DarthPrince's blog

Algorithm Gym :: Everything About Segment Trees

By DarthPrince, 3 years ago,

In the last lecture of **Algorithm Gym** (Data Structures), I introduced you Segment trees.

In this lecture, I want to tell you more about its usages and we will solve some serious problems together. Segment tree types :

Classic Segment Tree

Classic, is the way I call it. This type of segment tree, is the most simple and common type. In this kind of segment trees, for each node, we should keep some simple elements, like integers or boolians or etc.

This kind of problems don't have update queries on intervals.

Example 1 (Online):

Problem 380C - Sereja and Brackets :

For each node (for example x), we keep three integers : 1. `t[x]` = Answer for it's interval. 2. `o[x]` = The number of $($s after deleting the brackets who belong to the correct bracket sequence in this interval whit length `t[x]` . 3. `c[x]` = The number of $)$s after deleting the brackets who belong to the correct bracket sequence in this interval whit length `t[x]` .

Lemma : For merging to nodes $2x$ and $2x+1$ (children of node $2x+1$) all we need to do is this :

```
tmp = min(o[2 * x], c[2 * x + 1])
t[x] = t[2 * x] + t[2 * x + 1] + tmp
o[x] = o[2 * x] + o[2 * x + 1] - tmp
c[x] = c[2 * x] + c[2 * x + 1] - tmp
```

So, as you know, first of all we need a $build$ function which would be this : (as above) (C++ and $[l, r)$ is inclusive-outclusive)

```
void build(int id = 1,int l = 0,int r = n){
    if(r - l < 2){
        if(s[l] == '(')
            o[id] = 1;
        else
            c[id] = 1;
        return ;
    }
    int mid = (l+r)/2;
    build(2 * id,l,mid);
```

```
    build(2 * id + 1,mid,r);
    int tmp = min(o[2 * id],c[2 * id + 1]);
    t[id] = t[2 * id] + t[2 * id + 1] + tmp;
    o[id] = o[2 * id] + o[2 * id + 1] - tmp;
    c[id] = c[2 * id] + c[2 * id + 1] - tmp;
}
```

For queries, return value of the function should be 3 values : t, o, c which is the values I said above for the intersection of the node's interval and the query's interval (we consider query's interval is $[x, y)$), so in C++ code, return value is a `pair<int,pair<int,int> >` (`pair<t, pair<o,c> >`) :

```
typedef pair<int,int>pii;
typedef pair<int,pii>   node;
node segment(int x,int y,int id = 1,int l = 0,int r = n){
    if(l >= y || x >= r)   return node(0,pii(0,0));
    if(x <= l && r <= y)
        return node(t[id],pii(o[id],c[id]));
    int mid = (l+r)/2;
    node a = segment(x,y,2 * id,l,mid), b = segment(x,y,2 * id + 1,mid,r);
    int T, temp, O, C;
    temp = min(a.y.x , b.y.y);
    T = a.x + b.x + temp;
    O = a.y.x + b.y.x - temp;
        C = a.y.y + b.y.y - temp;
    return node(T,pii(O,C));
}
```

Example 2 (Offline): Problem KQUERY

Imagine we have an array $b_1, b_2, ..., b_n$ which, $b_i \in 0, 1$ and $b_i = 1$ if an only if $a_i > k$, then we can easily answer the query (i, j, k) in $O(log(n))$ using a simple segment tree (answer is $b_i + b_{i+1} + ... + b_j$).

We can do this ! We can answer the queries offline.

First of all, read all the queries and save them somewhere, then sort them in increasing order of k and also the array a in increasing order (compute the permutation $p_1, p_2, ..., p_n$ where $a_{p_1} \le a_{p_2} \le ... \le a_{p_n}$)

At first we'll set all array b to 1 and we will set all of them to 0 one by one.

Consider after sorting the queries in increasing order of their k, we have a permutation $w_1, w_2, ..., w_q$ (of $1, 2, ..., q$) where $k_{w_1} \le k_{w_2} \le k_{w_2} \le ... \le k_{w_q}$ (we keep the answer to the i - th query in ans_i .

Pseudo code : (all 0-based)

```
po = 0
for j = 0 to q-1
        while po < n and a[p[po]] <= k[w[j]]
                b[p[po]] = 0, po = po + 1
```

So, build function would be like this ($s[x]$ is the sum of b in the interval of node x) :

```
void build(int id = 1,int l = 0,int r = n){
        if(r - l < 2){
                s[id] = 1;
                return ;
```

9	Zlobober	141
9	adamant	141

View all →

→ Find user

Handle:

Find

→ Recent actions

Tommyr7 → Codeforces Round #462

praran26 → ICM Technex 2018 and Codeforces Round #463 (Div. 1 + Div. 2, combined)

harniver → ioIT — Open Contests 2017/18 — FINAL ROUND announce

SuperJava → Comparison between Cin&Cout and Scanf&Printf and C++ versions

mplaza → 10th edition of the programming marathon Deadline24

Opportunity → Is CF predictor (heroku app) down?

