数据结构与算法面试题

> 基于简历：LeetCode刷题240+（Medium 180+，Hard 40+），掌握常用数据结构与算法

一、链表（必考）

Q1: 反转链表（LeetCode 206）

标准实现：

```
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;
    while (curr) {
        ListNode* next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

递归实现：

```
ListNode* reverseList(ListNode* head) {
    if (!head || !head->next) return head;
    ListNode* newHead = reverseList(head->next);
    head->next->next = head;
    head->next = nullptr;
    return newHead;
}
```

关键点：

- 迭代法：三个指针（prev, curr, next）
- 递归法：理解递归返回的是新头节点
- 时间O(n)，空间O(1)（迭代）/ O(n)（递归栈）

Q2: 链表是否有环（LeetCode 141）

快慢指针：

```
bool hasCycle(ListNode *head) {
    if (!head || !head->next) return false;
```

```
    ListNode* slow = head;
    ListNode* fast = head->next;
    while (slow != fast) {
        if (!fast || !fast->next) return false;
        slow = slow->next;
        fast = fast->next->next;
    }
    return true;
}
```

找环入口（LeetCode 142）：

```
ListNode *detectCycle(ListNode *head) {
    ListNode* slow = head;
    ListNode* fast = head;

    // 判断是否有环
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) {
            // 有环，找入口
            ListNode* p = head;
            while (p != slow) {
                p = p->next;
                slow = slow->next;
            }
            return p;
        }
    }
    return nullptr;
}
```

原理：

- 快指针速度2倍，慢指针速度1倍
- 相遇时，从head和相遇点同时走，再次相遇即为环入口

Q3: 合并两个有序链表（LeetCode 21）

```
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;

    while (l1 && l2) {
        if (l1->val < l2->val) {
            tail->next = l1;
            l1 = l1->next;
```

```
            } else {
                tail->next = l2;
                l2 = l2->next;
            }
            tail = tail->next;
        }

        tail->next = l1 ? l1 : l2;
        return dummy.next;
    }
```

技巧： 使用dummy节点简化边界处理

二、栈与队列

Q4: 用栈实现队列（LeetCode 232）

```
class MyQueue {
private:
    stack<int> inStack, outStack;

    void transfer() {
        if (outStack.empty()) {
            while (!inStack.empty()) {
                outStack.push(inStack.top());
                inStack.pop();
            }
        }
    }

public:
    void push(int x) {
        inStack.push(x);
    }

    int pop() {
        transfer();
        int val = outStack.top();
        outStack.pop();
        return val;
    }

    int peek() {
        transfer();
        return outStack.top();
    }

    bool empty() {
        return inStack.empty() && outStack.empty();
```

```
    }
};
```

关键点：

- 两个栈：输入栈和输出栈
- pop/peek时，如果输出栈空，将输入栈全部转移
- 均摊时间复杂度O(1)

Q5: 有效的括号（LeetCode 20）

```
bool isValid(string s) {
    stack<char> stk;
    unordered_map<char, char> pairs = {
        {')', '('},
        {']', '['},
        {'}', '{'}
    };

    for (char c : s) {
        if (pairs.count(c)) {  // 右括号
            if (stk.empty() || stk.top() != pairs[c]) {
                return false;
            }
            stk.pop();
        } else {  // 左括号
            stk.push(c);
        }
    }

    return stk.empty();
}
```

Q6: 每日温度（LeetCode 739）- 单调栈

```
vector<int> dailyTemperatures(vector<int>& T) {
    int n = T.size();
    vector<int> result(n, 0);
    stack<int> stk;  // 存储下标

    for (int i = 0; i < n; i++) {
        while (!stk.empty() && T[i] > T[stk.top()]) {
            int idx = stk.top();
            stk.pop();
            result[idx] = i - idx;
        }
        stk.push(i);
```

```
    }

    return result;
}
```

单调栈技巧：

- 栈中元素保持单调递减
- 当前元素大于栈顶时弹出
- 应用：下一个更大元素、接雨水等

三、二叉树（高频考点）

Q7: 二叉树的最大深度（LeetCode 104）

递归：

```
int maxDepth(TreeNode* root) {
    if (!root) return 0;
    return max(maxDepth(root->left), maxDepth(root->right)) + 1;
}
```

迭代（层序遍历）：

