



Lab report

Course: Class Libraries and Data Structures

Semester: 1st semester of the academic year **2019-2020**

Major: Software Engineering

Class: 2018

Student Name: SONG, Xingjian (宋行健)

Student ID: 222019321062006

Teacher: ZHAO, Hengjun (赵恒军)

School of Computer and Information Science

Name	Binary Search Tree		
Date	Dec 26, 2019	Type	<input type="checkbox"/> Confirmatory <input checked="" type="checkbox"/> Design <input type="checkbox"/> Comprehensive
1. Objective & Requirements <ul style="list-style-type: none"> a) Understand the concept and property of binary search tree b) Get familiar with the insert, delete and find operations on binary search tree c) Grasp the design of recursive or iterative algorithms about binary search tree d) Learn how to do data analysis 			
2. Experimental environment (platform and software) Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions)			

1) Experimental content and design (Main Content, Procedure, Codes and Results)

Task 1

Accomplish the following tasks based on the codes to you:

- Implement a method that can return the height of a binary search tree
- Implement a method that can print a binary search tree in a tree-like way
- Generate a series of integers (of size n) randomly and insert them into an empty binary search tree, and compute the height of the tree. Repeat this process for many times and compute the average height of a binary search tree with n nodes. Try to analyze the relationship of the average tree height with $\log(n)$

Task (a):

首先定义了计算树高的迭代函数 `getHeightRec()`，它的参数是当前节点，函数从根节点进入，判断不是叶子节点后计算其左子树和右子树的树高，并返回其中高的那一个并加一（当前节点的一个树高），如此迭代，直到迭代到叶子节点后按顺序返回，最后输出树高。

部分代码如下：

//计算树高的迭代函数

```
template<typename T>
```

```
int BinSearchTree<T>::getHeightRec(Node* tempRoot) const {
```

```
    if (tempRoot == NULL) return -1; //空树的树高定义为-1
```

```
    int leftHeight = getHeightRec(tempRoot->left); //计算并返回左子树树高
```

```
    int rightHeight = getHeightRec(tempRoot->right); //计算并返回右子树树高
```

```
    //以左右子树高的为准，加上自己这个节点，返回当前树高
```

```
    if (leftHeight > rightHeight) return leftHeight+1;
```

```
    else return rightHeight+1;
```

```
}
```

```
template<typename T>
```

```
int BinSearchTree<T>::height() const
```

```
{
```

```
    return getHeightRec(root); //计算树高迭代函数
```

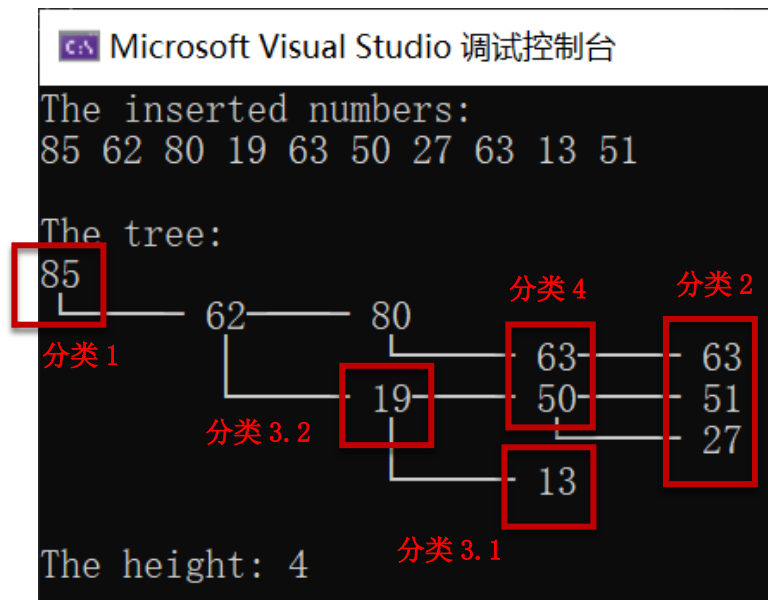
```
}
```

Task (b):

首先定义了打印树的迭代函数 `printTreeRec()`，它的参数是当前节点，函数从根节点进入，判断不是叶子节点后打印其左孩子和右孩子，如此迭代，直到迭代到叶子节点。

在打印过程中为了格式化，整体上分了四类节点：

- 分类 1：是分类 2 的一种特殊情况，这是根节点没有右孩子的情况，无法调用它的父母，所以先单独判断；
- 分类 2：右孩子链的结尾换行，此情况为了避免与左子树根节点的打印重合，又加入条件必须是父节点的右孩子节点；
- 分类 3：左子树开头情况，要统计 `tab` 数和“|”的打印位置；其以是否是叶子节点又分为两种情况；
- 分类 4：右子树链的普通情况，也是最简单的情况，因为它在一行的中间，所以不需要有特殊输出格式；



