Segment Trees

November 4, 2023

Hiten Goel (2022MCB1380) ,
Saharsh Saxena (2022CSB1116)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Neeraj Dwivedi

Summary: Our project basically aims for counting the number of clicks made on a different number of servers within a stipulated time and to have their count between specific intervals of time. Since this is a type of range query with dynamic point updates we have implemented this using a segment tree. For not complicating the process we are compiling the results for an hour on 3-4 servers. The updates have been implemented using re-entry of file till the user wants.

1. Introduction

Given an array arr[n]
In range query type problems we basically have 2 types of queries:
1: update arr[i]=x
2: find sum from arr[l] to arr[r]
There can be many solutions to it the famous ones being the normal array and the second being the prefix array.

1.1. Why segment trees

Before understanding why segment trees are used, we can look at the short comings of the earlier two data structures.
in simple array the update takes a time of O(1) by assigning arr[i]=x, but the range query takes O(n) time by
for(all i in range l to r)
sum+ =arr[i]
Prefix array is a data structure s.t: pre[i] contains the sum of all elements arr from 1 to i
pre[i] =pre[i-1] +arr[i]
This provides O(1) complexity for range query as sum=pre[r]-pre[l].But the point update now becomes O(n) as the update has to be made throughout the prefix array
for(all j in range i to n)
pre[j]=pre[j]-arr[i]+x This is where the segment trees become useful.they do both the queries in o(logn) time with O(nlogn) pre-computation.

2. Working of segment trees

Segment trees are easy to implement through arrays. It becomes complicated using linked list or pointers.
They are basically implemented by binary trees. The array elements are leaf nodes and for tree[i], the children are 2i and 2i+1 and the parent of tree[i] is (i-1)/2
Construction:
We start with a segment arr[0 . . . n-1]. and every time we divide the current segment into two (if it has not yet become a segment of length 1), and then call the same procedure on both halves, and for each such segment, we store the sum in the corresponding node.

All levels of the constructed segment tree will be completely filled except the last level. Also, the tree will be a Full Binary Tree because we always divide segment in two, at every level. Since the constructed tree is always a full binary tree with n leaves, there will be n-1 internal nodes. So the total number of nodes will be 2∗n − 1.
Height:
Height of the segment tree will be logN. Since the tree is represented using array and relation between parent and child indexes must be maintained, size of memory allocated for segment tree will be (2 ∗2log2n − 1). Here is the approach to construct and operate on segment tree
1. the leaf nodes are stored at i+n, so we can clearly insert all leaf nodes directly.

2. The next step is to build the tree and it takes O(n) time. The parent always has its less index than its children, so we just iterate over all the nodes in decreasing order, calculating the value of the parent node. It is equivalent to that inside the build function.
tree[i]=tree[2*i]+tree[2*i+1]
3. Updating a value at any position is also simple and the time taken will be proportional to the height of the tree. We only update values in the parents of the given node which is being changed. So to get the parent, we just go up to the parent node, which is p/2 or p»1, for node p. p1 turns (2*i) to (2*i + 1) and vice versa to get the second child of p.

4. Computing the sum also works in O(log(n)) time. If we have to find sum of [l,r), then we just have to iterate over their children depending on the overlapping of ranges and hence it is of max order height.

The theoretical time complexities of both previous implementation and this implementation is the same, but practically, it is found to be much more efficient as there are no recursive calls. We simply iterate over the elements that we need. Also, this is very easy to implement.

Time Complexities:
Tree Construction: O(n)
Query in Range: O(Log n)
Updating an element: O(Log n).

3. Figures, Tables and Algorithms

Figures, Tables and Algorithms have to contain a Caption that describes their content, and have to be properly referred in the text.

3.1. Figures

```
\includegraphics[options]{filename.xxx}
```

Here xxx is the correct format, e.g. `.png`, `.jpg`, `.eps`, ...

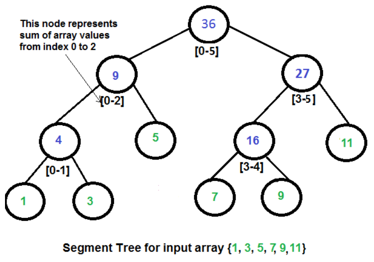
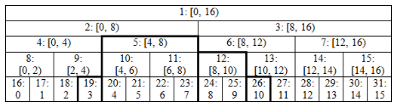

This node represents sum of array values from index 0 to 2
36
[0-5]
9
[0-2]
27
[3-5]
4
[0-1]
5
16
[3-4]
11
1
3
7
9
Segment Tree for input array {1, 3, 5, 7, 9, 11}

Figure 1: Formation of Segment Tree.



1: [0, 16)															
2: [0, 8)								3: [8, 16)							
4: [0, 4)				5: [4, 8)				6: [8, 12)				7: [12, 16)			
8: [0, 2)		9: [2, 4)		10: [4, 6)		11: [6, 8)		12: [8, 10)		13: [10, 12)		14: [12, 14)		15: [14, 16)	
16: 0	17: 1	18: 2	19: 3	20: 4	21: 5	22: 6	23: 7	24: 8	25: 9	26: 10	27: 11	28: 12	29: 13	30: 14	31: 15

Figure 2: Array implementation of segment tree.

3.2. Algorithms

Pseudo-algorithms can be written in LaTeX with the `algorithm` and `algorithmic` packages. An example is shown in Algorithm 3.

Algorithm 1 Building Segment Tree

```
1: for $i = 0; i < n; i++$ do
2:     tree[n+i]=arr[i]
3: end for
4: for $i = n - 1; i > 0; i--$ do
5:     tree[i] = tree[i<<1] + tree[i<<1 | 1]
6: end for
```

Algorithm 2 Updation in Segment Tree. arguments p,val

```
1: tree[p+n] = value;
   p = p+n;
2: for $(i = p; i > 1; i >>= 1)$ do
3:     $tree[i >> 1] = tree[i] + tree[i^1]$
4: end for
5: Final instructions
```

Algorithm 3 Sum in Segment Tree. arguments l,r

```
1: res = 0
2: for $(l+ = n, r+ = n; l < r; l >>= 1, r >>= 1)$ do
3:     if $l\%2 == 1$ then
4:         res + = tree[l++]
5:     end if
6:     if ( then
           $(r\%2 == 1))$ res + = tree[−−r]
7: 8:     end if
9: end for
10: return res
```

4. Problem Task

We calculate the no. of clicks made from a particular server within a time span of 1 hour and the information of the no. of clicks within a given range of time We first take the input using the file and then ask the user if he further wants to input or wants to find the sum in a given range.

5. Conclusions

We were able to get the range sum and update query by segment trees in most optimised time of O(logn).We saw that how it is the best form to implement this as we saw the other data structures like prefix array We implemented this to find the no. of clicks on no. of servers.

6. Bibliography

1. https://en.wikipedia.org/
2. https://cp-algorithms.com/
3. https://en.algorithmica.org/
4. https://planetmath.org/

Acknowledgements

We would like to thank our Course Instructor (Dr Anil Shukla) and Mentor (Neeraj Dwivedi) for providing us with such a significant opportunity to work on this project. We believe that we will indulge in more such projects in the future. We guarantee that this project was created entirely by us and is not a forgery. Finally, We'd like to express our gratitude to our parents and friends for their excellent comments and guidance during the completion of this project.