Competitive Programming and Mathematics Society

ICPC Workshop #2

Segment Trees

Ryan, Patrick and David

Attendance

CPMSOC

Agenda

CPMSOC

1. Segment Trees for range operations
2. Segment Tree Implementation (Recursive)
3. Segment Trees for Range Updates
4. Iterative Segment Trees
5. Wrap Up

Segment Trees

CPMSOC

Definition

A Segment Tree is a data structure that stores information about array intervals as a tree. It allows answering range query over an array efficiently, and also allow for efficient modification of the array.

For the most basic segment tree, we can use it to store info about interval's sum. Then it will support :

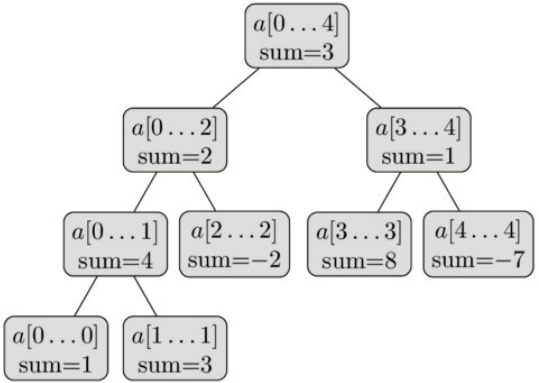
- The range query of sum. $O(\log n)$
- Update one index. $O(\log n)$
- Update a segment of indexes. $O(\log n)$

Which will out-perform naive iterative solution, or prefix-sum array.

A nice visualisation of Segment Tree

CPMSOC

Here is a visual representation of such a Segment Tree over the array $a = [1, 3, -2, 8, -7]$:



(Copied from cp-algorithms)

Construction of the segment tree

CPMSOC

We shall first consider what information we store in each node,

- The interval. (e.g start, end)
- Info about the interval. (e.g Sum of the interval, in our case)
- Sub-intervals. (e.g The child of the node)

We will take a recursive divide and conquer approach for constructing segment tree, this means we will recursively call our build function, if the interval can be divided into two. So we always start at the bottom level of the tree.

During construction of the segment tree, we have another merge step. In our case, we can always derive the sum of interval from the sum of two children's sum.

Sum query

CPMSOC

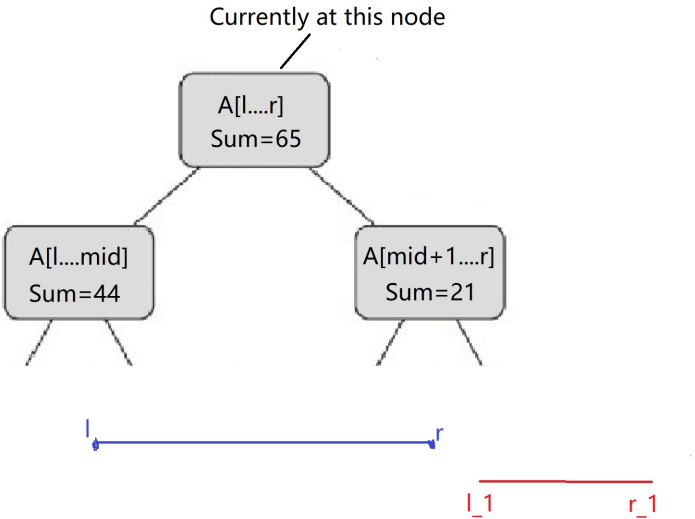
After construction of above data structure, we now able to answer sum queries. Suppose we want to compute the segment A[l_1::r_1], and we were at segment tree's node that represent interval A[l::r].

Our algorithm is simple, consist of two base case and a recursive case.

Click next slide to see nice visualization.

