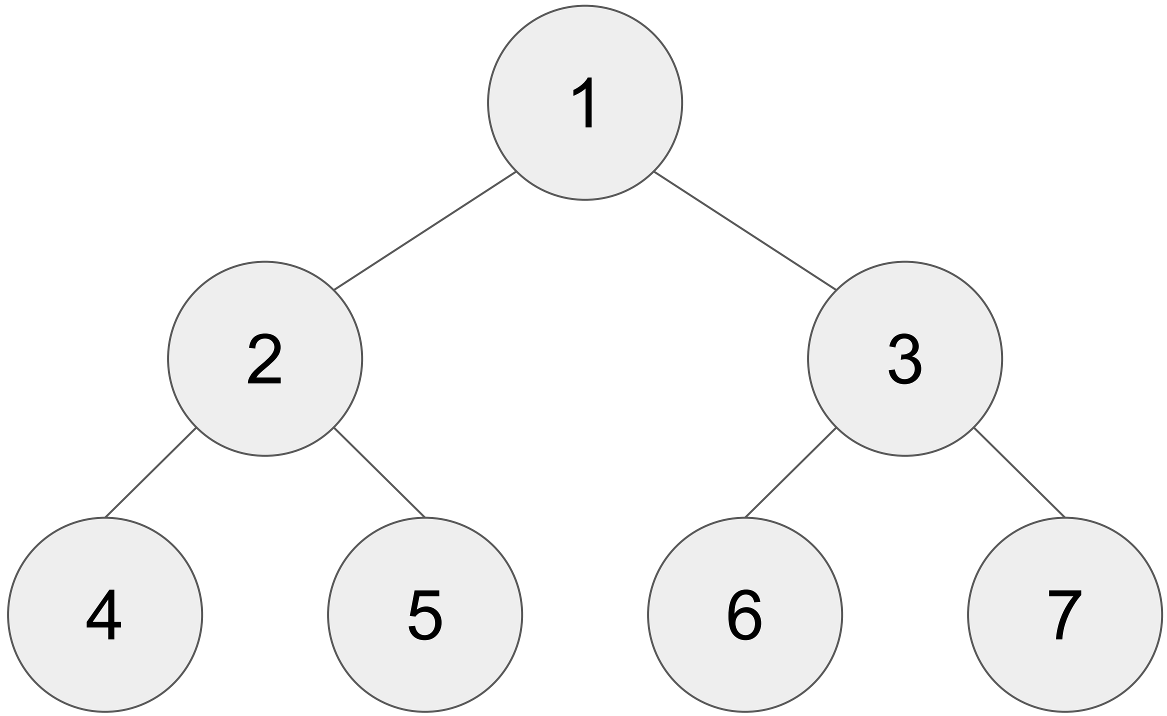
Problem C

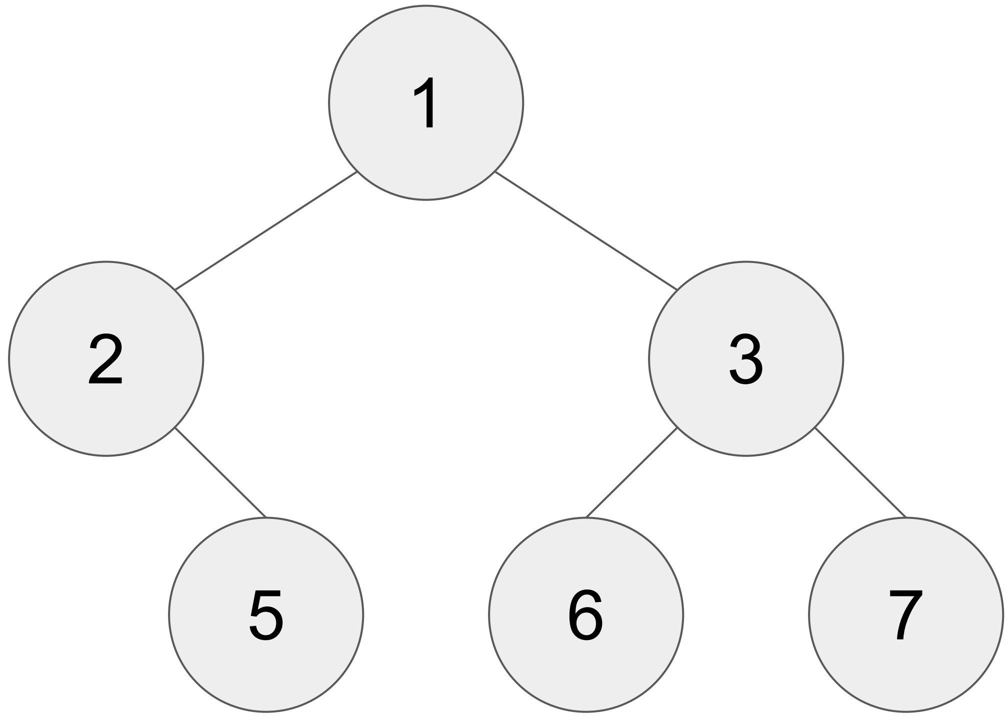