pllk → Baltic Olympiad in Informatics contest collection

AnasAbbas → DP Workaround

Tommyr7 → Codeforces Round #462 Editorial

sidhant → Tutorial on FFT/NTT — The tough made simple. (Part 2)

nerd → set index

Tomzik → Facebook interview

gKseni → Design competition for Codeforces T-shirts

ale64bit → Help with a problem of optimal partitioning

csacademy → Round #69 (Div. 2)

SevenDeadlySins → Mysterious_Present_4_D

Igorjan94 → C++17, competitive programming edition

madmaxhasan → I can't Hack Problem

Nickolas → Upcoming Marathon Match 98

GlebsHP → Yandex.Algorithm 2018

duckladydinh → Machine Learning Help! Hackerrank GS CodeSprint 2018 — Easy Problem!

masterbios → what is the error in it

ameyanator → Help needed to solve SPOJ question ADACYCLE. (Getting TLE)

noobied → Help!!

Luqman → Teams going to ACM ICPC World Finals 2018

Detailed →

```
        }
        int mid = (l+r)/2;
        build(2 * id, l, mid);
        build(2 * id + 1, mid, r);
        s[id] = s[2 * id] + s[2 * id + 1];
}
```

et An update function for when we want to st `b[p[po]] = 0` to update the segment tree:

```
void update(int p,int id = 1,int l = 0,int r = n){
        if(r - l < 2){
                s[id] = 0;
                return ;
        }
        int mid = (l+r)/2;
        if(p < mid)
                update(p, 2 * id, l, mid);
        else
                update(p, 2 * id + 1, mid, r);
        s[id] = s[2 * id] + s[2 * id + 1];
}
```

Finally, a function for sum of an interval

```
int sum(int x,int y,int id = 1,int l = 0,int r = n){// [x, y)
        if(x >= r or l >= y)    return 0;// [x, y) intersection [l,r) = empty
        if(x <= l && r <= y)    // [l,r) is a subset of [x,y)
                return s[id];
        int mid = (l + r)/2;
        return sum(x, y, id * 2, l, mid) +
               sum(x, y, id*2+1, mid, r) ;
}
```

So, in main function instead of that pseudo code, we will use this :

```
build();
int po = 0;
for(int y = 0;y < q;++ y){
        int x = w[y];
        while(po < n && a[p[po]] <= k[x])
                update(p[po ++]);
        ans[x] = sum(i[x], j[x] + 1); // the interval [i[x], j[x] + 1)
}
```

Lazy Propagation

I told you enough about lazy propagation in the last lecture. In this lecture, I want to solve ans example .

Example : Problem POSTERS.

We don't need all elements in the interval $[1, 10^7]$. The only thing we need is the set $s_1, s_2, ..., s_k$ where for each i, s_i is at least l or r in one of the queries.

We can use interval $1, 2, ..., k$ instead of that (each query is running in this interval, in code, we use 0-based, I mean $[0, k)$). For the i - th query, we will paint all the interval $[l, r]$ whit color i (1-based).

For each interval, if all it's interval is from the same color, I will keep that color for it and update the nodes using lazy propagation.

So,we will have a value $lazy$ for each node and there is no any build function (if $lazy[i] \neq 0$ then all the interval of node i is from the same color (color $lazy[i]$) and we haven't yet shifted the updates to its children. Every member of $lazy$ is 0 at first).

A function for shifting the updates to a node, to its children using lazy propagation :

```
void shift(int id){
        if(lazy[id])
                lazy[2 * is] = lazy[2 * id + 1] = lazy[id];
        lazy[id] = 0;
}
```

Update (paint) function (for queries) :

```
void upd(int x,int y,int color, int id = 0,int l = 0,int r = n){//painting the
interval [x,y) whith color "color"
        if(x >= r or l >= y)    return ;
        if(x <= l && r <= y){
                lazy[id] = color;
                return ;
        }
        int mid = (l+r)/2;
        shift(id);
        upd(x, y, color, 2 * id, l, mid);
        upd(x, y, color, 2*id+1, mid, r);
}
```

So, for each query you should call $upd(x, y+1, i)$ (i is the query's 1-base index) where $s_x = l$ and $s_y = r$.

At last, for counting the number of different colors (posters), we run the code below (it's obvious that it's correct) :

```
set <int> se;
void cnt(int id = 1,int l = 0,int r = n){
        if(lazy[id]){
                se.insert(lazy[id]);
                return ; // there is no need to see the children, because all
the interval is from the same color
        }
        if(r - l < 2)   return ;
        int mid = (l+r)/2;
        cnt(2 * id, l, mid);
        cnt(2*id+1, mid, r);
}
```

And answer will be `se.size()` .

Segment tree with vectors

In this type of segment tree, for each node we have a vector (we may also have some other variables beside this) .

