

**Lab Report**

**实验报告**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2023-2024** |
| **Major**: | Software Engineering |
| **Class**: | 2022 |
| **Student Name**: | 吴孜远 |
| **Student ID:** | 222022321062009 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Binary Search Tree  二分查找树 | | | |
| Date | | Dec，2023 | Type | | ☑Confirmatory （验证确认型）  ☑Design（设计型）  🗆Comprehensive（综合型） |
| 1. **Objective & Requirements（实验目的）**    1. Understand the concept and property of binary search tree   理解二分查找树的概念和性质   * 1. Get familiar with the insert, delete and find operations on binary search tree   熟悉二分查找树的节点插入、节点删除和元素查找操作   * 1. Grasp the design of recursive or iterative algorithms about binary search tree   掌握二分查找树上的递归和迭代算法的设计方法   * 1. Understand the concept of height of a binary search tree; can use iterative algorithms to compute the height of a binary search tree   掌握二分查找树的树高的定义，能利用递归或迭代算法计算二分查找树的树高   * 1. Understand the relationship between the size and height of a binary search tree; can deduce the relationship both mathematically and by experiment.   理解二分查找树树高和节点数目之间的关系，能够借助实验利用统计和方法定量分析二者之间的函数关系；能够对二者之间的关系进行数学上的推导。 | | | | | |
| 1. **Experimental environment (**platform and software**)（实验环境）**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. Experimental content and design (Main Content, Procedure, Codes and Results)（此部分应包含每一个实验内容的详细设计，含实验思路、详细实验步骤、核心代码说明等）   Task 1   1. Complete the implementation of the BST container based on the code sent to you. In particular, the method you are required to implement includes:  * BinSearchTree(); // default constructor * int size() const; // get the number of stored elements * int height() const; // compute the height * Iterator insert(const T& item); //insertion * ~BinSearchTree(); //destructor   It is suggested that some of the methods be implemented in a recursive way.  请基于所给代码实现BST类，至少应实现如下方法：   * BinSearchTree(); // 默认构造函数 * int size() const; // 获取存储元素的数目 * int height() const; // 获取树高 * Iterator insert(const T& item); // 元素的插入 * ~BinSearchTree(); // 析构函数   其中的部分方法建议用递归算法实现。   1. BinSearchTree()，构造函数要构造一个空树，既然root指向的是根节点，那么就让root指向空就好了。   template<typename T>  BinSearchTree<T>::BinSearchTree()  {      //Please implement this!      this->root = NULL;      count = 0;  }   1. size()，直接返回count，count就是在记录插入的节点数   template<typename T>  int BinSearchTree<T>::size() const  {      //Please implement this!      return count;  }   1. height()，这里我是用了递归的方法，一个数的树高=这个树的子树中树高最大的子树的树高+1，初始状态是空树，此时树高=-1。   template<typename T>  int BinSearchTree<T>::height() const  {      //Please implement this!      return treeHeight(root);  }  template<typename T>  int BinSearchTree<T>::treeHeight(Node\* currNode) const  {      //Please implement this!      if (currNode == NULL) {          return -1;      }      return 1 + std::max(treeHeight(currNode->left), treeHeight(currNode->right));  }   1. 这个insert方法我用了书上的实现策略，就是设置parent和child两个指针，child一层层向下走寻找合适位置，若item比child小，child就往左走，反之往右走，注意这里如果item等于child，child一样往右走。parent总是child的父母，直到child为NULL，此时item应该作为parent的子树，判断是左子树还是右子树就好了。   template<typename T>  typename BinSearchTree<T>::Iterator BinSearchTree<T>::insert(const T& item)  {      //Please implement this!      if (count == 0) { //树为空时          root = new Node;          root->left = NULL;          root->right = NULL;          root->item = item;      }      else {          //寻找合适的位置          Node\* parent = root->parent;          Node\* child = root;          while (child != NULL) {              parent = child;              if (item < child->item) {                  child = child->left;              }              else {                  child = child->right;              }          }          // 加入新节点          if (item < parent->item) { //item小于parent              Node\* newNode = new Node;              parent->left = newNode;              newNode->item = item;              newNode->parent = parent;              newNode->left = NULL;              newNode->right = NULL;              return Iterator(newNode);          }          else { //item不小于parent              Node\* newNode = new Node;              parent->right = newNode;              newNode->item = item;              newNode->parent = parent;              newNode->left = NULL;              newNode->right = NULL;              return Iterator(newNode);          }      }      count++;  }   1. 