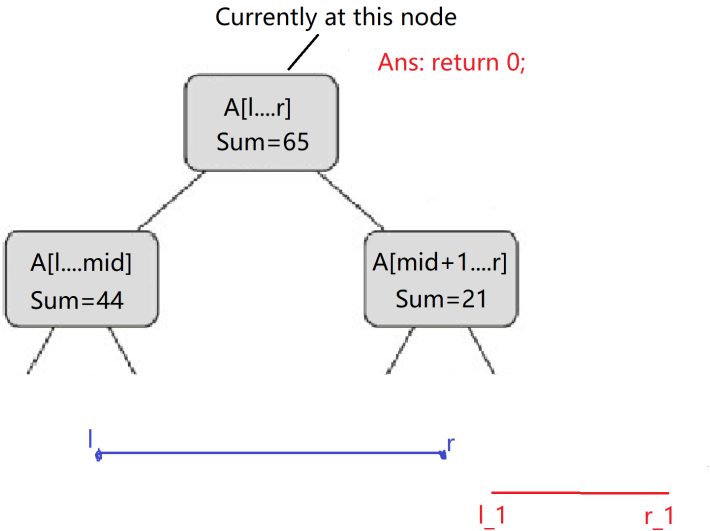
Base Case 1: No intersection

CPMSOC



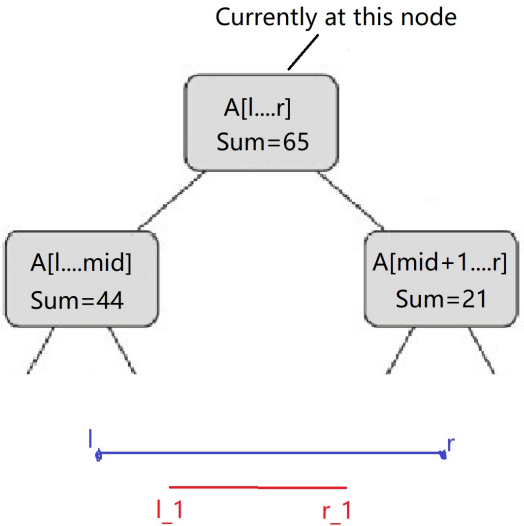
Base Case 1: No intersection

CPMSOC



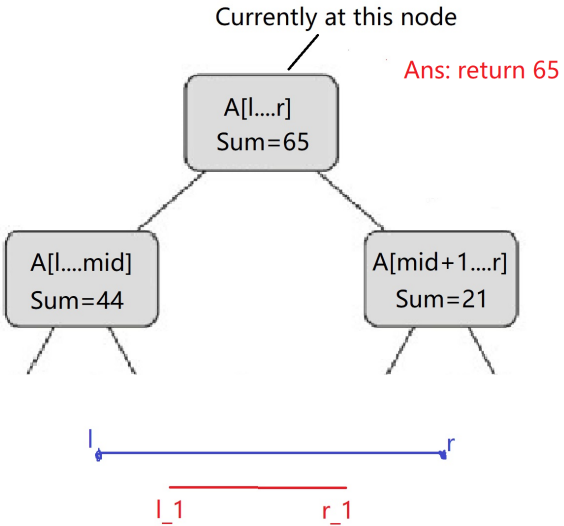
Base Case 2: Completely covered

CPMSOC



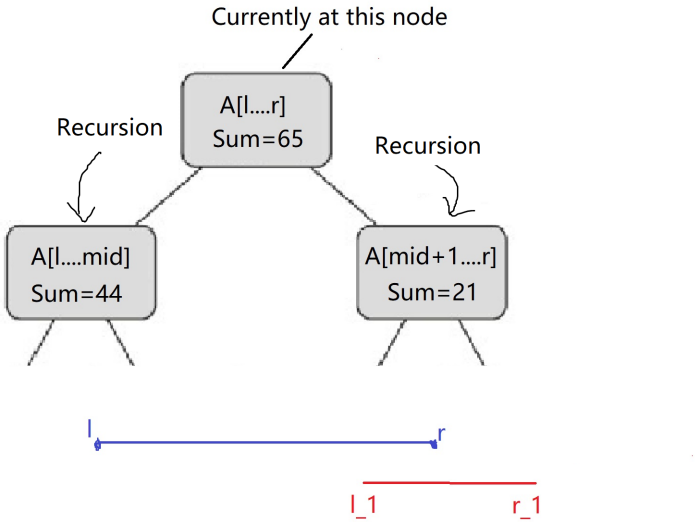
Base Case 2: Completely covered

CPMSOC



Recursive case: Partially intersected

CPMSOC



Point Updates on a Segment Tree

CPMSOC

Each element of the array is included in $O(log(N))$ nodes of the segment tree, one in each layer. Thus, we can achieve point updates in $O(log(N))$ by recursing top-down from the root and updating as we go.

We can complete this as long as we can update the leaf node easily, and updates can be easily performed as we move up the segtree. Almost all point operations can handle this, such as:

- Point Set
- Point Add / Subtract
- Point Set Insertion

Range Queries on a Segment Tree

CPMSOC

When answering range queries, the query range is decomposed into a set of disjoint, but adjacent subarrays. We can combine the precomputed answers for these subarrays to answer the original query. It can be shown that there at most $O(log(N))$ Segtree subarrays necessary for any interval $[L::R]$ in an array of length N.
This works as long as we have some method of combining the left and right subtrees in $O(1)$ time. Here are a couple examples of operations that we can do:

- Range Max / Min (commonly known as the RMQ problem)
- Range Sum
- Range Bitwise XOR
- Range Count Zeros / Search for K-th Zero
- Range Longest Increasing Subarray
- Range Mex (Minimum Excludant)

Segment Tree Implementation (Recursive)

CPMSOC

We implement a segment tree as an array where the node is numbered 1, and for each node v, the left child is $2 \quad v$ and the right child is $2 \quad v + 1$. We need a maximum of $4 \quad N$ nodes in our tree.

