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

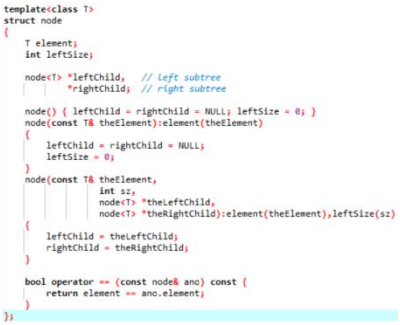
学号：202000130143	姓名： 郑凯饶	班级： 计科 20.1
实验题目：搜索树		
实验学时：2	实验日期： 12.8	

实验目的：
掌握二叉树结构的定义、描述方法、操作实现。

软件开发环境：
Windows 10 家庭中文版 64 位（10.0，版本 18363）
Dev-C++ IDE

1. 实验内容
创建带索引的二叉树类。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。

2. 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）
本来一开始想的是继承之前的二叉树类，但是考虑到之前代码体量达到 500 多行，自己也没有特别关注鲁棒性问题，还是决定定制搜索树类。



```
template<class T>
struct node
{
    T element;
    int leftSize;

    node<T> *leftChild,  // left subtree
            *rightChild; // right subtree

    node() { leftChild = rightChild = NULL; leftSize = 0; }
    node(const T& theElement):element(theElement)
    {
        leftChild = rightChild = NULL;
        leftSize = 0;
    }
    node(const T& theElement,
                int sz,
                node<T> *theLeftChild,
                node<T> *theRightChild):element(theElement),leftSize(sz)
    {
        leftChild = theLeftChild;
        rightChild = theRightChild;
    }

    bool operator == (const node& ano) const {
        return element == ano.element;
    }
};
```

首先，在之前的 BinaryTreeNode 中添加 leftSize 域作为 IBStree 的节点 node，记录节点左子树大小。

```
template<class T>
class IBStree {
    public:
        IBStree() {
            root = NULL;
            Size = 0;
        }

        node<T>* find(const T& ele);
        void insert(const T& ele);
        void erase(const T& ele);

        void getbyRank(int, int);

    private:
        int Size;
        node<T> *root;
};
```

Public 方法:
Find()：向下搜索，若目标值大于当前节点对右子树继续搜索，若小于则对左子树继续搜索，否则找到目标。实现如下：

```
while (p != NULL) {
    Xor ^= p->element;

    if (ele < p->element) p = p->leftChild;
    else {
        if (ele > p->element) p = p->rightChild;
        else {
            // match
            cout << Xor << '\n';
            return p;
        }
    }
}
```

Insert()：像 Find()方法中的搜索，并在搜索过程中记录路径，即把节点压入栈中，若找到输出 ‘0’，否则在相应位置创建新节点，并回溯路径对满足下述条件的节点，leftSize+=1

```
if (now->leftChild != NULL && now->leftChild->element == pre->element) {
```

Erase()： 操作和 Find()、Insert()类同，有两条路径需要记录，一条是根节点到删除节点的路径，另一条是删除节点到它的右子树中最小节点。异或计算时注意路径的完整性，待查询的元素也需要加入答案中。

GetByRank()：查找 leftSize 为 r-1 的节点：在左子树中查找 leftSize 为 r-1 的节点，或在右子树中查找 leftSize 为 r-1-（now->leftSize）的节点。

```
node<T> *now = root;
while (now->leftSize != r - 1) {
    if (now->leftSize > r - 1) {
        now = now->leftChild;
    }
    else if (now->leftSize < r - 1) {
        r -= now->leftSize + 1;
        now = now->rightChild;
    }
}

if (op == 0) {
    cout << "finding... " << now->element << '\n';
    find(now->element);
}
else if (op == 1) {
    erase(now->element);
}
```

3. 测试结果（测试输入，测试输出）

在 OJ 平台上成功提交。

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

这个实验中我在路径记录的时候细节比较马虎，弹栈、入栈的地方欠考虑导致内存访问异常，通过大量的调试解决了问题。但是我觉得可以在代码和题意上进行斟酌，能更快更准确地找出问题。

5. 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<bits/stdc++.h>
using namespace std;


template<class T>
struct node
{
 T element;
 int leftSize;

 node<T> *leftChild,   // left subtree
```

```
17.         *rightChild;  // right subtree
18.
19. node() { leftChild = rightChild = NULL; leftSize = 0; }
20. node(const T& theElement):element(theElement)
21. {
22.  leftChild = rightChild = NULL;
23.  leftSize = 0;
24. }
25. node(const T& theElement,
26.      int sz,
27.              node<T> *theLeftChild,
28.              node<T> *theRightChild):element(theElement),leftSize(sz)
29. {
30.  leftChild = theLeftChild;
31.  rightChild = theRightChild;
32. }
33.
34. bool operator == (const node& ano) const {
35.  return element == ano.element;
36. }
37.};
38.
39.template<class T>
40.class IBStree {
41. public:
42.  IBStree() {
43.   root = NULL;
44.   Size = 0;
45.  }
46.
47.  node<T>* find(const T& ele);
48.  void insert(const T& ele);
49.  void erase(const T& ele);
50.
51.  void getbyRank(int, int);
52.
53. private:
54.  int Size;
55.  node<T> *root;
56.};
57.
58.template<class T>
59.node<T>* IBStree<T>::find(const T& ele) {
60. int Xor = 0;
61. node<T> *p = root;
62.
```