部分代码如下：

```
template<typename T>
void BinSearchTree<T>::printTreeRec(Node* tempRoot) {
    //分类1: 根节点没有右孩子的情况，无法调用它的父母，所以先单独判断
    if(tempRoot==root && tempRoot->right==NULL)
        cout << tempRoot->item << "\n";
    //分类2: 右孩子链的结尾换行，此情况为了避免与左子树根节点的打印重合，又加入条件必须是父节点的右孩子节点
    else if (tempRoot->right == NULL && tempRoot->parent->right==tempRoot)
        cout << tempRoot->item<<"\n";
    //分类3: 左子树开头情况，要统计tab数和“|”的打印位置
    else if (tempRoot != root && tempRoot->parent->left == tempRoot) {
        Node* temp = tempRoot->parent; //创建临时节点，用来遍历当前节点的祖宗节点，判断tab数和“|”的打印位置
        vector<string> tab; //创建向量tab用来储存tab和“|”
        //遍历当前节点的祖宗节点
        while (temp != root) {
            if(temp->parent->left != NULL && temp->item != 101)
```

```

        tab.push_back("| \t"); //如果此节点未被标记, 则需要打印 "|" 连接
    else
        tab.push_back(" \t"); //如果此节点被标记为101则说明以打印, 不需要再打
    印 "|" 连接
    temp = temp->parent;
}

//将tab向量反向输出
for (int i = tab.size(); i > 0; i--) {
    cout << tab[i-1];
}
//分类3.1: 如果这个节点既是左子树开始, 又是左子树结尾
if (tempRoot->right == NULL) {
    cout << " ┌─── \t" << tempRoot->item << "\n";
    tempRoot->item = 101; //标记已输出的右子树根
}
//分类3.2: 如果这个节点仅是左子树开始, 他还有右孩子
else {
    cout << " ┌─── \t" << tempRoot->item << "─── \t";
    tempRoot->item = 101; //标记已输出的右子树根
}
}
else cout << tempRoot->item << "─── \t"; //分类4: 右子树链的普通情况

//递归代码
if (tempRoot->right != NULL) {
    printTreeRec(tempRoot->right);
}
if (tempRoot->left != NULL) {
    printTreeRec(tempRoot->left);
}
if (tempRoot->right == NULL && tempRoot->left == NULL) return;
}

template<typename T>
void BinSearchTree<T>::printTree()
{
    printTreeRec(root);
}

```

Task (c):

首先创建写入文件流用于储存节点数和其对应的树高, 便于后期分析。在这里模拟了 1 个节点到 800 个节点的二叉树, 每种情况都随机生成了 200 种情况进行计算平均树高。为了加大随机性, 将随机数的最大值提高到了 10000。

生成的数据通过 Excel 数据分析功能进行对数函数拟合, 通过拟合优度 R^2 得出二叉树的树高是否与节点数成对数关系。

部分代码如下:

```
int main()
```

```

{
    ofstream out;          //写入文件流;
    out.open("E:\\类库与数据结构\\实验报告\\lab-8\\data.txt");
    double Num = 200; //每个节点的模拟量
    int nodeNum = 800;   //节点个数模拟量
    int MAX = 10000;    //随机最大值
    srand((unsigned)time(0));

    //遍历有1个节点到有800个节点的情况
    for (int k = 1; k <= nodeNum; k++) {
        double averageHeight = 0;
        //每种节点情况随机生成200个模拟量，求出该节点的平均树高
        for (int j = 1; j <= Num; j++) {
            BinSearchTree<int> mybst;
            int item;
            for (int i = 0; i < k; i++) {
                item = rand() % MAX;
                mybst.insert(item);
            }
            averageHeight += mybst.height();    //平均树高
        }
        cout << k << "\t" << averageHeight / Num << endl;
        //将平均树高写入文件，便于后期分析
        out << k << "\t" << averageHeight / Num << endl;
    }
    out.close();
    cout << "FINISH!!" << endl;
}

```

3. Result analysis and discussion (Analysis of experimental results and summing up the harvest and the existing problems)

