大厂面试题 - Leetcode

王红元 coderwhy

目录
content

1 字符串大厂面试题

2 栈结构大厂面试题

3 队列结构大厂面试题

4 链表结构大厂面试题

5 二叉树大厂面试题

6 动态规划大厂面试题

14. 最长公共前缀（Google、字节、亚马逊等）

- 14. 最长公共前缀（Longest Common Prefix）
  - https://leetcode.cn/problems/longest-common-prefix/
- 题目：
  - 编写一个函数来查找字符串数组中的最长公共前缀。
  - 如果不存在公共前缀，返回空字符串 ""。

示例 1：

```
输入：strs = ["flower","flow","flight"]
输出："fl"
```

示例 2：

```
输入：strs = ["dog","racecar","car"]
输出：""
解释：输入不存在公共前缀。
```

14. 最长公共前缀 – 代码实现

```
function longestCommonPrefix(strs: string[]): string {
  if (strs.length === 0) return ""

  // 首先直接用第一个字符串作为公共前缀
  let prefix = strs[0]
  for (let i = 1; i < strs.length; i++) {
    while (strs[i].indexOf(prefix) !== 0) {
      // 每次长度-1
      prefix = prefix.slice(0, prefix.length - 1)
      if (prefix.length === 0) {
        return ""
      }
    }
  }

  return prefix
}
```

3. 无重复字符的最长子串（字节、亚马逊、阿里等）

- 3. 无重复字符的最长子串
  - https://leetcode.cn/problems/longest-substring-without-repeating-characters/description/
- 题目：
  - 给定一个字符串 s， 请你找出其中不含有重复字符的 **最长子串** 的长度。

示例 1:

```
输入: s = "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
```

示例 2:

```
输入: s = "bbbbb"
输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
```

示例 3:

```
输入: s = "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
```

3. 无重复字符的最长子串 – 代码实现

解题思路:
1.双指针

- right指针的作用是从头扫描到尾部
- left指针指向什么?
  - 默认指向0的位置
  - 在发现right指针出现了重复的字符时

```
function lengthOfLongestSubstring(s: string): number {
  const n = s.length

  // 1.定义需要用到的变量
  const map = new Map<string, number>()
  let maxLength = 0
  let left = 0
  for (let right = 0; right < n; right++) {
    const rightChar = s[right]

    // 保留最新的索引之前, 先判断是否之前这个字符已经出现过
    if (map.has(rightChar) && map.get(rightChar)! >= left) {
      left = map.get(rightChar)! + 1
    }

    map.set(rightChar, right)

    const currLength = right - left + 1
    maxLength = Math.max(currLength, maxLength)
  }

  return maxLength
};

console.log(lengthOfLongestSubstring("abcabcbb"))
```

5. 最长回文子串（亚马逊、华为、苹果等）

- 5. 最长回文子串
  - https://leetcode.cn/problems/longest-palindromic-substring/description/
- 题目：
  - 给你一个字符串 s，找到 s 中最长的回文子串。
  - 如果字符串的反序与原始字符串相同，则该字符串称为回文字符串。

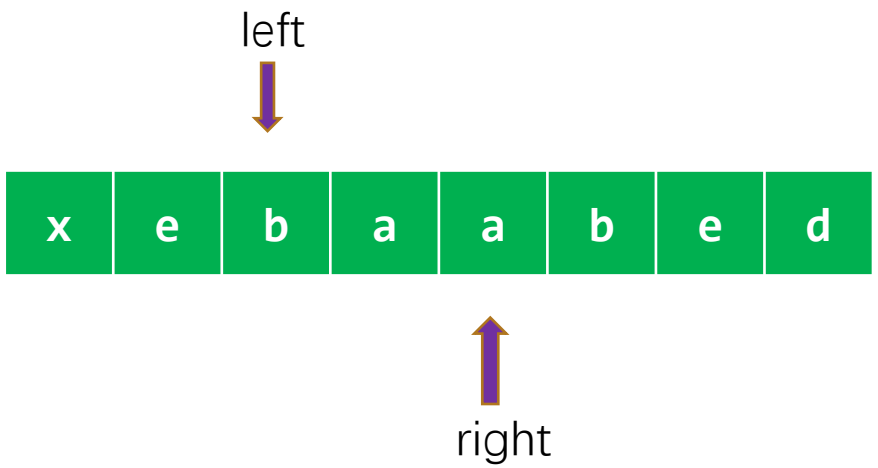
示例 1：

```
输入：s = "babad"
输出："bab"
解释："aba" 同样是符合题意的答案。
```

示例 2：

```
输入：s = "cbbd"
输出："bb"
```

5. 最长回文子串 – 代码实现




```ts
function centerExpand(s: string, left: number, right: number): number {
  while (left >= 0 && right < s.length && s[left] === s[right]) {
    left--
    right++
  }
  return right - left - 1
}
```

```ts
function longestPalindrome(s: string): string {
  const n = s.length
  // 边界判断的情况
  if (n <= 1) return s

  // 核心思路: 对称
  let start = 0
  let end = 0
  for (let i = 0; i < n; i++) {
    const length1 = centerExpand(s, i, i)
    const length2 = centerExpand(s, i, i + 1)
    const len = Math.max(length1, length2)

    if (len > end - start) {
      const left = i - Math.floor((len - 1) / 2)
      const right = i + Math.floor(len / 2)

      // 新的长度比原来保存的start/end要长, 重新赋值
      start = left
      end = right
    }
  }

  return s.substring(start, end + 1)
};
```

