

# 说明

## 一、运行环境

### 1. 语言: c++ 11

### 2. 编译器(g++)版本

```
~ ➤ g++ --version
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 9.1.0 (clang-902.0.39.1)
```

### 3. 编译命令

```
g++% -g -o %< -Wall -std=c++11
```

## 二、第一题

采用二叉链表存储表示，设计算法判断一个二叉树是否为完全二叉树

### 1. 测试

测试树🌲(输入为 `test_tree_1`) (输入时，用#代表空节点，按照中序输入)



测试结果如下

Yes  
No

### 2. 算法说明

时间复杂度:  $O(n)$

```
bool is_complete_binary_tree(Node * root) {
    bool flag = false;
```

```

queue<Node *> q;
q.push(root);
while (not q.empty()) {
    Node * cur = q.front();
    q.pop();

    if (cur -> r && ! cur -> l) return false;
    if (! cur -> l && ! cur -> r) flag = true;
    else if (flag && (cur -> l || cur -> r)) return false;

    if (cur -> l) q.push(cur -> l);
    if (cur -> r) q.push(cur -> r);
}

return true;
}

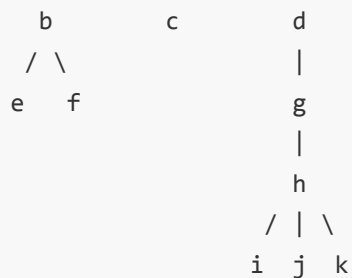
```

## 三、第二题

设森林采用二叉链表存储表示(孩子-兄弟表示法), (1)设计非递归算法实现森林的中序遍历;(2)计算森林中的叶子结点数;(3) 计算森林中一共有几棵树;(4)求森林中第一棵树的高度。

### 1. 测试

测试树🌲(输入为 `test_tree_2`) (输入方式同上)



测试结果如下

```

[in order]      efbcijkhgd
[leaf]          6
[tree]          3
[1st tree depth] 2

```

### 2. 算法说明

时间复杂度:  $O(n)$

```

void in_order_traverse(Node * root) {    // 中序遍历
    stack< Node * > s;
    Node * p = root;
    while (not s.empty() || p) {
        while (p) {
            s.push(p);
            p = p -> l;
        }
        if (not s.empty()) {
            p = s.top(); s.pop();
            cout << p -> data; p = p -> r;
        }
    }
}

```

```

int count_tree(Node * root) {    // 数树
    int ret = 0;
    Node * tree_root = root;
    while (tree_root)
        tree_root = tree_root -> r, ret ++;

    return ret;
}

```

```

int count_leaf(Node * root) {    // 数叶子
    int ret = 0;
    Node * cur = root;
    queue<Node *> q;

    q.push(root);

    while (not q.empty()) {
        cur = q.front(); q.pop();

        if (cur -> r) q.push(cur -> r);
        if (cur -> l) q.push(cur -> l);

        if (cur -> l == NULL) ret ++;
    }

    return ret;
}

```

```

int get_depth(Node * root) {    // 计算高度
    int ret = 1, cur_d;
    Node * cur = root -> l;

```

```

    if (! cur) return ret;

    queue< pair<Node *, int> > q;
    q.push(make_pair(cur, 2));

    while (not q.empty()) {
        cur = q.front().first;
        cur_d = q.front().second;
        q.pop();

        ret = max(ret, cur_d);

        if (cur -> l) q.push(make_pair(cur -> l, cur_d + 1));
        if (cur -> r) q.push(make_pair(cur -> r, cur_d));
    }


    return ret;
}

```

## 四、附加题第一题

实现二叉树先后遍历的非递归算法.

### 1. 测试

测试树  (输入为 `test_tree_3`) (输入方式同上)

```

      a
     / \
    b   c
   / \
  d   e

```

输出结果

```

[pre order]  abdec
[in order]   dbeac
[post order] debca

```

### 2. 算法说明

时间复杂度:  $O(n)$

```
void pre_order_traverse(Node * root) { // 先序遍历
```

```
    stack< Node * > s;
```

```
    Node * p = root;
```

```
    while (not s.empty() || p) {
```

```
        while (p) {
```

```
            cout << p -> data;
```

```
            s.push(p);
```

```
            p = p -> l;
```

```
        }
```

```
        if (not s.empty()) {
```

```
            p = s.top();
```

```
            s.pop();
```

```
            p = p -> r;
```

```
        }
```

```
    }
```

```
}
```

```
void in_order_traverse(Node * root) { // 中序遍历
```

```
    stack< Node * > s;
```

```
    Node * p = root;
```

```
    while (not s.empty() || p) {
```

```
        while (p) {
```

```
            s.push(p);
```

```
            p = p -> l;
```

```
        }
```

```
        if (not s.empty()) {
```

```
            p = s.top();
```

```
            s.pop();
```

```
            cout << p -> data;
```

```
            p = p -> r;
```

```
        }
```

```
    }
```

```
}
```

```
void post_order_traverse(Node * root) { // 后续遍历
```

```
    stack< Node * > s;
```

```
    Node * p = NULL, * pre = NULL;
```

```
    s.push(root);
```

```
    while (not s.empty()) {
```

```
        p = s.top();
```

```
        if ((p -> l == NULL && p -> r == NULL) || (p -> l == pre || p -> r == pre)) {
```

```


        s.pop(), pre = p;
        cout << p -> data;
    }
    else {
        if (p -> r) s.push(p -> r);
        if (p -> l) s.push(p -> l);
    }
}
}
}

```

## 附加题第二题

e树以双亲表示法存放，设计算法求树的高度

### 1. 测试

测试树  (输入为 `test_tree_4`) (输入方式：每一行第一个字母表示data，第二个表示parent编号)

```

    a
  / | \
 b  c  d
  / \
 e   f
   |
   g

```

输出结果

`[depth] 4.`