```
int maxDepth(TreeNode* root) {
    if (!root) return 0;
    queue<TreeNode*> q;
    q.push(root);
    int depth = 0;

    while (!q.empty()) {
        int size = q.size();
        for (int i = 0; i < size; i++) {
            TreeNode* node = q.front();
            q.pop();
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
        depth++;
    }

    return depth;
}
```

Q8: 二叉树的层序遍历（LeetCode 102）

```
vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> result;
    if (!root) return result;

    queue<TreeNode*> q;
    q.push(root);

    while (!q.empty()) {
        int size = q.size();
        vector<int> level;
        for (int i = 0; i < size; i++) {
            TreeNode* node = q.front();
            q.pop();
            level.push_back(node->val);
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
        result.push_back(level);
    }

    return result;
}
```

关键： 用size记录当前层的节点数

Q9: 二叉树的前中后序遍历（递归+迭代）

前序遍历（迭代）：

```
vector<int> preorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> stk;
    TreeNode* curr = root;

    while (curr || !stk.empty()) {
        while (curr) {
            result.push_back(curr->val);  // 访问
            stk.push(curr);
            curr = curr->left;
        }
        curr = stk.top();
        stk.pop();
        curr = curr->right;
    }

    return result;
}
```

中序遍历（迭代）：

```cpp
vector<int> inorderTraversal(TreeNode* root) {
    vector<int> result;
    stack<TreeNode*> stk;
    TreeNode* curr = root;

    while (curr || !stk.empty()) {
        while (curr) {
            stk.push(curr);
            curr = curr->left;
        }
        curr = stk.top();
        stk.pop();
        result.push_back(curr->val);  // 访问
        curr = curr->right;
    }

    return result;
}
```

Q10: 二叉树的最近公共祖先（LeetCode 236）

```cpp
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
    if (!root || root == p || root == q) return root;

    TreeNode* left = lowestCommonAncestor(root->left, p, q);
    TreeNode* right = lowestCommonAncestor(root->right, p, q);

    if (left && right) return root;  // p和q分别在左右子树
    return left ? left : right;      // p和q在同一侧
}
```

四、哈希表

Q11: 两数之和（LeetCode 1）

```cpp
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> map;  // value -> index
    for (int i = 0; i < nums.size(); i++) {
        int complement = target - nums[i];
        if (map.count(complement)) {
            return {map[complement], i};
        }
        map[nums[i]] = i;
    }
    return {};
}
```

Q12: 最长连续序列（LeetCode 128）

```
int longestConsecutive(vector<int>& nums) {
    unordered_set<int> set(nums.begin(), nums.end());
    int maxLen = 0;

    for (int num : set) {
        // 只从序列起点开始计算
        if (!set.count(num - 1)) {
            int curr = num;
            int len = 1;
            while (set.count(curr + 1)) {
                curr++;
                len++;
            }
            maxLen = max(maxLen, len);
        }
    }

    return maxLen;
}
```

关键优化： 只从序列起点开始，避免重复计算

五、排序与搜索

Q13: 快速排序

```
void quickSort(vector<int>& nums, int left, int right) {
    if (left >= right) return;

    int pivot = partition(nums, left, right);
    quickSort(nums, left, pivot - 1);
    quickSort(nums, pivot + 1, right);
}

int partition(vector<int>& nums, int left, int right) {
    int pivot = nums[right];
    int i = left - 1;

    for (int j = left; j < right; j++) {
        if (nums[j] <= pivot) {
            i++;
            swap(nums[i], nums[j]);
        }
    }
```

```
    swap(nums[i + 1], nums[right]);
    return i + 1;
}
```

复杂度：

- 平均O(n log n)
- 最坏O(n²)（数组已排序）
- 空间O(log n)（递归栈）

Q14: 归并排序

```
void mergeSort(vector<int>& nums, int left, int right) {
    if (left >= right) return;

    int mid = left + (right - left) / 2;
    mergeSort(nums, left, mid);
    mergeSort(nums, mid + 1, right);
    merge(nums, left, mid, right);
}

void merge(vector<int>& nums, int left, int mid, int right) {
    vector<int> temp(right - left + 1);
    int i = left, j = mid + 1, k = 0;

    while (i <= mid && j <= right) {
        if (nums[i] <= nums[j]) {
            temp[k++] = nums[i++];
        } else {
            temp[k++] = nums[j++];
        }
    }

    while (i <= mid) temp[k++] = nums[i++];
    while (j <= right) temp[k++] = nums[j++];

    for (int i = 0; i < temp.size(); i++) {
        nums[left + i] = temp[i];
    }
}
```

特点：

- 稳定排序
- 时间O(n log n)
- 空间O(n)

Q15: 二分查找（LeetCode 704）

```cpp
int search(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            return mid;
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1;
}
```

二分查找模板（左边界）：

```cpp
int leftBound(vector<int>& nums, int target) {
    int left = 0, right = nums.size();

