山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名：黄瑞哲 | | 班级：计科17.3 |
| 实验题目：堆及其应用 | | | |
| 实验学时：2 | | 实验日期：2018.12.6 | |
| 实验目的：   1. 掌握堆结构的定义、描述方法、操作定义的实现 2. 掌握堆结构的应用 | | | |
| 软件环境：  Visual Studio Community 2017 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求） 2. 创建最小堆类，最小堆的储存结构用数组。提供操作：插入、删除、初始化 3. 接受键盘录入的一系列整数，以文本的方式输出其对应的最小堆。 4. 对建立好的最小堆，键盘输入插入元素，输出插入操作完成后的堆（可以文本形式表示）：键盘输入删除元素，输出删除操作完成后的堆。 5. 键盘输入n，随机生成n个0-1000之间的整数，输出堆排序的排序过程 6. 键盘输入字符个数n，以(c, w)的形式依次表示字符和字符出现的频率，字符互不相同，输出Huffman树（可用文本形式）和每个字符的Huffman编码。 7. 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） 8. 堆是一种完全二叉树，可以用数组描述。数组中第k个元素的左右儿子分别为2\*k, 2\*k+1，他的父亲为[k/2]。 9. 堆的插入：将元素放到数组末尾并不断向上调整，直至满足堆的性质。 10. 堆的删除：找到删除元素后将其删除，并将数组末尾的元素放置该位置，并对这棵子树进行调整，直至满足堆的性质。 11. 堆排序：建立最大堆，每次取top后将其放在数组的第i位上，i从n减到1。 12. Huffman树：初始有n棵树，将其放到堆中，每次取两个top后将其合成一棵树并再次放入堆中，直至堆中只剩一个元素。 13. 测试结果（测试输入，测试输出，结果分析）      1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   Huffman树的结果与手工模拟的结果一致  堆排序结果为递增数列  插入元素和删除元素后的结果与手工模拟的结果一致   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）   /\*Heap.h\*/  #ifndef Heap\_  #define Heap\_  #include <stdexcept>  #include <cstring>  #include <iostream>  template<typename T, typename Pr = std::greater<T> >  class Heap {  public:  Heap(int = 10);  Heap(const Heap<T, Pr>&);  ~Heap() {  delete[] heap;  }  bool empty() const { return heapSize == 0; }  int size() const { return heapSize; }  const T& top() {  if (heapSize < 1) throw std::out\_of\_range("heap is empty");  return heap[1];  }  void pop();  void push(const T&);  void init(T\*, T\*);  void erase(const T&);  void output(std::ostream&, char = '\n');  protected:  int heapSize;  int arrayLength;  T\* heap;  Pr cmp;  };  template<typename T, typename cmp>  Heap<T, cmp>::Heap(int initialCapacity) : cmp() {  if (initialCapacity < 1) throw std::out\_of\_range("the initial Capacity of heap mut > 0");  arrayLength = initialCapacity + 1;  heap = new T[arrayLength];  heapSize = 0;  }  template<typename T, typename Pr>  Heap<T, Pr>::Heap(const Heap<T, Pr>& theHeap) {  arrayLength = theHeap.arrayLength;  heapSize = theHeap.heapSize;  heap = new T[arrayLength];  copy(theHeap.heap, theHeap.head + heapSize + 1, heap);  }  template<typename T, typename Pr>  void Heap<T, Pr>::pop() {  heap[1] = heap[heapSize--];  int k = 1, c = 2;  while (c <= heapSize) {  if (c < heapSize && !cmp(heap[c], heap[c | 1])) c |= 1;  if (cmp(heap[k], heap[c])) break;  std::swap(heap[k], heap[c]);  k = c;  c <<= 1;  }  }  template<typename T, typename Pr>  void Heap<T, Pr>::push(const T& element) {  if (heapSize >= arrayLength - 1) {  T\* temp = new T[arrayLength \* 2];  for (int i = 1; i <= heapSize; ++i) temp[i] = heap[i];  delete[] heap;  heap = temp;  arrayLength \*= 2;  }  heap[++heapSize] = element;  int k = heapSize;  while (k != 1) {  if (cmp(heap[k], heap[k >> 1])) std::swap(heap[k], heap[k >> 1]);  k >>= 1;  }  }  template<typename T, typename Pr>  void Heap<T, Pr>::init(T\* begin, T\* end) {  delete[] heap;  heapSize = end - begin;  arrayLength = heapSize;  heap = new T[arrayLength + 1];  for (int i = 1; i <= heapSize; ++i) heap[i] = \*(begin + i - 1);  for (int rt = heapSize / 2; rt; --rt) {  int k = rt;  int c = k << 1;  while (c <= heapSize) {  if (c < heapSize && !cmp(heap[c], heap[c | 1])) c |= 1;  if (cmp(heap[k], heap[c])) break;  std::swap(heap[k], heap[c]);  k = c;  c <<= 1;  }  }  }  template<typename T, typename Pr>  void Heap<T, Pr>::erase(const T& element) {  int k = heapSize;  for (int i = 1; i <= heapSize; ++i) {  if (heap[i] == element) {  heap[i] = heap[heapSize];  heap[heapSize--].