Cumulative Code

Submits: 2

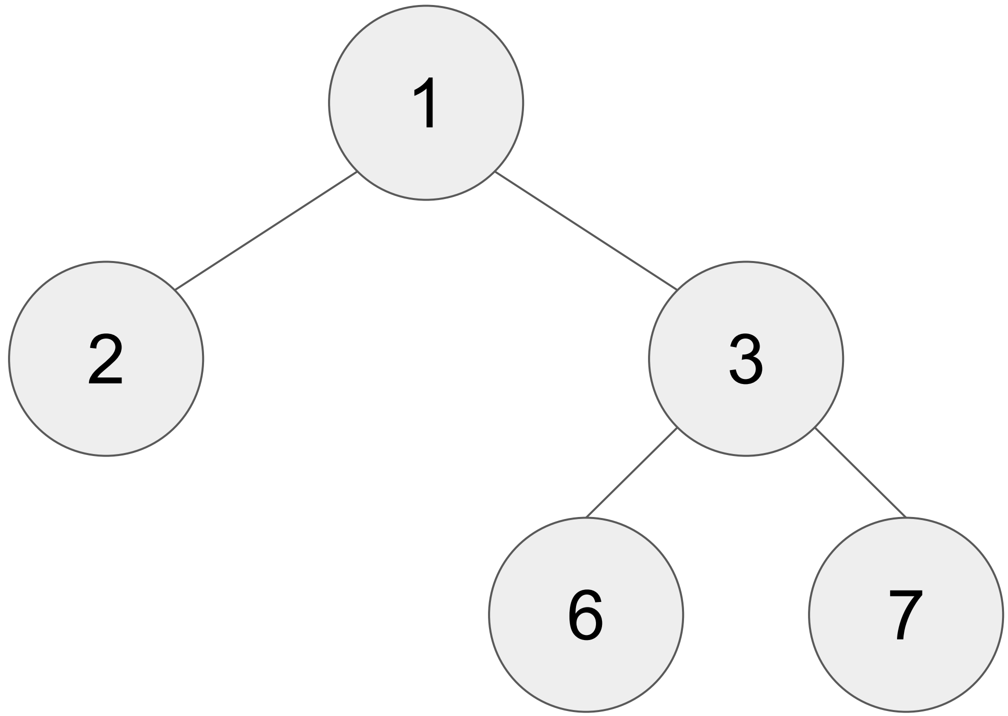
Accepted: ?

