数据结构作业10

二叉树的遍历：（代码顺序如下）

1. 二叉树前序遍历的递归算法
2. 二叉树中序遍历的递归算法
3. 二叉树后序遍历的递归算法
4. 二叉树前序遍历的两种非递归算法
5. 二叉树中序遍历的非递归算法
6. 二叉树后序遍历的非递归算法

#include <iostream>

#include <vector>

#include <stack>

#include <queue>

using namespace std;

typedef struct BTNode{

char val;

struct BTNode\* left;

struct BTNode\* right;

}TreeNode;

vector<char> PreOrder(TreeNode\* b){

vector<char> res;

if(b){

res.push\_back(b->val);

vector<char> left = PreOrder(b->left);

vector<char> right = PreOrder(b->right);

res.insert(res.end(), left.begin(), left.end());

res.insert(res.end(), right.begin(), right.end());

}

return res;

}

vector<char> InOrder(TreeNode\* b){

vector<char> res;

if(b){

vector<char> left = InOrder(b->left);

res.insert(res.end(), left.begin(), left.end());

res.push\_back(b->val);

vector<char> right = InOrder(b->right);

res.insert(res.end(), right.begin(), right.end());

}

return res;

}

vector<char> PostOrder(TreeNode\* b){

vector<char> res;

if(b){

vector<char> left = PostOrder(b->left);

vector<char> right = PostOrder(b->right);

res.insert(res.end(), left.begin(), left.end());

res.insert(res.end(), right.begin(), right.end());

res.push\_back(b->val);

}

return res;

}

vector<char> PreOrder1(TreeNode\* b){

vector<char> res;

stack<TreeNode\*> s;

if(b) s.push(b); // 检查b是否为NULL

while(!s.empty()){ // 只要栈不为空就继续

TreeNode\* p = s.top(); //取栈顶元素

s.pop(); //栈顶元素出栈

res.push\_back(p->val); //访问栈顶元素

if(p->right) s.push(p->right); //右孩子入栈

if(p->left) s.push(p->left); //左孩子入栈

}

return res;

}

vector<char> PreOrder2(TreeNode\* b){

vector<char> res;

stack<TreeNode\*> s;

TreeNode\* p = b;

while(p || !s.empty()){ //栈不为空或者p不为空时循环

while(p){ //一直向左走

res.push\_back(p->val);

s.push(p);

p = p->left;

}

if(!s.empty()){ //栈不为空时

p = s.top(); //取栈顶元素

s.pop(); //栈顶元素出栈

p = p->right; //转而处理右子树

}

}

return res;

}

vector<char> InOrder1(TreeNode\* b){

vector<char> res;

stack<TreeNode\*> s;

TreeNode\* p = b;

while(p || !s.empty()){ //栈不为空或者p不为空时循环

while(p){ //一直向左走

s.push(p);

p = p->left;

}

if(!s.empty()){ //栈不为空时

p = s.top();//取栈顶元素

s.pop();//栈顶元素出栈

res.push\_back(p->val); //访问栈顶元素

p = p->right;//转而处理右子树

}

}

return res;

}

vector<char> PostOrder1(TreeNode\* b){

vector<char> res;

stack<TreeNode\*> s;

TreeNode\* p = b;

bool flag;

TreeNode\* pre = NULL;

do{

while(p){

s.push(p);

p = p->left;

}

pre = NULL;

flag = true; //flag为真表示当前节点的右子树还未被访问,正在处理栈顶元素

while(!s.empty() && flag){ //栈不为空且flag为真时循环

p = s.top();

s.pop();

if(p->right == pre){ //右子树已被访问

res.push\_back(p->val);

pre = p;

}else{

s.push(p);

p = p->right; //转而处理右子树

flag = false; //右子树未被访问

}

}

}while(!s.empty());

return res;

}

vector<char> bfs(TreeNode\* b){

vector<char> res;

queue<TreeNode\*> q;

TreeNode\* p = b;

q.push(p);

while(!q.empty()){

p = q.front();

q.pop();

res.push\_back(p->val);

if(p->left) q.push(p->left);

if(p->right) q.push(p->right);

}

return res;

}

求出前序遍历序列为ABDEHJKLMNCFGI

中序遍历序列为DBJHLKMNEAFCGI

后序遍历序列为 DJLNMKHEBFIGCA

根据前+中或中+后构建二叉树

TreeNode\* buildTree(vector<char>& pre, vector<char>& in){  
 if(pre.empty() || in.empty()) return **NULL**;  
 TreeNode\* root = new TreeNode;  
 root->val = pre[0];  
 vector<char> pre\_left, pre\_right, in\_left, in\_right;  
 int i = 0;  
 for(; i < in.size(); i++){  
 if(in[i] == pre[0]) break;  
 in\_left.push\_back(in[i]);  
 }  
 for(i = i + 1; i < in.size(); i++){  
 in\_right.push\_back(in[i]);  
 }  
 for(i = 1; i <= in\_left.size(); i++){  
 pre\_left.push\_back(pre[i]);  
 }  
 for(; i < pre.size(); i++){  
 pre\_right.push\_back(pre[i]);  
 }  
 root->left = buildTree(pre\_left, in\_left);  
 root->right = buildTree(pre\_right, in\_right);  
 return root;  
}

TreeNode\* buildTree1(vector<char>& in, vector<char>& post){  
 if(in.empty() || post.empty()) return **NULL**;  
 TreeNode\* root = new TreeNode;  
 root->val = post[post.size() - 1];  
 vector<char> in\_left, in\_right, post\_left, post\_right;  
 int i = 0;  
 for(; i < in.size(); i++){  
 if(in[i] == post[post.size() - 1]) break;  
 in\_left.push\_back(in[i]);  
 }  
 for(i = i + 1; i < in.size(); i++){  
 in\_right.push\_back(in[i]);  
 }  
 for(i = 0; i < in\_left.size(); i++){  
 post\_left.push\_back(post[i]);  
 }  
 for(; i < in\_right.size(); i++){  
 post\_right.push\_back(post[i]);  
 }  
 root->left = buildTree1(in\_left, post\_left);  
 root->right = buildTree1(in\_right, post\_right);  
 return root;  
}

实验4：

先序序列：abcedfhgij

中序序列：ecbhfdiga

后序序列：echfjigdba

实验4和8图如下