    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] < target) {
            left = mid + 1;
        } else {
            right = mid;
        }
    }

    return left;
}
```

六、贪心算法

Q16: 跳跃游戏（LeetCode 55）

```cpp
bool canJump(vector<int>& nums) {
    int maxReach = 0;
    for (int i = 0; i < nums.size(); i++) {
        if (i > maxReach) return false;
        maxReach = max(maxReach, i + nums[i]);
    }
    return true;
}
```

Q17: 买卖股票的最佳时机（LeetCode 121）

```cpp
int maxProfit(vector<int>& prices) {
    int minPrice = INT_MAX;
    int maxProfit = 0;

    for (int price : prices) {
        minPrice = min(minPrice, price);
        maxProfit = max(maxProfit, price - minPrice);
    }

    return maxProfit;
}
```

七、动态规划

Q18: 爬楼梯（LeetCode 70）

```cpp
int climbStairs(int n) {
    if (n <= 2) return n;
    int dp0 = 1, dp1 = 2;
    for (int i = 3; i <= n; i++) {
        int dp2 = dp0 + dp1;
        dp0 = dp1;
        dp1 = dp2;
    }
    return dp1;
}
```

Q19: 最长递增子序列（LeetCode 300）

DP解法O(n²)：

```cpp
int lengthOfLIS(vector<int>& nums) {
    int n = nums.size();
    vector<int> dp(n, 1);
    int maxLen = 1;

    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (nums[i] > nums[j]) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
```

```
            maxLen = max(maxLen, dp[i]);
        }

        return maxLen;
    }
```

贪心+二分O(n log n)：

```
int lengthOfLIS(vector<int>& nums) {
    vector<int> tails;
    for (int num : nums) {
        auto it = lower_bound(tails.begin(), tails.end(), num);
        if (it == tails.end()) {
            tails.push_back(num);
        } else {
            *it = num;
        }
    }
    return tails.size();
}
```

Q20: 最长公共子序列（LeetCode 1143）

```
int longestCommonSubsequence(string text1, string text2) {
    int m = text1.size(), n = text2.size();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (text1[i-1] == text2[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
            } else {
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
        }
    }

    return dp[m][n];
}
```

八、回溯算法

Q21: 全排列（LeetCode 46）

```cpp
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> result;
        vector<int> path;
        vector<bool> used(nums.size(), false);
        backtrack(nums, path, used, result);
        return result;
    }

    void backtrack(vector<int>& nums, vector<int>& path,
                   vector<bool>& used, vector<vector<int>>& result) {
        if (path.size() == nums.size()) {
            result.push_back(path);
            return;
        }

        for (int i = 0; i < nums.size(); i++) {
            if (used[i]) continue;
            path.push_back(nums[i]);
            used[i] = true;
            backtrack(nums, path, used, result);
            path.pop_back();
            used[i] = false;
        }
    }
};
```

Q22: 子集（LeetCode 78）

```cpp
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> result;
        vector<int> path;
        backtrack(nums, 0, path, result);
        return result;
    }

    void backtrack(vector<int>& nums, int start,
                   vector<int>& path, vector<vector<int>>& result) {
        result.push_back(path);

        for (int i = start; i < nums.size(); i++) {
            path.push_back(nums[i]);
            backtrack(nums, i + 1, path, result);
            path.pop_back();
        }
```

```
    }
};
```

九、滑动窗口

Q23: 无重复字符的最长子串（LeetCode 3）

```
int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> window;
    int left = 0, maxLen = 0;

    for (int right = 0; right < s.size(); right++) {
        char c = s[right];
        window[c]++;

        while (window[c] > 1) {
            char d = s[left];
            window[d]--;
            left++;
        }

        maxLen = max(maxLen, right - left + 1);
    }

    return maxLen;
}
```

Q24: 最小覆盖子串（LeetCode 76）

```
string minWindow(string s, string t) {
    unordered_map<char, int> need, window;
    for (char c : t) need[c]++;

    int left = 0, right = 0;
    int valid = 0;
    int start = 0, len = INT_MAX;

    while (right < s.size()) {
        char c = s[right];
        right++;
        if (need.count(c)) {
            window[c]++;
            if (window[c] == need[c]) {
                valid++;
            }
        }
```

```
        while (valid == need.size()) {
            if (right - left < len) {
                start = left;
                len = right - left;
            }

            char d = s[left];
            left++;
            if (need.count(d)) {
                if (window[d] == need[d]) {
                    valid--;
                }
                window[d]--;
            }
        }
    }

    return len == INT_MAX ? "" : s.substr(start, len);
}
```

十、高频算法模板总结

1. 二分查找模板

```
// 基础二分
int binarySearch(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) return mid;
        else if (nums[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

// 左边界
int leftBound(vector<int>& nums, int target) {
    int left = 0, right = nums.size();
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] < target) left = mid + 1;
        else right = mid;
    }
    return left;
}