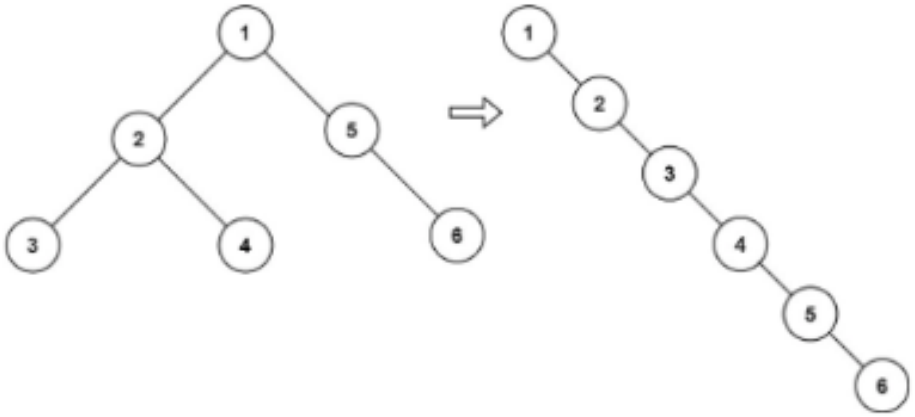
动态规划 – Manacher算法（课下扩展）

```typescript
function longestPalindrome(s: string): string {
  const n = s.length;
  const dp: boolean[][] = [];

  let ans = '';

  // 枚举子串的长度 l
  for (let l = 0; l < n; ++l) {
    // 枚举子串的起始位置 i
    for (let i = 0; i + l < n; ++i) {
      const j = i + l; // 子串的结束位置

      // 边界条件
      if (l === 0) {
        dp[i] = dp[i] || [];
        dp[i][j] = true;
      } else if (l === 1) {
        dp[i] = dp[i] || [];
        dp[i][j] = s[i] === s[j];
      } else {
        dp[i] = dp[i] || [];
        // 状态转移方程
        dp[i][j] = s[i] === s[j] && dp[i + 1][j - 1];
      }

      // 如果当前的子串是回文串，并且长度大于目前已知的最长回文串
      if (dp[i][j] && l + 1 > ans.length) {
        ans = s.substring(i, i + l + 1);
      }
    }
  }

  return ans;
}
```

```typescript
function longestPalindrome(s: string): string {
  const t = `#${s.split("").join("#")}#` // 在字符串 s 的每个字符之间添加 #，得到新字符
  const n = t.length // t 的长度
  const f = Array(n).fill(0) // 存储以每个字符为中心的最长回文子串的半径
  let maxRight = 0,
    center = 0 // 最长回文子串能够到达的最右端的位置和对应的中心位置
  let maxLen = 0,
    centerIndex = 0 // 最长回文子串的长度和对应的中心位置
  for (let i = 0; i < n; i++) {
    // 如果 i 在 maxRight 的左侧，那么可以利用对称性减少计算量
    f[i] = i < maxRight ? Math.min(f[2 * center - i], maxRight - i) : 1

    // 尝试将当前回文子串的中心扩展到 i
    while (i - f[i] >= 0 && i + f[i] < n && t[i - f[i]] === t[i + f[i]]) {
      f[i]++
    }
    // 如果当前回文子串的右端超过了之前的最长回文子串的右端，更新相关变量
    if (i + f[i] - 1 > maxRight) {
      maxRight = i + f[i] - 1
      center = i
    }
    // 如果当前回文子串的长度超过了之前的最长回文子串的长度，更新相关变量
    if (f[i] > maxLen) {
      maxLen = f[i]
      centerIndex = i
    }
  }

  // 根据中心位置和长度构造最长回文子串并返回
  const start = (centerIndex - maxLen + 1) / 2 // 最长回文子串在字符串 s 中的起始位置
  return s.slice(start, start + maxLen - 1) // 提取最长回文子串并返回
}
```

114. 二叉树展开为链表（微软、字节、谷歌）

- 114. 二叉树展开为链表
  - https://leetcode.cn/problems/flatten-binary-tree-to-linked-list/
- 题目：
  - 给你二叉树的根结点 root， 请你将它展开为一个单链表：
    - ✓ 展开后的单链表应该同样使用 TreeNode， 其中 right 子指针指向链表中下一个结点，而左子指针始终为 null 。
    - ✓ 展开后的单链表应该与二叉树 **先序遍历** 顺序相同。



相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

微软 Microsoft 7　Facebook 5　字节跳动 3　谷歌 Google 2

苹果 Apple 2

相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

彭博 Bloomberg 8　优步 Uber 3　高盛集团 Goldman Sachs 2　阿里巴巴 1

今日头条 1　英伟达 NVIDIA 1　eBay 1　VMware 1　甲骨文 Oracle 1

拼多多 1

114. 二叉树展开为链表 – 代码实现

- 可以使用栈结构：
  - 将当前节点的右子树和左子树依次压入栈中。
  - 然后再取出栈顶元素，将其左子树设为空，右子树设为栈顶元素。
  - 再继续将新的右子树和左子树压入栈中，重复这个过程直到栈为空。

```
function flatten(root: TreeNode | null): void {
  if (root === null) return;

  const stack = [root];
  let prev: TreeNode | null = null;

  while (stack.length) {
    const curr = stack.pop()!;
    if (prev !== null) {
      prev.left = null;
      prev.right = curr;
    }

    const left = curr.left;
    const right = curr.right;
    if (right !== null) {
      stack.push(right);
    }
    if (left !== null) {
      stack.push(left);
    }

    prev = curr;
  }
}
```

150. 逆波兰表达式求值（亚马逊、谷歌、腾讯等）

- 150. 逆波兰表达式求值
  - https://leetcode.cn/problems/evaluate-reverse-polish-notation/description/
- 题目：
  - 给你一个字符串数组 tokens ， 表示一个根据 逆波兰表示法 表示的算术表达式。
  - 请你计算该表达式。返回一个表示表达式值的整数。
  - **注意：**
    - ✓ 有效的算符为 '+'、'-'、'*' 和 '/' 。
    - ✓ 每个操作数（运算对象）都可以是一个整数或者另一个表达式。
    - ✓ 两个整数之间的除法总是 **向零截断** 。
    - ✓ 表达式中不含除零运算。
    - ✓ 输入是一个根据逆波兰表示法表示的算术表达式。
    - ✓ 答案及所有中间计算结果可以用 **32 位** 整数表示。

150. 逆波兰表达式求值 – 代码实现

- 使用了一个栈来存储数字和运算符。
  - 遍历 tokens 数组，遇到数字时将其压入栈中，遇到运算符时从栈中弹出两个数字并进行相应的计算，将计算结果再压入栈中。
  - 最后栈中剩下的数字就是表达式的值。

```ts
function isOperator(token: string): boolean {
  return (
    token === '+' ||
    token === "-" ||
    token === '*' ||
    token === "/"
  )
}
```

```ts
function evalRPN(tokens: string[]): number {
  const stack: number[] = []
  for (const token of tokens) {
    if (isOperator(token)) {
      const num1 = stack.pop()!
      const num2 = stack.pop()!
      switch (token) {
        case '+':
          stack.push(num1 + num2)
          break
        case '-':
          stack.push(num1 - num2)
          break
        case '*':
          stack.push(num1 * num2)
          break
        case '/':
          stack.push(Math.trunc(num1 + num2))
          break
      }
    } else {
      stack.push(Number(token))
    }
  }
  return stack[0]
}
```

```ts
function evalRPN(tokens: string[]): number {
  const stack: number[] = []

  for (let i = 0; i < tokens.length; i++) {
    const token = tokens[i]
    if (token === "+") {
      const num2 = stack.pop()!
      const num1 = stack.pop()!
      stack.push(num1 + num2)
    } else if (token === "-") {
      const num2 = stack.pop()!
      const num1 = stack.pop()!
      stack.push(num1 - num2)
    } else if (token === "*") {
      const num2 = stack.pop()!
      const num1 = stack.pop()!
      stack.push(num1 * num2)
    } else if (token === "/") {
      const num2 = stack.pop()!
      const num1 = stack.pop()!
      stack.push(Math.trunc(num1 / num2))
    } else {
      stack.push(Number(token))
    }
  }
  return stack.pop()!
}
```