Author: Ivan Paljak, Luka Kalinovčić

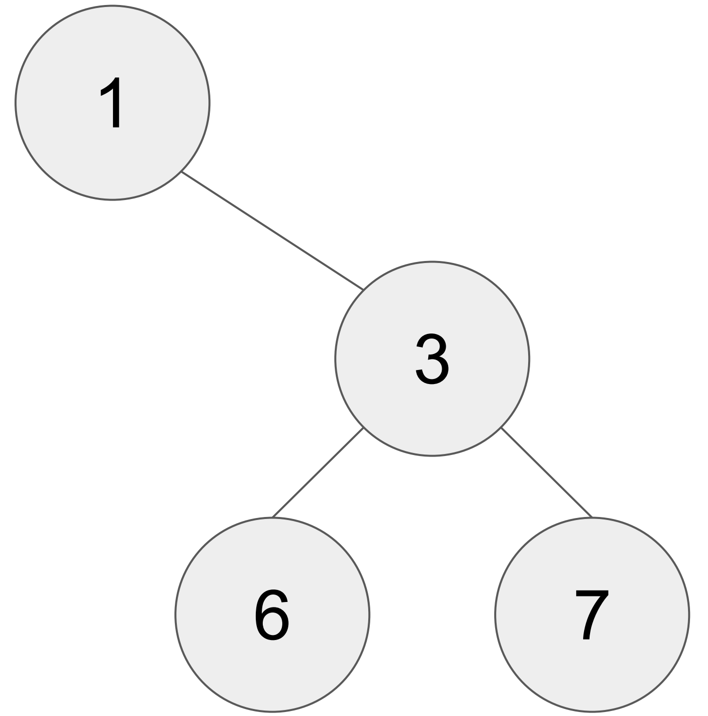

1
2
3
4
5
6
7



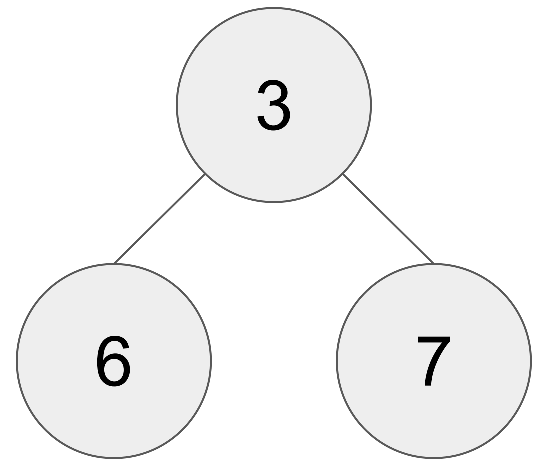
Code: 2



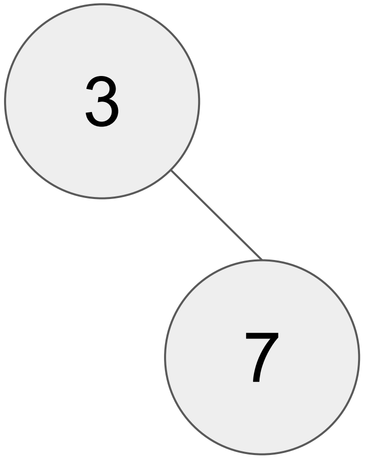
Code: 2 2



