

master ▾

...

leetcode-master / problems / 算法模板.md



zy6p Update 算法模板.md ...

History

2 contributors

299 lines (252 sloc) 8.71 KB

刷题 微信群

知识星球 代码随想录

欢迎大家[参与本项目](#)，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

二分查找法

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int n = nums.size();
        int left = 0;
        int right = n; // 我们定义target在左闭右开的区间里，[left, right)
        while (left < right) { // 因为left == right的时候，在[left, right)是无效的空间
            int middle = left + ((right - left) >> 1);
            if (nums[middle] > target) {
                right = middle; // target 在左区间，因为是左闭右开的区间，nums[middle]一定
                // 不是我们的目标值，所以right = middle，在[left, middle)中继续寻找目标值
            } else if (nums[middle] < target) {
                left = middle + 1; // target 在右区间，在 [middle+1, right)中
            } else { // nums[middle] == target
                return middle; // 数组中找到目标值的情况，直接返回下标
            }
        }
        return right;
    }
};
```

KMP

```

void kmp(int* next, const string& s){
    next[0] = -1;
    int j = -1;
    for(int i = 1; i < s.size(); i++){
        while (j >= 0 && s[i] != s[j + 1]) {
            j = next[j];
        }
        if (s[i] == s[j + 1]) {
            j++;
        }
        next[i] = j;
    }
}

```

二叉树

二叉树的定义：

```

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

```

深度优先遍历（递归）

前序遍历（中左右）

```

void traversal(TreeNode* cur, vector<int>& vec) {
    if (cur == NULL) return;
    vec.push_back(cur->val);    // 中 ， 同时也是处理节点逻辑的地方
    traversal(cur->left, vec);  // 左
    traversal(cur->right, vec); // 右
}

```

中序遍历（左中右）

```

void traversal(TreeNode* cur, vector<int>& vec) {
    if (cur == NULL) return;
    traversal(cur->left, vec);  // 左
    vec.push_back(cur->val);    // 中 ， 同时也是处理节点逻辑的地方
    traversal(cur->right, vec); // 右
}

```

后序遍历（左右中）

```

void traversal(TreeNode* cur, vector<int>& vec) {
    if (cur == NULL) return;

```

```

traversal(cur->left, vec); // 左
traversal(cur->right, vec); // 右
vec.push_back(cur->val);    // 中 ， 同时也是处理节点逻辑的地方
}

```

深度优先遍历（迭代法）

相关题解：[0094.二叉树的中序遍历](#)

前序遍历（中左右）

```

vector<int> preorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> st;
    if (root != NULL) st.push(root);
    while (!st.empty()) {
        TreeNode* node = st.top();
        if (node != NULL) {
            st.pop();
            if (node->right) st.push(node->right); // 右
            if (node->left) st.push(node->left);   // 左
            st.push(node);                       // 中
            st.push(NULL);
        } else {
            st.pop();
            node = st.top();
            st.pop();
            result.push_back(node->val);          // 节点处理逻辑
        }
    }
    return result;
}

```

中序遍历（左中右）

```

vector<int> inorderTraversal(TreeNode* root) {
    vector<int> result; // 存放中序遍历的元素
    stack<TreeNode*> st;
    if (root != NULL) st.push(root);
    while (!st.empty()) {
        TreeNode* node = st.top();
        if (node != NULL) {
            st.pop();
            if (node->right) st.push(node->right); // 右
            st.push(node);                       // 中
            st.push(NULL);
            if (node->left) st.push(node->left);   // 左
        } else {
            st.pop();
            node = st.top();
            st.pop();
            result.push_back(node->val);          // 节点处理逻辑
        }
    }
}

```

```
    return result;
}
```

后序遍历（左右中）

```
vector<int> postorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> st;
    if (root != NULL) st.push(root);
    while (!st.empty()) {
        TreeNode* node = st.top();
        if (node != NULL) {
            st.pop();
            st.push(node);           // 中
            st.push(NULL);
            if (node->right) st.push(node->right); // 右
            if (node->left) st.push(node->left);   // 左
        } else {
            st.pop();
            node = st.top();
            st.pop();
            result.push_back(node->val);          // 节点处理逻辑
        }
    }
    return result;
}
```

广度优先遍历（队列）

相关题解：[0102.二叉树的层序遍历](#)

```
vector<vector<int>> levelOrder(TreeNode* root) {
    queue<TreeNode*> que;
    if (root != NULL) que.push(root);
    vector<vector<int>> result;
    while (!que.empty()) {
        int size = que.size();
        vector<int> vec;
        for (int i = 0; i < size; i++) { // 这里一定要使用固定大小size，不要使用que.size()
            TreeNode* node = que.front();
            que.pop();
            vec.push_back(node->val);    // 节点处理的逻辑
            if (node->left) que.push(node->left);
            if (node->right) que.push(node->right);
        }
        result.push_back(vec);
    }
    return result;
}
```

可以直接解决如下题目：

- [0102.二叉树的层序遍历](#)
- [0199.二叉树的右视图](#)
- [0637.二叉树的层平均值](#)
- [0104.二叉树的最大深度（迭代法）](#)
- [0111.二叉树的最小深度（迭代法）](#)
- [0222.完全二叉树的节点个数（迭代法）](#)

二叉树深度

```
int getDepth(TreeNode* node) {  
    if (node == NULL) return 0;  
    return 1 + max(getDepth(node->left), getDepth(node->right));  
}
```

二叉树节点数量

```
int countNodes(TreeNode* root) {  
    if (root == NULL) return 0;  
    return 1 + countNodes(root->left) + countNodes(root->right);  
}
```

回溯算法

```
void backtracking(参数) {  
    if (终止条件) {  
        存放结果;  
        return;  
    }  
  
    for (选择: 本层集合中元素 (树中节点孩子的数量就是集合的大小)) {  
        处理节点;  
        backtracking(路径, 选择列表); // 递归  
        回溯, 撤销处理结果  
    }  
}
```

并查集

```
int n = 1005; // 更具题意而定  
int father[1005];  
  
// 并查集初始化  
void init() {
```

```
        for (int i= 0; i < n; ++i) {
            father[i] = i;
        }
    }
    // 并查集里寻根的过程
    int find(int u) {
        return u == father[u] ? u : father[u] = find(father[u]);
    }
    // 将v->u 这条边加入并查集
    void join(int u, int v) {
        u = find(u);
        v = find(v);
        if (u == v) return ;
        father[v] = u;
    }
    // 判断 u 和 v是否找到同一个根
    bool same(int u, int v) {
        u = find(u);
        v = find(v);
        return u == v;
    }
}
```

(持续补充ing)

其他语言版本

Java:

Python:

Go:

- 作者微信: [程序员Carl](#)
- B站视频: [代码随想录](#)
- 知识星球: [代码随想录](#)