Example : Online approach for problem KQUERYO (I added this problem as the online version of KQUERY):

It will be nice if for each node, with interval $[l, r)$ such that $i \le l \le r \le j + 1$ and this interval is maximal (it's parent's interval is not in the interval $[i, j + 1)$), we can count the answer.

For that propose, we can keep all elements of $a_l, a_{l+1}, ..., a_r$ in increasing order and use binary search for counting. So, memory will be $O(n.log(n))$ (each element is in $O(log(n))$ nodes). We keep this sorted elements in verctor $v[i]$ for i - th node. Also, we don't need to run sort on all node's vectors, for node i, we can merge $v[2 * i]$ and $v[2 * id + 1]$ (like merge sort) .

So, build function is like below :

```
void build(int id = 1,int l = 0,int r = n){
        if(r - l < 2){
                v[id].push_back(a[l]);
                return ;
        }
        int mid = (l+r)/2;
        build(2 * id, l, mid);
        build(2*id+1, mid, r);
        merge(v[2 * id].begin(), v[2 * id].end(), v[2 * id + 1].begin(), v[2 *
id + 1].end(), back_inserter(v[id])); // read more about back_inserter in
http://www.cplusplus.com/reference/iterator/back_inserter/
}
```

And function for solving queries :

```
int cnt(int x,int y,int k,int id = 1,int l = 0,int r  = n){// solve the query
(x,y-1,k)
        if(x >= r or l >= y)    return 0;
        if(x <= l && r <= n)
                return v[id].size() - (upper_bound(v[id].begin(), v[id].end(),
k) - v[id].begin());
        int mid = (l+r)/2;
        return cnt(x, y, k, 2 * id, l, mid) +
                   cnt(x, y, k, 2*id+1, mid, r) ;
}
```

Another example : Component Tree

Segment tree with sets

In this type of segment tree, for each node we have a `set` or `multiset` or `hash_map` (here) or `unorderd_map` or etc (we may also have some other variables beside this) .

Consider this problem :

We have n vectors, $a_1, a_2, ..., a_n$ and all of them are initially empty. We should perform m queries on this vectors of two types :

1. $A\ p\ k$ Add number $k$at the end of a_p
2. $C\ l\ r\ k$ print the number $\sum_{i=l}^{r} count(a_i, k)$ where $count(a_i, k)$ is the number of occurrences of k in a_i .

For this problem, we use a segment tree where each node has a `multiset` , node i with interval $[l, r)$ has a `multiset` $s[i]$ that contains each number k exactly $\sum_{i=l}^{r} count(a_i, k)$ times (memory would be $O(q.log(n))$) .

For answer query $C\ x\ y\ k$, we will print the sum of all $s_x.count(k)$ where if the interval of node x is $[l, r)$, $x \le l \le r \le y + 1$ and its maximal (its parent doesn't fulfill this condition) .

We have no build function (because vectors are initially empty). But we need an add function :

```
void add(int p,int k,int id = 1,int l = 0,int r = n){// perform query A p k
        s[id].insert(k);
        if(r - l < 2)   return ;
        int mid = (l+r)/2;
        if(p < mid)
                add(p, k, id * 2, l, mid);
        else
                add(p, k, id*2+1, mid, r);
}
```

And the function for the second query is :

```
int ask(int x,int y,int k,int id = 1,int l = 0,int r = n){// Answer query C x
y-1 k
        if(x >= r or l >= y)    return 0;
        if(x <= l && r <= y)
                return s[id].count(k);
        int mid = (l+r)/2;
        return ask(x, y, k, 2 * id, l, mid) +
                   ask(x, y, k, 2*id+1, mid, r) ;
}
```

Segment tree with other data structures in each node

From now, all the other types of segments, are like the types above.

$2D$ Segment trees

In this type of segment tree, for each node we have another segment tree (we may also have some other variables beside this) .

Segment trees with tries

In this type of segment tree, for each node we have a trie (we may also have some other variables beside this) .

Segment trees with DSU

In this type of segment tree, for each node we have a disjoint set (we may also have some other variables beside this) .

Example : Problem 76A - Gift, you can read my source code (8613428) with this type of segment trees .

Segment trees with Fenwick

In this type of segment tree, for each node we have a Fenwick (we may also have some other variables beside this) . Example :

Consider this problem :

We have n vectors, $a_1, a_2, ..., a_n$ and all of them are initially empty. We should perform m queries on this vectors of two types :

1. $A\ p\ k$ Add number $k$at the end of a_p
2. $C\ l\ r\ k$ print the number $\sum_{i=l}^{r} count(a_i, j)$ for each $j \leq k$ where $count(a_i, k)$ is the number of occurrences of k in a_i .

For this problem, we use a segment tree where each node has a `vector` , node i with interval $[l, r)$ has a `set` $v[i]$ that contains each number k if and only if $\sum_{i=l}^{r} count(a_i, k) \neq 0$ (memory would be $O(q.log(n))$) (in increasing order).

