**重 庆 大 学**

**学 生 实 验 报 告**

**实验课程名称 数据结构与算法**

**开课实验室 D1349**

**学 院 软件学院 年级 2016 专业班 软件2班**

**学 生 姓 名 丁子元 学 号 20161616**

**开 课 时 间 2016 至 2017 学年第 1 学期**

|  |  |
| --- | --- |
| **总 成 绩** |  |
| **教师签名** | **文俊浩** |

**软件学院制**

**《数据结构与算法》实验报告**

**开课实验室：DS1501 2017 年 10 月　10 日**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | | 软件学院 | 年级、专业、班 | | 2016级2班 | 姓名 | 丁子元 | | 成绩 | |  |
| 课程  名称 | | 数据结构与算法 | | 实验项目  名 称 | 堆 | | | 指导教师 | | **文俊浩** | |
| 教师评语 | 教师签名：  年 月 日 | | | | | | | | | | |
| **一、实验目的**  1 练习二叉树的C++实现  2 练习堆的C++实现  **二、实验原理**  1 二叉树、二叉检索树的定义和操作方法  2堆的定义和操作方法 | | | | | | | | | | | |

|  |
| --- |
| **三、使用仪器、材料**  Vs2017  **四、实验步骤**  (1) 定义二叉树的结点类BinNode；  (2) 实现二叉检索树的结点类BSTNode ；  (3) 实现二叉检索树BST；  (4) 设计程序对上述实现进行测试，具体要求见教材P126 5.10题。  (1) 参考教材中的代码实现最大堆heap；  (2) 设计程序对最大堆中的insert、removefirst、remove等函数进行测试，记录程序执行结果；  (3) 修改最大堆的代码，实现最小堆；  (4) 设计程序对最小堆中的insert、removefirst、remove等函数进行测试，记录程序执行结果； |

