广州大学学生实验报告

开课实验室： 电子楼 416B　　2018 年 6 月 3 日

学院	计算机科学与网络工程学院	年级、专业、班	软件 171	姓名	谢金宏	学号	1706300001
实验课程名称	数据结构					成绩	
实验项目名称	实验五 查找算法					指导老师	杜娇

一、实验目的

1. 巩固二叉树的实现方法。
2. 理解二叉排序树的递归插入和递归查找算法思想并用代码实现。
3. 理解哈希查找算法思想并用代码实现。

二、使用仪器、器材

4. 使用 Windows 操作系统的微型计算机一台。
5. 符合 C++11 标准的 C++编程软件。

三、实验内容及原理

二叉排序树

1. 编程实现二叉排序树
2. 产生 10 个随机数。将这些随机数插入排序树中，利用排序树输入升序序列和降序序列。

哈希表

1. 编程实现哈希表，哈希表采用除留余数法和线性探查法。
2. 产生 10 个随机数。将这些随机数分别插入长度为 13 和 15 的哈希表中，检查哈希表的形态。

四、实验过程原始数据记录

本次实验中使用的源代码：


BinarySearchTree.cpp


HashTable.cpp

辅助函数

```
// 生成随机数
vector<int> Generate10RandomNumbers()
{
    vector<int> random_numbers;
    for (int i = 0; i < 10; ++i)
        random_numbers.push_back(rand());
    return random_numbers;
}

// 打印数字
void ShowNumbers(vector<int> numbers)
{
    for (int number : numbers)
    {
        cout << number << ' ';
    }
    cout << endl;
}
```

二叉排序树的定义

```
// 二叉查找树的结点
struct Node
{
    int value;
    Node *left_child = nullptr;
    Node *right_child = nullptr;

    Node(int value, Node *left_child = nullptr, Node *right_child = nullptr)
    {
        this->value = value;
        this->left_child = left_child;
        this->right_child = right_child;
    }
};

// 二叉查找树
class BinarySearchTree
{
private:
    Node *root = nullptr;
```

```
    // 从树中删除 node 结点
    void removeNode(Node * node)
    {
        if (node != nullptr) {
            removeNode(node->left_child);
            removeNode(node->right_child);
        }
        delete node;
    }

public:
    BinarySearchTree() = default;
    BinarySearchTree(int value)
    {
        root = new Node(value);
    }

    ~BinarySearchTree()
    {
        removeNode(root);
    }

    // 向树中添加值为 value 的结点
    void AddNode(int value)
    {
        Node *current = root, *parent = nullptr;
        while (current != nullptr)
        {
            if (current->value == value) return;  // 树中已存在结点，不进行添加

            parent = current;
            current = (current->value < value ? current->right_child : current-
>left_child);  // 根据根节点的值决定向左搜索或向右搜索
        }
        if (parent == nullptr) {  // 处理空树的情况
            root = new Node(value);
            return;
        }
        if (parent->value < value) parent->right_child = new Node(value);
        if (parent->value > value) parent->left_child = new Node(value);
    }
```

```
    // 向树中添加取值为 values 数组中各元素值的结点
    void AddNodes(vector<int> values)
    {
        for (int value : values) {
            AddNode(value);
        }
    }
    // 按升序打印结点值
    vector<int> GetAscendOrderNodeValue()
    {
        vector<int> values;

        function<void(Node *)> visitNode = [&](Node *node){
            if (node == nullptr) return;
            visitNode(node->left_child);
            values.push_back(node->value);
            visitNode(node->right_child);
        };

        visitNode(root);
        return values;
    }

    // 按降序打印结点值
    vector<int> GetDescendOrderNodeValue()
    {
        vector<int> values;

        function<void(Node *)> visitNode = [&](Node *node){
            if (node == nullptr) return;
            visitNode(node->right_child);
            values.push_back(node->value);
            visitNode(node->left_child);
        };

        visitNode(root);
        return values;
    }
};
```

二叉排序树测试

```cpp
int main()
{
    BinarySearchTree tree;
    vector<int> random_numbers(Generate10RandomNumbers());
    tree.AddNodes(random_numbers);
    cout << "随机产生的数字："; ShowNumbers(random_numbers);
    cout << "升序序列："; ShowNumbers(tree.GetAscendOrderNodeValue());
    cout << "降序序列："; ShowNumbers(tree.GetDescendOrderNodeValue());
}
```

