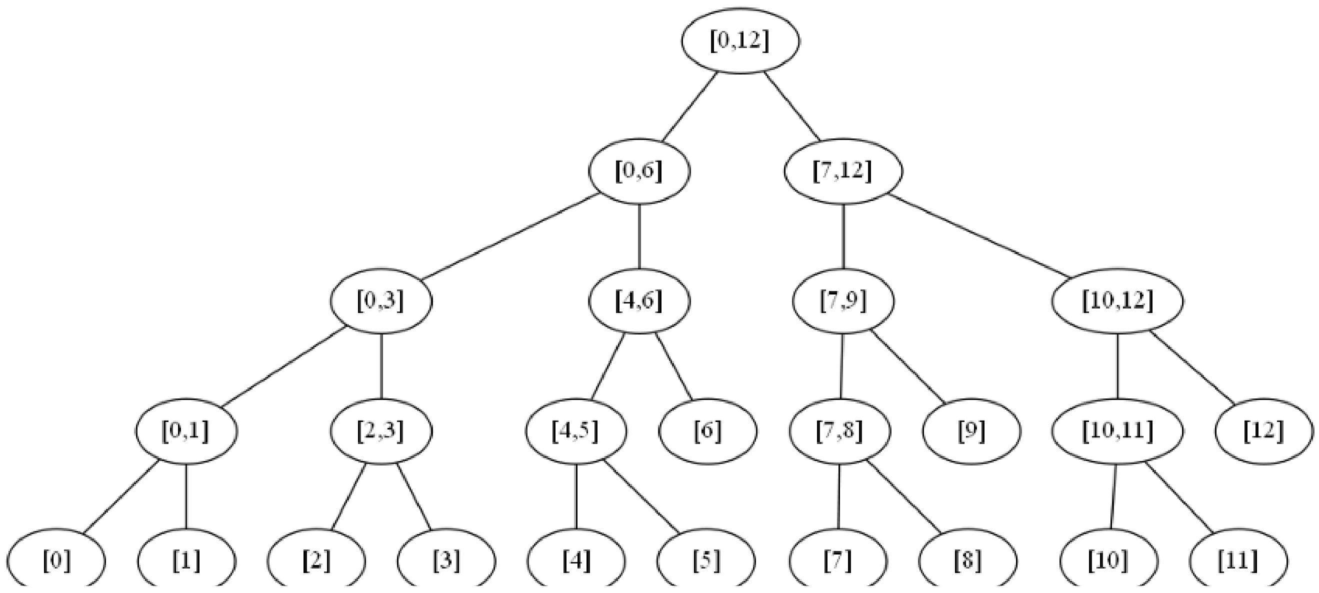
Introduction to Segment Tree

Bharat Kul Ratan Follow
Oct 15, 2017 · 6 min read

Segment tree is a data structure similar to heap. Every node of this tree has zero or two children. Nodes having zero child are called leaf nodes and represent the single element from the given array for which Segment tree would be built. Each node of the tree will hold information about a range from the given array. The closer the node to root, the more range it covers. Because it's a tree, we can perform queries in **O(log2N)** where N is number of elements given. Here base of this logarithm is two as every node of this tree will have at most two child nodes. Left child will hold information for left half of the parent range and right child will hold info for remaining right half. For the interval [A, B] ([A,B] represents closed interval for A and B) in the given array, the Segment tree would be defined in the following recursive manner:
* The root node will hold the information for the interval [A, B]
* For A < B, the left child will hold the information for interval [A, (A+B)/2] and right child will hold the information for the interval [((A+B)/2)+1, B]

For example, Segment tree for an array having thirteen elements in an array [0, 12] would look like,



Segment tree for thirteen elements.