Code: 2 2 1

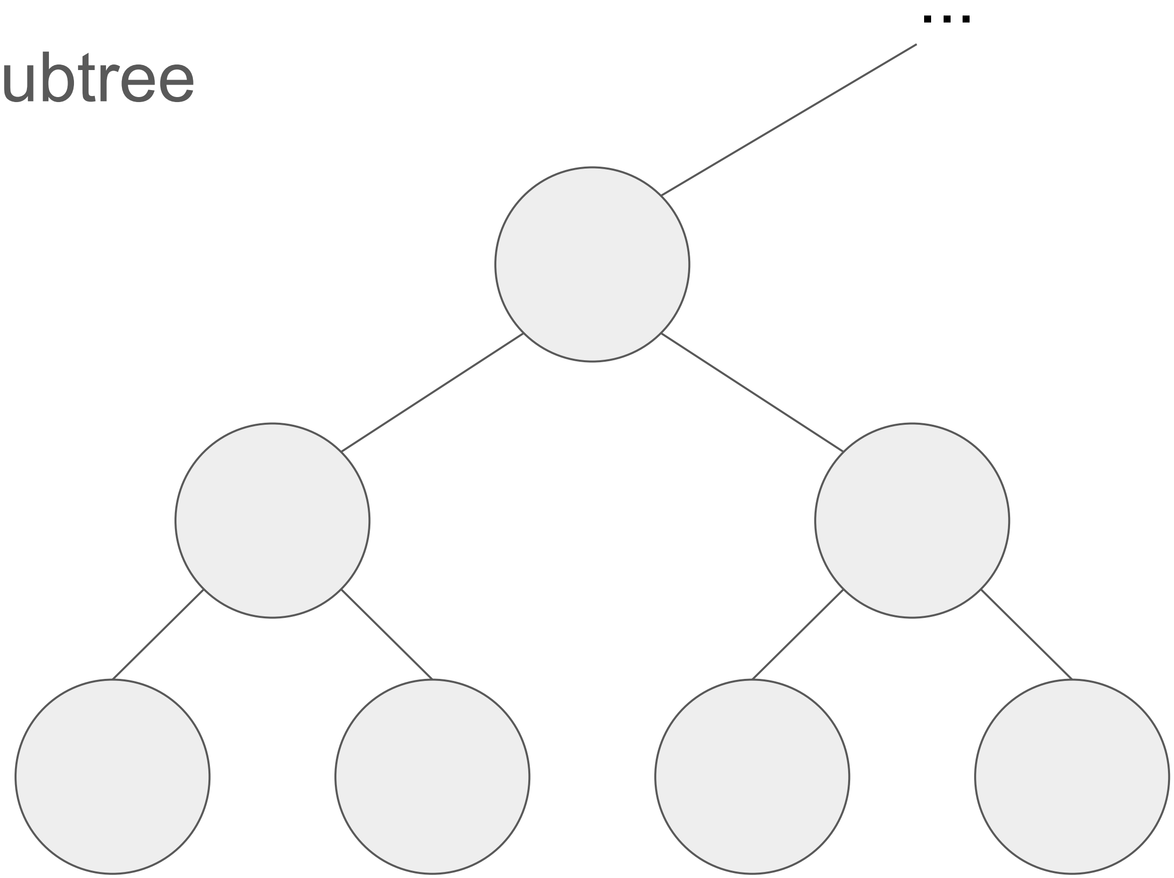


Code: 2 2 1 3



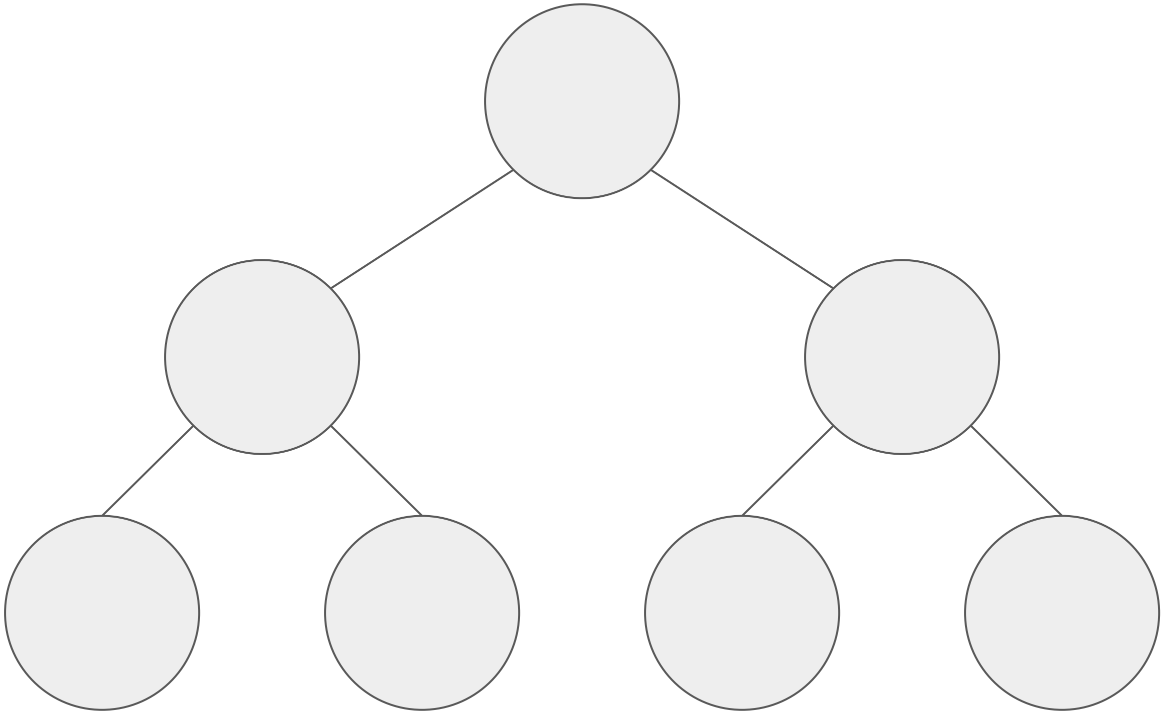
Code: 2 2 1 3 3

Type A subtree



The removal order: left subtree, right subtree, root node.

