My experience

Monday, December 24, 2012

Segment Trees and lazy propagation

In this topic i will explain a very interesting data structure that can be used to solve a specific set of problems. I will start by explaining its definition and the proceeding with an example problem to solve with it.

Table of contents:

What is segment trees?

Segment Trees is a Tree data structure for storing intervals, or segments, It allows querying which of the stored segments contain a given point. It is, in principle, a static structure; that is, its content cannot be modified once the structure is built. It only uses O(N lg(N)) storage.

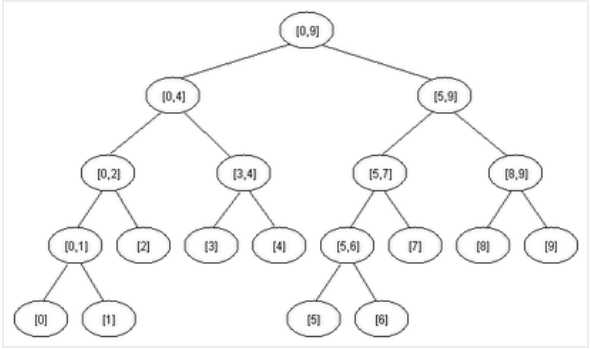
A segment trees has only three operations: build_tree, update_tree, query_tree.

Building tree: To init the tree segments or intervals values
Update tree: To update value of an interval or segment
Query tree: To retrieve the value of an interval or segment

Example Segment Tree:

- The first node will hold the information for the interval **[i, j]**
- If i<j the left and right son will hold the information for the intervals **[i, (i+j)/2]** and **[(i+j)/2+1, j]**

Notice that the height of a segment tree for an interval with **N** elements is **[logN] + 1**. Here is how a segment tree for the interval **[0, 9]** would look like:



Order of growth of segment trees operations

- **build_tree:** O(N lg(N))
- **update_tree:** O(lg(N + k))
- **query_tree:** O(lg(N + k))

K = Number of retrieved intervals or segments

Show me your code

```
1 /**
2  * In this code we have a very large array called arr, and very large set of operations
3  * Operation #1: Increment the elements within range [i, j] with value val
```

```
 * Operation #2: Get max element within range [i, j]
 * Build tree: build_tree(1, 0, N-1)
 * Update tree: update_tree(1, 0, N-1, i, j, value)
 * Query tree: query_tree(1, 0, N-1, i, j)
 */

#include<iostream>
#include<algorithm>
using namespace std;

#include<string.h>
#include<math.h>

#define N 20
#define MAX (1+(1<<6)) // Why? :D
#define inf 0x7fffffff

int arr[N];
int tree[MAX];

/**
 * Build and init tree
 */
void build_tree(int node, int a, int b) {
    if(a > b) return; // Out of range

        if(a == b) { // Leaf node
                tree[node] = arr[a]; // Init value
                return;
        }

        build_tree(node*2, a, (a+b)/2); // Init left child
        build_tree(node*2+1, 1+(a+b)/2, b); // Init right child

        tree[node] = max(tree[node*2], tree[node*2+1]); // Init root value
}

/**
 * Increment elements within range [i, j] with value value
 */
void update_tree(int node, int a, int b, int i, int j, int value) {

        if(a > b || a > j || b < i) // Current segment is not within range [i, j]
                return;

        if(a == b) { // Leaf node
                tree[node] += value;
                return;
        }

        update_tree(node*2, a, (a+b)/2, i, j, value); // Updating left child
        update_tree(1+node*2, 1+(a+b)/2, b, i, j, value); // Updating right child

        tree[node] = max(tree[node*2], tree[node*2+1]); // Updating root with max value
}

/**
 * Query tree to get max element value within range [i, j]
 */
int query_tree(int node, int a, int b, int i, int j) {

        if(a > b || a > j || b < i) return -inf; // Out of range

        if(a >= i && b <= j) // Current segment is totally within range [i, j]
                return tree[node];

        int q1 = query_tree(node*2, a, (a+b)/2, i, j); // Query left child
        int q2 = query_tree(1+node*2, 1+(a+b)/2, b, i, j); // Query right child

        int res = max(q1, q2); // Return final result

        return res;
}

int main() {
        for(int i = 0; i < N; i++) arr[i] = 1;
```

```
80
81        build_tree(1, 0, N-1);
82
83        update_tree(1, 0, N-1, 0, 6, 5); // Increment range [0, 6] by 5
84        update_tree(1, 0, N-1, 7, 10, 12); // Incremenet range [7, 10] by 12
85        update_tree(1, 0, N-1, 10, N-1, 100); // Increment range [10, N-1] by 100
86
87        cout << query_tree(1, 0, N-1, 0, N-1) << endl; // Get max element in range [0, N-1]
88}
```

