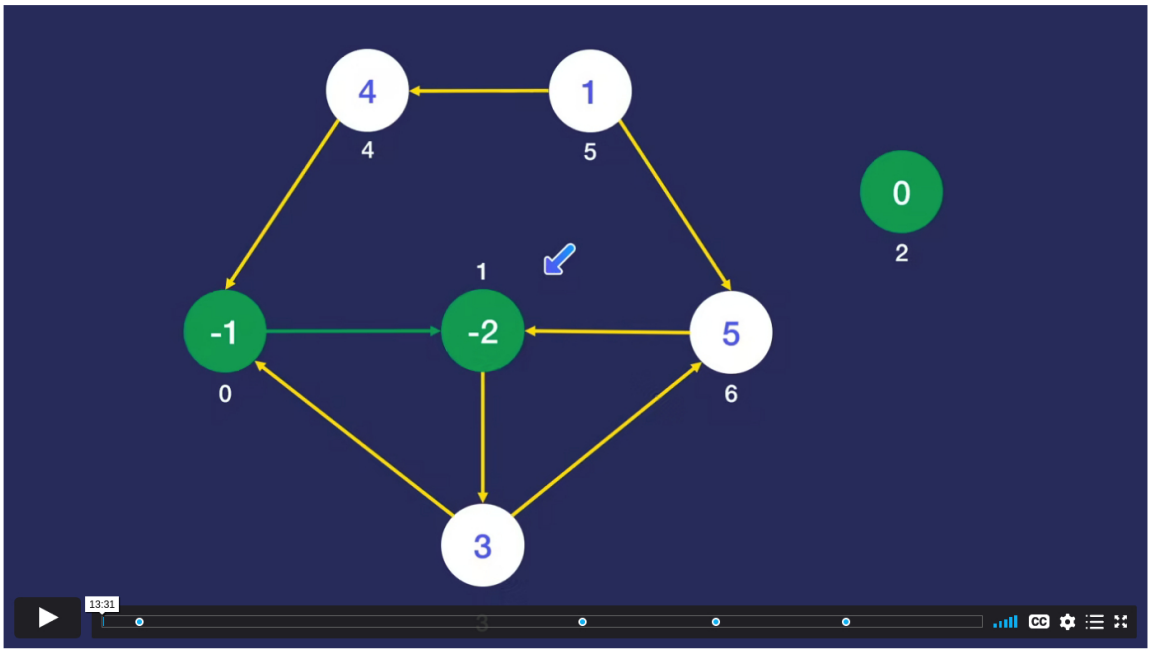
Average Rating: 4.45 (40 votes) Premium

Video Solution



Solution Article

You probably can guess from the problem title, this is the third problem in the series of Jump Game problems. Those problems are similar, but have considerable differences, making their solutions quite different.

Here, two approaches are introduced: *Breadth-First Search* approach and *Depth-First Search* approach.

Approach 1: Breadth-First Search

Most solutions start from a brute force approach and are optimized by removing unnecessary calculations. Same as this one.

A naive brute force approach is to iterate all the possible routes and check if there is one reaches `zero`. However, if we already checked one index, we do not need to check it again. We can mark the index as visited by make it negative.

C++ Java Python3 Copy

```
        queue<int> q;
        q.push(start);

        while (!q.empty()) {
            int node = q.front();
            q.pop();

            // check if we reached zero
            if (arr[node] == 0) {
                return true;
            }
            if (arr[node] > 0) {
                // check available next steps
                if (node + arr[node] < n) {
                    q.push(node + arr[node]);
                }
                if (node - arr[node] >= 0) {
                    q.push(node - arr[node]);
                }
                // mark as visited
                arr[node] = -arr[node];
            }
        }
        return false;
    }
};
```

Complexity Analysis

Assume N is the length of `arr`.

- Time complexity: $\mathcal{O}(N)$ since we will visit every index at most once.
- Space complexity : $\mathcal{O}(N)$ since it needs `q` to store next index. In fact, `q` would keep at most two levels of nodes. Since we got two children for each node, the traversal of this solution is a binary tree. The maximum number of nodes within a single level for a binary tree would be $\frac{N}{2}$, so the maximum length of `q` is $\mathcal{O}(\frac{N}{2} + \frac{N}{2}) = \mathcal{O}(N)$.

