0032. 最长有效括号

ITCharge 大约 5 分钟

- 标签：栈、字符串、动态规划
- 难度：困难

题目链接

- [0032. 最长有效括号 - 力扣]

题目大意

描述：给定一个只包含 `'('` 和 `')'` 的字符串。

要求：找出最长有效（格式正确且连续）括号子串的长度。

说明：

- $0 \le s.length \le 3 * 10^4$。
- `s[i]` 为 `'('` 或 `')'` 。

示例：

- 示例 1：

```py
输入：s = "(()"
输出：2
解释：最长有效括号子串是 "()"
```

- 示例 2：

```py
输入：s = ")()())"
输出：4
解释：最长有效括号子串是 "()()"
```

解题思路

思路 1：动态规划

1. 划分阶段

按照最长有效括号子串的结束位置进行阶段划分。

2. 定义状态

定义状态 `dp[i]` 表示为：以字符 `s[i]` 为结尾的最长有效括号的长度。

3. 状态转移方程

- 如果 `s[i] == '('`，此时以 `s[i]` 结尾的子串不可能构成有效括号对，则 `dp[i] = 0`。
- 如果 `s[i] == ')'`，我们需要考虑 `s[i - 1]` 来判断是否能够构成有效括号对。
  - 如果 `s[i - 1] == '('`，字符串形如 `......()`，此时 `s[i - 1]` 与 `s[i]` 为 `()`，则：
    - `dp[i]` 取决于「以字符 `s[i - 2]` 为结尾的最长有效括号长度」+「`s[i - 1]` 与 `s[i]` 构成的有效括号对长度（`2`）」，即 `dp[i] = dp[i - 2] + 2`。
    - 特别地，如果 `s[i - 2]` 不存在，即 `i - 2 < 0`，则 `dp[i]` 直接取决于「`s[i - 1]` 与 `s[i]` 构成的有效括号对长度（`2`）」，即 `dp[i] = 2`。
  - 如果 `s[i - 1] == ')'`，字符串形如 `......))`，此时 `s[i - 1]` 与 `s[i]` 为 `))`。那么以 `s[i - 1]` 为结尾的最长有效长度为 `dp[i - 1]`，则我们需要看 `i - 1 - dp[i - 1]` 位置上的字符 `s[i - 1 - dp[i - 1]]`是否与 `s[i]` 匹配。
    - 如果 `s[i - 1 - dp[i - 1]] == '('`，则说明 `s[i - 1 - dp[i - 1]]`与 `s[i]` 相匹配，此时我们需要看以 `s[i - 1 - dp[i - 1]]` 的前一个字符 `s[i - 1 - dp[i - 2]]` 为结尾的最长括号长度是多少，将其加上 ``s[i - 1 - dp[i - 1]] 与 s[i]`，从而构成更长的有效括号对：
      - `dp[i]` 取决于「以字符 `s[i - 1]` 为结尾的最长括号长度」+「以字符 `s[i - 1 - dp[i - 2]]` 为结尾的最长括号长度」+「`s[i - 1 - dp[i - 1]]` 与 `s[i]` 的长度（`2`）」，即 `dp[i] = dp[i - 1] + dp[i - dp[i - 1] - 2] + 2`。
      - 特别地，如果 `s[i - dp[i - 1] - 2]` 不存在，即 `i - dp[i - 1] - 2 < 0`，则 `dp[i]` 直接取决于「以字符 `s[i - 1]` 为结尾的最长括号长度」+「`s[i - 1 - dp[i - 1]]` 与 `s[i]` 的长度（`2`）」，即 `dp[i] = dp[i - 1] + 2`。

4. 初始条件

- 默认所有以字符 `s[i]` 为结尾的最长有效括号的长度为 `0`，即 `dp[i] = 0`。

5. 最终结果

根据我们之前定义的状态，`dp[i]` 表示为：以字符 `s[i]` 为结尾的最长有效括号的长度。则最终结果为 `max(dp[i])`。

思路 1：代码

```py
class Solution:
    def longestValidParentheses(self, s: str) -> int:
        dp = [0 for _ in range(len(s))]
        ans = 0
        for i in range(1, len(s)):
            if s[i] == '(':
                continue
            if s[i - 1] == '(':
                if i >= 2:
                    dp[i] = dp[i - 2] + 2
                else:
                    dp[i] = 2
            elif i - dp[i - 1] > 0 and s[i - dp[i - 1] - 1] == '(':
                if i - dp[i - 1] >= 2:
                    dp[i] = dp[i - 1] + dp[i - dp[i - 1] - 2] + 2
                else:
                    dp[i] = dp[i - 1] + 2
            ans = max(ans, dp[i])

        return ans
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为字符串长度。
- **空间复杂度**：$O(n)$。

思路 2：栈

1. 定义一个变量 `ans` 用于维护最长有效括号的长度，初始时， `ans = 0` 。
2. 定义一个栈用于判定括号对是否匹配（栈中存储的是括号的下标），栈底元素始终保持「最长有效括号子串的开始元素的前一个元素下标」。
3. 初始时，我们在栈中存储 `-1` 作为哨兵节点，表示「最长有效括号子串的开始元素的前一个元素下标为 `-1` 」，即 `stack = [-1]` ，
4. 然后从左至右遍历字符串。
    1. 如果遇到左括号，即 `s[i] == '('` ，则将其下标 `i` 压入栈，用于后续匹配右括号。
    2. 如果遇到右括号，即 `s[i] == ')'` ，则将其与最近的左括号进行匹配（即栈顶元素），弹出栈顶元素，与当前右括号进行匹配。弹出之后：
        1. 如果栈为空，则说明：
            1. 之前弹出的栈顶元素实际上是「最长有效括号子串的开始元素的前一个元素下标」，而不是左括号 `(` ，此时无法完成合法匹配。
            2. 将当前右括号的坐标 `i` 压入栈中，充当「下一个有效括号子串的开始元素前一个下标」。
        2. 如果栈不为空，则说明：
            1. 之前弹出的栈顶元素为左括号 `(` ，此时可完成合法匹配。
            2. 当前合法匹配的长度为「 右括号的下标 `i` 」 - 「最长有效括号子串的开始元素的前一个元素下标」。即 `i - stack[-1]` 。
            3. 更新最长匹配长度 `ans` 为 `max(ans, i - stack[-1])` 。
5. 遍历完输出答案 `ans` 。

思路 2：代码

```py
class Solution:
    def longestValidParentheses(self, s: str) -> int:
        stack = [-1]
        ans = 0
        for i in range(len(s)):
            if s[i] == '(':
                stack.append(i)
            else:
                stack.pop()
```

```python
                if stack:
                    ans = max(ans, i - stack[-1])
                else:
                    stack.append(i)
        return ans
```

思路 2：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为字符串长度。
- **空间复杂度**：$O(n)$。

参考资料

- 【题解】动态规划思路详解（C++）——32.最长有效括号
- 【题解】32. 最长有效括号 - 力扣（Leetcode）
- 【题解】【Nick~Hot一百题系列】超简单思路栈！