剑指 Offer 09. 用两个栈实现队列（字节、百度等）

- 剑指 Offer 09. 用两个栈实现队列：
  - https://leetcode.cn/problems/yong-liang-ge-zhan-shi-xian-dui-lie-lcof/description/
- 题目：
  - 用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1)

示例 1：

```
输入：
["CQueue","appendTail","deleteHead","deleteHead","deleteHead"]
[[],[3],[],[],[]]
输出：[null,null,3,-1,-1]
```

示例 2：

```
输入：
["CQueue","deleteHead","appendTail","appendTail","deleteHead","deleteHead"]
[[],[],[5],[2],[],[]]
输出：[null,-1,null,null,5,2]
```

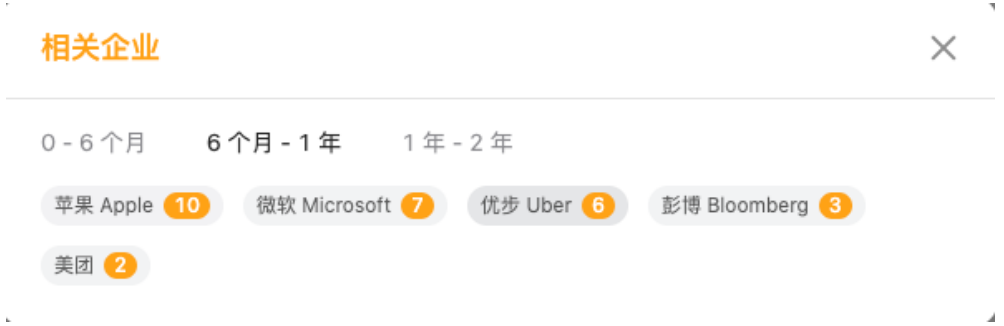
用两个栈实现队列 – 代码实现

- 使用两个栈 s1 和 s2：s1 用来插入元素，s2 用来删除元素
  - 其中插入元素只需要将元素插入 s1 即可
  - 删除元素则需要分情况：
    - ✓ 如果 s2 不为空，直接弹出 s2 的栈顶元素；
    - ✓ 如果 s2 为空，将 s1 中的元素逐个弹出并压入 s2，然后弹出 s2 的栈顶元素；

```
class CQueue {
  private s1: number[] = []
  private s2: number[] = []

  constructor() {}

  appendTail(value: number): void {
    this.s1.push(value)
  }

  deleteHead(): number {
    if (this.s2.length > 0) {
      // s2不为空，直接从s2弹出
      return this.s2.pop()!
    } else if (this.s1.length > 0) {
      // s2为空，s1不为空
      while (this.s1.length > 0) {
        this.s2.push(this.s1.pop()!)
      }
      return this.s2.pop()!
    } else {
      return -1
    }
  }
}
```

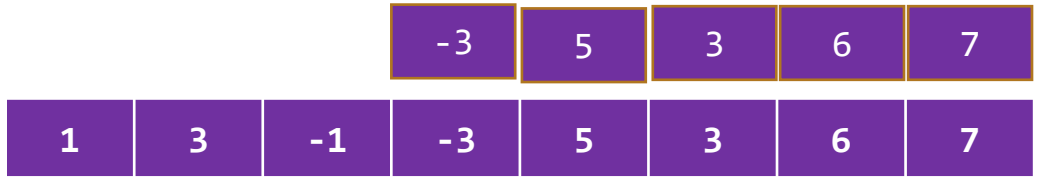
239. 滑动窗口最大值（亚马逊、谷歌、字节等）

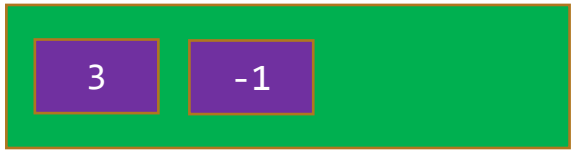
- 239. 滑动窗口最大值
  - https://leetcode.cn/problems/sliding-window-maximum/description/



- 首先，我们需要定义一个双端队列 deque 用来存储下标，一个空数组 res 用来存储结果。
- 接着，我们遍历整个数组，对于当前的数字 nums[i]，如果双端队列 deque 不为空，并且当前数字 nums[i] 大于等于队列末尾的数字，则我们弹出队列末尾的数字，直到队列为空或者当前数字 nums[i] 小于队列末尾的数字。这样可以保证队列中的数字是单调递减的。然后，我们将当前数字的下标 i 入队。
- 接下来，我们需要保证队列中的数字是在滑动窗口范围内的。如果队列头部的数字的下标小于等于 i - k，说明这个数字已经不在滑动窗口内，我们需要弹出队列头部的数字。
- 最后，如果当前下标 i 大于等于 k - 1，我们将队列头部数字所对应的 nums 中的数字加入到结果数组 res 中。

双端队列的实现





滑动窗口	最大值
[1, 3, -1]	3
[3, -1, -3]	3
[-1, -3, 5]	5
[-3, 5, 3]	5
[5, 3, 6]	6
[3, 6, 7]	7

1.创建一个双端队列(数组)
2.遍历每3个元素
- 每遍历到一个元素, 就将其添加到队列尾部中
  - 如果新添加的元素, 比队列原来尾部的元素要大
  - 那么之前尾部的元素删除掉

```
function maxSlidingWindow(nums: number[], k: number): number[] {
  const n = nums.length

  // 窗口双端队列结构
  const deque: number[] = []
  const res: number[] = []

  // 遍历每一个元素
  for (let i = 0; i < n; i++) {
    // 将元素放入到队列的尾部
    while (deque.length && nums[i] > nums[deque[deque.length - 1]]) {
      deque.pop()
    }
    deque.push(i)

    // 判断目前队列的头部元素的索引是否在范围之内
    while (deque[0] <= i - k) {
      deque.shift()
    }

    // 获取到头部的值, 作为最大值
    if (i >= k - 1) {
      const max = nums[deque[0]]
      res.push(max)
    }
  }

  return res
}
```