First of all, we will read all queries, store them and for each query of type A, we will insert k in v for all nodes that contain p (and after all of them, we sort these vectors using merge sort and run unique function to delete repeated elements) .

Then, for each node i, we build a vector $fen[i]$ with size $|s[i]|$ (initially 0).

Insert function :

```
void insert(int p,int k,int id = 1,int l = 0,int r = n){//      perform query A
p k
        if(r - l < 2){
                v[id].push_back(k);
                return ;
        }
        int mid = (l+r)/2;
        if(p < mid)
                insert(p, k, id * 2, l, mid);
        else
                insert(p, k, id*2+1, mid, r);
}
```

Sort function (after reading all queries) :

```
void SORT(int id = 1,int l = 0,int r = n){
        if(r - l < 2)
                return ;
        int mid = (l+r)/2;
        SORT(2 * id, l, mid);
        SORT(2*id+1, mid, r);
        merge(v[2 * id].begin(), v[2 * id].end(), v[2 * id + 1].begin(), v[2 *
id + 1].end(), back_inserter(v[id])); // read more about back_inserter in
http://www.cplusplus.com/reference/iterator/back_inserter/
        v[id].resize(unique(v[id].begin(), v[id].end()) - v[id].begin());
        fen[id] = vector<int> (v[id].size() + 1, 0);
}
```

Then for all queries of type A, for each node x containing p we will run :

```
for(int i = a + 1;i < fen[x].size(); i += i & -i)       fen[x][i] ++;
```

Where $v[x][a] = k$. Code :

```
void upd(int p,int k, int id = 1,int l = 0,int r = n){
        int a = lower_bound(v[id].begin(), v[id].end(), k) - v[id].begin();
        for(int i = a + 1; i < fen[id].size(); i += i & -i )
                fen[id][i] ++ ;
        if(r - l < 2)   return;
        int mid = (l+r)/2;
        if(p < mid)
                upd(p, k, 2 * id, l, mid);
```

```
        else
                upd(p, k, 2*id+1, mid, r);
}
```

And now we can easily compute the answer for queries of type C :

```
int ask(int x,int y,int k,int id = 1,int l = 0,int r = n){// Answer query C x
y-1 k
        if(x >= r or l >= y)    return 0;
        if(x <= l && r <= y){
                int a = lower_bound(v[id].begin(), v[id].end(), k) -
v[id].begin();
                int ans = 0;
                for(int i = a + 1; i > 0; i -= i & -i)
                        ans += fen[id][i];
                return ans;
        }
        int mid = (l+r)/2;
        return ask(x, y, k, 2 * id, l, mid) +
                   ask(x, y, k, 2*id+1, mid, r) ;
}
```

Segment tree on a rooted tree

As you know, segment tree is for problems with array. So, obviously we should convert the rooted tree into an array. You know DFS algorithm and starting time (the time when we go into a vertex, starting from 1). So, if s_v is starting time of v, element number s_v (in the segment tree) belongs to the vertex number v and if $f_v = max(s_u) + 1$ where u is in subtree of v, the interval $[s_v, f_v)$ shows the interval of subtree of v (in the segment tree) .

Example : Problem 396C - On Changing Tree

Consider h_v height if vertex v (distance from root).

For each query of first of type, if u is in subtree of v, its value increasing by $x + (h_u - h_v) \times -k = x + k(h_v - h_u) = x + k \times h_v - k \times h_u$. So for each u, if s is the set of all queries of first type which u is in the subtree of their v, answer to query $2\ u$ is $\sum_{i \in s}(k_i \times h_{v_i} + x_i) - h_u \times \sum_{i \in s} k_i$, so we should calculate two values $\sum_{i \in s}(k_i \times h_{v_i} + x_i)$ and $\sum_{i \in s} k_i$, we can answer the queries. So, we for each query, we can store values in all members of its subtree ($[s_v, f_v)$).

So for each node of segment tree, we will have two variables $hkx = \sum x + h \times k$ and $sk = \sum k$ (we don't need lazy propagation, because we only update maximal nodes).

Source code of update function :

```
void update(int x,int k,int v,int id = 1,int l = 0,int r = n){
        if(s[v] >= r or l >= f[v])      return ;
        if(s[v] <= l && r <= f[v]){
                hkx[id] = (hkx[id] + x) % mod;
                int a = (1LL * h[v] * k) % mod;
                hkx[id] = (hkx[id] + a) % mod;
                sk[id] = (sk[id] + k) % mod;
                return ;
        }
        int mid = (l+r)/2;
        update(x, k, v, 2 * id, l, mid);
        update(x, k, v, 2*id+1, mid, r);
}
```

Function for 2nd type query :

```
int ask(int v,int id = 1,int l = 0,int r = n){
        int a = (1LL * h[v] * sk[id]) % mod;
        int ans = (hkx[id] + mod - a) % mod;
        if(r - l < 2)   return ans;
        int mid = (l+r)/2;
        if(s[v] < mid)
                return (ans + ask(v, 2 * id, l, mid)) % mod;
                return (ans + ask(v, 2*id+1, mid, r)) % mod;
}
```

Persistent Segment Trees

In the last lecture, I talked about this type of segment trees, now I just want to solve an important example.