|  |
| --- |
| **五、实验过程原始记录(数据、图表、计算等)**  (1) 定义二叉树的结点类BinNode；  template <typename E>  class BinNode  {  private:  E it; // The node’s value  BinNode\* lc; // Pointer to left child  BinNode\* rc; // Pointer to right child  public:  // Two constructors -- with and without initial values  BinNode() { lc = rc = NULL; }  BinNode(E e, BinNode\* l = NULL, BinNode\* r = NULL)  {  it = e; lc = l; rc = r;  }  // ˜BinNode() {} // Destructor  // Functions to set and return the value and key  E& element() { return it; }  // Functions to set and return the children  inline BinNode\* left() const { return lc; }  void setLeft(BinNode<E>\* b) { lc = (BinNode\*)b; }  inline BinNode\* right() const { return rc; }  void setRight(BinNode<E>\* b) { rc = (BinNode\*)b; }  // Return true if it is a leaf, false otherwise  bool isLeaf() { return (lc == NULL) && (rc == NULL); }  };  (2) 实现二叉检索树的结点类BSTNode ；  #include <iostream>  #include "BinNode.h"  using namespace std;  // Simple binary tree node implementation  template <typename Key, typename E>  class BSTNode : public BinNode<E> {  private:  Key k; // The node’s key  E it; // The node’s value  BSTNode\* lc; // Pointer to left child  BSTNode\* rc; // Pointer to right child  public:  // Two constructors -- with and without initial values  BSTNode() { lc = rc = NULL; }  BSTNode(Key K, E e, BSTNode\* l = NULL, BSTNode\* r = NULL)  {  k = K; it = e; lc = l; rc = r;  }  //˜BSTNode() {} // Destructor  // Functions to set and return the value and key  E& element() { return it; }  void setElement(const E& e) { it = e; }  Key& key() { return k; }  void setKey(const Key& K) { k = K; }  // Functions to set and return the children  inline BSTNode\* left() const { return lc; }  void setLeft(BinNode<E>\* b) { lc = (BSTNode\*)b; }  inline BSTNode\* right() const { return rc; }  void setRight(BinNode<E>\* b) { rc = (BSTNode\*)b; }  // Return true if it is a leaf, false otherwise  bool isLeaf() { return (lc == NULL) && (rc == NULL); }  };  (3) 实现二叉检索树BST；  BSTree<int, int\*> tree;  int n = 11;  srand((unsigned)time(NULL)); //用当前系统时间设置种子    cout << "Size: " << tree.size() << "\n";  tree.insert(15, new int(15));  for (int j = 0; j < n-1; j++)  {  int k = rand() % (n + 1);  tree.insert(k, new int(k));  }    tree.print();  cout << "Size: " << tree.size() << "\n";  tree.remove(15);  tree.print();  cout << "Size: " << tree.size() << "\n";    return 0;  (4) 设计程序对上述实现进行测试，具体要求见教材P126 5.10题。  1将此段代码放在BST.h  template <typename Key, typename E>  int BSTree<Key, E>::Sum(BSTNode<Key, E>\* root,int level) const{  static int sum=0;  if (root == NULL) return sum;  if (root->right()==NULL&&root->left()==NULL) {  sum=sum+root->key();  return sum;  }  if(root->right()!=NULL||root->left()!=NULL)  {  sum=sum+root->key();  Sum(root->right(),level+1);  Sum(root->left(),level+1);  }    return sum;  } 2 将此段代码放在main.cpp  BSTree<int, int\*> tree;  int n = 11;  srand((unsigned)time(NULL)); //用当前系统时间设置种子    cout << "Size: " << tree.size() << "\n";  tree.insert(15, new int(15));  for (int j = 0; j < n-1; j++)  {  int k = rand() % (n + 1);  tree.insert(k, new int(k));  }    tree.print();  cout << "Size: " << tree.size() << "\n";  //tree.remove(15);  // tree.print();  // cout << "Size: " << tree.size() << "\n";  //  cout<<"Sum = "<<tree.Sum(tree.root,0);  return 0;  (1) 参考教材中的代码实现最大堆heap；  private:  E\* Heap; // Pointer to the heap array  int maxsize; // Maximum size of the heap  int n; // Number of elements now in the heap  // Helper function to put element in its correct place  void siftdown(int pos) {  while (!isLeaf(pos)) { // Stop if pos is a leaf  int j = leftchild(pos); int rc = rightchild(pos);  if ((rc < n) && Comp::prior(Heap[rc], Heap[j]))  j = rc; // Set j to greater child’s value  if (Comp::prior(Heap[pos], Heap[j])) return; // Done  swap(Heap, pos, j);  pos = j; // Move down  }  }  public:  heap(E\* h, int num, int max) // Constructor  {  Heap = h; n = num; maxsize = max; buildHeap();  }  int size() const // Return current