

◀ Previous (/articles/queue-reconstruction-by-height/) Next ▶ (/articles/verify-preorder-serialization-of-a-binary-tree/)

1137. N-th Tribonacci Number [↗](#) (/problems/n-th-tribonacci-number/)

July 31, 2019 | 2.2K views

Average Rating: 5 (6 votes)

The Tribonacci sequence T_n is defined as follows:

$T_0 = 0$, $T_1 = 1$, $T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for $n \geq 0$.

Given n , return the value of T_n .

Example 1:

```
Input: n = 4
Output: 4
Explanation:
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4
```

Example 2:

```
Input: n = 25
Output: 1389537
```

Constraints:

- $0 \leq n \leq 37$
- The answer is guaranteed to fit within a 32-bit integer, ie. $\text{answer} \leq 2^{31} - 1$.

Solution

Possible Solutions: Space vs Performance Optimisation

There could be two approaches here. The first one is to optimise the performance, and the second one is to minimize the space used.

Let's start from the performance optimisation.

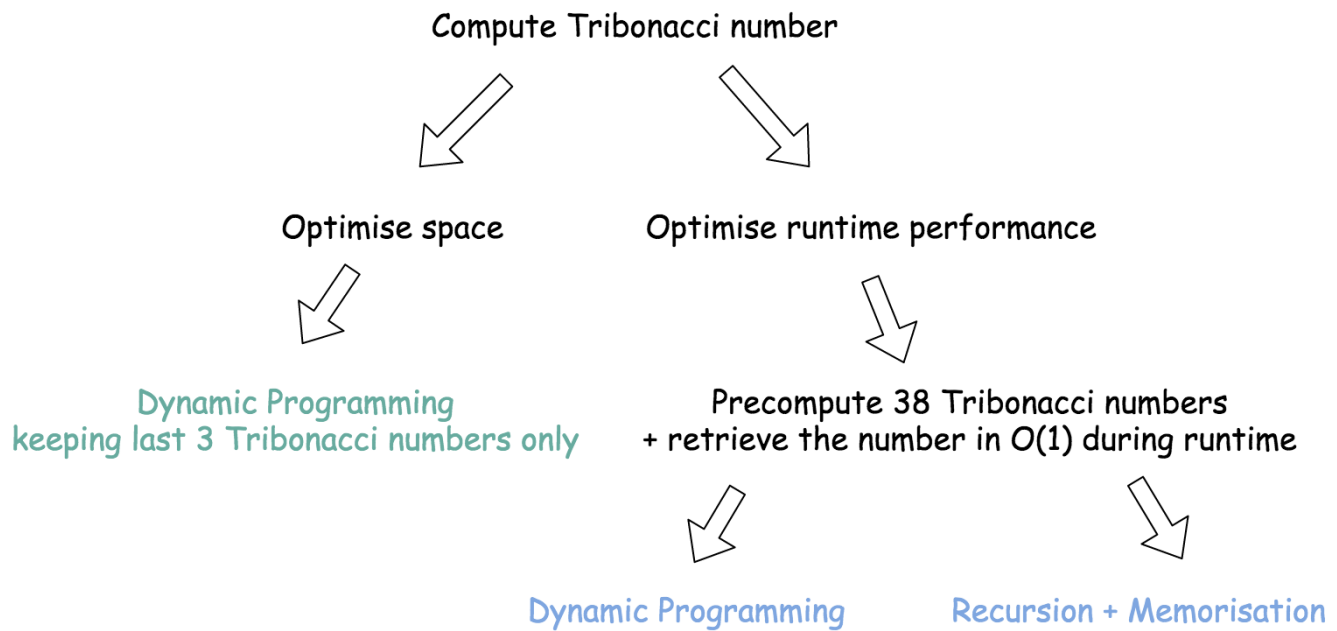
Since n is known to be less than 38, it's enough to precompute all 38 Tribonacci numbers once, store them in a static variable of the class `Solution`, and then just retrieve a needed number in a constant time during the testcase execution.

How to make the preliminary computations?

Two ideas could work here pretty well : recursion with memorisation and dynamic programming. They both need N operations to compute N Tribonacci numbers. Simple recursion like $\text{tribonacci}(k) = \text{tribonacci}(k - 1) + \text{tribonacci}(k - 2) + \text{tribonacci}(k - 3)$ is out of consideration here because it will result in exponential time complexity 3^N .

Approach with preliminary computations has perfect $O(1)$ runtime performance but needs in $O(N)$ space to keep N Tribonacci numbers. In some quite rare cases it's crucial to optimise the space used above the performance.

In such a situation no more space-consuming static variables are allowed and one could use dynamic programming approach keeping not more than 3 Tribonacci numbers in memory.