Example : Problem MKTHNUM

First approach : $O((n+m).log^2(n))$

I won't discuss this approach, it's using binary search an will get TLE.

Second approach : $O((n+m).log(n))$

This approach is really important and pretty and too useful :

Sort elements of a to compute permutation $p_1, p_2, ..., p_n$ such that $a_{p_1} \leq a_{p_2} \leq ... \leq a_{p_n}$ and $q_1, q_2, ..., q_n$ where, for each i, $p_{q_i} = i$.

We have an array $b_1, b_2, ..., b_n$ (initially 0) and a persistent segment tree on it.

Then n step,for each i, starting from 1, we perform $b_{q_i} = 1$.

Lest $sum(l, r, k)$ be $b_l + b_{l+1} + ... + b_r$ after k - th update (if $k = 0$, it equals to 0)

As I said in the last lecture, we have an array $root$ and the root of the empty segment tree, ir . So for each query $Q(x, y, k)$, we need to find the first i such that $sum(1, i, r)$ - $sum(1, i, l - 1) > k - 1$ and answer will be a_{p_i}. (I'll explain how in the source code) :

Build function (s is the sum of the node's interval):

```
void build(int id = ir,int l = 0,int r = n){
        s[id] = 0;
        if(r - l < 2)
                return ;
        int mid = (l+r)/2;
        L[id] = NEXT_FREE_INDEX ++;
        R[id] = NEXT_FREE_INDEX ++;
        build(L[id], l, mid);
        build(R[id], mid, r);
        s[id] = s[L[id]] + s[R[id]];
}
```

Update function :

```
int upd(int p, int v,int id,int l = 0,int r = n){
        int ID =  NEXT_FREE_INDEX ++; // index of the node in new version of
```

```
segment tree
        s[ID] = s[id] + 1;
        if(r - l < 2)
                return ID;
        int mid = (l+r)/2;
        L[ID] = L[id], R[ID] = R[id]; // in case of not updating the interval
of left child or right child
        if(p < mid)
                L[ID] = upd(p, v, L[ID], l, mid);
        else
                R[ID] = upd(p, v, R[ID], mid, r);
        return ID;
}
```

Ask function (it returns i, so you should print a_{p_i} :

```
int ask(int id, int ID, int k, int l = 0,int r = n){// id is the index of the
node after l-1-th update (or ir) and ID will be its index after r-th update
        if(r - l < 2)   return l;
        int mid = (l+r)/2;
        if(s[L[ID]] - s[L[id]] >= k)// answer is in the left child's interval
                return ask(L[id], L[ID], k, l, mid);
        else
                return ask(R[id], R[ID], k - (s[L[ID]] - s[L[id]] ), mid, r);//
there are already s[L[ID]] - s[L[id]] 1s in the left child's interval
}
```

As you can see, this problem is too tricky.

If there is any error or suggestion let me know.

segment tree, algorithms, tutorial

+322 DarthPrince 3 years ago 63

Comments (63)

Write comment?

1e9

3 years ago, # | +18

Another nice blog. Thanks. And if possible write a blog on Graph + DP.

→ Reply

sheri.mori

3 years ago, # ^ | +2

YES,I would love it to learn DP from **PrinceOfPersia**'s blog

→ Reply

Duxar

3 years ago, # | ← Rev. 3 +11

Why there is no section only for algorithms and data structures on CF? I keep 2-3 tabs open all the time to avoid losing posts like this one.

Keep up the good work!

→ Reply

3 years ago, # ^ | +9

Nice idea. Meanwhile until the idea is implemented, you can click on

pranshul

the star at the end of the post so that it is added to your favorite blogs and you can always get back to it in future. You can also mark you favorite users(concept of friends) , problems, and solutions.

→ Reply

Duxar

3 years ago, # ^ | ☆ ← Rev. 2 0

Yes, favorite! Nice feature, thanks (Also it was a memento to read them if I want to close the tab, now I will keep only one tab open just for that). I forgot about it. Now, how do I solve the problem of finding older posts about algorithms and data structures?

→ Reply

JoudZouzou

3 years ago, # ^ | ☆ +1

here are some

→ Reply

MatRush

3 years ago, # | ☆ 0

Asnwer -> Answer :-)

→ Reply

DarthPrince

3 years ago, # ^ | ☆ 0

Thanks, fixed.

→ Reply

aangairbender

3 years ago, # ^ | ☆ 0

why downvote? He's an author))

→ Reply

ddt

3 years ago, # | ☆ +5

Thanks a lot for the wonderful article :)

→ Reply

hiukim

3 years ago, # | ☆ +6

A very good summary on segment tree! Thanks.

→ Reply

Queen_sacrifice

3 years ago, # | ☆ ← Rev. 2 0

prince of persia did your online for kquery get AC?