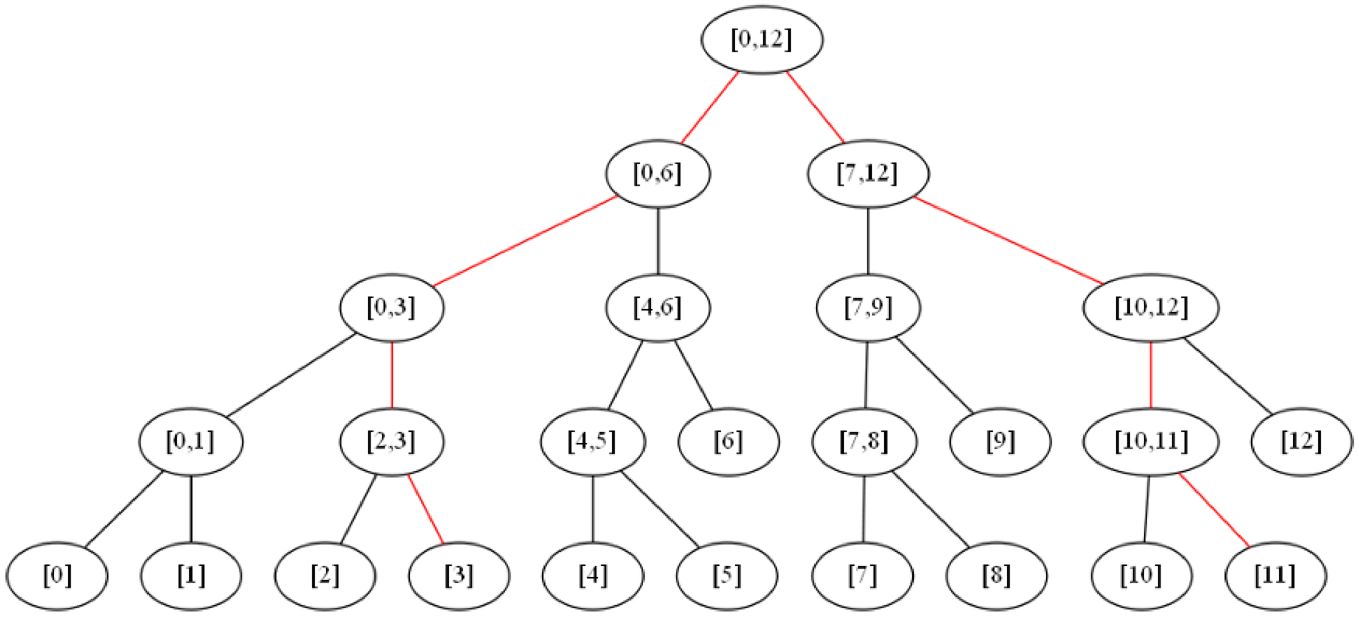
Let's take an example. Suppose we are given an array of 13 integers and numerous queries to find out maximum value between a given range. To perform our queries in O(log2N) time we will build a Segment tree from this array. Each of its node will hold maximum value for a particular interval for which this node is responsible for as shown in the above figure. The tree is built in O(N) and let us do queries in O(log(N)). Here is the code for the above example. In this code the Segment tree is built for an array and queries maximum value for a range.

```
import java.util.Random;
import java.util.Arrays;

class SegmentTree{
 // arr is given array
 // segTree is our Segment tree in form of array
 public int arr [], segTree [];

 SegmentTree(int arrayLength){
  arr = new int [arrayLength];
  Random generator = new Random();

  for (int I=0; I<arrayLength; I++){
   //putting random values in array
   arr[I] = generator.nextInt(100);
  }

//We will be building seg tree for array of length 13
  //for now let's take segTree length 32

  int segTreeLen = 32;
  segTree = new int [segTreeLen];
  Arrays.fill (segTree, 0);
 }

 /**
  * @param node is the root node of tree
  * @param left is the beginning index of array
  * @param right is the ending index of array
  **/
 private void buildSegmentTree(int node, int left, int right){
  if (left == right){
   segTree [ node ] = arr[left];
  }
  else {
   //recursively approach left half of tree until a leaf is found
```

```
      buildSegmentTree (2 * node, left, (left+right)/2 );

      //recursively build the right half of tree
      buildSegmentTree (2 * node + 1, ((left+right)/2)+1, right);

      // value in left child of node
      int valueInLeftChild = segTree [2 * node];

      // value in right child of node
      int valueInRightChild = segTree [ 2 * node +1];


      // comapare both childs for greater value
      // and assign to node that value
      if ( valueInLeftChild  >= valueInRightChild ){
       segTree [ node ] = valueInLeftChild;
      }
      else segTree [ node ] = valueInRightChild;
     }
    }

    /**
     * @param node is the root node of tree
     * @param left is the beginning of the array index
     * @param right is ending index of array
     * @param start is the beginning index of the query
     * @param end is ending index of query
     **/
    private int query (int node, int left, int right, int start, int
   end){

     // if query range is out of array interval no valid value is
   returned
     if ( start > right || end < left){
      return -1;
     }

     /**
      * if query range includes the array interval
      * return the value of node in the tree
      * which holds the maximum value for the range
      * Segment tree saves our computation in this manner
      * because segTree[treeNode] already holds maximum value for this
   range
      **/
     if ( start <= left &&  end >= right){
      return segTree [node];
     }

     int half = (left+right)/2;

     //Computing maximum value in left half of the tree
     int leftMax = query (2*node, left, half, start, end);
    //Computing maximum value in right half of the tree
    int rightMax = query (2*node+1, half+1, right, start, end);

    int leftCorner, rightCorner;
    leftCorner =(start<left)?left:start;
    rightCorner = (right<end)?right:end;

    /**
     * Below if-else statements updates the node value
     * according to the maxium value from its child node
     **/
    if (leftMax == -1){
     System.out.println ("max value is " + rightMax + " between index "
  + leftCorner  + " and " + rightCorner);
     return segTree[node] = rightMax;
    }
    if (rightMax == -1){
     System.out.println ("max value is " + leftMax + " between index "
  + leftCorner + "  " + rightCorner);
     return segTree[node] = leftMax;
    }
    if (leftMax > rightMax){
     System.out.println ("max value is " + leftMax + " between index "
  + leftCorner + "  " + rightCorner);
     return segTree[node] = leftMax;
    }
    else {
     System.out.println ("max value is " + rightMax + " between index "
  + leftCorner + "  " + rightCorner);
     return segTree[node] = rightMax;
    }
   }


   public static void main (String ... args){
    //Build the Segment tree for an array having 13 elements
    SegmentTree tree = new SegmentTree (13);

    //Print the given array
    System.out.println (Arrays.toString(tree.arr));

    // 0 is the left index from array and 12 is the right index
    // 1 is choosen as root node because we need to calculate left and
  right child
    // by using formula 2*node and 2*node+1 which won't work for 0
    tree.buildSegmentTree(1, 0, 12);

    // Uncomment below line to print the Segment tree
    //System.out.println (Arrays.toString(tree.segTree));

    // We are finding maximum value between index 3 and 11 of given
  array for now
    System.out.println ("Max value between index 3 and 11 is " +
  tree.query (1, 0, 12, 3, 11));
```

```
      }
    }
```

