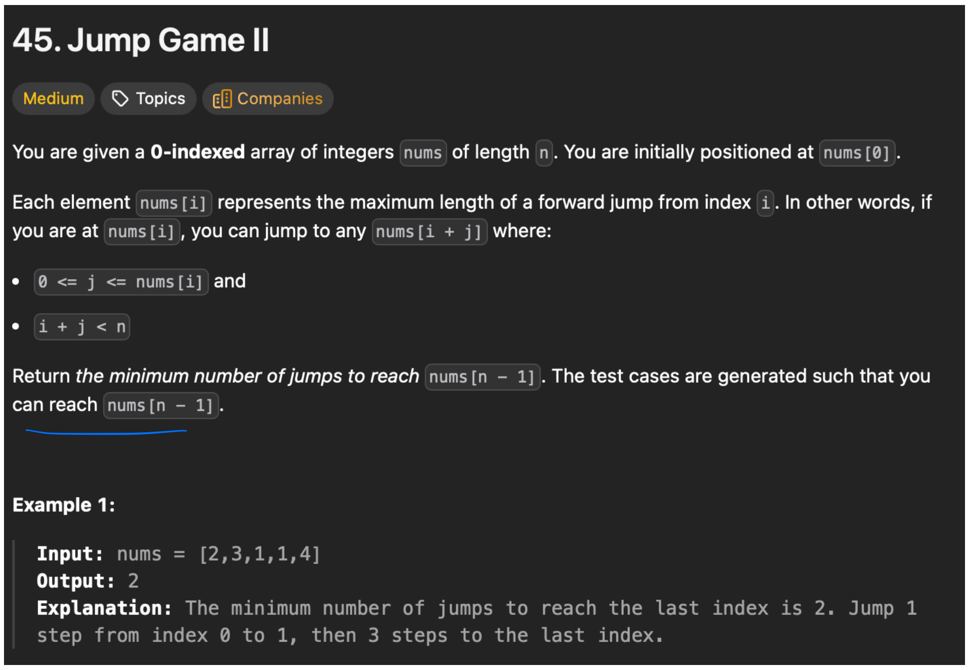


45. Jump Game II

Medium · Topics · Companies

You are given a **0-indexed** array of integers `nums` of length `n`. You are initially positioned at `nums[0]`.

Each element `nums[i]` represents the maximum length of a forward jump from index `i`. In other words, if you are at `nums[i]`, you can jump to any `nums[i + j]` where:

- `0 <= j <= nums[i]` and
- `i + j < n`

Return *the minimum number of jumps to reach* `nums[n - 1]`. The test cases are generated such that you can reach `nums[n - 1]`.

Example 1:

```
Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1
step from index 0 to 1, then 3 steps to the last index.
```

greedy / deque ?

i = 1 . nums[1] = 3 { i = 2 X ; 3 n[3] = 1 - i = 4 ; 4 ✓

→ nums[0] = 2 { i = 2 nums[2] = 1 — i = 3

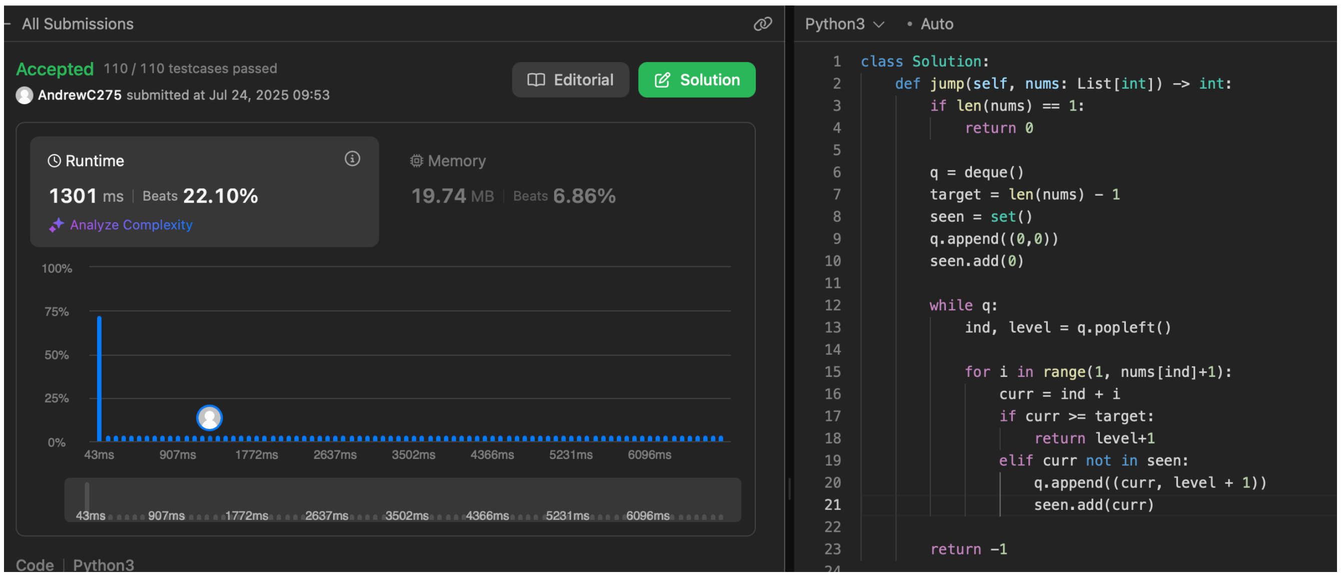
q = [0] l = 0 , reach = False

q = [1, 2] l = 1 , reach = False

q = [~~2~~, 3, 4, ~~3~~] , l = 2, reach = True

seen dup

return



All Submissions

Accepted 110 / 110 testcases passed

AndrewC275 submitted at Jul 24, 2025 09:53

Editorial | Solution

Runtime: 1301 ms | Beats 22.10%

Analyze Complexity

Memory: 19.74 MB | Beats 6.86%

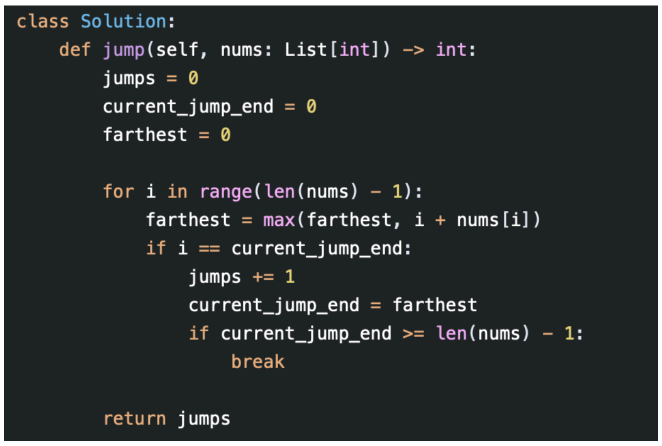
Code | Python3

Python3 • Auto

```python
class Solution:
    def jump(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return 0

        q = deque()
        target = len(nums) - 1
        seen = set()
        q.append((0,0))
        seen.add(0)

        while q:
            ind, level = q.popleft()

            for i in range(1, nums[ind]+1):
                curr = ind + i
                if curr >= target:
                    return level+1
                elif curr not in seen:
                    q.append((curr, level + 1))
                    seen.add(curr)

        return -1
```

this is actually BFS

Greedy:



```python
class Solution:
    def jump(self, nums: List[int]) -> int:
        jumps = 0
        current_jump_end = 0
        farthest = 0

        for i in range(len(nums) - 1):
            farthest = max(farthest, i + nums[i])
            if i == current_jump_end:
                jumps += 1
                current_jump_end = farthest
                if current_jump_end >= len(nums) - 1:
                    break

        return jumps
```