→ Reply

DarthPrince

3 years ago, # ^ | ☆ ← Rev. 2 -16

I didn't submit it. It was just an example of segment tree with vector .

And by the way, prince, not price.

→ Reply

3 years ago, # ^ | ☆ +8

typo fixed. :)

→ Reply

→ Reply

Queen_sacrifice

P_Nyagolov

3 years ago, # ^ | ☆ 0

This was my first idea when I was solving the problem and it didn't get AC(I don't remember it was because of TLE or MLE).

→ Reply

Queen_sacrifice

3 years ago, # ^ | ☆ 0

i couldn't get AC with online. i dont know if any online submission would pass.

→ Reply

DarthPrince

3 years ago, # ^ | ☆ ← Rev. 2 -16

Use `getchar_unlocked` or `buffer` for reading inputs and `printf` for printing the output.

→ Reply

Queen_sacrifice

3 years ago, # ^ | ☆ 0

tried fread(). also, instead of vector, used static arrays. still TLE

→ Reply

mruxim

3 years ago, # | ☆ +37

it looks like $O((n+m).log^3(n))$ solution for problem MKTHNUM can fit in time limit! code :)

→ Reply

DarthPrince

3 years ago, # ^ | ☆ -31

→ Reply

The comment is hidden because of too negative feedback, click here to view it

Junior94

3 years ago, # ^ | ☆ +29

Why not both?

→ Reply

DarthPrince

3 years ago, # ^ | ☆ -39

→ Reply

The comment is hidden because of too negative feedback, click here to view it

halfo

3 years ago, # ^ | ☆ +19

The best algorithm is the one that works (i.e. gets AC) and easier to think of and to write.

→ Reply

3 years ago, # ^ | ☆ ← Rev. 2 0

PrinceOfPersia i'm so pissed watching

nashir

PrinceOfPersia I'm so pissed watching your comment getting down voted -_-

people can be so ungrateful :/

→ Reply

300pound

3 years ago, # | ☆ ← Rev. 2 0

Thanks again to PrinceOfPersia for this great tutorial

Just a tip,when I was solving 380 Problem C with guidance from the code above, I got snagged at the querying input format.

To get the value of a specific cell 0, the function to call is segment(0,1); Hence since the input adheres to 1-based indexing, segment(l-1,r) will do it.

→ Reply

burakcetin

3 years ago, # | ☆ +3

Nice tutorial!! It will take some time reading all of it though B).

→ Reply

grayhathacker

3 years ago, # | ☆ -8

this tutorial extracted the fear of segment tree out of me... thanx :)

→ Reply

shubham_466

3 years ago, # | ☆ -8

Great tutorial!! **PrinceOfPersia** can u write a blog on BIT? That would be a lot of help!!

→ Reply

surjection

3 years ago, # | ☆ 0

Great tutorial! Thanks for your effort.

Did you submit the solution you described here for SPOJ POSTERS? I ask because I am not convinced that a solution using segment trees and co-ordinate compression is possible. The only implementations I have found that try to do this fail on the following test case:

```
1
3
3 7
1 3
7 10
```

The expected output is `3` but segment tree/coordinate compression solutions give `2`.

Can you point me to an implementation using segment trees and co-ordinate compression that can deal with this case? My own attempt is here.

→ Reply

nashir

3 years ago, # ^ | ☆ ← Rev. 6 0

compression can be done for this problem.but you have to be implement it a bit differently. author missed to point it out.

while compressing the highest relative difference has to be 2 or greater, not 1 like general compression.

otherwise failure will occur like you've stated above.

So while compressing you'll have to include the numbers before and

So while compressing you'll have to include the numbers before and after of every 'special' numbers (l or r for every queries)

Then it will work just fine.

here is my implementation. :)

→ Reply


sleepify

8 months ago, # ^ | ☆ ← Rev. 2 0

There is a bug in the author's code for POSTERS. Change id = 0 to id = 1 in the upd function. Here's my implementation. Code

→ Reply


mowji

3 years ago, # | ☆ 0

Very grateful keep on doing your job!

→ Reply


anajin

3 years ago, # | ☆ 0

great job! Can you give me more problems solved by offline SegmentTree/BIT ? Thanks in advance.

→ Reply


terribleimposter

3 years ago, # | ☆ +8

According to C++ reference, multiset::count is linear in number of matches. Hence in your subsection — Segment tree with sets — if say 1 is inserted into the first vector n times and the query is to count the number of occurences of 1 in all the vectors, then that query actually takes O(n) time. There can be n such queries resulting in O(n^2) overall running time. Is there a workaround to this?

→ Reply


kien_coi_1997

3 years ago, # | ☆ 0

It is possible to combine AVL tree vs Segment tree to have a new structure called Segment tree with insert and delete operators.

http://codeforces.com/blog/entry/12285

Just for fun! It is quite useless.

