Data Structure: Segment Tree

CUTTING EDGE VISIONARIES

Topics in brief

- What is data structure?
- Why do we need Segment tree?
- Principle of Segment tree
- Properties (Execution time, Space consumed)
- Some Questions

What is Data Structure?

- Efficient way to organize data.
- Can help in performing task efficiently by reducing the time required.

- As you will see now, the way we organise data matters a lot! It can reduce minutes to seconds!

Problem Statement

- You are given an array of n integers (a[0] to a[n-1]).
- You have to answer m queries.
- In each query, you are given two integers, l and r (0<=l,r<=n-1). Your job is to print the largest number in the range a[l], a[l+1], a[l+2],...., a[r-1], a[r].
- n is in the order of 10^6
- m is in the order of 10^5.

- **Real world data is usually as large as this**

Problem Statement

Let n=16 and the array be as follows.

Array elements	30	9	62	2	6	39	22	77	67	51	83	12	19	49	3	99
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Query: 1 3
Result: 62

Query: 0 15
Result: 99

Query: 5 9
Result: 77

Query: 12 12
Result: 19

Simple solution: Naïve approach

- Iterate from l to r and keep track of the maximum number found.
- Eg. For query 2 7, look at all the numbers from 2 till 7. Easy to find the max number this way.
- In form of a 'for' loop:

```
int max=array[l];
for(i=l;i<=r;i++)
{
  If(array[i]>max)
      max=array[i];
}
```

Though very simple to understand, this method is very time consuming.

- For each query, we have to look at (r-l+1) numbers. i.e. we have to perform (r-l+1) operations.
- n can be as large as 10^6. Therefore, the worst case is when l=0 and r=999999. 10^6 operations are required to find the answer in this case.
- As mentioned before, we can have as many as 10^5 queries.
- So, if all queries exhibit the worst case, we have to perform a total of (10^6) X (10^5) = 10^11 operations.

Analysis of execution time

- On a normal machine with 2.5 Ghz processor,

  10^7 or less operations take very less time (<.1 second)

  10^8 operations take about 0.5 - 1 second

  10^9 operations take about 3 - 4 seconds.

  ....

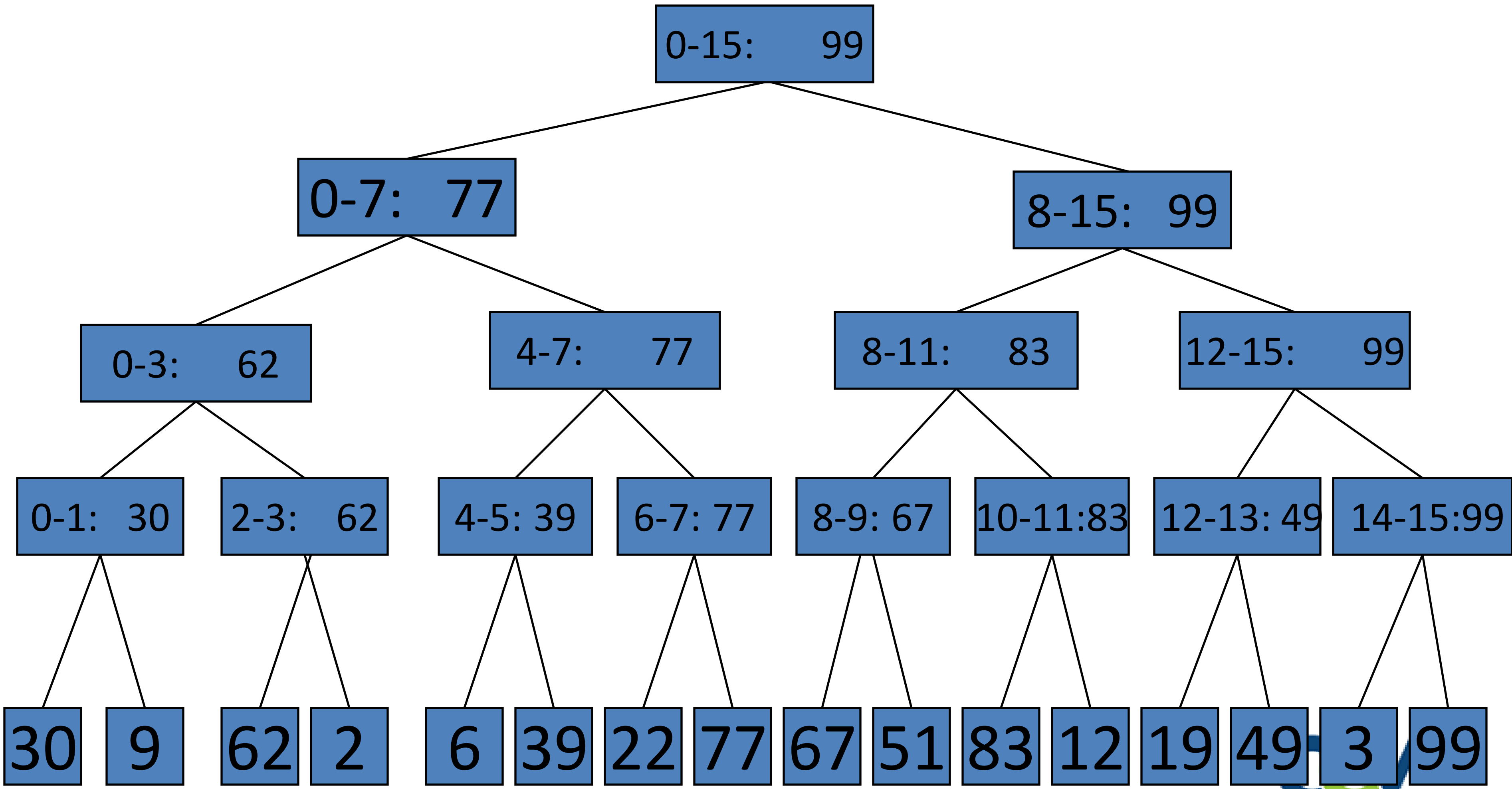
  10^11 operations will take unacceptably long time!

