

**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**: | 朱昊 |
| **Student ID:** | 222022321062008 |
| **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. 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。  默认构造函数如下：  BinSearchTree<T>::BinSearchTree()  { //根据二叉查找树的结构，默认有一个根节点，只不过初始化时全部都为NULL  //Please implement this!  root = new Node;  root->left = NULL;  root->right = NULL;  root->parent = NULL;  root->item = NULL;  count = 1;  }  Size（）函数如下：  template<typename T>  int BinSearchTree<T>::size() const  {  //Please implement this!  return this->count;  }  获取树高的方法如下：  template<typename T>  int BinSearchTree<T>::findheight(Iterator i) const  {//findheight函数用递归的算法去解决这个问题  if (i.curr == NULL) {  return -1;//如果迭代器指向的为空则说明到了叶节点，，返回-1  }  else {  return 1 + (int)fmax(findheight((i.curr)->left), findheight((i.curr)->right));//递归的调用findheight函数，只不过调用之前先+1  }  }  template<typename T>  int BinSearchTree<T>::height() const  {  //Please implement this!  Iterator itr(root);  //由于height函数不带参数，所以在包装一个findheight函数  return findheight(itr);  }  元素的插入实现如下：  template<typename T>  typename BinSearchTree<T>::Iterator BinSearchTree<T>::insert(const T& item)  {  //Please implement this!  if (root->item == NULL) {//若一开始只有一个根节点则且元素还是空，说明是空树，则直接修改就行  root->item = item;  }  else{//不是空树  //创建两个Node\* ，node1用来找要插入的位置，node2使其成为插入位置的parent  Node\* node1 = root;  Node\* node2 = NULL;  while (node1 != NULL) {// 找到要插入节点的位置  node2 = node1;  if (item < node1->item) {// 如果要插入的值小于当前节点值，移动到左子节点  node1 = node1->left;  }  else {// 否则移动到右子节点  node1 = node1->right;  }  }  if (item < node2->item) { // 将新节点插入为左子节点  node2->left = new Node;  node2->left->item = item;  node2->left->left = NULL;  node2->left->right = NULL;  node2->left->parent = node2;  count++; // 将新节点插入为左子节点  return Iterator(node2->left); // 将新节点插入为左子节点  }  else { // 将新节点插入为左子节点  node2->right = new Node;  node2->right->item = item;  node2->right->left = NULL;  node2->right->right = NULL;  node2->right->parent = node2;  count++;// 更新树的节点计数  return Iterator(node2->right);// 返回新插入节点的迭代器  }  }  }  析构二叉查找树如下：  template<typename T>  void BinSearchTree<T>::deleteitem(Iterator itr) {  if (itr.curr == NULL) {  //如果当前节点为空，直接返回，无需删除  }  else {  deleteitem((itr.curr)->left);// 递归删除左子节点及其子节点  deleteitem((itr.curr)->right); // 递归删除右子节点及其子节点  delete itr.curr;  }  }  template<typename T>  BinSearchTree<T>::~BinSearchTree()  {  //Please implement this!  Iterator i(root);// 创建迭代器指向根节点  deleteitem(i); // 调用删除节点及其子节点的函数，从根节点开始删除整棵树  }  Main函数里生成一系列随机数逐个插入，并获取BST平均树高和节点数目的代码如下：  srand(time(0));  for (int m = 1; m < 11; m++) {  double totalheight = 0;  for (int i = 1; i < 1001; i++) {  BinSearchTree<int> t2;  for (int j = 1; j < m + 1; j++) {  t2.insert(rand());  //cout << rand() << endl;  }  totalheight += t2.height();  }  cout << "------------" << endl;  double averageheight = totalheight / 1000;  cout << "插入了" << m << "个元素,平均高度为：" << averageheight << endl;  }  最外层每次循环都会创建1000个插入同样的节点个数的二叉树并统计平均高度  最终根据打印结果（如下图）    拟合图如下： | | | | | |
| 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？    ）  实验结果：符合预期  实验收获：通过实现了二叉搜索树的基本操作，包括插入、查找树高、获取节点数量以及析构函数的实现，增进了我对二叉搜索树数据结构的理解，并提高我在算法和数据结构领域的实践能力。同时，实验结果的分析和总结可以让我更深入地探索树结构的特性和数学规律。  思考题1：  思考题2：  // 辅助函数：打印特定数量的空格  template<typename T>  void BinSearchTree<T>::printSpaces(int count) {  for (int i = 0; i < count; ++i) {  std::cout << " ";  }  }  template<typename T>  void BinSearchTree<T>::printTree(Node\* root, int space)  {  //如果根节点为空，直接返回  if (root == nullptr) {  return;  }  //增加空格数以表示层级  space += 5;  // 遍历右子树，打印节点值  printTree(root->right, space);  if (root->right != NULL) {  printSpaces(space+5);std::cout << "|";  }  std::cout << std::endl;    // 打印当前节点，带有适当的空格  printSpaces(space);  if (root == this->root) {  std::cout <<"----"<< root->item<<std::endl;  }  else {  std::cout << "----" << root->item << std::endl;  }  if (root == this->root) {  }else if (root->parent->left == root) {  }  else if (root->left == NULL && root->right == NULL) {  printSpaces(space);  std::cout << "|";  }  if (root->left != NULL) {  printSpaces(space);  std::cout << "|"<<" |";  }  // 遍历左子树，打印节点值  printTree(root->left, space);  }  template<typename T>  void BinSearchTree<T>::print() {  printTree(this->root, 0);  }  实现的打印样式如下：采用递归的思想，但仍有不足，不知道怎么解决 | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |