**附件：**

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：202000130138 | 姓名：宋璎航 | | 班级： 20.3 |
| 实验题目：搜索树 | | | |
| 实验学时：2 | | 实验日期： 2021/12/11 | |
| 实验目的：  掌握二叉搜索树结构的定义、描述方法、操作实现。 | | | |
| 软件开发环境：  VSC | | | |
| 1. 实验内容   **1、题目描述**：  创建带索引的二叉搜索树类。存储结构使用链表，提供操作:插入、删除、按名次删除、查找、按名次查找、升序输出所有元素。  **输入输出格式**：  **输入**：  输入第一行一个数字m (m<=1000000)，表示有m个操作。  接下来m行，每一行有两个数字a，b；  当输入的第一个数字 a 为 0 时，输入的第二个数字 b 表示向搜索树中插入 b；  当输入的第一个数字 a 为 1 时，输入的第二个数字 b 表示向搜索树中查找 b；  当输入的第一个数字 a 为 2 时，输入的第二个数字 b 表示向搜索树中删除 b；  当输入的第一个数字 a 为 3 时，输入的第二个数字 b 表示查找搜索树中名次为 b 的元素；  当输入的第一个数字 a 为 4 时，输入的第二个数字 b 表示删除搜索树中名次为 b 的元素；  **输出**：  对于输入中的每一种操作，输出执行操作的过程中依次比较的元素值的异或值。   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）   即建立一个二叉搜索树并实现各种操作（插入，查找，删除，按名次查找，按名次删除）   1. 测试结果（测试输入，测试输出）  输入 13  0 6  0 7  0 4  0 5  0 1  1 5  0 7  3 3  2 4  1 5  3 4  4 3  0 4  **输出**  0  6  6  2  2  7  0  7  2  3  1  6  3   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   结果均正确   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）   #include <iostream>  #include <queue>  #include <stack>  using namespace std;  template <class T>  struct binaryTreeNode  {  T element;  binaryTreeNode \*leftchild;  binaryTreeNode \*rightchild;  int leftsize;  binaryTreeNode()  {  element = 0;  leftchild = NULL;  rightchild = NULL;  leftsize = 0;  }  binaryTreeNode(const T &theelement)  {  element = theelement;  leftchild = NULL;  leftsize = 0;  rightchild = NULL;  }  binaryTreeNode(const T &theelement, binaryTreeNode \*l, binaryTreeNode \*r)  {  element = theelement;  leftchild = l;  leftsize = 0;  rightchild = r;  }  };  template <class T>  class binarySearchTree  {  private:  int size;  binaryTreeNode<T> \*root;  public:  binarySearchTree()  {  size = 0;  root = NULL;  }  int getsize()  {  return size;  }  void push(const T &value)  {  int ans = 0;  stack<binaryTreeNode<T> \*> s;  if (root == NULL)  {  root = new binaryTreeNode<T>(value);  size++;  cout << 0 << endl;  return;  }  binaryTreeNode<T> \*p = root;  while (1)  {  if (value > p->element)  {  ans = ans ^ p->element;  if (p->rightchild == NULL)  {  binaryTreeNode<T> \*t = new binaryTreeNode<T>(value);  p->rightchild = t;  break;  }  else  {  p = p->rightchild;  }  }  else if (value < p->element)  {  s.push(p);  ans = ans ^ p->element;  if (p->leftchild == NULL)  {  binaryTreeNode<T> \*t = new binaryTreeNode<T>(value);  p->leftchild = t;  break;  }  else  {  p = p->leftchild;  }  }  else if (value == p->element)  {  cout << 0 << endl;  return;  }  }  while (!s.empty())  {  binaryTreeNode<T> \*temp = s.top();  temp->leftsize++;  s.pop();  }  cout << ans << endl;  size++;  }  void search(const T &value)  {  int ans = 0;  binaryTreeNode<T> \*p = root;  while (p != NULL && p->element != value)  {  ans = ans ^ p->element;  if (p->element > value)  {  p = p->leftchild;  continue;  }  if (p->element < value)  {  p = p->rightchild;  }  }  if (p == NULL)  {  cout << 0 << endl;  }  else  {  ans = ans ^ p->element;  cout << ans << endl;  }  }  void delete\_Node(binaryTreeNode<T> \*d, binaryTreeNode<T> \*pd)  {  binaryTreeNode<T> \*child = NULL;  if (d->leftchild != NULL)  child = d->leftchild;  else  child = d->rightchild;  if (d == pd->leftchild)  {  pd->leftchild = child;  }  else  {  pd->rightchild = child;  }  delete d;  }  