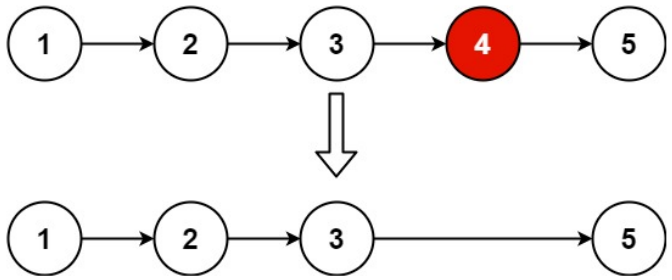
动态规划实现（课下扩展）

```ts
function maxSlidingWindow(nums: number[], k: number): number[] {
  const n = nums.length;
  if (n * k === 0) return [];
  if (k === 1) return nums;

  const left = new Array(n);
  left[0] = nums[0];
  const right = new Array(n);
  right[n - 1] = nums[n - 1];
  for (let i = 1; i < n; i++) {
    if (i % k === 0) left[i] = nums[i];
    else left[i] = Math.max(left[i - 1], nums[i]);
    const j = n - i - 1;
    if ((j + 1) % k === 0) right[j] = nums[j];
    else right[j] = Math.max(right[j + 1], nums[j]);
  }

  const output = new Array(n - k + 1);
  for (let i = 0; i < n - k + 1; i++) {
    output[i] = Math.max(right[i], left[i + k - 1]);
  }

  return output;
}
```

19. 删除链表的倒数第 N 个结点（字节、谷歌等）

- 19. 删除链表的倒数第 N 个结点
  - https://leetcode.cn/problems/remove-nth-node-from-end-of-list/description/
- 题目：
  - 给你一个链表，删除链表的倒数第 n 个结点，并且返回链表的头结点。

示例 1：




```
输入：head = [1,2,3,4,5], n = 2
输出：[1,2,3,5]
```

示例 2：

```
输入：head = [1], n = 1
输出：[]
```

示例 3：

```
输入：head = [1,2], n = 1
输出：[1]
```

相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

字节跳动 7　亚马逊 5　谷歌 Google 2　微软 Microsoft 2　彭博 Bloomberg 2

相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

Facebook 17　苹果 Apple 9　优步 Uber 2　eBay 2　英伟达 NVIDIA 2　英特尔 Intel 2

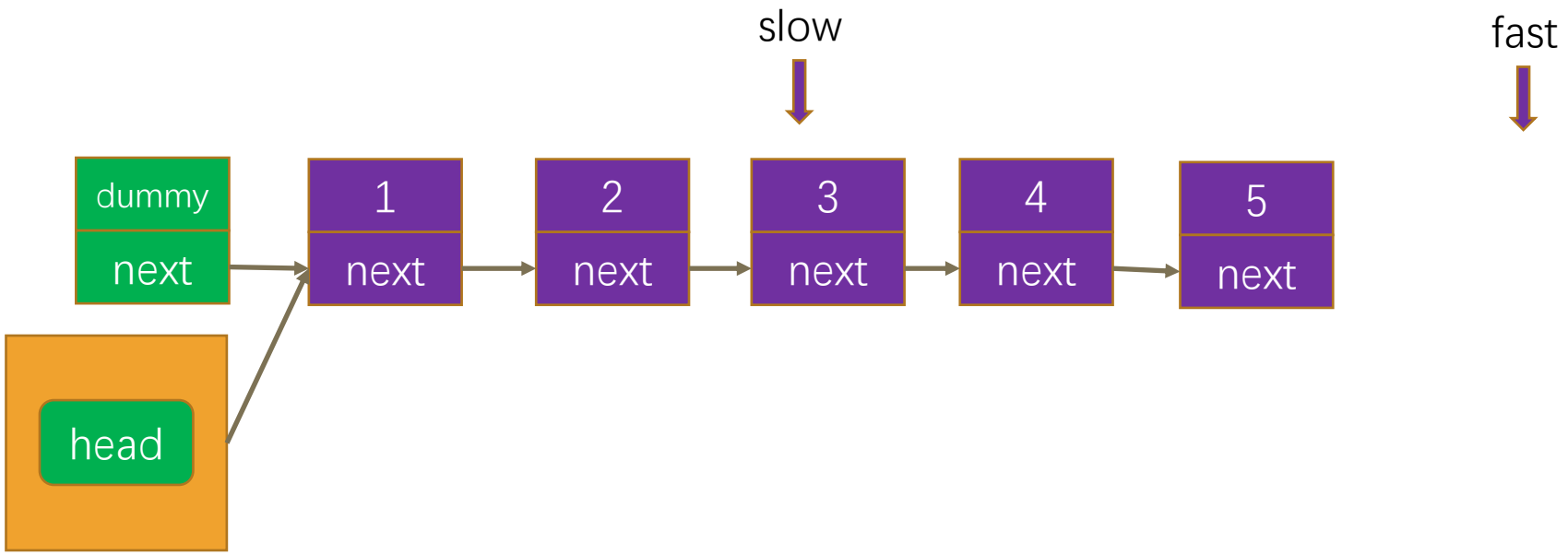
相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

华为 5　美团 5　百度 4　阿里巴巴 4　多益网络 3　VMware 3　PayPal 3　蚂蚁集团 2　腾讯 2　Shopee 2

删除链表的倒数第 N 个结点 – 代码实现

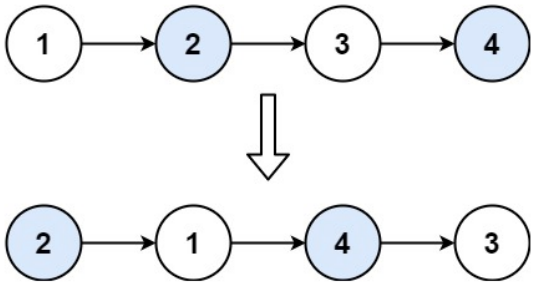
- 可以使用双指针来解决这个问题。
  - 首先让快指针先移动 n 步，然后让慢指针和快指针一起移动，直到快指针到达链表末尾。
  - 此时慢指针所指的节点就是要删除的节点的前一个节点，可以将其指向下下个节点，从而删除倒数第 n 个节点。
  - 其中 dummy 节点是为了方便处理边界情况而添加的。



24. 两两交换链表中的节点（谷歌、亚马逊、微软等）

- 24. 两两交换链表中的节点
  - https://leetcode.cn/problems/swap-nodes-in-pairs/description/
- 题目：
  - 给你一个链表，两两交换其中相邻的节点，并返回交换后链表的头节点。你必须在不修改节点内部的值的情况下完成本题（即，只能进行节点交换）。