For this example we are asking for maximum value between range [3,11]. In the figure below, path is show in the red which includes the starting and ending index 3 and 11 respectively. Query starts from the root node. As it continues down, those nodes will not be queried down further which are completely in the queried range.



We've build a static Segment tree. Static means once the tree is built it does not change. We perform queries which are read-only, they don't modify the tree. Let's take another operation of updating an element in array as well as in the tree. Again this point update operation would be performed in O(log(N)). And it's code is almost similar to *buildSegmentTree()*. Here's the code for updating point values:

```
  private void update (int nodeToUpdate, int valueToUpdate){
    update (nodeToUpdate, valueToUpdate, 1, 0, 12);
   }

   /**
    * @param nodeToUpdate is the node to be updated in array and tree
    * @param valueToUpdate is the value to be updated at the 
  nodeToUpdate
    * @param node root node of tree
    * @param left is the beginning index of array
    * @param right is the ending index of array
    **/
```

```
 private void update(int nodeToUpdate, int valueToUpdate, int node,
int left, int right){
  if (left == right){
   segTree [ node ] = arr[left];
   // only the below if condition is different from
buildSegmentTree()
   // which does the job of point update.
   if ( left == nodeToUpdate){
    segTree[node] = arr[left] = valueToUpdate;
   }
  }
  else {
   update (nodeToUpdate, valueToUpdate, 2 * node, left,
(left+right)/2 );
   update (nodeToUpdate, valueToUpdate, 2 * node + 1,
((left+right)/2)+1, right);


   int valueInLeftChild = segTree [2 * node];
   int valueInRightChild = segTree [ 2 * node +1];

   if ( valueInLeftChild  >= valueInRightChild ){
    segTree [ node ] = valueInLeftChild;
   }
   else segTree [ node ] = valueInRightChild;
  }
 }
```

We need to call this function from main() as

```
tree.update (nodeToUpdate, valueToUpdate);
// e.g. tree.update (4, 99);
```

In this article we learned how to build a Segment tree, performing queries over it and do point updates. In case of any doubts regarding this tutorial, please drop a comment. In the next part of this series we will learn about updating the tree using **lazy propagation** which is a very useful technique and saves a lot of calculation. So stay tuned. Have a nice day!!!

Programming   Data Structures   Java   Tutorial