Segment Tree

The segment tree is one of the most useful data structures in competitive programming. For what tasks is it needed? Let's start with the most basic task.

Sum on a segment

Suppose we have an array a_i of n elements, and we want to be able to do two operations with it:

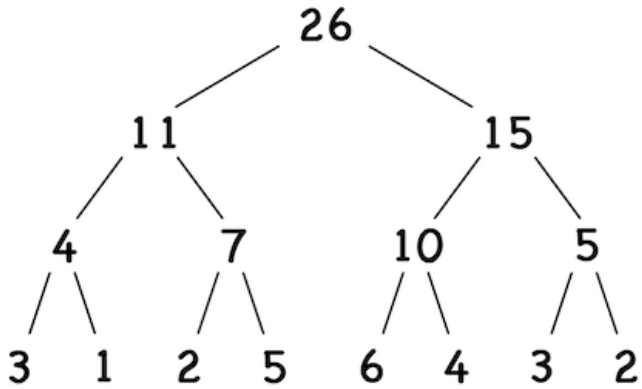
- `set(i, v)`: set the element with index i to v.
- `sum(l, r)`: find the sum on the segment from l to $r-1$.

Note that in the request for the sum we take the left border l inclusive, and the right border r exclusive. So we will do in all cases when we talk about segments.

Structure of the segment tree

Let's imagine that we need to build a segment tree for the following array:

The segment tree be constructed as follows. This is a binary tree, in the leaves of which there are elements of the original array, and each internal node contains the sum of the numbers in its children.

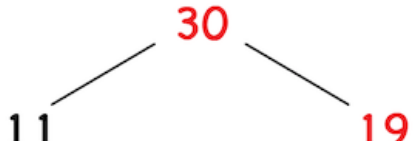


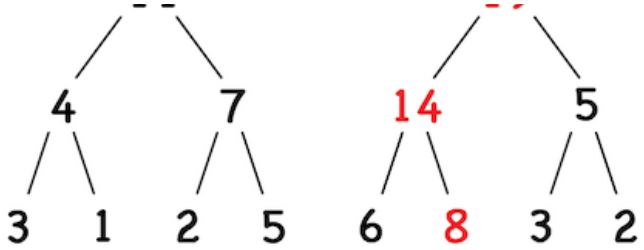
Note that the tree turned out so beautiful, because the length of the array was a power of two. If the length of the array is not a power of two, you can extend the array with zeroes to the nearest power of two. In this case, the length of the array will increase no more than twice, so the asymptotic time complexity of the operations will not change.

Now let's look at how to do operations on such a tree.

Operation set

Let's start with the operation `set`. When the element of the array changes, you need to change the corresponding number in the leaf node of the tree, and then recalculate the values that will change from this. These are the values that are higher up the tree from the modified leaf. We can simply recalculate the value in each node as the sum of the values in children.

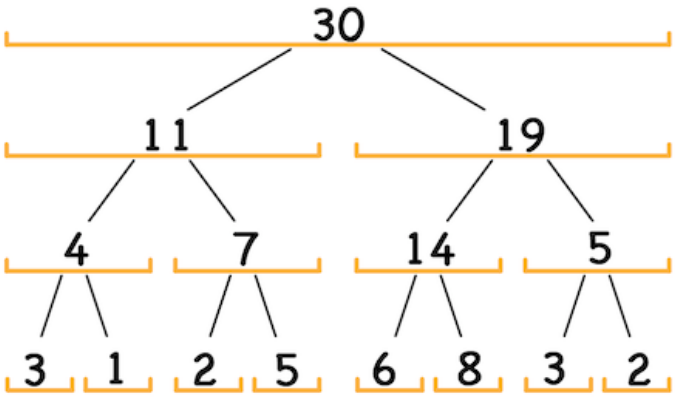




When performing such an operation, we need to recalculate one node on each layer of the tree. We have only $\log n$ layers, so the operation time will be $O(\log n)$.

Operation sum

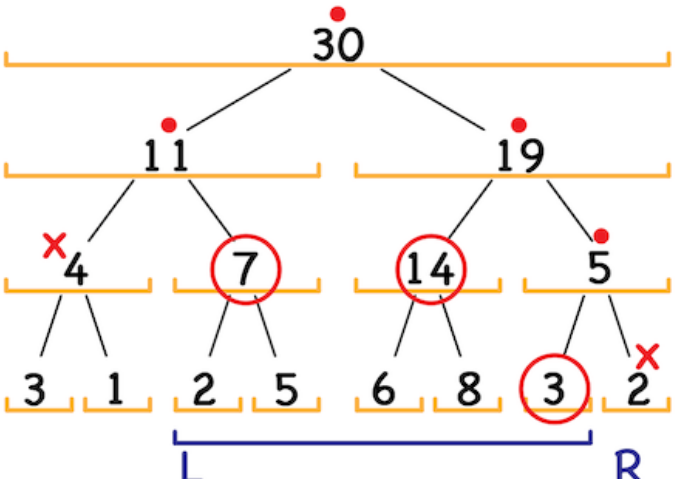
Now let's look at how to calculate the sum on a segment. To do this, let's first see what kind of numbers are written in the nodes of the segment tree. Note that these numbers are the sums on some segments of the original array.



