Welcome

Title

Hello, and welcome to competitive programming.

Today we are going to introduce an advanced data structure call a segment tree. It allows up to ask questions about ranges very efficiently.

Objectives

Your objectives is to be able to explain the math behind segment trees, and to be able to implement one. You will be able to use them to answer a Range Minimum Query and a Range Sum Query.

Range Minimum Query

The Problem

Suppose you have a large array of integers and want to be able to select a subarray and ask what is the index of the smallest number in that range.

The naive way of doing this would involve an Order n linear search through the subrange. This is fine if you only have to do this query once, but if you have to do it many times then this is not efficient enough.

Here's an example array with 8 elements. If I ask about the range from 3 to 7, then the index with the minimum is 5. If I ask about range 0 to 3, then the index with the

minimum is 3.

The structure

To answer these kinds of queries, we are going to use a segment tree. This first part will be a static structure, in which we initialize a segment tree an perform multiple queries. The second part will be a dynamic segment tree where we can update the information in it.

We will use a an array based tree, kind of like we do with heaps. Index 1 will be the root of the tree, and will represent the entire range. For a given index p, $2p$ will be the left child, and represent the left half of the initial range. Similarly, index $2p+1$ will be the right child, and represent the right half of that range.

For our example, element 1 of the segment tree represents the entire range. Element 2 is the left child, and represents the left half of that range, or elements 0 through 3, inclusively.

If this is your first time seeing a segment tree, it would be a good idea for you to try to reproduce this tree yourself to be sure you understand it.

Queries

There are two kinds of queries you will see. The first is when we want to query a range that has an exact representation in the tree.

If we query range [0,7] or range [4,5], these can be answered with a simple lookup.

If we have a query like [1,4] it is a bit more complex. We need to consult three separate nodes of the segment tree to know the right answer.

Code

Here is the code to do this kind of query. I stole it from the competitive programming textbook.

The function takes five parameters. The first is the index p into the segment tree. The next two are the left and right bounds of the segment corresponding to the node p. The last two are the range we want to query.

The first thing we do is check to see if the range [L,R] is completely disjoint from the query range [i,j]. If so, we return -1. Next we check if the range [L,R] is completely inside range [i,j]. If it is, we return the data in the segment tree.

Now we recursive. We divide the region in half and check each side. This gives us p1 and p2. If either of them are -1 then we return the other side.

If both of them have data, then we do a lookup to see which of the two minimums should win.

Code for Building

We build the segment tree recursively. Here, p is the index into the segment tree, and L and R are the bounds that correspond to that node.

The base case is when L and R are the same, in which case we store the index. For a recursive case, we check the left and right halves to see which side wins.

Dynamic Updates

Segment trees are not the best structure if we do not change the underlying structure after we have built the tree. There is a dynamic programming solution that works better, and we'll cover that later in the course.

But suppose we need to update this tree, suppose by changing an individual element. In this case, we simply find the node corresponding to the element by

recursing down through the tree, and then rebuilding the nodes back to the root if the new value becomes a minimum.

Range Sum

There are other problems you can solve with this data structure too; the ranged sum query is one of them. Dynamic updates of a single element can be done in O(log n) time, but there is a more advanced version of this structure than can update entire ranges in O(log n) time. These are call segment trees with lazy update, and will be the subject of a future lecture.

Next
Transcript — The Sieve of Eratosthenes →