https://leetcode-cn.com/problems/valid-permutations-for-di-sequence/solution/di-xu-lie-de-you-xiao-pai-lie-by-leetcode/
https://leetcode.com/problems/valid-permutations-for-di-sequence/discuss/168278/C%2B%2BJavaPython-DP-Solution-O(N2)

方法一：动态规划
当我们已经确定了排列中的前 i 个元素 P[0], P[1], ..., P[i - 1] 时，我们需要通过字符串 S 的第 i - 1 位 S[i - 1] 和 P[i - 1] 共同确定下一个元素 P[i]。这说明，P[i - 1] 之前的元素 P[0], P[1], .., P[i - 2] 都是无意义的，有意义的是 P[i - 1] 和剩下未选出的 n + 1 - i 个元素的相对大小。例如当 n 的值为 5 时，我们已经确定的排列为 2, 3, 4，未选择的元素为 0, 1, 5，那么有意义的状态是排列 ?, ?, 2 以及未选择的元素 0, 1, 3，其中 ? 表示我们不关心的元素，0, 1, 2, 3 表示 P[i - 1] 和未选择元素的相对大小。

这样我们就可以用动态规划解决这道题目。我们用 dp(i, j) 表示确定了排列中到 P[i] 为止的前 i + 1 个元素，并且 P[i] 和未选择元素的相对大小为 j 的方案数（即未选择的元素中，有 j 个元素比 P[i] 小）。在状态转移时，dp(i, j) 会从 dp(i - 1, k) 转移而来，其中 k 代表了 P[i - 1] 的相对大小。如果 S[i - 1] 为 D，那么 k 不比 j 小；如果 S[i - 1] 为 I，那么 k 必须比 j 小。

JavaPython
```
class Solution {
    public int numPermsDISequence(String S) {
        int MOD = 1_000_000_007;
        int N = S.length();

        // dp[i][j] : Num ways to place P_i with relative rank j
        int[][] dp = new int[N+1][N+1];
        Arrays.fill(dp[0], 1);

        for (int i = 1; i <= N; ++i) {
            for (int j = 0; j <= i; ++j) {
                if (S.charAt(i-1) == 'D') {
                    for (int k = j; k < i; ++k) {
                        dp[i][j] += dp[i-1][k];
                        dp[i][j] %= MOD;
                    }
                } else {
                    for (int k = 0; k < j; ++k) {
                        dp[i][j] += dp[i-1][k];
                        dp[i][j] %= MOD;
                    }
                }
            }
        }
```

```
        int ans = 0;
        for (int x: dp[N]) {
            ans += x;
            ans %= MOD;
        }

        return ans;
    }
}
```

动态规划优化

我们发现，在上面动态规划的状态转移中，当 S[i - 1] 为 I 时，dp(i, j) 比 dp(i, j - 1) 多出了 dp(i - 1, j - 1) 这一项；当 S[i - 1] 为 D 时，dp(i, j) 比 dp(i, j + 1) 多出了 dp(i - 1, j) 这一项，因此可以不用 dp(i, j) 都计算一遍对应的 dp(i - 1, k) 的和，而是用

```
dp(i, j) = dp(i, j - 1) + dp(i - 1, j - 1)  if S[i - 1] == 'I'
dp(i, j) = dp(i, j + 1) + dp(i - 1, j)      if S[i - 1] == 'D'
```

代替之，减少时间复杂度。

Python

```
from functools import lru_cache

class Solution:
    def numPermsDISequence(self, S):
        MOD = 10**9 + 7
        N = len(S)

        @lru_cache(None)
        def dp(i, j):
            # How many ways to place P_i with relative rank j?
            if not(0 <= j <= i):
                return 0
            if i == 0:
                return 1
            elif S[i-1] == 'D':
                return (dp(i, j+1) + dp(i-1, j)) % MOD
            else:
                return (dp(i, j-1) + dp(i-1, j-1)) % MOD

        return sum(dp(N, j) for j in range(N+1)) % MOD
```

复杂度分析

时间复杂度：O(N^3)O(N
3
)，如果使用动态规划优化，复杂度降为 O(N^2)O(N
2
)。

空间复杂度：O(N^2)O(N
2
)。

方法二：分治
我们同样可以使用分治算法（实际上是一种区间动态规划）解决这道题目。首先我们考虑将 0 放在哪里，可以发现 0 要么放在 DI 对应的位置，要么放在排列的开头且对应的字符为 I，要么放在排列的结尾且对应的字符为 D。将 0 放好后，排列被分成了 0 左侧和右侧两部分，每个部分各是一个对应长度的有效排列问题。

设左侧的长度为 x，排列的方案数为 f(x)，右侧的长度为 y，排列的方案数为 f(y)，在合并时，我们需要在 x + y 中选择 x 个数分给左侧，剩余的 y 个数分给右侧，因此合并后的方案数为 binom(x + y, x) * f(x) * f(y)，其中 binom 为组合数。

Python
```
from functools import lru_cache

class Solution:
    def numPermsDISequence(self, S):
        MOD = 10**9 + 7

        fac = [1, 1]
        for x in range(2, 201):
            fac.append(fac[-1] * x % MOD)
        facinv = [pow(f, MOD-2, MOD) for f in fac]

        def binom(n, k):
            return fac[n] * facinv[n-k] % MOD * facinv[k] % MOD

        @lru_cache(None)
        def dp(i, j):
            if i >= j: return 1
            ans = 0
            n = j - i + 2
            if S[i] == 'I': ans += dp(i+1, j)
            if S[j] == 'D': ans += dp(i, j-1)
```

```
        for k in range(i+1, j+1):
            if S[k-1:k+1] == 'DI':
                ans += binom(n-1, k-i) * dp(i, k-2) % MOD * dp(k+1, j) % MOD
                ans %= MOD
        return ans

    return dp(0, len(S) - 1)
```

复杂度分析

时间复杂度：O(N^3)O(N
3
)。

空间复杂度：O(N^2)O(N
2
)。

作者：LeetCode
链接：https://leetcode-cn.com/problems/valid-permutations-for-di-sequence/solution/di-xu-lie-de-you-xiao-pai-lie-by-leetcode/
来源：力扣（LeetCode）