```
63. while (p != NULL) {
64.  Xor ^= p->element;
65.
66.  if (ele < p->element) p = p->leftChild;
67.  else {
68.   if (ele > p->element) p = p->rightChild;
69.   else {
70.    // match
71.    cout << Xor << '\n';
72.    return p;
73.   }
74.  }
75. }
76.
77. // not match
78. cout << 0 << '\n';
79. return NULL;
80.
81.}
82.
83.template<class T>
84.void IBStree<T>::insert(const T& ele) {
85. int Xor = 0;
86. node<T> *p = root, *pp = NULL;
87.
88. stack<node<T>* > path;
89.
90. while (p != NULL) {
91.  path.push(p);
92.  Xor ^= p->element;
93.
94.  pp = p;
95.
96.  if (ele < p->element) p = p->leftChild;
97.  else {
98.   if (ele > p->element) p = p->rightChild;
99.   else {
100.       cout << 0 << '\n';
101.       return;
102.      }
103.     }
104.    }
105.
106.    cout << Xor << '\n';
107.
108.    node<T> *newNode = new node<T>(ele);
```

```
109.     if (root != NULL) {
110.      if (ele < pp->element) pp->leftChild = newNode;
111.      else pp->rightChild = newNode;
112.     }
113.     else {
114.      root = newNode;
115.      Size++;
116.      return;
117.     }
118.     Size++;
119.
120.     if (path.empty()) return;
121.     auto pre = newNode;
122.
123.     while (!path.empty()) {
124.      auto now = path.top();
125.      path.pop();
126.
127.      if (now->leftChild != NULL && now->leftChild->element == pre->element) {
128.       now->leftSize++;
129.      }
130.      pre = now;
131.     }
132.    }
133.
134.    template<class T>
135.    void IBStree<T>::erase(const T& ele) {
136.     int Xor = 0;
137.     node<T> *p = root, *pp = NULL;
138.
139.     stack<node<T>* > path;
140.
141.     while (p != NULL && p->element != ele) {
142.      path.push(p);  // push at least one or return on 145
143.      Xor ^= p->element;
144.      pp = p;
145.      if (ele < p->element) p = p->leftChild;
146.      else p = p->rightChild;
147.     }
148.
149.     if (p == NULL) {
150.      cout << 0 << '\n';
151.      return;
152.     }
153.
154.     // have found ele
```

```
155.     Xor ^= p->element;
156.     path.push(p);
157.     cout << Xor << '\n';
158.
159.     // path1
160.     auto pre = path.top();
161.     path.pop();
162.
163.     while (!path.empty()) {
164.      auto now = path.top();
165.      path.pop();
166.      if (now->leftChild != NULL && now->leftChild->element == pre->element) now
    ->leftSize--;
167.      pre = now;
168.     }
169.
170.     // 2 孩子
171.     if (p->leftChild != NULL && p->rightChild != NULL) {
172.      node<T> *s = p->rightChild, *ps = p;
173.      while (s->leftChild != NULL) {
174.       path.push(s);
175.       ps = s;
176.       s = s->leftChild;
177.      }
178.
179.      // path2
180.      while (!path.empty()) {
181.       auto now = path.top();
182.       path.pop();
183.       now->leftSize -= 1;
184.      }
185.      // 替换
186.      node<T> *q = new node<T>(s->element, p->leftSize, p->leftChild, p->rightCh
    ild);
187.      if (pp == NULL) {
188.       root = q;
189.      }
190.      else if (p == pp->leftChild) {
191.       pp->leftChild = q;
192.      }
193.      else pp->rightChild = q;
194.
195.      // 转换为删除 s
196.      if (ps == p) pp = q;
197.      else pp = ps;
198.
```

```
199.      delete p;
200.      p = s;
201.     }
202.
203.     // 1 孩子
204.     node<T> *c;
205.     if (p->leftChild != NULL) {
206.      c = p->leftChild;
207.     }
208.     else c = p->rightChild;
209.
210.     if (p == root) {
211.      root = c;
212.     }
213.     else {
214.      if (p == pp->leftChild) {
215.       pp->leftChild = c;
216.      }
217.      else pp->rightChild = c;
218.     }
219.     Size--;
220.     delete p;
221.    }
222.
223.    template<class T>
224.    void IBStree<T>::getbyRank(int r, int op) {
225.     if (r <= 0 || r > Size) {
226.      cout << 0 << '\n';
227.      return;
228.     }
229.
230.     node<T> *now = root;
231.     while (now->leftSize != r - 1) {
232.      if (now->leftSize > r - 1) {
233.       now = now->leftChild;
234.      }
235.      else if (now->leftSize < r - 1) {
236.       r -= now->leftSize + 1;
237.       now = now->rightChild;
238.      }
239.     }
240.
241.     if (op == 0) {
242.    //  cout << "finding... " << now->element << '\n';
243.      find(now->element);
244.     }
```

```
245.     else if (op == 1) {
246.      erase(now->element);
247.     }
248.    }
249.
250.    void solve() {
251.     IBStree<int> tree;
252.
253.     int m; cin >> m;
254.     for (int i = 1, a, b; i <= m; i++) {
255.      cin >> a >> b;
256.      switch (a) {
257.       case 0:
258.        tree.insert(b);
259.        break;
260.       case 1:
261.        tree.find(b);
262.        break;
263.       case 2:
264.        tree.erase(b);
265.        break;
266.       case 3:
267.        tree.getbyRank(b, 0);
268.        break;
269.       case 4:
270.        tree.getbyRank(b, 1);
271.        break;
272.      }
273.     }
274.    }
275.
276.    int main(){
277.     solve();
278.     return 0;
279.    }
```