~T();  k = i;  break;  }  }  int c = k << 1;  while (c <= heapSize) {  if (c < heapSize && !cmp(heap[c], heap[c << 1])) c |= 1;  if (cmp(heap[k], heap[c])) break;  std::swap(heap[k], heap[c]);  k = c;  c <<= 1;  }  }  template<typename T, typename Pr>  void Heap<T, Pr>::output(std::ostream& out, char ends) {  for (int i = 1; i <= heapSize; ++i) out << heap[i] << ' ';  out << ends;  }  #endif //Heap\_  /\*Huffman.h\*/  #ifndef Huffman\_  #define Huffman\_  #include <utility>  #include <queue/linkedQueue.h>  #include "Heap.h"  #include "binaryTreeNode.h"  using namespace std;  template<typename T>  class Compare {  typedef pair<T, int> pti;  public:  bool operator()(binaryTreeNode<pti>\* a, binaryTreeNode<pti>\* b) const {  return a->element.second < b->element.second;  }  };  template<typename T>  class Huffman {  typedef pair<T, int> pti;  private:  Heap<binaryTreeNode<pti>\*, Compare<T>> h;  binaryTreeNode<pti>\* root;  public:  Huffman(int n, pti\* a);  ~Huffman();  binaryTreeNode<pti>\* create();  void preOrder(binaryTreeNode<pti>\*);  void inOrder(binaryTreeNode<pti>\*);  void postOrder(binaryTreeNode<pti> \*);  void levelOrder(binaryTreeNode<pti>\*);  void destroy(binaryTreeNode<pti>\*);  };  template <typename T>  Huffman<T>::Huffman(int n, pti\* a) {  while (!h.empty()) h.pop();  for (auto i = 0; i < n; ++i) h.push(new binaryTreeNode<pair<T, int>>(a[i]));  root = nullptr;  }  template <typename T>  Huffman<T>::~Huffman() {  destroy(root);  }  template <typename T>  binaryTreeNode<pair<T, int> >\* Huffman<T>::create() {  while(h.size() > 1) {  binaryTreeNode<pair<T, int>>\* t1 = h.top(); h.pop();  binaryTreeNode<pair<T, int>>\* t2 = h.top(); h.pop();  binaryTreeNode<pair<T, int>>\* t = new binaryTreeNode<pair<T, int>>(make\_pair(0, t1->element.second + t2->element.second));  t->leftChild = t1;  t->rightChild = t2;  h.push(t);  }  root = h.top(); h.pop();  return root;  }  template <typename T>  void Huffman<T>::preOrder(binaryTreeNode<pti>\* rt) {  if (rt == nullptr) return;  cout << rt->element.second << ' ';  preOrder(rt->leftChild);  preOrder(rt->rightChild);  }  template <typename T>  void Huffman<T>::inOrder(binaryTreeNode<pti>\* rt) {  if (rt == nullptr) return;  preOrder(rt->leftChild);  cout << rt->element.second << ' ';  preOrder(rt->rightChild);  }  template <typename T>  void Huffman<T>::postOrder(binaryTreeNode<pti>\* rt) {  if (rt == nullptr) return;  preOrder(rt->leftChild);  preOrder(rt->rightChild);  cout << rt->element.second << ' ';  }  template <typename T>  void Huffman<T>::levelOrder(binaryTreeNode<pti>\* rt) {  linkedQueue<binaryTreeNode<pti>\*> q;  q.push(rt);  while(!q.empty()) {  binaryTreeNode<pti>\* p = q.front(); q.pop();  if (p == nullptr) continue;  cout << p->element.second << ' ';  q.push(p->leftChild);  q.push(p->rightChild);  }  }  template <typename T>  void Huffman<T>::destroy(binaryTreeNode<pti>\* rt) {  if (rt == nullptr) return;  destroy(rt->leftChild);  destroy(rt->rightChild);  delete rt;  }  #endif //Huffman\_  /\*pch.h\*/  #ifndef PCH\_H  #define PCH\_H  #include <tree/Heap.h>  #include <tree/binaryTreeNode.h>  #include <tree/Huffman.h>  #endif //PCH\_H  /\*exe10.cpp\*/  #include "pch.h"  #include <iostream>  #include <utility>  #include <string>  #include <cstdlib>  #include <ctime>  using namespace std;  void dfs(binaryTreeNode<pair<char, int>>\* t, string str) {  if (t->leftChild == nullptr) {  cout << t->element.first << ' ' << str << endl;  return;  }  else dfs(t->leftChild, str + '0');  if (t->rightChild == nullptr) {  cout << t->element.first << ' ' << str << endl;  return;  }  else dfs(t->rightChild, str + '1');  }  void heapSort() {  int n;  cin >> n;  srand(time(nullptr));  int \*a = new int[n];  for (int i = 0; i < n; ++i) {  a[i] = rand() % 1000;  cout << a[i] << ' ';  }  cout << endl;  Heap<int> h;  h.init(a, a + n);  for(int i=n-1;i>=0;--i) {  int x = h.top();  h.pop();  a[i] = x;  h.output(cout, '|');  for (int j = i; j < n; ++j) cout << ' ' << a[j];  cout << endl;  }  }  void task1() {  int n, x;  cin >> n;  const auto a = new int[n];  for (int i = 0; i < n; ++i) cin >> a[i];  Heap<int, less<>> h;  h.init(a, a + n);  h.output(cout);  cout << "插入的元素: ";  cin >> x;  h.push(x);  h.output(cout);  cout << "删除的元素: ";  cin >> x;  h.erase(x);  h.output(cout);  }  void task3() {  int n;  cout << "字符个数: ";  cin >> n;  const auto a = new pair<char, int>[n];  for (int i = 0; i < n; ++i) {  char c;  int w;  cin >> c >> w;  a[i] = make\_pair(c, w);  }  Huffman<char> huff(n, a);  binaryTreeNode<pair<char, int>>\* root = huff.create();  cout << "层次遍历: ";  huff.levelOrder(root);  cout << endl;  cout << "前序遍历: ";  huff.preOrder(root);  cout << endl;  cout << "编码为:" << endl;  dfs(root, "");  }  int main()  {  task1();  heapSort();  task3();  return 0;  } | | | |