Principle of Segment tree

- The principle of segment tree is:
- If we have two arrays

  {3,5,2} and {6,22,1,6,10}.
- And the maximum integer of these arrays is m1=5 and m2=22 respectively. Then, the maximum of the array obtained by combining both these array {3,5,2,6,22,1,6,10} is MAX(m1,m2)=MAX(5,22)=22.
- i.e. We don't need to look at the whole combined array again. We can deduce the maximum of the whole array just by looking at the maximum of the individual arrays

Segment tree



Observations

- In a segment tree, no matter what the query is, we will required to look at <= 2 * log2(n) numbers.
- So, operations required to answer each query = 2 * log2(10^6) = 40 , in the worst case.

  And as there are 10^5 queries, the total number of operations required in answering all queries = 10^5 X 40 = 4 X 10^6 (which is just fine)

- Segment trees consume less time..
- But is the output given by this algorithm correct??

- There is always a trade off between memory and time. We can either use less memory or less time. Not both.
- Segment tree uses less time in exchange for some more memory.
- Maximum number of nodes in a segment tree???
- Time required to build??
- Time for each query??

- In this example, the segment tree was constructed to find MAXIMUM of a range.
- The same can be done to find MINIMUM of a range.
- Q.- What about finding SUM of a range?

Can they be optimised using Segment Trees??

- SUM of a range queries can be done using Segment trees. But a much more efficient approach is possible that that.

Sum of a range query (alternate approach)

- We have the following array:

Array elements	7	9	3	2	6	15	10	17	2	4	3	1	11	1	0	5
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Query: 1 3
Result: 14

Query: 7 9
Result: 23

Query: 0 15
Result: 96

Query: 9 9
Result: 4

CUTTING EDGE VISIONARIES

Array 1:

Array elements	7	9	3	2	6	15	10	17	2	4	3	1	11	1	0	5
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

We construct this new **Array 2:**

Array elements	7	16	19	21	27	42	52	69	71	75	78	79	90	91	91	96
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Array2[i]=summation(j=0 to j=i) (array1[j])

- Now to answer each query we need only one step:

  For each query: l r , the answer is

  array2[r]-array2[l-1]

  Eg. For the query 4 8, the answer is

  array2[8]-array2[3]=71-21=50

  For the query 6 15, the answer is

  array[15]-array[5]=96-42=54

- So each query is answered in just 1 operation as opposed to log2(n) operations in case of segment tree.
- This is known as dynamic programming.

- What if some array element needs to be changed??

- In case of the old Naïve approach, if we want to change any array element, it can be done in only one operation.
- But, in segment trees, changing of an array element required log2(n) steps again.
- Even though updation is slower in Segment trees, still Query answering is so fast that it doesn’t matter that updation is a little slow.

- Visit www.codeaccepted.wordpress.com for more algorithms that are used in competitive programming.