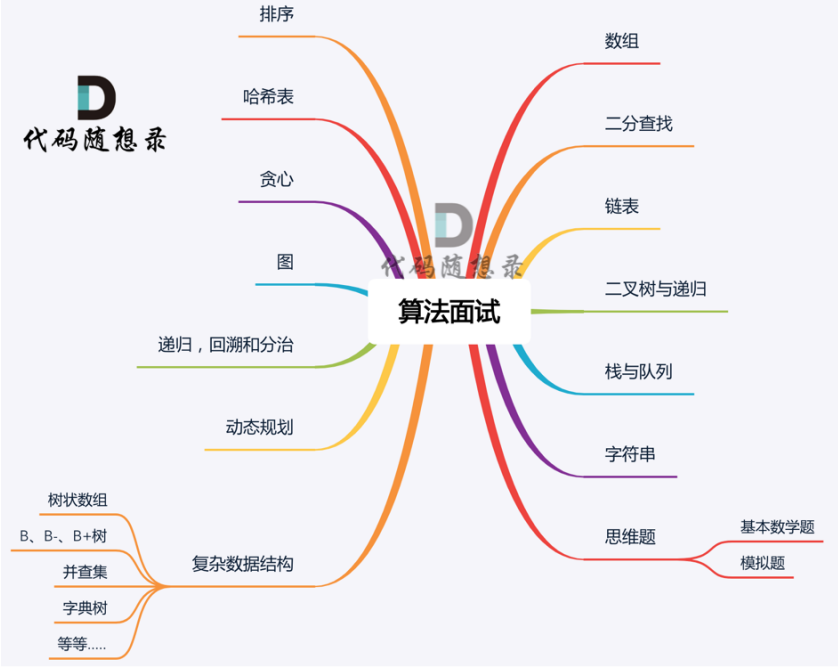
目录：

算法面试思维导图



算法文章精选

- 程序员算法面试中，必须掌握的数组理论知识
- 这五道数组相关的面试题，你一定要会！
- 这六道哈希表相关的面试题，你一定要会！
- 刷leetcode的时候，究竟什么时候可以使用库函数，什么时候不要使用库函数，过来人来说一说
- 关于链表，你该了解这些！
- 链表：听说用虚拟头节点会方便很多？
- 链表：一道题目考察了常见的五个操作！
- 链表：听说过两天反转链表又写不出来了？
- 链表：环找到了，那入口呢？
- 关于哈希表，你该了解这些！
- 哈希表：可以拿数组当哈希表来用，但哈希值不要太大
- 哈希表：哈希值太大了，还是得用set
- 哈希表：今天你快乐了么？
- 哈希表：map等候多时了
- 哈希表：其实需要哈希的地方都能找到map的身影
- 哈希表：这道题目我做过？
- 哈希表：解决了两数之和，那么能解决三数之和么？
- 双指针法：一样的道理，能解决四数之和
- 精选链表相关的面试题
- 精选字符串相关的面试题
- 精选栈与队列相关的面试题
- 精选二叉树相关的面试题
- 精选递归与回溯面试题

（持续更新中....）

LeetCode 刷题攻略

刷题顺序：建议先从同一类型里题目开始刷起，同一类型里再从简单到中等到困难刷起，题型顺序建议：**数组-> 链表-> 哈希表->字符串->栈与队列->树**。

这里我总结了各个类型的经典题目，**初学者可以按照如下顺序来刷题**，算法老手可以按照这个list查缺补漏！

- 数组经典题目

    - 0035.搜索插入位置
    - 0027.移除元素
    - 0026.删除排序数组中的重复项
    - 0209.长度最小的子数组
    - 0059.螺旋矩阵II
- 链表经典题目
    - 0203.移除链表元素
    - 0707.设计链表
    - 0206.翻转链表
    - 面试题02.07.链表相交
    - 0142.环形链表II
- 哈希表经典题目
    - 0242.有效的字母异位词
    - 0383.赎金信
    - 0575.分糖果
    - 0349.两个数组的交集
    - 0202.快乐数
    - 0001.两数之和
    - 0454.四数相加II
    - 0015.三数之和
    - 0018.四数之和
    - 0219.存在重复元素II
    - 0220.存在重复元素III
- 字符串经典题目
    - 0344.反转字符串
    - 0541.反转字符串II
    - 剑指Offer05.替换空格
    - 0151.翻转字符串里的单词
    - 延伸左旋转字符串（剑指offer上的题目）
    - 0028.实现strStr()
    - 0459.重复的子字符串
- 双指针法经典题目
    - 0015.三数之和
    - 0018.四数之和