This Gist brought to you by GitHub. segment_tree.cpp view raw

Lazy Propagation

Sometimes a segment tree operation wouldn't survive if the problem constraints is too large, here it come lazy propagation along with the segment tree.

In the current version when we update a range, we branch its childs even if the segment is covered within range. In the lazy version we only mark its child that it needs to be updated and update it when needed.

```
   /**
    * In this code we have a very large array called arr, and very large set of operations
1 2 * Operation #1: Increment the elements within range [i, j] with value val
3 4 * Operation #2: Get max element within range [i, j]
5 6 * Build tree: build_tree(1, 0, N-1)
7 8 * Update tree: update_tree(1, 0, N-1, i, j, value)
9   * Query tree: query_tree(1, 0, N-1, i, j)
10  */
11
12 #include<iostream>
13 #include<algorithm>
14 using namespace std;
15
16 #include<string.h>
17 #include<math.h>
18
19 #define N 20
20 #define MAX (1+(1<<6)) // Why? :D
21 #define inf 0x7fffffff
22
23 int arr[N];
24 int tree[MAX];
25 int lazy[MAX];
26
27 /**
28  * Build and init tree
29  */
30 void build_tree(int node, int a, int b) {
31         if(a > b) return; // Out of range
32
33         if(a == b) { // Leaf node
34                 tree[node] = arr[a]; // Init value
35                 return;
36         }
37
38         build_tree(node*2, a, (a+b)/2); // Init left child
39         build_tree(node*2+1, 1+(a+b)/2, b); // Init right child
40
41         tree[node] = max(tree[node*2], tree[node*2+1]); // Init root value
42 }
43
44 /**
45  * Increment elements within range [i, j] with value value
46  */
47 void update_tree(int node, int a, int b, int i, int j, int value) {
48
49         if(lazy[node] != 0) { // This node needs to be updated
50                 tree[node] += lazy[node]; // Update it
51
52                 if(a != b) {
53                         lazy[node*2] += lazy[node]; // Mark child as lazy
54                         lazy[node*2+1] += lazy[node]; // Mark child as lazy
55                 }
56
```

```
57                 lazy[node] = 0; // Reset it
58         }
59
60         if(a > b || a > j || b < i) // Current segment is not within range [i, j]
61                 return;
62
63         if(a >= i && b <= j) { // Segment is fully within range
64                 tree[node] += value;
65
66                 if(a != b) { // Not leaf node
67                         lazy[node*2] += value;
68                         lazy[node*2+1] += value;
69                 }
70
71                 return;
72         }
73
74         update_tree(node*2, a, (a+b)/2, i, j, value); // Updating left child
75         update_tree(1+node*2, 1+(a+b)/2, b, i, j, value); // Updating right child
76
77         tree[node] = max(tree[node*2], tree[node*2+1]); // Updating root with max value
78 }
79
80 /**
81  * Query tree to get max element value within range [i, j]
82  */
83 int query_tree(int node, int a, int b, int i, int j) {
84
85         if(a > b || a > j || b < i) return -inf; // Out of range
86
87         if(lazy[node] != 0) { // This node needs to be updated
88                 tree[node] += lazy[node]; // Update it
89
90                 if(a != b) {
91                         lazy[node*2] += lazy[node]; // Mark child as lazy
92                         lazy[node*2+1] += lazy[node]; // Mark child as lazy
93                 }
94
95                 lazy[node] = 0; // Reset it
96         }
97
98         if(a >= i && b <= j) // Current segment is totally within range [i, j]
99                 return tree[node];
100
101        int q1 = query_tree(node*2, a, (a+b)/2, i, j); // Query left child
102        int q2 = query_tree(1+node*2, 1+(a+b)/2, b, i, j); // Query right child
103
104        int res = max(q1, q2); // Return final result
105
106        return res;
107}
108
109int main() {
110        for(int i = 0; i < N; i++) arr[i] = 1;
111
112        build_tree(1, 0, N-1);
113
114        memset(lazy, 0, sizeof lazy);
115
116        update_tree(1, 0, N-1, 0, 6, 5); // Increment range [0, 6] by 5
117        update_tree(1, 0, N-1, 7, 10, 12); // Incremenet range [7, 10] by 12
118        update_tree(1, 0, N-1, 10, N-1, 100); // Increment range [10, N-1] by 100
119
           cout << query_tree(1, 0, N-1, 0, N-1) << endl; // Get max element in range [0, N-1]
   }
```

