403. Frog Jump

QuestionEditorial Solution
[My Submissions]

- Total Accepted: **715**
- Total Submissions: **1819**
- Difficulty: **Hard**

A frog is crossing a river. The river is divided into x 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 is able to cross the river by landing on the last stone. Initially, the frog is on the first stone and assume the first jump must be 1 unit.

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

Note:

- The number of stones is ≥ 2 and is $< 1{,}100$.
- Each stone's position will be a non-negative integer $< 2^{31}$.
- The first stone's position is always 0.

Example 1:

```
[0,1,3,5,6,8,12,17]

There are a total of 8 stones.
The first stone at the 0th unit, second stone at the 1st unit,
third stone at the 3rd unit, and so on...
The last stone at the 17th unit.

Return true. 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:

```
[0,1,2,3,4,8,9,11]

Return false.   There is no way to jump to the last stone as
the gap between the 5th and 6th stone is too large.
```

DFS

```
class Solution {
```

```
public:
    //zzw
    bool canCross(vector<int>& stones) {
        int res=0;
        helper(stones,0,1,res);
        return res==stones.size()-1;
    }

    void helper(vector<int>& stones, int begin, int jump, int &res)
    {
        int n = stones.size();
        if(begin>=n-1 || res>=n-1) return;
        for(int i=begin+1;i<n;i++)
        {
            int dist = stones[i]-stones[begin];
            //at the start stones[0] must be 1
            if(dist!=1 && i==1) return;
            if(jump==dist || jump==dist-1 || jump==dist+1)
            {
                res = max(res,i);
                helper(stones,i,dist,res);
            }
        }
    }
};
```

Dynamic Programming

```
//author zzw
class Solution {
public:
    bool canCross(vector<int>& stones) {
        int n=stones.size();
        vector<int>solutions(n,-1);
        solutions[0]=0;
        for(int i=1;i<n;i++)
        {
            for(int j=i-1;j>=0;j--)
            {
                int needjump = stones[i]-stones[j];
                if(abs(needjump-solutions[j])<=1 && solutions[j]!=-1)
                solutions[i]=needjump;
            }
        }
        return solutions[n-1]>-1;
    }
};
```