We need three functions for a basic segment tree:

build(int v, int tl, int tr)
set(int v, int tl, int tr, int x, int val)
sum(int v, int tl, int tr, int l, int r)

v, tl and tr store information about the node we are currently at.
l, r, x and val store information about the operation we are performing.

Range Updates on a Segment Tree

CPMSOC

For a range update, the worst case is that the update affects every node ($O(N)$ time complexity). We can get this down to $O(log(N))$ by observing that there are a minimum of $O(log(N))$ nodes necessary to cover the any range.

This requires us to perform ***Lazy Updates***, where we only $push$ range updates to a node's children when we need to. This requires us to store information at each node about updates that are yet to be propagated.

We create a function $push(node\_number)$, which propagates updates to the children. We call this function before recursing to child nodes. This comes at with a constant factor cost increase to memory (for storing the extra value), but achieves the necessary $O(log(N))$ per update.

Problem: Global Warming

CPMSOC

You are given a list of $predictions$ about the temperature over the next N days. The i-th prediction states that from day L_i to R_i inclusive, the temperature will reach a maximum of X_i. Determine if it is possible for all the predictions to be correct (i.e there exists some array of temperatures such that each prediction is fulfilled).

Problem: Global Warming

CPMSOC

You are given a list of $predictions$ about the temperature over the next N days. The i-th prediction states that from day L_i to R_i inclusive, the temperature will reach a maximum of X_i. Determine if it is possible for all the predictions to be correct (i.e there exists some array of temperatures such that each prediction is fulfilled).

Can we define the operations we need our data structure to support?

Problem: Global Warming

CPMSOC

You are given a list of $predictions$ about the temperature over the next N days. The i-th prediction states that from day L_i to R_i inclusive, the temperature will reach a maximum of X_i. Determine if it is possible for all the predictions to be correct (i.e there exists some array of temperatures such that each prediction is fulfilled).

Can we define the operations we need our data structure to support?

We need a data structure which supports both range set and range max.

Problem: Global Warming

CPMSOC

You are given a list of $predictions$ about the temperature over the next N days. The i-th prediction states that from day L_i to R_i inclusive, the temperature will reach a maximum of X_i. Determine if it is possible for all the predictions to be correct (i.e there exists some array of temperatures such that each prediction is fulfilled).

Can we define the operations we need our data structure to support?

We need a data structure which supports both range set and range max.

Range set operations override each other. Can we define an ordering to them so that we do not lose information?

Problem: Global Warming

CPMSOC

You are given a list of $predictions$ about the temperature over the next N days. The i-th prediction states that from day L_i to R_i inclusive, the temperature will reach a maximum of X_i. Determine if it is possible for all the predictions to be correct (i.e there exists some array of temperatures such that each prediction is fulfilled).

Can we define the operations we need our data structure to support?

We need a data structure which supports both range set and range max.

Range set operations override each other. Can we define an ordering to them so that we do not lose information?

Applying the operations in decreasing order of temperature will ensure that more restrictive (lower temperature) predictions will not be overridden by less restrictive (higher temperature) predictions.

Iterative Segment Trees

CPMSOC

Iterative segment trees are an alternative to the recursive version, which is preferable in certain scenarios. https://codeforces.com/blog/entry/18051 is a very high-quality blog post

Iterative Segment Tree Differences:

- The structure of the segment tree is more closely tied to the structure of the array.
  - The segtree node corresponding to the i-th element is located at $i + N$
  - Traversal is easier. To go up, divide the index by 2. To go to the left child, multiply by 2. To go to the right child, multiply by 2 and add 1.
- Construction of the segment tree is completed bottom-up, layer by layer
- Queries are answered outside-in
- Modifications are completed bottom-up
- Only uses 2 N Memory rather than 4 N
- Generally a constant factor more time efficient
- Much more lightweight implementation (most functions are a single for loop)
- Inclusive-Exclusive ranges

Wrap Up

CPMSOC

- Problems:
  - Implement a basic Segment Tree with operations of your choice (I suggest one of point set, point add, range set and one of range sum or range max)
  - Global Warming
  - Cannons (Applying Segment Trees to DP) (ask for link)
- Investigate persistent segment trees and lazy creation
- Another Workshop on this Friday, 12-2
- ICPC Divisionals this Saturday
- Attendance Form
- Weekly Social Sessions, every Wednesday