heap size  {  return n;  }  bool isLeaf(int pos) const // True if pos is a leaf  {  return (pos >= n / 2) && (pos < n);  }  int leftchild(int pos) const  {  return 2 \* pos + 1;  } // Return leftchild position  int rightchild(int pos) const  {  return 2 \* pos + 2;  } // Return rightchild position  int parent(int pos) const // Return parent position  {  return (pos - 1) / 2;  }  void buildHeap() // Heapify contents of Heap  {  for (int i = n / 2 - 1; i >= 0; i--) siftdown(i);  }  // Insert "it" into the heap  void insert(const E& it) {  Assert(n < maxsize, "Heap is full");  int curr = n++;  Heap[curr] = it; // Start at end of heap  // Now sift up until curr’s parent > curr  while ((curr != 0) &&  (Comp::prior(Heap[curr], Heap[parent(curr)]))) {  swap(Heap, curr, parent(curr));  curr = parent(curr);  }  }    // Remove first value  E removefirst() {  Assert(n > 0, "Heap is empty");  swap(Heap, 0, --n); // Swap first with last value  if (n != 0) siftdown(0); // Siftdown new root val  return Heap[n]; // Return deleted value  }  // Remove and return element at specified position  E remove(int pos) {  Assert((pos >= 0) && (pos < n), "Bad position");  if (pos == (n - 1)) n--; // Last element, no work to do  else  {  swap(Heap, pos, --n); // Swap with last value  while ((pos != 0) &&  (Comp::prior(Heap[pos], Heap[parent(pos)]))) {  swap(Heap, pos, parent(pos)); // Push up large key  pos = parent(pos);  }  if (n != 0) siftdown(pos); // Push down small key  }  return Heap[n];  }  (2) 设计程序对最大堆中的insert、removefirst、remove等函数进行测试，记录程序执行结果；  int test[10] = {1,1,3,1,1};  heap<int, int> heap(test,10,11);  heap.insert(2);  for (int i = 0; i <= 10; i++)  cout << test[i] << " ";  cout << "\n" << "insert finish" << "\n";  heap.remove(3);  for (int i = 0; i <= 9; i++)  cout << test[i] << " ";  cout << "\n"<< "remove 3 finish" << "\n";      heap.removefirst();  for(int i=0;i<=8;i++)  cout<< test[i]<<" ";  cout<<"\n"<<"removefirst finish" << "\n";  return 0;  (3) 修改最大堆的代码，实现最小堆；  #include <assert.h>  template <typename E, typename Comp> class heap  {  private:  E\* Heap;  int maxsize;  int n;      void siftdown(int pos) {  while (!isLeaf(pos)) {  int j = leftchild(pos); int rc = rightchild(pos);  if ((rc < n) && (Heap[j]>Heap[rc]))  j = rc;  if ((Heap[j]>Heap[pos] )) return;  swap(Heap[pos], Heap[j]);  pos = j;  }  }    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 leftchild(int pos) const  { return 2\*pos + 1; }  int rightchild(int pos) const  { return 2\*pos + 2; }  int parent(int pos) const  { 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[parent(curr)]> Heap[curr]))) {  swap(Heap[curr], Heap[parent(curr)]);  curr = parent(curr);  }  }  E removefirst() {  assert (n > 0, "Heap is empty");  swap(Heap[0], Heap[--n]);  if (n != 0) siftdown(0);  return Heap[n];  }    E remove(int pos) {  assert((pos >= 0) && (pos < n), "Bad position");  if (pos == (n-1)) n--;  else  {  swap(Heap[pos], Heap[--n]);  while ((pos != 0) &&  ((Heap[parent(pos)]>Heap[pos]))) {  swap(Heap[pos], Heap[parent(pos)]);  pos = parent(pos);  }  if (n != 0) siftdown(pos);  }  return Heap[n];  }  };  (4) 设计程序对最小堆中的insert、removefirst、remove等函数进行测试，记录程序执行结果；  #include <iostream>  #include "minheap.h"  using namespace std;  int main()  {  int test[10] = { 1,1,3,1,1 };  heap<int, int> heap(test, 10, 11);  heap.insert(2);  for (int i = 0; i <= 10; i++)  cout << test[i] << " ";  cout << "\n" << "insert finish" << "\n";  heap.remove(3);  for (int i = 0; i <= 9; i++)  cout << test[i] << " ";  cout << "\n" << "remove 3 finish" << "\n";  heap.removefirst();  for (int i = 0; i <= 8; i++)  cout << test[i] << " ";  cout << "\n" << "removefirst finish" << "\n";  return 0;  } |

|  |
| --- |
| **六、实验结果及分析**  (1) 定义二叉树的结点类BinNode；  (2) 实现二叉检索树的结点类BSTNode ；  (3) 实现二叉检索树BST；    (4) 设计程序对上述实现进行测试，具体要求见教材P126 5.10题。    (1) 参考教材中的代码实现最大堆heap；  (2) 设计程序对最大堆中的insert、removefirst、remove等函数进行测试，记录程序执行结果；    (3) 修改最大堆的代码，实现最小堆；  (4) 设计程序对最小堆中的insert、removefirst、remove等函数进行测试，记录程序执行结果； |