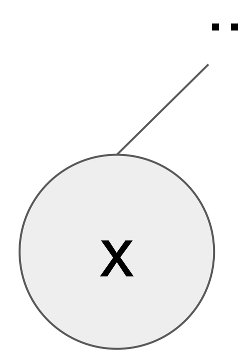
Type B subtree



The removal order: left subtree, root node, right subtree.

In the analysis we'll focus on type A trees only. Type B is dealt with the same way.

Let's start simple and find a recursive formula $f_x(k)$ to sum up the code generated by a type A subtree of depth k, where root is labeled with number x.

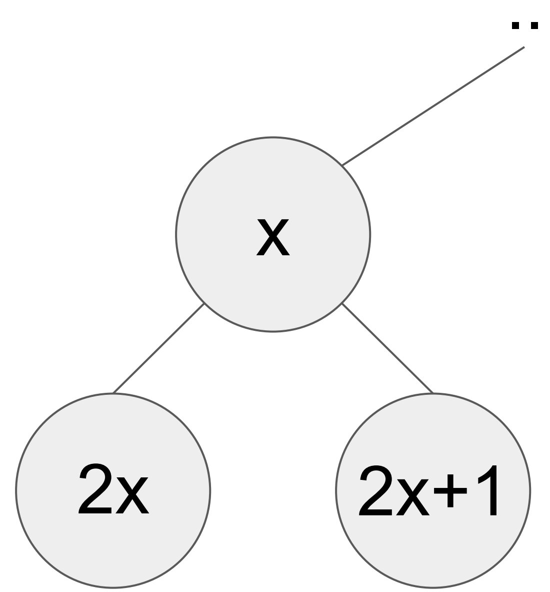


For k = 1, there is only a single node in the subtree.

As we remove it, we append (x div 2) to the code.