void erase\_by\_value(const T &value, bool output)  {  binaryTreeNode<T> \*p = root;  binaryTreeNode<T> \*pp = NULL;  stack<binaryTreeNode<T> \*> s;  int ans = 0;  while (1)  {  if (p == NULL)  {  if (output)  cout << 0 << endl;  return;  }  if (value == p->element)  {  ans = ans ^ p->element;  break;  }  if (value > p->element)  {  ans = ans ^ p->element;+  pp = p;  p = p->rightchild;  }  else  {  s.push(p);  ans = ans ^ p->element;  pp = p;  p = p->leftchild;  }  }  while (!s.empty())  {  binaryTreeNode<T> \*temp = s.top();  temp->leftsize--;  s.pop();  }  if (p == root)  {  if (p->leftchild == NULL && p->rightchild == NULL)  {  delete p;  root = NULL;  }  else if (p->leftchild == NULL && p->rightchild != NULL)  {  root = p->rightchild;  delete p;  p = NULL;  }  else if (p->leftchild != NULL && p->rightchild == NULL)  {  root = p->leftchild;  delete p;  p = NULL;  }  else  {  binaryTreeNode<T> \*right\_min = p->rightchild;  binaryTreeNode<T> \*prm = p;  while (right\_min->leftchild != NULL)  {  right\_min->leftsize--;  prm = right\_min;  right\_min = right\_min->leftchild;  }  p->element = right\_min->element;  delete\_Node(right\_min, prm);  }  }  else  {  if (p->leftchild == NULL && p->rightchild == NULL)  {  if (p == pp->leftchild)  pp->leftchild = NULL;  if (p == pp->rightchild)  pp->rightchild = NULL;  p = NULL;  delete p;  }  else if (p->leftchild == NULL && p->rightchild != NULL)  {  delete\_Node(p, pp);  }  else if (p->leftchild != NULL && p->rightchild == NULL)  {  delete\_Node(p, pp);  }  else  {  binaryTreeNode<T> \*right\_min = p->rightchild;  binaryTreeNode<T> \*prm = p;  while (right\_min->leftchild != NULL)  {  right\_min->leftsize--;  prm = right\_min;  right\_min = right\_min->leftchild;  }  p->element = right\_min->element;  delete\_Node(right\_min, prm);  }  }  size--;  if (output)  cout << ans << endl;  }  void erase\_by\_rank(int rank)  {  int sum = size;  int ans = 0;  if (rank > size || rank <= 0)  {  cout << 0 << endl;  return;  }  binaryTreeNode<T> \*p = root;  while (1)  {  if (rank > p->leftsize + 1)  {  ans = ans ^ p->element;  sum = sum - p->leftsize - 1;  rank = rank - p->leftsize - 1;  p = p->rightchild;  }  else if (rank == p->leftsize + 1)  {  ans = ans ^ p->element;  cout << ans << endl;  erase\_by\_value(p->element, false);  return;  }  else if (rank < p->leftsize + 1)  {  ans = ans ^ p->element;  sum = p->leftsize;  p = p->leftchild;  }  }  }  void find\_by\_rank(int rank)  {  int sum = size;  int ans = 0;  if (rank > size || rank <= 0)  {  cout << 0 << endl;  return;  }  binaryTreeNode<T> \*p = root;  while (1)  {  if (rank > p->leftsize + 1)  {  ans = ans ^ p->element;  sum = sum - p->leftsize - 1;  rank = rank - p->leftsize - 1;  p = p->rightchild;  }  else if (rank == p->leftsize + 1)  {  ans = ans ^ p->element;  cout << ans << endl;  return;  }  else if (rank < p->leftsize + 1)  {  ans = ans ^ p->element;  sum = p->leftsize;  p = p->leftchild;  }  }  }  };  int main()  {  binarySearchTree<int> t;  int m;  cin >> m;  for (int i = 1; i <= m; i++)  {  int k, b;  cin >> k >> b;  if (k == 0)  {  t.push(b);  }  if (k == 1)  {  t.search(b);  }  if (k == 2)  {  t.erase\_by\_value(b, true);  }  if (k == 3)  {  t.find\_by\_rank(b);  }  if (k == 4)  {  t.erase\_by\_rank(b);  }  }  } | | | |