析构函数也是用了递归方法，要删除这个树，必须要先删除这个树的子树，再删除根，最简形式是这个树为空，直接返回，什么都不用删除。   template<typename T>  BinSearchTree<T>::~BinSearchTree()  {      //Please implement this!      deleteNode(root);  }  template<typename T>  void BinSearchTree<T>::deleteNode(Node\* temp)  {      //递归的方法，从下往上，先删子树再删自己。      if (temp == nullptr) {          return ;      }      deleteNode(temp->left);      deleteNode(temp->right);      delete temp;  }   1. Generate a series of integers randomly and insert them into an empty binary search tree, and compute the height of the tree. Repeat this for a number of times and compute the average height of a BST of size n. Try to discover the mathematical relationship between the size and the height of a BST based on your analysis.   生成一系列随机数，通过其从空树开始通过逐个插入的方式构造一个BST，计算该BST的树高；变换随机数序列的数值范围和数目，多次重复实验，获取n个节点BST平均树高的统计量；对所得结果进行数据拟合等分析，获取BST平均树高和节点数目之间的数学关系。  这里我使用了三层for循环，最内层是生成节点数为n的BST，其中每个节点都是1-100的随机数；中间层是为了直接生成20次，取树高平均值；最外层循环设置不同的节点数n。这样就只需运行一次就可以获得多组n对应的平均数高。  代码：  int main() {      srand((unsigned)time(0));      int sum = 0;      double aveHeight = 0;      for (int n = 10; n < 60; n = n + 5) {          cout << n << ", ";      }      cout << endl;      for (int n = 10; n < 60; n = n + 5) { //测试节点          sum = 0;          for (int i = 0; i < 20; i++) { //每个节点数生成20次树，取树高平均值              BinSearchTree<int>\* bst = new BinSearchTree<int>;              for (int i = 0; i < n; i++) { //生成一个n个结点的树                  bst->insert(rand() % 100 + 1);                }              sum = sum + bst->height();                //cout << bst->height() << " ";          }          aveHeight = sum / 20.0;          cout << aveHeight << ",";      }      return 0;  }  输出：  拟合曲线图：    拟合函数：y = 3.23\*log(n) – 2.90  符合对数关系。 | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）（此部分应包含实验结果，对实验结果的分析，实验收获的总结，实验中存在问题的讨论等；另外，需要回应一下如下思考题：1. 如何推导证明Binary Search Tree的平均树高与结点数成对数关系？2. 如何实现树状打印方法：void printTree(); // print the tree-shape of the BST   获取类似（但不局限于）下图所示的显示效果？    ）  思考题1：  我只能说如果这个树是平衡树或者是一个满树，它的节点数与树高一定满足关系：h = floor(log(n))，这时显然的，因为最后一层的节点数大约是是整个树的一半。最理想的情况，根是50，当我们插入一个数能正好构成平衡树甚至满树。实际上这个问题要用概率论的方法求解，所以说实话我不能准确推导出这个问题。  思考题2：  使用递归的方法：  printTreeStructure接受一个Node指针表示当前子树的根节点，一个字符串prefix表示当前节点的前缀，以及一个bool值isLeft表示当前节点是否是左子树的根节点。  如果当前节点存在（非NULL），则输出当前节点的值，前缀字符串，以及表示连接关系的箭头（"└── "）。  然后，递归调用printTreeStructure函数，处理右和左子树，传递更新后的前缀字符串和信息表示当前节点是否为左子树的根节点。  如果当前节点为NULL，则输出NULL。这是为了避免判断不清是表示左子树还是右子树。  template<typename T>  void BinSearchTree<T>::printTree()  {      //Please think about how to implement this!      printTreeStructure(root," ",false);  }  template <typename T>  void BinSearchTree<T>::printTreeStructure(Node\* root, std::string prefix, bool isLeft)  {      if (root) {          std::cout << prefix;          std::cout << (isLeft ? "└── " : "└── ");          std::cout << root->item << std::endl;          printTreeStructure(root->right, prefix + (isLeft ? "│   " : "    "), true);          printTreeStructure(root->left, prefix + (isLeft ? "│   " : "    "), false);      }      else {          std::cout << prefix;          std::cout << (isLeft ? "└── " : "└── ");          std::cout << "NULL" << std::endl;      }  }  打印效果如下： | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |