Description | Editorial | Solutions (1.8K) | Submissions

403. Frog Jump

Hard 4.9K 229

Companies

A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.

Given a list of `stones`' positions (in units) in sorted **ascending order**, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be `1` unit.

If the frog's last jump was `k` units, its next jump must be either `k - 1`, `k`, or `k + 1` units. The frog can only jump in the forward direction.

Example 1:

```
Input: stones = [0,1,3,5,6,8,12,17]
Output: true
Explanation: The frog can jump to the last stone by jumping 1 unit to the 2nd stone, then 2 units to the 3rd stone, then 2 units to the 4th stone, then 3 units to the 6th stone, 4 units to the 7th stone, and 5 units to the 8th stone.
```

Example 2:

```
Input: stones = [0,1,2,3,4,8,9,11]
Output: false
Explanation: There is no way to jump to the last stone as the gap between the 5th and 6th stone is too large.
```

Java | Auto

```java
class Solution {
    HashMap<Integer, Integer> m = new HashMap<>();
    int[][] dp;

    boolean solve(int i, int k, int[] stones) {
        if (i == stones.length - 1) {
            return true;
        }

        if (dp[i][k] != -1) {
            return dp[i][k] == 1;
        }

        boolean k0 = false, kp = false, k1 = false;

        if (m.containsKey(stones[i] + k)) {
            k0 = solve(m.get(stones[i] + k), k, stones);
```

Ln 1, Col 1

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

stones =
[0,1,3,5,6,8,12,17]

Output

true

Console | Run | Submit

Description | Editorial | Solutions (2.4K) | Submissions

Press Esc to exit full screen

334. Increasing Triplet Subsequence

Medium 7.2K 352

Companies

Given an integer array `nums`, return `true` *if there exists a triple of indices* `(i, j, k)` *such that* `i < j < k` *and* `nums[i] < nums[j] < nums[k]`. If no such indices exists, return `false`.

Example 1:

> **Input:** nums = [1,2,3,4,5]
> **Output:** true
> **Explanation:** Any triplet where i < j < k is valid.

Example 2:

> **Input:** nums = [5,4,3,2,1]
> **Output:** false
> **Explanation:** No triplet exists.

Example 3:

> **Input:** nums = [2,1,5,0,4,6]
> **Output:** true
> **Explanation:** The triplet (3, 4, 5) is valid because nums[3] == 0 < nums[4] == 4 < nums[5] == 6.

Constraints:

- $1 <= nums.length <= 5 * 10^5$
- $-2^{31} <= nums[i] <= 2^{31} - 1$

Java | Auto

```java
    public boolean increasingTriplet(int[] nums) {
        int max1 = Integer.MAX_VALUE;
        int max2 = Integer.MAX_VALUE;
        for(int n : nums) {
            if(n <= max1) max1 = n;
            else if(n <= max2) max2 = n;
            else return true;
        }
        return false;
    }
}
```

Ln 12, Col 2

Testcase | Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[1,2,3,4,5]

Output

true

Console | Run | Submit