→ Reply


suraj021

3 years ago, # | ☆ 0

PrinceOfPersia this is a wonderful article on segment tree and thanks a lot for this . But seriously i am having hard time understanding the logic used in the solution of very first. :(Can you please EXPLAIN ?

→ Reply


suraj021

3 years ago, # ^ | ☆ 0

I got the logic . Thanks

→ Reply


sierra101

3 years ago, # | ☆ 0

how to scale values in a given range say [L,R] with a constant say C,by segment tree or BIT.

Thanks in advance :D

→ Reply

zxqfl

3 years ago, # ^ | ☆ +5

Values are integers → brute force, scaling more than 64 times would cause overflow anyway

Values are floating-point → $\log a + \log b = \log(ab)$

→ Reply

sierra101

3 years ago, # ^ | ☆ 0

could u provide a link to implementation of code and problems on same :D

→ Reply

zxqfl

3 years ago, # ^ | ☆ 0

I don't have those things; since I assume you're talking about the floating-point case, I'll go into a little more detail:

When you have an array A, instead of adding $A[i]$ to your BIT, add $\ln A[i]$. When you want to scale by C in the range $[l, r]$, instead add $\ln C$ in the range $[l, r]$. When you want to find the value of $A[i]$, the value is given by $e^{A[i]}$.

→ Reply

rumman_sust

3 years ago, # | ☆ 0

Such an helpful post. Thank you very much. If you have enough time please write about line sweeping with segment tree :)

→ Reply

1theweblover007

2 years ago, # | ☆ +4

```
void build(int id = 1,int l = 0,int r = n){
    if(r - l < 2){
        if(s[l] == '(')
            o[id] = 1;
        else
            c[id] = 1;
        return ;
    }
    int mid = (l+r)/2;
    build(2 * id,l,mid);
    build(2 * id + 1,mid,r);
    int tmp = min(o[2 * id],c[2 * id + 1]);
    t[id] = t[2 * id] + t[2 * id + 1] + tmp;
    o[id] = o[2 * id] + o[2 * id + 1] - tmp;
    c[id] = c[2 * id] + c[2 * id + 1] - tmp;
}
```

PrinceOfPersia Should'nt it be- build(2 * id + 1,mid+1,r); I think probably you have missed "mid+1" part.

→ Reply

numbertheorist17

2 years ago, # ^ | ☆ +23

Actually, it is correct as written (it should not be mid+1). The reason is the way the code is written is for the interval $[l, r)$, and he splits the interval into $[l, mid)$ and $[mid, r)$.

→ Reply

no0neHearsUsButTheSky

2 years ago, # | ☆ 0

Great editorial **AmirMohammad Dehghan** . Thanks a lot :)

→ Reply

rajanparmar94

2 years ago, # | ☆ ← Rev. 2 -13

nice editorial

→ Reply

sanket407

2 years ago, # | ☆ 0

getting WA for posterS ![user:amd] Used seg trees to store all posters from position 1 to max of queries.

Then Lazy propogation is used and after all poster queries did a last passing of lazy values till bottom.

Then after all poster queries counted values at bottom nodes (stored in wall[])

Can anyone come up with test cases where my code getting WA?

sol: http://pastebin.com/KYcUvpku

→ Reply

wts

22 months ago, # | ☆ 0

Thanks for the lecture. I was wondering what is the good approach if the input values are too large to fit in an array (e.g.the order 10^18). The updates are in ranges e.g. add x -> a,b The queries are also in ranges e.g. find sum -> c,d

→ Reply

samier_aldroubi

22 months ago, # ^ | ☆ 0

Compress the numbers :) .

→ Reply

adamant

22 months ago, # ^ | ☆ ← Rev. 4 0

```
const int maxn = 1e5, logn = 30, inf = 1e9;

int add[2 * maxn * logn], sum[2 * maxn * logn];
int L[2 * maxn * logn], R[2 * maxn * logn];
int sz = 2;

int create(int &v)
{
    if(!v)
        v = sz++;
    return v;
}

void upd(int a, int b, int c, int v = 1, int l = 0, int r =
inf)
{ // arr[a..b) += c
    if(a <= l && r <= b)
    {
        add[v] += c;
        sum[v] += (r - l) * c;
        return;
    }
    if(r <= a || b <= l)

        return;
```

```
        return;
    int m = (l + r) / 2;
    upd(a, b, c, create(L[v]), l, m);
    upd(a, b, c, create(R[v]), m, r);
    sum[v] = (r - l) * add[v] + sum[L[v]] + sum[R[v]];
}

int get(int a, int b, int v = 1, int l = 0, int r = inf)
{ // sum[a..b)
    if(a <= l && r <= b)
        return sum[v];
    if(r <= a || b <= l)
        return 0;
    int m = (l + r) / 2;
    return get(a, b, L[v], l, m) + get(a, b, R[v], m, r) +
           max(0, min(r, b) - max(l, a)) * add[v];
}
```

→ Reply

Vendetta.

21 month(s) ago, # | ☆ 0

Thanks a lot. That was really helpful.