示例 1：



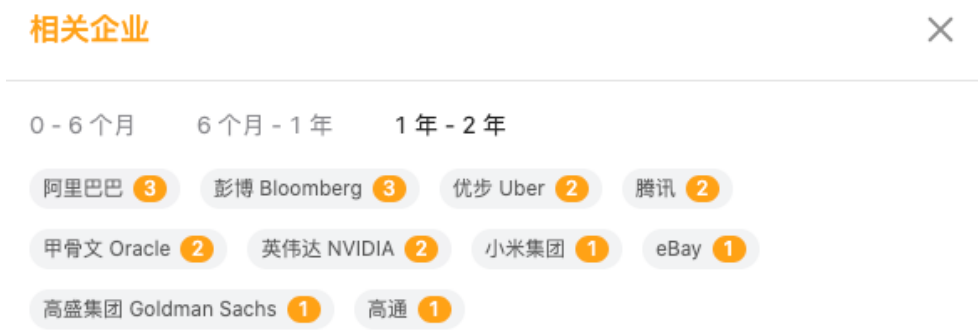

```
输入：head = [1,2,3,4]
输出：[2,1,4,3]
```

示例 2：

```
输入：head = []
输出：[]
```

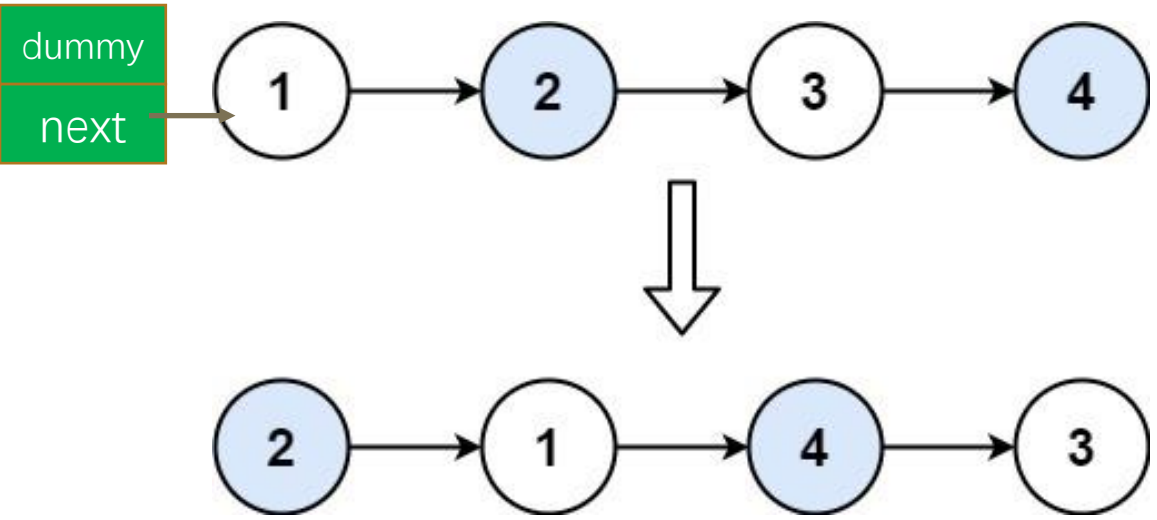
示例 3：

```
输入：head = [1]
输出：[1]
```



24. 两两交换链表中的节点 – 代码实现

- 实现思路：
  - 首先添加一个 dummy 节点；
  - 创建一个p节点，默认指向虚拟节点（这里因为有虚拟节点，所以可以直接调用next）；
  - 使用一个指针 p 依次指向每组相邻的节点，然后交换这两个节点的位置，直到遍历完整个链表。




```
function swapPairs(head: ListNode | null): ListNode | null {
  // 1.创建虚拟节点
  const dummy = new ListNode(0)
  dummy.next = head

  // 2.创建current节点，指向虚拟节点
  let current = dummy
  while (current.next && current.next.next) {
    // 将接下来的两个节点取出
    const node1 = current.next
    const node2 = current.next.next

    // 交换node1和node2的位置
    current.next = node2
    node1.next = node2.next
    node2.next = node1

    // 开始进行下一次的交换
    current = node1
  }

  return dummy.next
};
```

144. 二叉树的前序遍历（谷歌、微软、苹果等）

- 144. 二叉树的前序遍历
  - https://leetcode.cn/problems/binary-tree-preorder-traversal/

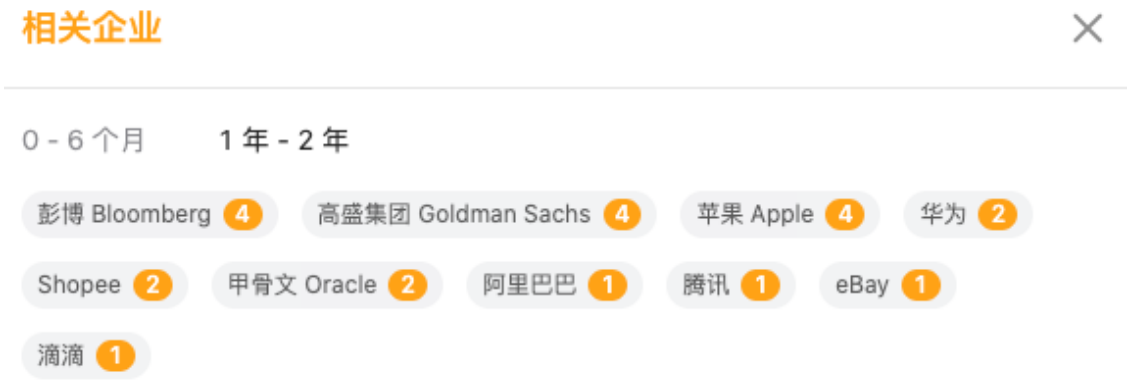
144. 二叉树的前序遍历（课下完成）

```ts
function preorderTraversal(root: TreeNode | null): number[] {
  const result: number[] = []

  function traverse(node: TreeNode | null) {
    if (!node) {
      return
    }

    result.push(node.val)
    traverse(node.left)
    traverse(node.right)
  }

  traverse(root)

  return result
}
```

```ts
function preorderTraversal(root: TreeNode | null): number[] {
  const res: number[] = []
  const stack: TreeNode[] = []

  if (!root) {
    return res
  }

  stack.push(root)

  while (stack.length > 0) {
    const node = stack.pop()!
    res.push(node.val)

    if (node.right) {
      stack.push(node.right)
    }

    if (node.left) {
      stack.push(node.left)
    }
  }

  return res
}
```

