**实验五实验报告**

**2112066 于成俊**

1. **实验题目**

**实现对于二叉搜索树的如下操作:**

**1. 给定序列，使用逐点插入法构建二叉搜索树;**

**2.使用非递归的方法按照升序序列输出上述二叉搜索树的关键字;**

**3. 判断上述二叉搜索树是否为AVL搜索树，若不是，则将其转化为AVL搜索树。**

**备注:输出树时将结点的左右孩子结点在括号内输出，若为空则输出#**

1. **核心代码**

**①节点类 （定义三个指针）**

**char c;**

**int data;**

**//用于实现树**

**Node\* left;**

**Node\* right;**

**//用于实现队列**

**Node\* next;**

**②队列类**

**Node\* front1;//用于队列弹出，弹出时往后移**

**Node\* front2;//一直不动，用于从头查找**

**Node\* rear;**

**void pop() {**

**if (front1== NULL)return;**

**front1= front1->next;**

**}**

**③树类**

1. **ListQueue Q;//以队列形式存储**

**Node\* root;**

**2.看看这个节点的左右指针是否为空**

**Node\*& look(Node\* n) {**

**if (n->left == NULL)return n->left;**

**else {**

**if (n->right == NULL) return n->right;**

**else look(n->next);**

**}**

**}**

**3.创建一个空的二叉树**

**void build() {**

**Node\* tmp = new Node();**

**if (root == NULL) {**

**root = tmp;**

**Q.push(tmp);**

**}**

**else {**

**look(Q.front1) = tmp;**

**Q.push(tmp);**

**if (Q.front1->right != NULL) {**

**Q.pop();**

**}**

**}**

**}**

**4. 插入函数**

**void Insert(Node\* root, int x) {**

**if (this->root->c == '#') {**

**this->root->c = '+';**

**this->root->data = x;**

**return;**

**}**

**if ((root->left->c == '#') && x < root->data) {**

**root->left->data = x;**

**root->left->c = '+';**

**return;**

**}**

**if ((root->left->c != '#') && x < root->data) {**

**Insert(root->left, x);**

**return;**

**}**

**if ((root->right->c == '#') && x > root->data) {**

**root->right->data = x;**

**root->right->c = '+';**

**return;**

**}**

**if ((root->right->c != '#') && x > root->data) {**

**Insert(root->right, x);**

**return;**

**}**

**}**

**5. 返回节点高度**

**int height(Node\* root) {**

**if (root == NULL || root->c == '#')return 0;**

**if (root != NULL) {**

**int a = height(root->left);**

**int b = height(root->right);**

**return 1 + ((a > b) ? a : b);**

**}**

**}**

**6.判断是否为AVL**

**bool check(Node\* root) {**

**if (root == NULL) {**

**ceng = 0;**

**if (key) {**

**return true;**

**}**

**else {**

**key = true;**

**return false;**

**}**

**}**

**bool T = true;**

**for (int i = 1; i <= power2(ceng); i++) {**

**if (root->c == '#') {}**

**else {**

**T = false;**

**if (root->left->c == '#' && root->right->c == '#') {}**

**if (root->left->c == '#' && root->right->c != '#') {**

**if (height(root->right) >= 2) key = false;**

**}**

**if (root->left->c != '#' && root->right->c == '#') {**

**if (height(root->left) >= 2)key = false;**

**}**

**if (root->left->c != '#' && root->right->c != '#') {**

**if (abs(height(root->left) - height(root->right)) >= 2) {**

**key = false;**

**}**

**}**

**}**

**if (i == power2(ceng) && T) {**

**ceng = 0;**

**if (key) {**

**return true;**

**}**

**else {**

**key = true;**

**return false;**

**}**

**}**

**root = root->next;**

**}**

**ceng += 1;**

**check(root);**

**}**

**7.按升序序列输出（中序遍历）**

**int\* sortprint(Node\* root, int n) {**

**//栈**

**int top1 = -1;**

**Node\*\* H = new Node \* [n];**

**int top2 = -1;**

**int\* p = new int[n];**

**//当栈为空且所指根节点为空，说明该二叉树遍历完成**

**while (root->c != '#' || top1 != -1) {**

**while (root->c != '#') {**

**H[++top1] = root;**

**root = root->left;**

**}**

**//退出循环，说明左子树遍历完毕，开始遍历右子树**

**if (top1 != -1) {**

**root = H[top1--];**

**cout << root->data << " ";**

**p[++top2] = root->data;**

**root = root->right;**

**}**

**}**

**return p;**

**}**

**8.打印树**

**void print(Node\* root) {**

**if (root == NULL) {**

**ceng = 0;**

**return;**

**}**

**bool T = true;**

**for (int i = 1; i <= power2(ceng); i++) {**

**if (root->c == '#') {}**

**else {**

**T = false;**

**if (root->left->c == '#' && root->right->c == '#') {**

**cout << root->data << "(#,#) ";**

**}**

**if (root->left->c == '#' && root->right->c != '#') {**

**cout << root->data << "(#," << root->right->data << ") ";**

**}**

**if (root->left->c != '#' && root->right->c == '#') {**

**cout << root->data << "(" << root->left->data << ",#) ";**

**}**

**if (root->left->c != '#' && root->right->c != '#') {**

**cout << root->data << "(" << root->left->data << ", " << root->right->data << ") ";**

**}**

**}**

**if (i == power2(ceng) && T) {**

**ceng = 0;**

**return;**

**}**

**root = root->next;**

**}**

**cout << endl;**

**ceng += 1;**

**print(root);**

**}**

**9.构造平衡二叉搜索树（用二分的方法）**

**void AVL(tree t,int a[],int m,int n) {**

**int start = m;**

**int end = n;**

**if (start == end) {**

**t.Insert(t.root, a[(start + end) / 2]);**

**return;**

**}**

**else {**

**t.Insert(t.root, a[(start + end) / 2]);**

**if ((start + end) / 2 - 1 < start) {}**

**else {**

**AVL(t, a, start, (start + end) / 2 - 1);**

**}**

**if ((start + end) / 2 + 1>end) {**

**}**

**else {**

**AVL(t, a, (start + end) / 2 + 1, end);**

**} }}**

1. **算法分析**

**①空间复杂度：以为要按节点的数量来构造满二叉树，所以共 -1个节点，所以为O()**

**②时间复杂度**

1. **中序遍历 O(logn)**
2. **构造AVL O(()**
3. **打印树 O()**
4. **心得体会**
5. **用栈实现中序遍历（非递归）**
6. **用二分的方法可以构造AVL**