    - 0026.删除排序数组中的重复项
    - 0206.翻转链表
    - 0142.环形链表II
    - 0344.反转字符串
    - 剑指Offer05.替换空格
- 栈与队列经典题目
    - 0232.用栈实现队列
    - 0225.用队列实现栈
    - 0020.有效的括号
    - 1047.删除字符串中的所有相邻重复项
    - 0239.滑动窗口最大值
    - 0347.前K个高频元素
- 二叉树经典题目
    - 0144.二叉树的前序遍历
    - 0094.二叉树的中序遍历
    - 0145.二叉树的后序遍历
    - 0102.二叉树的层序遍历
    - 0199.二叉树的右视图
    - 0226.翻转二叉树
    - 0101.对称二叉树
    - 0104.二叉树的最大深度
    - 0111.二叉树的最小深度
    - 0222.完全二叉树的节点个数
    - 0654.最大二叉树
    - 0617.合并二叉树
    - 0700.二叉搜索树中的搜索
    - 0098.验证二叉搜索树
    - 0701.二叉搜索树中的插入操作
    - 0450.删除二叉搜索树中的节点

(持续补充ing)

算法模板

H2 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;
    }
};

```

H2 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;
    }
}
```

H2 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) {}
};
```

H3 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
}
```

中序遍历（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
}
```

H3 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;
}
```

H3 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
- 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）

H3 二叉树深度

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

H3 二叉树节点数量

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

（持续补充ing）

H1 LeetCode 最强题解:

题目	类型	难度	解题方法
0001.两数之和	数组	简单	**暴力 哈希**
0015.三数之和	数组	中等	**双指针 哈希**

题目	类型	难度	解题方法
0017.电话号码的字母组合	回溯	中等	**回溯**
0018.四数之和	数组	中等	**双指针**
0020.有效的括号	栈	简单	**栈**
0021.合并两个有序链表	链表	简单	**模拟**
0026.删除排序数组中的重复项	数组	简单	**暴力 快慢指针/快慢指针**
0027.移除元素	数组	简单	**暴力 双指针/快慢指针/双指针**
0028.实现strStr()	字符串	简单	**KMP**
0035.搜索插入位置	数组	简单	**暴力 二分**
0037.解数独	回溯	困难	**回溯**
0039.组合总和	数组/回溯	中等	**回溯**
0040.组合总和II	数组/回溯	中等	**回溯**
0046.全排列	回溯	中等	**回溯**
0047.全排列II	回溯	中等	**回溯**
0051.N皇后	回溯	困难	**回溯**
0052.N皇后II	回溯	困难	**回溯**
0053.最大子序和	数组	简单	**暴力 贪心** 动态规划 分治

题目	类型	难度	解题方法
0059.螺旋矩阵II	数组	中等	**模拟**
0077.组合	回溯	中等	**回溯**
0078.子集	回溯/数组	中等	**回溯**
0083.删除排序链表中的重复元素	链表	简单	**模拟**
0090.子集II	回溯/数组	中等	**回溯**
0093.复原IP地址	回溯	中等	**回溯**
0094.二叉树的中序遍历	树	中等	**递归 迭代/栈**
0098.验证二叉搜索树	树	中等	**递归**
0100.相同的树	树	简单	**递归**
0101.对称二叉树	树	简单	**递归 迭代/队列/栈**
0102.二叉树的层序遍历	树	中等	**广度优先搜索/队列**
0104.二叉树的最大深度	树	简单	**递归 迭代/队列/BFS**
0110.平衡二叉树	树	简单	**递归**
0111.二叉树的最小深度	树	简单	**递归 队列/BFS**
0131.分割回文串	回溯	中等	**回溯**
0142.环形链表II	链表	中等	**快慢指针/双指针**