94. 二叉树的中序遍历（亚马逊、谷歌、阿里等）

- 94. 二叉树的中序遍历
  - https://leetcode.cn/problems/binary-tree-inorder-traversal/



94. 二叉树的中序遍历（课下完成）

```ts
function inorderTraversal(root: TreeNode | null): number[] {
  const result: number[] = [];

  function traversal(node: TreeNode | null) {
    if (!node) {
      return;
    }

    traversal(node.left);
    result.push(node.val);
    traversal(node.right);
  }

  traversal(root);
  return result;
}
```

```ts
function inorderTraversal(root: TreeNode | null): number[] {
  const stack: TreeNode[] = [];
  const res: number[] = [];

  let curr = root;

  while (curr || stack.length > 0) {
    while (curr) {
      stack.push(curr);
      curr = curr.left;
    }
    curr = stack.pop()!;
    res.push(curr.val);
    curr = curr.right;
  }

  return res;
}
```

145. 二叉树的后序遍历（字节、腾讯、苹果等）

- 145. 二叉树的后序遍历
  - https://leetcode.cn/problems/binary-tree-postorder-traversal/

145. 二叉树的后序遍历 （课下完成）

```ts
function postorderTraversal(root: TreeNode | null): number[] {
  const res: number[] = [];

  function traversal(node: TreeNode | null) {
    if (!node) {
      return;
    }
    traversal(node.left);
    traversal(node.right);
    res.push(node.val);
  }

  traversal(root);

  return res;
}
```

```ts
function postorderTraversal2(root: TreeNode | null): number[] {
  const res: number[] = [];
  const stack: TreeNode[] = [];

  if (root) {
    stack.push(root);
  }

  while (stack.length) {
    const node = stack.pop()!;
    res.push(node.val);
    if (node.left) {
      stack.push(node.left);
    }
    if (node.right) {
      stack.push(node.right);
    }
  }

  return res.reverse();
}
```

102. 二叉树的层序遍历（字节、亚马逊、阿里等）

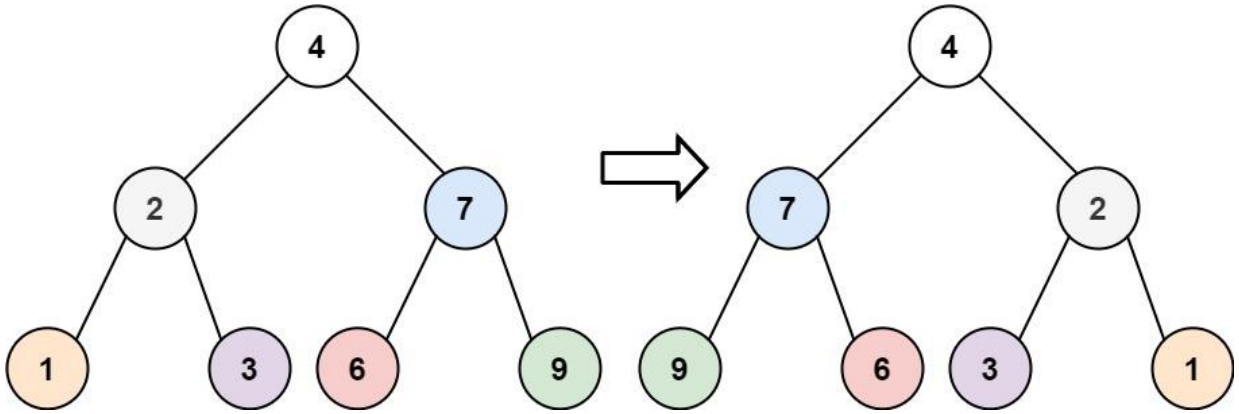
- 102. 二叉树的层序遍历
  - https://leetcode.cn/problems/binary-tree-level-order-traversal/description/

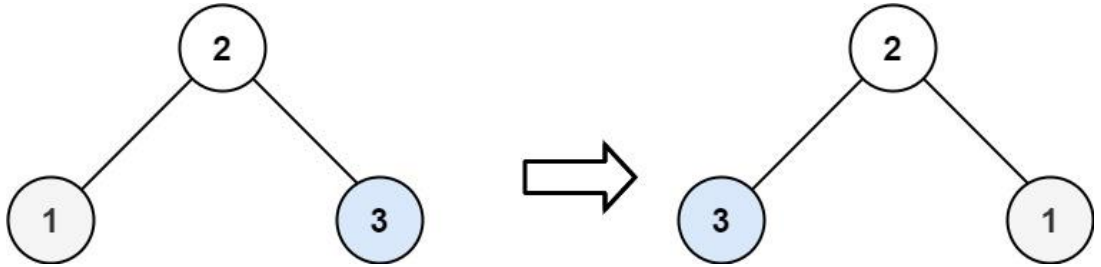
102. 二叉树的层序遍历（课下完成）

```ts
function levelOrder(root: TreeNode | null): number[][] {
  if (!root) {
    return [];
  }
  let result: number[][] = [];
  let queue: TreeNode[] = [root];
  while (queue.length) {
    let level: number[] = [];
    let levelSize = queue.length;
    for (let i = 0; i < levelSize; i++) {
      let node = queue.shift()!;
      level.push(node.val);
      if (node.left) {
        queue.push(node.left);
      }
      if (node.right) {
        queue.push(node.right);
      }
    }
    result.push(level);
  }
  return result;
}
```

226. 翻转二叉树（亚马逊、微软、腾讯登）

- 226. 翻转二叉树（Max Howell去Google面试没有写出来的题目）
  - https://leetcode.cn/problems/invert-binary-tree/description/





相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

亚马逊 8　微软 Microsoft 2

相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

Facebook 7　谷歌 Google 5　苹果 Apple 3　优步 Uber 3

字节跳动 2　彭博 Bloomberg 2

相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

高盛集团 Goldman Sachs 3　VMware 2　领英 LinkedIn 2

甲骨文 Oracle 2　百度 1　腾讯 1　滴滴 1　eBay 1　小米集团 1

陌陌 1

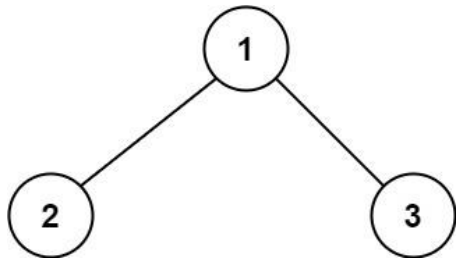
226. 翻转二叉树

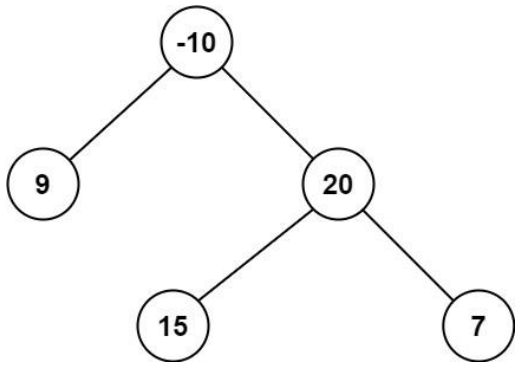
```ts
function invertTree(root: TreeNode | null): TreeNode | null {
  if (root == null) {
    return null;
  }

  const left = root.left;
  root.left = invertTree(root.right);
  root.right = invertTree(left);
  return root;
}
```

```ts
function invertTree(root: TreeNode | null): TreeNode | null {
  if (!root) {
    return null;
  }

  const stack = [root];

  while (stack.length) {
    const node = stack.pop();
    const temp = node!.left;
    node!.left = node!.right;
    node!.right = temp;
    if (node!.left) {
      stack.push(node!.left);
    }
    if (node!.right) {
      stack.push(node!.right);
    }
  }

  return root;
}
```