哈希表定义

```cpp
// 使用除留余数法和线性探查法的哈希表
// 不能插入负数
class HashTable
{
private:
    int size;
    int *table;

    // 获取 number 在除留余数法中定义的哈希表键 key
    int getKey(int number)
    {
        return number % size;
    }

    // 获取当前 key 在线性探查法中定义的相邻的下一个 key
    int getNextKey(int key)
    {
        return (key + 1) % size;
    }
```

```cpp
public:
    HashTable(int size) {
        assert(size > 0);
        this->table = new int[size];
        this->size = size;
        memset(this->table, -1, sizeof(int)*size);
    }

    ~HashTable() {
        delete[] table;
    }


    // 向哈希表中插入 number
    void Insert(int number) {
        int key = getKey(number);
        while (table[key] != -1 && table[key] != number) {
            key = getNextKey(key);
        }
        table[key] = number;
    }
```

```
    // 查找哈希表中是否存在 number
    void Find(int number) {
        int key = getKey(number);
        while (table[key] != -1 && table[key] != number) {
            key = getNextKey(key);
        }
        if (table[key] == number) {
            cout << "302 - Found number " << number << " at key " << key << "." <<
endl;
        }
        else {
            cout << "404 - Number " << number << " not found." << endl;
        }
    }


    // 展示哈希表
    void Show() {
        cout << '[' << size << ']' << ' ';
        for (int i = 0; i < size; ++i) {
            cout << table[i] << ' ';
        }
        cout << endl;
    }
};
```

哈希表测试函数

```
int main()
{
    vector<int> random_numbers(Generate10RandomNumbers());

    cout << "产生的随机数为: " << endl;
    for (int number : random_numbers) {
        cout << number << ' ';
    }
    cout << endl;

    HashTable hash_table_length_13(13);
    HashTable hash_table_legnth_15(15);

    cout << "对表长度为 13 的哈希表: " << endl;
    for (int number : random_numbers) {
        hash_table_length_13.Insert(number);
        hash_table_length_13.Find(number);
    }
    hash_table_length_13.Show();

    cout << "对表长度为 15 的哈希表: " << endl;
    for (int number : random_numbers) {
        hash_table_legnth_15.Insert(number);
        hash_table_legnth_15.Find(number);
    }
    hash_table_legnth_15.Show();
}
```

五、实验结果及分析

完成了本次实验的全部要求。
程序的测试结果如下：

二叉排序树

随机产生的数字：1481765933 1085377743 1270216262 1191391529 812669700 553475508 445349752 1344887256 730417256 1812158119

升序序列：445349752 553475508 730417256 812669700 1085377743 1191391529 1270216262 1344887256 1481765933 1812158119

降序序列：1812158119 1481765933 1344887256 1270216262 1191391529 1085377743 812669700 730417256 553475508 445349752

哈希表

产生的随机数为：

1481765933 1085377743 1270216262 1191391529 812669700 553475508 445349752 1344887256 730417256 1812158119

对表长度为 13 的哈希表：

302 - Found number 1481765933 at key 11.

302 - Found number 1085377743 at key 8.

302 - Found number 1270216262 at key 3.

302 - Found number 1191391529 at key 4.

302 - Found number 812669700 at key 12.

302 - Found number 553475508 at key 1.

302 - Found number 445349752 at key 5.

302 - Found number 1344887256 at key 0.

302 - Found number 730417256 at key 10.

302 - Found number 1812158119 at key 6.

[13] 1344887256 553475508 -1 1270216262 1191391529 445349752 1812158119 -1 1085377743 -1 730417256 1481765933 812669700

对表长度为 15 的哈希表：

302 - Found number 1481765933 at key 8.

302 - Found number 1085377743 at key 3.

302 - Found number 1270216262 at key 2.

302 - Found number 1191391529 at key 14.

302 - Found number 812669700 at key 0.

302 - Found number 553475508 at key 4.

302 - Found number 445349752 at key 7.

302 - Found number 1344887256 at key 6.

302 - Found number 730417256 at key 11.

302 - Found number 1812158119 at key 5.

[15] 812669700 -1 1270216262 1085377743 553475508 1812158119 1344887256 445349752 1481765933 -1 -1 730417256 -1 -1 1191391529