0055. 跳跃游戏

ITCharge 大约 3 分钟

- 标签：贪心、数组、动态规划
- 难度：中等

题目链接

- 0055. 跳跃游戏 - 力扣

题目大意

描述：给定一个非负整数数组 `nums` ，数组中每个元素代表在该位置可以跳跃的最大长度。开始位置位于数组的第一个下标处。

要求：判断是否能够到达最后一个下标。

说明：

- $1 \leq nums.length \leq 3 \times 10^4$。
- $0 \leq nums[i] \leq 10^5$。

示例：

- 示例 1：

```py
输入：nums = [2,3,1,1,4]
输出：true
解释：可以先跳 1 步，从下标 0 到达下标 1，然后再从下标 1 跳 3 步到达最后一个下标。
```

- 示例 2：

```py
输入：nums = [3,2,1,0,4]
输出：false
解释：无论怎样，总会到达下标为 3 的位置。但该下标的最大跳跃长度是 0 ， 所以永远不可能到达最后一个下标。
```

解题思路

思路 1：贪心算法

如果我们能通过前面的某个位置 j，到达后面的某个位置 i，则我们一定能到达区间 $[j, i]$ 中所有的点（$j \leq i$）。

而前面的位置 j 肯定也是通过 j 前面的点到达的。所以我们可以通过贪心算法来计算出所能到达的最远位置。具体步骤如下：

1. 初始化能到达的最远位置 max_i 为 0。
2. 遍历数组 `nums` 。
3. 如果能到达当前位置，即 $max_i \leq i$，并且当前位置 + 当前位置最大跳跃长度 > 能到达的最远位置，即 $i + nums[i] > max_i$，则更新能到达的最远位置 max_i。
4. 遍历完数组，最后比较能到达的最远位置 max_i 和数组最远距离 `size - 1` 的关系。如果 $max_i >= len(nums)$，则返回 `True` ，否则返回 `False` 。

思路 1：代码

```py
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        size = len(nums)
        max_i = 0
        for i in range(size):
            if max_i >= i and i + nums[i] > max_i:
                max_i = i + nums[i]

        return max_i >= size - 1
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 是数组 `nums` 的长度。
- **空间复杂度**：

思路 2：动态规划

1. 划分阶段

按照位置进行阶段划分。

2. 定义状态

定义状态 `dp[i]` 表示为：从位置 0 出发，经过 $j \leq i$，可以跳出的最远距离。

3. 状态转移方程

- 如果能通过 $0 \sim i - 1$ 个位置到达 i，即 $dp[i - 1] \leq i$，则 $dp[i] = max(dp[i - 1], i + nums[i])$。
- 如果不能通过 $0 \sim i - 1$ 个位置到达 i，即 $dp[i - 1] < i$，则 $dp[i] = dp[i - 1]$。

4. 初始条件

初始状态下，从 0 出发，经过 0，可以跳出的最远距离为 `nums[0]`，即 `dp[0] = nums[0]`。

5. 最终结果

根据我们之前定义的状态，`dp[i]` 表示为：从位置 0 出发，经过 $j \leq i$，可以跳出的最远距离。则我们需要判断 `dp[size - 1]` 与数组最远距离 `size - 1` 的关系。

思路 2：代码

py

```
class Solution:
    def canJump(self, nums: List[int]) -> bool:
        size = len(nums)
        dp = [0 for _ in range(size)]
        dp[0] = nums[0]
        for i in range(1, size):
```

```
            if i <= dp[i - 1]:
                dp[i] = max(dp[i - 1], i + nums[i])
            else:
                dp[i] = dp[i - 1]
        return dp[size - 1] >= size - 1
```

思路 2：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 是数组 `nums` 的长度。
- **空间复杂度**：$O(n)$。