124. 二叉树中的最大路径和（亚马逊、微软、腾讯等）

- 124. 二叉树中的最大路径和
  - https://leetcode.cn/problems/binary-tree-maximum-path-sum/description/
- 题目：
  - **路径** 被定义为一条从树中任意节点出发，沿父节点-子节点连接，达到任意节点的序列。同一个节点在一条路径序列中 **至多出现一次** 。该路径 **至少包含一个** 节点，且不一定经过根节点。
  - **路径和** 是路径中各节点值的总和。
  - 给你一个二叉树的根节点 root ， 返回其 **最大路径和** 。





相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

亚马逊 3　微软 Microsoft 2

相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

字节跳动 17　Facebook 10　谷歌 Google 7　苹果 Apple 3

彭博 Bloomberg 2　三星 2　美团 2　华为 2　甲骨文 Oracle 2

Shopee 2

相关企业

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

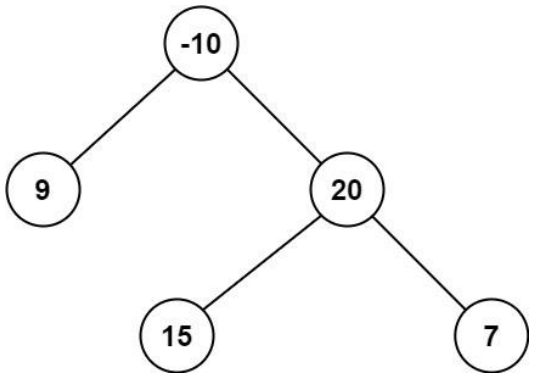
腾讯 5　百度 4　优步 Uber 3　滴滴 2　推特 Twitter 2

PayPal 2　领英 LinkedIn 2　小米集团 2　摩根士丹利 Morgan Stanley 2

京东 1

124. 二叉树中的最大路径和

- 这道题目可以使用深度优先搜索（DFS）来解决。
  - 我们可以从根节点开始递归，遍历二叉树中的所有节点。
  - 对于每个节点，我们需要计算经过该节点的最大路径和。
    - ✓ 在计算经过该节点的最大路径和时，我们需要考虑到左子树和右子树是否能够贡献最大路径和。
    - ✓ 如果左子树的最大路径和大于0，那么我们就将其加入到经过该节点的最大路径和中；
    - ✓ 如果右子树的最大路径和大于0，那么我们就将其加入到经过该节点的最大路径和中。
  - 最后，我们将经过该节点的最大路径和与已经计算出的最大路径和进行比较，取两者中的较大值。




```
function maxPathSum(root: TreeNode | null): number {
  let maxSum = -Infinity;

  function dfs(node: TreeNode | null): number {
    if (!node) return 0;

    // 左边可以提供的最大值
    const leftSum = Math.max(dfs(node.left), 0);
    // 右边可以提供的最大值
    const rightSum = Math.max(dfs(node.right), 0);

    // 加上自己(val)之后获取到的最大值
    const pathSum = node.val + leftSum + rightSum;
    // 是否是目前找到的最大值
    maxSum = Math.max(maxSum, pathSum);

    // 将自己+左/右的最大值返回出去
    return node.val + Math.max(leftSum, rightSum);
  }

  dfs(root);

  return maxSum;
}
```

62. 不同路径（亚马逊、谷歌、字节、腾讯等）

- 62. 不同路径
  - https://leetcode.cn/problems/unique-paths/description/
- 一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为 “Start”）。
- 机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为 “Finish”）。
- 问总共有多少条不同的路径?

输入： m = 3, n = 7
输出： 28

示例 2：

输入： m = 3, n = 2
输出： 3
解释：
从左上角开始，总共有 3 条路径可以到达右下角。
1. 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右
3. 向下 -> 向右 -> 向下

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

亚马逊 9　谷歌 Google 4　字节跳动 3　彭博 Bloomberg 3

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

微软 Microsoft 10　苹果 Apple 9　Facebook 7　华为 2

0 - 6 个月　6 个月 - 1 年　1 年 - 2 年

甲骨文 Oracle 3　摩根士丹利 Morgan Stanley 2　高盛集团 Goldman Sachs 2

优步 Uber 2　Wish 2　快手 1　腾讯 1　eBay 1　Citadel 1

PayPal 1

动态规划的实现思路

- **这个题目和跳楼梯其实是一类题目。**
- **设 dp[i][j] 表示从起点到网格的 (i, j) 点的不同路径数。**
- **对于每个格子，由于机器人只能从上面或左边到达该格子，因此有以下两种情况：**
  - 从上面的格子到达该格子，即 dp[i][j] = dp[i-1][j]；
  - 从左边的格子到达该格子，即 dp[i][j] = dp[i][j-1]。
    - ✓ 因此，到达网格的 (i, j) 点的不同路径数就等于到达上面格子的路径数加上到达左边格子的路径数。
  - 动态转移方程为：dp[i][j] = dp[i-1][j] + dp[i][j-1]；
- **初始状态：**对于边界情况，起点的路径数为 1，即 dp[0][0] = 1。
- **计算最终状态：**dp[m-1][n-1]


```
function uniquePaths(m: number, n: number): number {
    let dp = Array.from({length: m}, () => Array(n).fill(0))
    for (let i = 0; i < m; i++) {
        dp[i][0] = 1
    }
    for (let j = 0; j < n; j++) {
        dp[0][j] = 1
    }
    for (let i = 1; i < m; i++) {
        for (let j = 1; j < n; j++) {
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
        }
    }
    return dp[m-1][n-1]
};
```

不同路径的组合数学（课下扩展）

- 我们可以使用组合数学的方法，通过计算总共需要向下和向右走的步数，从而计算不同的路径数目。
- 假设总共需要向下走 n 步，向右走 m 步，则路径的总长度为 n + m，其中需要选择 n 个位置向下走，因此路径的总数目为 C(n + m, n)。

```
function uniquePaths(m: number, n: number): number {
  // 总共需要走的步数
  const total = m + n - 2;
  // 向下走的步数
  const down = n - 1;
  // 不同路径的数目
  let ans = 1;
  // 使用组合数学的方法计算不同路径数目
  for (let i = 1; i <= down; i++) {
    ans *= total - down + i; // C(total-down+i, i)的分子
    ans /= i; // C(total-down+i, i)的分母
  }
  // 返回计算结果
  return ans;
}
```

剑指 Offer 47. 礼物的最大价值（字节、谷歌等）

- 剑指 Offer 47. 礼物的最大价值
  - https://leetcode.cn/problems/li-wu-de-zui-da-jie-zhi-lcof/description/
- 题目：
  - 在一个 m*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物?