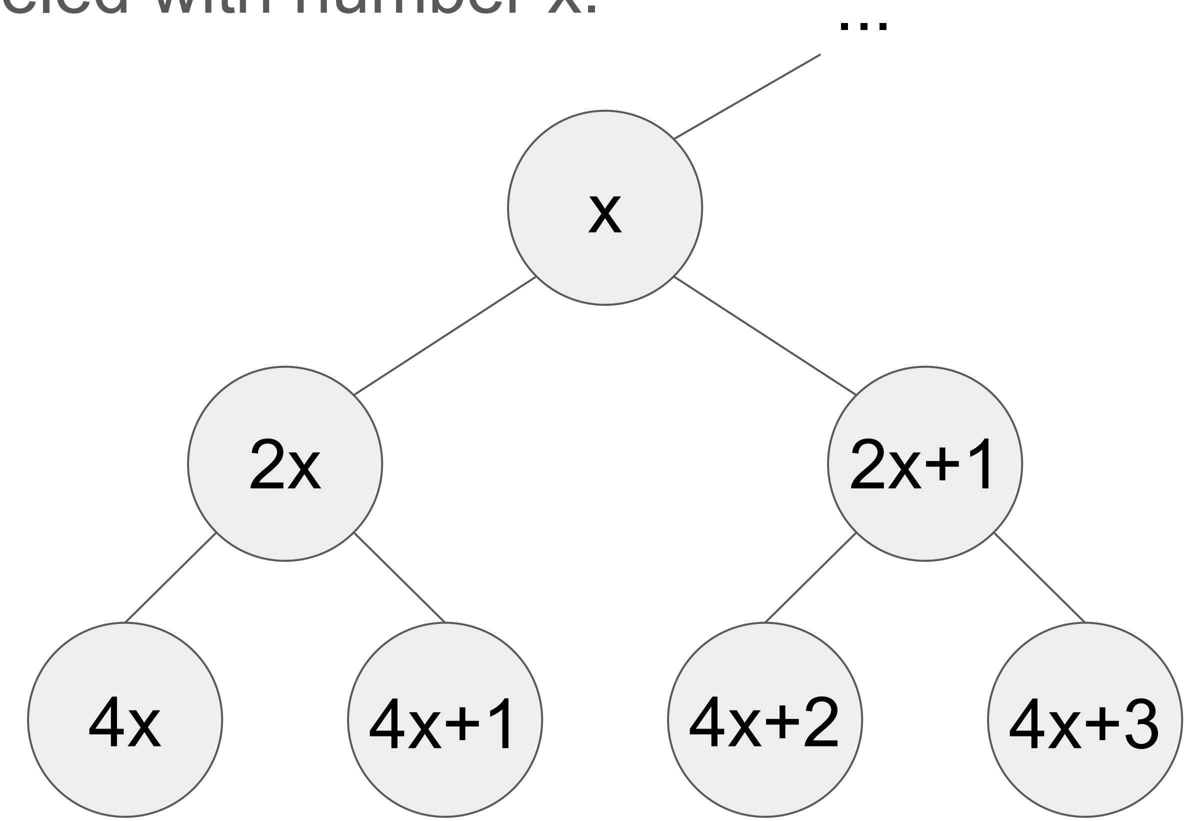
$f_x(1) = (x \text{ div } 2)$

Let's start simple and find a recursive formula $f_x(k)$ to sum up the code generated by a type A subtree of depth k, where root is labeled with number x.



$f_x(2) = x + x + (x \text{ div } 2) = 2x + (x \text{ div } 2)$

Let's start simple and find a recursive formula $f_x(k)$ to sum up the code generated by a type A subtree of depth k, where root is labeled with number x.



$f_x(3) = 2x + 2x + x + 2x+1 + 2x+1 + x + (x \text{ div } 2)$

$= 10x + 2 + (x \text{ div } 2)$

In general, $f_x(k) = a_k \cdot x + b_k + c_k \cdot (x \text{ div } 2)$ and we can compute it recursively:

$$f_x(k) = f_{2x}(k-1) + f_{2x+1}(k-1) + (x \text{ div } 2)$$

$$\begin{aligned} f_{2x}(k-1) &= a_{k-1} \cdot 2x + b_{k-1} + c_{k-1} \cdot (2x \text{ div } 2) \\ &= (2a_{k-1} + c_{k-1})x + b_{k-1} \end{aligned}$$

$$\begin{aligned} f_{2x+1}(k-1) &= a_{k-1} \cdot (2x + 1) + b_{k-1} + c_{k-1} \cdot ((2x + 1) \text{ div } 2) \\ &= (2a_{k-1} + c_{k-1})x + a_{k-1} + b_{k-1} \end{aligned}$$

$$f_x(k) = (4a_{k-1} + 2c_{k-1})x + a_{k-1} + 2b_{k-1} + (x \text{ div } 2)$$

$$a_k = 4a_{k-1} + 2c_{k-1} \qquad b_k = a_{k-1} + 2b_{k-1} \qquad c_k = 1$$

Now, let's come up with a formula that only sums up code elements at indices in the query

$$Q = \{a, a + d, a + 2\cdot d, ..., a + (m - 1)\cdot d\}.$$

Let $next_Q(i)$ be the smallest index in Q greater than or equal to i.

Let $g_x(k, i)$ be the sum of elements at the required indices, given a subtree of depth k with root labeled x, and given that there are already i elements in the output code before we process the subtree.

$$\begin{aligned} g_x(k, i) \quad &= g_{2x}(k-1, i) + g_{2x+1}(k-1, i + 2^{k-1} - 1) \\ &+ ((i + 2^k - 1) \in Q)\cdot(x \text{ div } 2) \end{aligned}$$

The recursive formula we have is still summing elements one-by-one. We need to optimize it a bit.

1) If no index in $[i + 1, i + 2^k - 1]$ is in query Q, return 0 immediately.

2) Memoize function calls where:

- $k \leq K/2$ and
- $[i + 1, i + 2^k - 1]$ is entirely within the query interval $[a, a + a + (m - 1) \cdot d]$.

The key for the memoization is $(k, next_Q(i) - i)$.

Because of 1), $next_Q(i) \leq i + 2^k - 1$, so we have $O(2^{K/2})$ states to memoize.

The remaining cases where we don't return 0 or memoize are:

1) Cases for type B subtrees. There are only O(K) such function calls.

2) Cases with $k > K/2$. There are $O(2^{K/2})$ function calls.

3) Cases where $[i + 1, i + 2^k - 1]$ intersects with the query interval $[a, a + a + (m - 1) \cdot d]$, but is not entirely within. There are only O(K) such function calls.

Overall complexity of the algorithm is $O(2^{K/2})$ per query.