Testcase Run Code Result Debugger

Accepted Runtime: 4 ms

Your input [4,2,3,0,3,1,2] 5

Output true Diff

Expected true

Console Use Example Testcases

Approach 2: Depth-First Search

DFS is similar to BFS but differs in the order of searching. You should consider DFS when you think of BFS.

Still, we make the value negative to mark as visited.

C++ | Java | Python3 — Copy

```cpp
class Solution {
public:
    bool canReach(vector<int>& arr, int start) {
        if (start >= 0 and start < arr.size() and arr[start] >= 0) {
            if (arr[start] == 0) {
                return true;
            }
            arr[start] = -arr[start];
            return canReach(arr, start + arr[start]) or canReach(arr, start - arr[start]);
        }
        return false;
    }
};
```

Complexity Analysis

Assume N is the length of `arr`.

- Time complexity: $\mathcal{O}(N)$, since we will visit every index only once.
- Space complexity: $\mathcal{O}(N)$ since it needs at most $\mathcal{O}(N)$ stacks for recursions.

Report Article Issue

Comments: 18 — Best | Most Votes | Newest to Oldest | Oldest to Newest

Type comment here... (Markdown is supported)

Post

justinsingh2015 ★ 56 August 16, 2020 7:43 AM

I do not believe that the BFS solution should be described as brute force when it has the same runtime complexity as the DFS solution.

53 Show 2 replies Reply

dana-n ★ 246 Last Edit: November 29, 2020 1:56 AM

Stop modifying input parameters! A function should have no side effects on input. You can use something like a local boolean array to track whether an index has been visited.

42 Show 10 replies Reply

sriharik ★ 1105 May 5, 2021 10:01 PM

The DFS solution if you all are interested:

```java
public boolean canReach(int[] arr, int start) {
    return helper(arr, start, new HashSet<>());
}

private boolean helper(int[] arr, int idx, Set<Integer> set) {
    if (idx < 0 || idx >= arr.length) return false;
    if (arr[idx] == 0) return true;
    if (set.contains(idx)) return false;
    set.add(idx);
```

Read More

4 Reply

himani_01 ★ 129 December 4, 2020 10:51 AM

We are making the arr[i] negative to mark it as visited. But in DFS shouldn't we revert it back to its original value so that backtracking works fine. In dfs if one path doesn't work for us then we go back(backtrack) and traverse the next one. So don't we need the value of arr[i] intact so that we can do the backtrack and continue properly ?

For example in my code, I have used a Set of visited indices, so that i dont go in loop.
I am also removing the index from the set whenever backtracking.

```java
private boolean recur(int[] nums, int i, Set<Integer> s){
    if(i >= nums.length || i < 0)
        return false;
    if(nums[i] ==0)
```

Read More

1 Show 1 reply Reply

samy_vilar ★ 5 November 12, 2020 1:16 PM

Surprised the editorial didn't use a stack for its DFS approach, granted the recursive approach is quite elegant, though unlike its preceding BFS it would require unwinding the stack when reaching a feasible entry, unlike the former which simply returns true; it would probably be a decent exercise for those that want to better understand DFS, specially post-order traversals with a stack, though with that said the overall space usage would probably remain the same either way, may be slighter better or worse depending on micro-optimizations in either cases ...

1 Reply

rizwankhan131199 ★ 3 August 3, 2020 3:34 AM

Great Post !

1 Reply

syfasyfa ★ 1 August 13, 2020 7:56 AM

> You should consider DFS when you think of BFS.

Why? There's no difference in time and space complexity.

1 Show 5 replies Reply

bcb98801xx 156 November 22, 2020 9:26 PM

My BFS beats DFS in Golang, so why DFS is better?

0 Reply

dareyy 5 September 20, 2020 8:20 AM

why not use a deque in the bfs solution?

0 Show 3 replies Reply

rishabhgpt3 61 September 29, 2021 2:05 AM

lol what is meant by `You should consider DFS when you think of BFS`

0 Reply