示例 1:

输入:

```
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
```

输出: 12

解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物

剑指 Offer 47. 礼物的最大价值

1	3	1
1	5	1
4	2	1

1	4	5
2	9	10
6	11	12

```ts
function maxValue(grid: number[][]): number {
  // 1.获取m*n
  const m = grid.length
  const n = grid[0].length

  // 2.初始化dp保存每个格子的最大值
  const dp: number[][] = Array.from({ length: m }, () => {
    return Array(n).fill(0)
  })
  dp[0][0] = grid[0][0]

  // 3.设置初始化值
  for (let i = 1; i < m; i++) {
    dp[i][0] = dp[i-1][0] + grid[i][0]
  }
  for (let j = 1; j < n; j++) {
    dp[0][j] = dp[0][j-1] + grid[0][j]
  }

  // 4.遍历每个位置，求出最大值
  for (let i = 1; i < m; i++) {
    for (let j = 1; j < n; j++) {
      dp[i][j] = Math.max(dp[i-1][j], dp[i][j-1]) + grid[i][j]
    }
  }
  return dp[m-1][n-1]
}
```

300. 最长递增子序列（亚马逊、谷歌、字节、华为等）

- 300. 最长递增子序列（Longest Increasing Subsequence，简称LIS）
  - https://leetcode.cn/problems/longest-increasing-subsequence/description/
- 题目：
  - 给你一个整数数组 nums ，找到其中最长严格递增子序列的长度。
  - **子序列** 是由数组派生而来的序列，删除（或不删除）数组中的元素而不改变其余元素的顺序。例如，[3,6,2,7] 是数组 [0,3,1,6,2,2,7] 的子序列。

示例 1：

```
输入：nums = [10,9,2,5,3,7,101,18]
输出：4
解释：最长递增子序列是 [2,3,7,101]，因此长度为 4 。
```

示例 2：

```
输入：nums = [0,1,0,3,2,3]
输出：4
```

示例 3：

```
输入：nums = [7,7,7,7,7,7,7]
输出：1
```

300. 最长递增子序列 – 动态规划

- 这道题目可以使用动态规划来解决。
- 定义状态：设dp[i]表示以第i个元素结尾的最长上升子序列的长度。
- 状态转移方程：
  - 对于每个i，我们需要找到在[0, i-1]范围内比nums[i]小的元素，以这些元素结尾的最长上升子序列中最长的那个子序列的长度。
  - 然后将其加1即可得到以nums[i]结尾的最长上升子序列的长度。
  - 状态转移方程为：dp[i] = max(dp[j]) + 1，其中j < i且nums[j] < nums[i]。
- 初始状态：对于每个i，dp[i]的初始值为1，因为每个元素本身也可以作为一个长度为1的上升子序列。
- 最终计算结果：最长上升子序列的长度即为dp数组中的最大值。

```
function lengthOfLIS(nums: number[]): number {
  const n = nums.length
  const dp = new Array(n).fill(1)
  let max = dp[0]
  for (let i = 1; i < n; i++) {
    for (let j = 0; j < i; j++) {
      // 找到前面一个比i位置小的数值
      if (nums[j] < nums[i]) {
        dp[i] = Math.max(dp[i], dp[j] + 1)
      }
    }
    max = Math.max(max, dp[i])
  }

  return max
}
```

动态规划的计算过程

原数组

10	9	2	4	3	7	101	18

dp数组初始化值

1	1	1	1	1	1	1	1

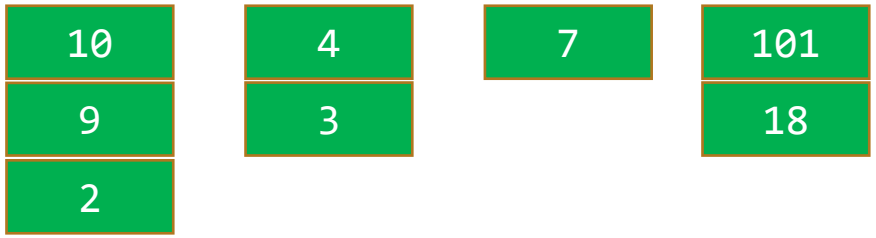
dp计算过程

1	1	1	dp(2)+1=2	dp(2)+1=2	dp(3)+1=3	dp(5)+1=4	dp(5)+1=4
0	1	2	3	4	5	6	7

动态规划算法的时间复杂度为O(n^2)，空间复杂度也为O(n)，其中n为原数组的长度。

贪心 + 二分查找的思考过程

原数组



- 维护一个数组tails，用于记录扫描到的元素应该存放的位置。
- 扫描原数组中的每个元素num，在tails数组中找是否有比自己更大的值。
  - 如果有，那么找到对应位置，并且让num作为该位置的最小值；
  - 如果没有，那么直接放到tails数组的尾部；
- tails数组的长度，就是最长递增子序列的长度；

- **为什么这么神奇刚好是数组的长度呢？（了解）**
- 情况一：如果是逆序的时候，一定会一直在一个上面加加加
- 情况二：一旦出现了比前面的最小值的值大的，那么就一定会增加一个新的数列，说明在上升的过程
- 情况三：如果之后出现一个比前面数列小的，那么就需要重新计算序列

贪心算法+二分查找的时间复杂度为O(nlogn)，空间复杂度为O(n)

贪心 + 二分查找的代码实现

```ts
function lengthOfLIS(nums: number[]): number {
  const tails: number[] = []

  for (const num of nums) {
    let left = 0
    let right = tails.length - 1

    while (left <= right) {
      const mid = Math.floor((left + right) / 2)
      if (num <= tails[mid]) { // 向左边找
        right = mid -1
      } else { // 向右边找
        left = mid + 1
      }
    }
    if (left === tails.length) {
      tails.push(num)
    } else {
      tails[left] = num
    }
  }

  return tails.length
}
```