题目	类型	难度	解题方法
0144.二叉树的前序遍历	树	中等	**递归 迭代/栈**
0145.二叉树的后序遍历	树	困难	**递归 迭代/栈**
0151.翻转字符串里的单词	字符串	中等	**模拟/双指针**
0155.最小栈	栈	简单	**栈**
0199.二叉树的右视图	二叉树	中等	**广度优先遍历/队列**
0202.快乐数	哈希表	简单	**哈希**
0203.移除链表元素	链表	简单	**模拟 虚拟头结点**
0205.同构字符串	哈希表	简单	**哈希**
0206.翻转链表	链表	简单	**双指针法 递归**
0209.长度最小的子数组	数组	中等	**暴力 滑动窗口**
0216.组合总和III	数组/回溯	中等	**回溯**
0219.存在重复元素II	哈希表	简单	**哈希**
0222.完全二叉树的节点个数	树	简单	**递归**
0225.用队列实现栈	队列	简单	**队列**
0226.翻转二叉树	二叉树	简单	**递归 迭代**
0232.用栈实现队列	栈	简单	**栈**

题目	类型	难度	解题方法
0237.删除链表中的节点	链表	简单	**原链表移除 添加虚拟节点** 递归
0239.滑动窗口最大值	滑动窗口/队列	困难	**单调队列**
0242.有效的字母异位词	哈希表	简单	**哈希**
0344.反转字符串	字符串	简单	**双指针**
0347.前K个高频元素	哈希/堆/优先级队列	中等	**哈希/优先级队列**
0349.两个数组的交集	哈希表	简单	**哈希**
0350.两个数组的交集II	哈希表	简单	**哈希**
0383.赎金信	数组	简单	**暴力 字典计数 哈希**
0434.字符串中的单词数	字符串	简单	**模拟**
0450.删除二叉搜索树中的节点	树	中等	**递归**
0454.四数相加II	哈希表	中等	**哈希**
0459.重复的子字符串	字符创	简单	**KMP**
0491.递增子序列	深度优先搜索	中等	**深度优先搜索/回溯**
0541.反转字符串II	字符串	简单	**模拟**
0575.分糖果	哈希表	简单	**哈希**
0617.合并二叉树	树	简单	**递归 迭代**

题目	类型	难度	解题方法
0654.最大二叉树	树	中等	**递归**
0700.二叉搜索树中的搜索	树	简单	**递归 迭代**
0701.二叉搜索树中的插入操作	树	简单	**递归 迭代**
0705.设计哈希集合	哈希表	简单	**模拟**
0707.设计链表	链表	中等	**模拟**
1047.删除字符串中的所有相邻重复项	栈	简单	**栈**
剑指Offer05.替换空格	字符串	简单	**双指针**
面试题02.07.链表相交	链表	简单	**模拟**

持续更新中....

关于作者

大家好，我是程序员Carl，ACM 校赛、黑龙江省赛、东北四省赛金牌，和亚洲区域赛铜牌获得者，哈工大计算机硕士毕业，先后在腾讯和百度从事后端技术研发，CSDN博客专家。对算法和C++后端技术有一定的见解，利用工作之余重新刷leetcode。

加我的微信，备注：组队刷题， 拉你进刷题群，每天一道经典题目分析，而且题目不是孤立的，每一道题目之间都是有关系的，都是由浅入深一脉相承的，所以学习效果最好是每篇连续着看，也许之前你会某些知识点，但是一直没有把知识点串起来，这里每天一篇文章就会帮你把知识点串起来。我也花了不少精力来整理我的题解，**而且我不会在群里发任何广告，纯自己学习和分享。 欢迎你的加入！**

我的公众号

更多精彩文章持续更新，微信搜索：「代码随想录」第一时间围观，关注后回复：「简历模板」「java」「C++」「python」「算法与数据结构」等关键字就可以获得我多年整理出来的学习资料。

每天8：35准时为你推送一篇经典面试题目，帮你梳理算法知识体系，轻松学习算法！