// 右边界
int rightBound(vector<int>& nums, int target) {
    int left = 0, right = nums.size();
```

```
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] <= target) left = mid + 1;
        else right = mid;
    }
    return left - 1;
}
```

2. 滑动窗口模板

```
int slidingWindow(string s, string t) {
    unordered_map<char, int> need, window;
    for (char c : t) need[c]++;

    int left = 0, right = 0;
    int valid = 0;

    while (right < s.size()) {
        char c = s[right];
        right++;
        // 窗口内数据更新
        ...

        while (window needs shrink) {
            char d = s[left];
            left++;
            // 窗口内数据更新
            ...
        }
    }
}
```

3. 回溯算法模板

```
void backtrack(路径, 选择列表) {
    if (满足结束条件) {
        result.add(路径);
        return;
    }

    for (选择 in 选择列表) {
        做选择;
        backtrack(路径, 选择列表);
        撤销选择;
    }
}
```

4. BFS模板

```
int BFS(Node start, Node target) {
    queue<Node> q;
    unordered_set<Node> visited;

    q.push(start);
    visited.insert(start);
    int step = 0;

    while (!q.empty()) {
        int size = q.size();
        for (int i = 0; i < size; i++) {
            Node cur = q.front();
            q.pop();

            if (cur == target) return step;

            for (Node next : cur.adj()) {
                if (!visited.count(next)) {
                    q.push(next);
                    visited.insert(next);
                }
            }
        }
        step++;
    }
    return -1;
}
```

5. DFS模板

```
void DFS(Node node, unordered_set<Node>& visited) {
    if (visited.count(node)) return;
    visited.insert(node);

    // 处理node

    for (Node next : node.adj()) {
        DFS(next, visited);
    }
}
```

十一、面试现场编程技巧

1. 问题分析流程

1. **理解题意**：复述题目，确认输入输出
2. **示例分析**：手动模拟1-2个例子

3. **边界条件**：空数组、单元素、重复元素等
4. **时间空间复杂度**：先说明目标复杂度

2. 编码注意事项

```
// ✅ 好的习惯
// 1. 变量名有意义
int maxProfit;  // 而不是 int mp;

// 2. 边界检查
if (nums.empty()) return 0;

// 3. 注释关键步骤
// 窗口右移，更新窗口内数据
right++;

// 4. 空行分隔逻辑块

// 5. 及时测试
// 写完一段逻辑就用例子验证
```

3. 优化思路

1. **暴力 → 优化**：先说暴力解法，再优化
2. **空间换时间**：哈希表、缓存
3. **预处理**：排序、前缀和
4. **双指针**：滑动窗口、快慢指针
5. **分治**：归并、快排

4. 常见坑

```
// ❌ 整数溢出
int mid = (left + right) / 2;
// ✅ 正确
int mid = left + (right - left) / 2;

// ❌ 数组越界
if (nums[i+1] > nums[i])
// ✅ 正确
if (i+1 < nums.size() && nums[i+1] > nums[i])

// ❌ 死循环
while (left < right) {
    int mid = (left + right) / 2;
    if (...) left = mid;  // 可能死循环
}
// ✅ 正确
left = mid + 1;
```

十二、LeetCode高频题清单

Easy（必刷）

- [ ] 1. 两数之和
- [ ] 20. 有效的括号
- [ ] 21. 合并两个有序链表
- [ ] 70. 爬楼梯
- [ ] 104. 二叉树的最大深度
- [ ] 121. 买卖股票的最佳时机
- [ ] 141. 环形链表
- [ ] 206. 反转链表

Medium（重点）

- [ ] 3. 无重复字符的最长子串
- [ ] 15. 三数之和
- [ ] 46. 全排列
- [ ] 53. 最大子数组和
- [ ] 78. 子集
- [ ] 102. 二叉树的层序遍历
- [ ] 146. LRU缓存
- [ ] 200. 岛屿数量
- [ ] 236. 二叉树的最近公共祖先
- [ ] 300. 最长递增子序列

Hard（挑战）

- [ ] 23. 合并K个升序链表
- [ ] 42. 接雨水
- [ ] 76. 最小覆盖子串
- [ ] 124. 二叉树中的最大路径和
- [ ] 239. 滑动窗口最大值

面试建议

1. **手写代码要规范**：变量命名清晰、逻辑分块
2. **边界条件要考虑**：空数组、单元素、负数等
3. **复杂度要分析**：说明时间和空间复杂度
4. **优化思路要清晰**：从暴力到优化的演进过程
5. **测试用例要验证**：写完后自己跑一遍简单例子

刷题策略：质量 > 数量，理解 > 记忆