In this case, for example, the number in the root is the sum over the entire array, and the numbers in the leaves are the sum over the segment of one element.

Let's try to build the sum on the segment $[l..r)$ from these already calculated sums. To do this, run a recursive traversal of the segment tree. In this case, we will interrupt recursion in two situations.

- The segment corresponding to the current node does not intersect the segment $[l..r)$. In this case, all the elements in this subtree are outside the area in which we need to calculate the sum, so we can stop the recursion.
- The segment corresponding to the current node is entirely nested in the segment $[l..r)$. In this case, all the elements in this subtree are in the area in which we need to calculate the sum, so we need to add to the answer their sum, which is recorded in the current node.



Here, the crosses indicate the vertices at which the recursion broke off in the first cutoff, and the vertices in which the number was added to the answer are circled.

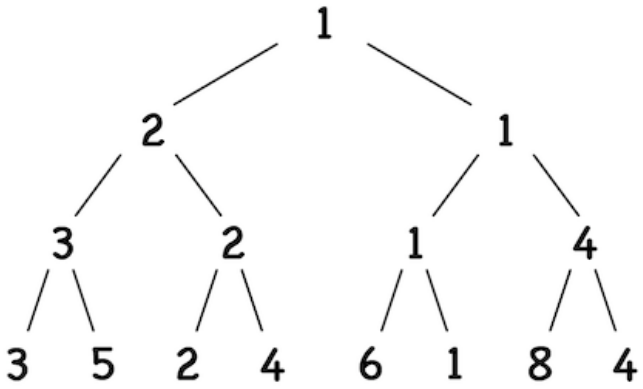
How long does such a tree traversal work? To answer this question, we need to understand how many nodes none of the cutoffs will happen in, and we will need to go deeper into the tree. Each such case gives us a new branch of recursion. It turns out that there will be quite a few such nodes. The fact is that in order for none of the cutoffs to work, the segment corresponding to the node of the tree must intersect the query segment, but not be contained in it entirely. This is only possible if it contains one of the boundaries of the segment $[l..r)$. But on each layer of the tree of segments there can be no more than one segment containing each of the boundaries. Thus, there can be no more than $2 \log n$ nodes at which cutoffs did not work, and, therefore, the general asymptotic time of this procedure will be $O(\log n)$.

Minimum and other functions

What other operations can be done using the segment tree? Instead of the sum, you can calculate other functions on the interval, for example, a minimum. Add the following operation to the tree:

- `min(l, r)`, which returns the minimum on the segment $a[l..r-1]$.

How to handle such an operation using the segment tree. Let's build the same tree as for the sum, only in each node record not the sum of the elements in children, but the minimum.



The `set` operation is performed as before. You need to replace the element in the leaf node and then recalculate the values up the tree to the root. The operation `min` is performed in the same way as `sum`: you need to traverse the tree, while doing the same cutoffs, while the segment will be divided into several segments, on which we already know the minimum. Taking a minimum of these numbers, we get a minimum over the entire segment. The operation time will also be $O(\log n)$.

Other functions

In addition to the sum and the minimum, using the segment tree, you can calculate any associative operation. The operation $\otimes$ is called associative if its result does not depend on the order in which it is calculated, that is, if $(a \otimes b) \otimes c = a \otimes (b \otimes c)$.

For example, in addition to the sum and the minimum, associative operations are:

- multiplication (including modulo multiplication, matrix multiplication, etc.),
- bitwise operations: `&`, `|`, `^`,
- the largest common divisor (in this case, it should be noted that the calculation time of the GCD is not $O(1)$, so the operation time will be longer.

In simpler terms, a function can be used for a segment tree, if you know the result of its calculation for two halves of a segment, you can quickly calculate its result for the entire segment.

Consider a more nontrivial operation. Suppose we want to be able to calculate on a segment the number of elements equal to the minimum on this segment. To add such an operation, let's store in each node a pair of numbers (min, cnt): the minimum value on the segment and the number of elements equal to this minimum.

Now let's see how to recalculate the value of such a pair in a segment, knowing the values in halves. Suppose that the values are (m_1, c_1) and (m_2, c_2) on each half of the segment, let's see how the values of m_1 and m_2 are related. If $m_1 < m_2$, then the minimum on the segment is m_1, and the number of elements equal to the minimum is c_1, so for a large segment we get the value (m_1, c_1). Similarly, if $m_1 > m_2$, then the value for a large segment is (m_2, c_2). If $m_1 = m_2$, then the minimums at half are the same. The total number of elements equal to the minimum is $c_1 + c_2$, so we get the pair $(m_1, c_1 + c_2)$. We construct a segment tree in which such a pair is recorded in each node, and we will update it using the described function. As a result, we obtain a segment tree with the required operation.