Approach 1: Space Optimisation : Dynamic Programming

- If $n < 3$ the answer is obvious.
- Otherwise initiate the first three numbers $x = 0$, $y = z = 1$ and proceed to the loop of $n - 2$ steps. At each step:
 - Replace x by y .
 - Replace y by z .
 - Replace z by $x + y + z$.
- Return z .

Implementation

Java Python

 Copy

```
1 class Solution {
2     public int tribonacci(int n) {
3         if (n < 3) return n == 0 ? 0 : 1;
4
5         int tmp, x = 0, y = 1, z = 1;
6         for (int i = 3; i <= n; ++i) {
7             tmp = x + y + z;
8             x = y;
9             y = z;
10            z = tmp;
11        }
12        return z;
13    }
14 }
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$.
- Space complexity : constant space to keep the last three Fibonacci numbers.


Approach 2: Performance Optimisation : Recursion with Memorisation

- Precompute 38 Tribonacci numbers:
 - Initiate array of precomputed Tribonacci numbers `nums` by zeros and initiate the first three numbers.
 - Return `helper(n - 1)`.
- Recursive function `helper(k)` :
 - If `k == 0`, return 0.
 - If `nums[k] != 0`, return `nums[k]`.
 - Otherwise, `nums[k] = helper(k - 1) + helper(k - 2) + helper(k - 3)`. Return `nums[k]`.
- Retrieve needed Tribonacci number from the array of precomputed numbers.

Implementation

Java

Python

 Copy

```
1 class Tri {
2     private int n = 38;
3     public int[] nums = new int[n];
4
5     int helper(int k) {
6         if (k == 0) return 0;
7         if (nums[k] != 0) return nums[k];
8
9         nums[k] = helper(k - 1) + helper(k - 2) + helper(k - 3);
10        return nums[k];
11    }
12
13    Tri() {
14        nums[1] = 1;
15        nums[2] = 1;
16        helper(n - 1);
17    }
18 }
19
20 class Solution {
21     public static Tri t = new Tri();
22     public int tribonacci(int n) {
23         return t.nums[n];
24     }
25 }
```

Complexity Analysis

- Time complexity : $\mathcal{O}(1)$ to retrieve preliminary computed Tribonacci number, and 38 operations for the preliminary computations.
- Space complexity : constant space to keep an array of 38 Tribonacci numbers.


Approach 3: Performance Optimisation : Dynamic Programming

- Precompute 38 Tribonacci numbers:
 - Initiate an array of precomputed Tribonacci numbers `nums` by zeros and initiate the first three numbers.
 - Perform the loop for `i` in a range from 3 to 38. Compute at each step the new Tribonacci number: `nums[i] = helper(i - 1) + helper(i - 2) + helper(i - 3)`.
- Retrieve needed Tribonacci number from the array of precomputed numbers.

Implementation

Java

Python

 Copy


```
1 class Tri {
2     private int n = 38;
3     public int[] nums = new int[n];
4     Tri() {
5         nums[1] = 1;
6         nums[2] = 1;
7         for (int i = 3; i < n; ++i)
8             nums[i] = nums[i - 1] + nums[i - 2] + nums[i - 3];
9     }
10 }
11
12 class Solution {
13     public static Tri t = new Tri();
14     public int tribonacci(int n) {
15         return t.nums[n];
16     }
17 }
```

Complexity Analysis

- Time complexity : $\mathcal{O}(1)$ to retrieve preliminary computed Tribonacci number, and 38 operations for the preliminary computations.
- Space complexity : constant space to keep an array of 38 Tribonacci numbers.

Analysis written by @liaison (<https://leetcode.com/liaison/>) and @andvary (<https://leetcode.com/andvary/>)

Rate this article:

 Previous (</articles/queue-reconstruction-by-height/>)

Next  (</articles/verify-preorder-serialization-of-a-binary-tree/>)


Comments: 2

Sort By ▼

Type comment here... (Markdown is supported)

 Preview

Post

msnitish (msnitish) ★4  August 7, 2019 1:55 AM

⋮

Wow! I was waiting for this kind of vivid explanation.

1 ^ v | Share | Reply

huxyluts (huxyluts) ★ 2 ⌚ August 5, 2019 11:23 PM



How does this last solution (Approach 3) become $O(1)$, I thought it's $O(1)$? @andvary
(<https://leetcode.com/andvary>)

1 ^ v | Share | Reply

SHOW 5 REPLIES

Copyright © 2019 LeetCode

[Help Center \(/support/\)](/support/) | [Terms \(/terms/\)](/terms/) | [Privacy Policy \(/privacy/\)](/privacy/)

 [United States \(/region/\)](/region/)