对于 Task(a)，在计算树高时，把问题递归化成，二叉树的树高等于左右子树较高的树高加一。

对于 Task(b)，在打印时最重要的是给二叉树的节点进行合理而全面的情况分析。在本实验中我将其分为 4 类节点，因为是横向打印二叉树，所以右孩子的打印不需要换行，只需要考虑这个节点是否还有右孩子，有的话继续打印，没有则进行换行操作，并回溯到其父节点判断是否有左孩子。整个遍历过程是一个深度优先搜索。

其中比较复杂的情况是每行打印的第一个左孩子需要考虑到 tab 数和“|”的打印位置。对于这种情况，建立一个 vector 向量进行数据储存，每次换行后从该节点向根回溯，遇到没有输出的左孩子储存“|\t”，其他则只储存“\t”，然后进行倒序输出，并将自身标记为 101 表示已输出（随机生成的最大值为 100 的情况），后面不用再打印“|”符号了。


其中比较容易出 bug 的地方是根节点和节点的双重身份（例如既是左孩子又是一行的最后一个），为此我分了 4 个判断语句进行准确的分类，排除了这些 bug。

对于 Task(c)，起初我认为与上次排队论的模拟相似，只是固定了节点数，然后计算平均树高和理论树高，后来发现误差很大。后来在老师的指导下，明白了不仅要对节点数进行模拟计算平均数高，还要通过对各种节点数做分析找出他们之间的函数关系。

对数据的处理可以将树高取对数后与节点数线性拟合，但是这里我用了一个比较“懒”的方法，通过 Excel 中对数拟合的功能直接拟合出对数关系式和拟合优度 R^2 ，但我相信这个功能算法的本质也是将其转换为对数后进行计算的。

实验结果：

Task (a)(b):



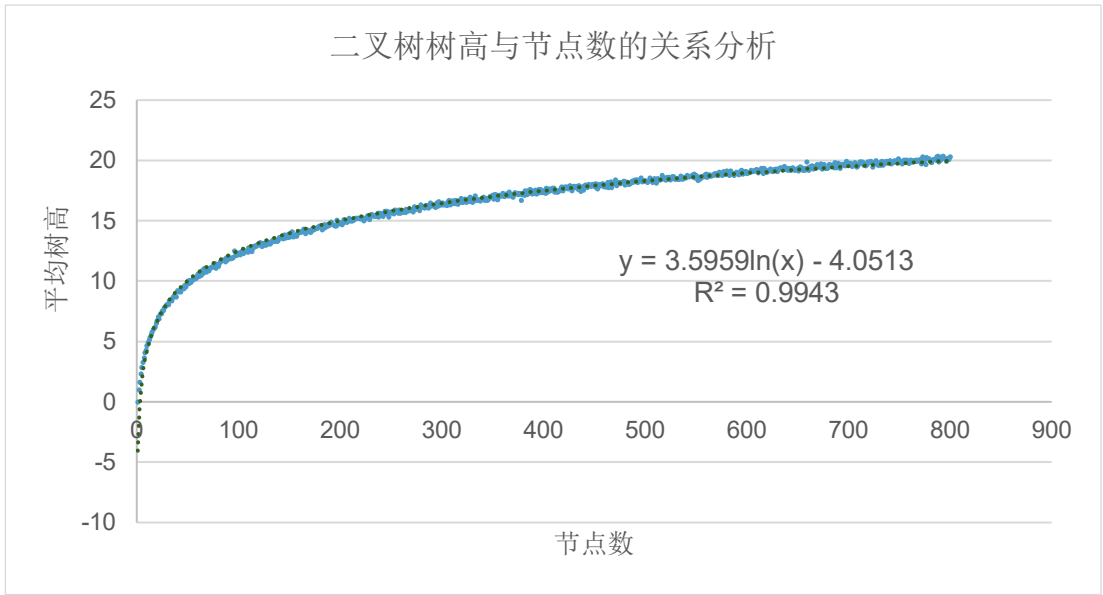
```
Microsoft Visual Studio 调试控制台
The inserted numbers:
0 90 98 67 66 39 42 26 44 53

The tree:
0----- 90----- 98
      |----- 67
      |----- 66
      |----- 39----- 42----- 44----- 53
      |----- 26

The height: 7

E:\类库与数据结构\实验报告\lab-8\BinSearchTree\Debug\BinSearchTree.exe (进程 15532)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .
```

Task (c):



利用 Excel 数据分析功能，将数据进行曲线拟合，并计算拟合优度，得出 $R^2=0.9943$ ，十分接近 1，拟合度很好，说明二叉树的树高与节点数呈现对数关系。

Comments & Evaluation	Content & Design (A-E)	
	Procedure & Codes (A-E)	
	Results (A-E)	
	Analysis & Discussion (A-E)	
	Score (A-E): Feedback comments:	