[pos1] = Heap[pos2];  Heap[pos2] = temp;  }  template <typename E>  class heap  {  private:  E\* Heap;  int maxsize;  int n; //当前堆内元素个数  void siftdown(int pos){  // cout<<pos<<endl;  while(!isLeaf(pos)){  // cout<<"current: "<<Heap[pos]<<endl;  int j = leftchild(pos);  int rc = rightchild(pos);  // cout<<j<<"-"<<Heap[j]<<endl;  if(rc < n)  // cout<<rc<<"-"<<Heap[rc]<<endl;  if((rc < n) && ((Heap[rc]->result(\*Heap[rc])) < (Heap[j]->result(\*Heap[j])))) {  j = rc;  // cout<<"right better"<<endl;  }  if((Heap[pos]->result(\*Heap[pos])) < (Heap[j]->result(\*Heap[j]))) {  // cout<<"pos not change"<<endl;  return;  }  swap(Heap , pos , j);  pos = j;  // cout<<"siftdown called"<<endl;  }  }  public:  heap(E\* h , int num , int max){ Heap = h; n = num; maxsize = max; buildHeap(); }  int size() const{ return n; }  bool isLeaf(int pos) const{ return (pos >= n/2 && pos < n); }  int rightchild(int pos){ return 2\*pos + 2; }  int leftchild(int pos){ return 2\*pos + 1; }  int parent(int pos){ return (pos - 1) / 2; }  void buildHeap(){  for(int i = n/2 - 1 ; i >= 0 ; i--){  siftdown(i);  }  }  void insert(const E& it){  Assert(n < maxsize , "Heap is full");  int curr = n++;  Heap[curr] = it;  while((curr != 0) && ((Heap[curr]->result(\*Heap[curr])) < (Heap[parent(curr)]->result(\*Heap[parent(curr)])))){  swap(Heap , curr , parent(curr));  curr = parent(curr);  }  }  E removefirst(){  Assert(n > 0 , "Heap is empty");  swap(Heap , 0 , --n);  if(n != 0)  siftdown(0);  return Heap[n];  }  E remove(int pos){  Assert(n >= 0 , "Bad position");  if(pos == n - 1) n--;  else{  swap(Heap , pos , --n);  while((pos != 0) && ((Heap[pos]->result(\*Heap[pos])) < (Heap[parent(pos)]->result(\*Heap[parent(pos)])))){  swap(Heap , pos , parent(pos));  pos = parent(pos);  }  if(n != 0)  siftdown(pos);  }  return Heap[n];  }  };  int main()  {  /\* test Rational class  Rational a(2,3);  Rational b(4,6);  Rational c(1,3);  Rational d(2,1);  Rational e(4,2);  cout<<(a==b)<<endl;  cout<<(d!=e)<<endl;  cout<<(a>c)<<endl;  cout<<(b<c)<<endl;  \*/  /\* Heap check  Rational\* tree[5];  for(int i = 0 ; i < 5 ; i++){  int n ; int d ; cin >> n >> d;  Rational\* t1 = new Rational(n , d);  tree[i] = t1;  }  heap<Rational\*>(tree,5,5);  for(int i = 0 ; i < 5 ; i++){  cout<<tree[i]->result(\*tree[i])<<" ";  }  \*/  /\*project\*/  int n; cin >> n;  Rational\* tree[n];  for(int i = 0 ; i < n ; i++){  int n ; int d ; cin >> n >> d;  Rational\* t1 = new Rational(n , d);  tree[i] = t1;  }  heap<Rational\*> h(tree,n,n);  for(int i = 0 ; i < n ; i++){  if(tree[i]->D != 1)  cout<<tree[i]->N<<"/"<<tree[i]->D<<" ";  else  cout<<tree[i]->N<<" ";  }  cout<<endl;  for(int i = 0 ; i < n ; i++){  Rational\* t = h.removefirst();  if(t->D != 1)  cout<<t->N<<"/"<<t->D<<" ";  else  cout<<t->N<<" ";  }  } | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   为了方便Rational类的比较，加入了result成员函数来得到有理数的double值。    验证Rational有理数类是否实现成功  Z%~9HU_E@%5J2V5~GXXPO`L  发现判断模块实现成功，现开始堆的设计  IMG_256  发现堆的序列与理论序列一致。  现 实现题目要求，发现需要以最简分数形式输出，若分母唯一，则直接输出整数，所以添加了简易的（效率不高的）huajian成员函数    添加了最后输出的判断模块，若分母为一则输出整数    最后得到目标结果  IMG_256 | | | | | | |

`