This Gist brought to you by GitHub. lazy_segment_tree.cpp view raw

Sample Problems to try

- Quadrant Queries
- D-Query

References

- Wiki
- Topcoder tutorials

Posted by Hussein El-Sayed at 4:58 AM

Recommend this on Google

11 comments:

donnghi December 24, 2012 at 12:25 PM

In lazy's version, i thinks it's better if you replace update tree[2*node] and tree[2*node+1] in 49th, 50th, 80th and 81th line by lazy[2*node] and lazy[2*node+1].

Its reason is your query is not really come down to higher level, so lazy[] should be updated

Reply

Hussein El-Sayed December 24, 2012 at 10:44 PM

I can't understand you :)

Reply

donnghi December 25, 2012 at 3:03 AM

So, in line 80th:
tree[node*2] += lazy[node]; // Mark child as lazy
tree[node*2+1] += lazy[node]; // Mark child as lazy

=> replaced by:
lazy[node*2] += lazy[node];
lazy[node*2+1] += lazy[node];

It's correct ?

Reply

Hussein El-Sayed December 25, 2012 at 3:29 AM

Yes you are totally right :).. thanks for correcting me ;)..

Reply

Hussein El-Sayed December 25, 2012 at 3:30 AM

The same at line 49 and 50, updated check it now and tell me :)

Reply

Sandipan Manna December 31, 2012 at 12:13 PM

update_tree(1, 0, N-1, 0, 6, 5); // Increment range [0, 6] by 5
update_tree(1, 0, N-1, 7, 10, 12); // Incremenet range [7, 10] by 12
update_tree(1, 0, N-1, 10, N-1, 100); // Increment range [10, N-1] by 100

So, The maximum element in the whole array should be 101
and your final array becomes
[6 6 6 6 6 6 6 13 13 13 13 101 101 101 101 101 101 101 101]

but your program gives output as 117 !!!

Reply

Hussein El-Sayed January 1, 2013 at 3:06 AM

No it should be 113, however the size of the array needs to be (1+(1<<6)) as it should be 2^(1+lg N)..

Also there was some checks needed to be added in the lazy version.. please check it and get back to me.

Reply

Sandipan Manna January 11, 2013 at 11:19 PM

Yes your segment tree size should be
int x = (int)(ceil(log2(N)))+1;
size = (1 << x);
This one!

Reply

InfiniteComplexity February 16, 2013 at 3:27 AM

Wow! Thanks for this post, it was very helpful! However, I'm trying to implement another update_tree_val function that sets the values from a range to one value. e.g. update_tree_val(3,7,4) would set the range [3,7] to the value of 4. How can I do this using lazy propagation on your tree?

Thanks

Reply

Replies

Hussein El-Sayed February 16, 2013 at 11:33 PM

It would be the same but without incrementing..

Reply

PRASHANTHSOUNDAR February 19, 2013 at 9:37 AM

hi, as a beginner to seg tree and lazy propagation i found this post very useful . i tried solving http://www.spoj.com/problems/HORRIBLE/
using slightly modified but same method as this but i am getting WA. can u help?..
http://ideone.com/3rBgSC

Reply

Enter your comment...

Comment as: Google Account

Publish Preview

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)