→ Reply

Gamerrishad

21 month(s) ago, # | ☆ 0

Shouldn't it be "y"?? Under the section "Segment tree with vectors"??

And function for solving queries :

```
int cnt(int x,int y,int k,int id = 1,int l = 0,int r  = n){// solve the query (x,y-1,k)
        if(x >= r or l >= y)    return 0;
        if(x <= l && r <= n)
               return v[id].size() - (upper_bound(v[id].begin(), v[id].end(), k) - v[id].begin());
        int mid = (l+r)/2;
        return cnt(x, y, k, 2 * id, l, mid) +
                  cnt(x, y, k, 2*id+1, mid, r) ;
}
```

Another example : Component Tree

→ Reply

marcospqlixo

18 months ago, # | ☆ +1

awesome tutorial, thanks amd!

→ Reply

rakib_ruet_13

14 months ago, # | ☆ 0

A great blog on segment tree. But could not figure out Persistent Segment Tree very well :(.it seems for me, i Need more details.

→ Reply

Dpman

11 months ago, # ^ | ☆ 0

You can read it from here : https://blog.anudeep2011.com/persistent-segment-trees-explained-with-spoj-problems/

→ Reply

paulabhik47

10 months ago, # | ☆ 0

shouldn't in sereja and brackets 380 c it should be t[x] = t[2 * x] + t[2 * x + 1] + 2*tmp ?

→ Reply

9 months ago, # ^ | ☆ 0

It depends on the merging step. If you do t[x] = t[2 * x] + t[2 * x + 1] +

jcbbigcrane

2*tmp, then you don't need to print 2*answer. Otherwise you have to print 2*answer because of the merging of both the opening and closing brackets.

→ Reply

lsiddiqsunny

8 months ago, # | ☆ 0

How to get number of elements in a range [l,r] which are **greater than x** and **less then y** by **segment tree**?

→ Reply

KyleWalker

8 months ago, # ^ | ☆ ← Rev. 2 0

you can save numbers [l,r] sorted in each node this can be done O(n.lgn) with merge sort and use binary search to find how many numbers are greater than x and less then y in nodes . each query can be done in O(lg^2 (n)).

→ Reply

vertika2011

8 months ago, # | ☆ 0

Really nice tutorial.

→ Reply

ravikiran0606

2 months ago, # | ☆ +1

In "Segment tree with sets" section of the blog, if we use segment tree with each node as multiset, can we handle range updates along with range queries?

For example, In this problem,

We have n vectors, a1, a2, ..., an and all of them are initially empty. We should perform m queries on this vectors of two types :

1) A l r k Add number k to the elements of each a[i], l<=i<=r.

2) C l r k print the sum of the number of occurrences of k in each a[i], l<=i<=r.

How to solve this kind of problem?

→ Reply

SinByCos

2 months ago, # | ☆ 0

Great blog! In case someone had problems for the persistent segment tree (implementation or MKTHNUM problem) they can check this out: https://blog.anudeep2011.com/persistent-segment-trees-explained-with-spoj-problems/

→ Reply

gupta_rishabh

new, 5 weeks ago, # | ☆ ← Rev. 2 0

In the problem Sereja and Brackets , the segment tree approach is giving TLE on test case 13

http://codeforces.com/contest/380/problem/C

typedef pair<int,int> pii; typedef pair<int,pii> node;

void buildTree(int * ansArr , int *OArr , int *CArr , string arr , int index , int s , int e) { if(s>e){ return; }

if(s==e){

```
if(s==e){
    if(arr[s]=='('){
       OArr[index] = 1;
       CArr[index] = 0;
    }
    else if(arr[s]==')'){
       OArr[index] = 0;
       CArr[index] = 1;
    }
    ansArr[index] = 0;
    return;
}

int mid = (s+e)/2;
buildTree(ansArr,OArr,CArr,arr,2*index,s,mid);
buildTree(ansArr,OArr,CArr,arr,2*index+1,mid+1,e);
int tmp = min(OArr[2*index] , CArr[2*index+1]);
ansArr[index] = (ansArr[2*index]) + (ansArr[2*index+1]) + tmp;
OArr[index] = OArr[2*index]  +  OArr[2*index+1]- tmp;
CArr[index] = CArr[2*index] + CArr[2*index+1] - tmp;
return;

}

node query(int * ansArr , int *OArr , int * CArr , int index , int s , int e , int qs , int
qe){ No Overlap if(qs>e || qe<s){ return node(0,pii(0,0)); } Complete Overlap
if(s>=qs && e<=qe){ return node(ansArr[index] , pii(OArr[index] , CArr[index])); }

int mid = (s + e)/2;
node  nodeLeft = query(ansArr,OArr,CArr , 2*index , s , mid , qs ,
qe);
node  nodeRight = query(ansArr,OArr,CArr , 2*index+1,mid+1,e,qs,qe);

int tmp = min(nodeLeft.second.first , nodeRight.second.second);
int ans = nodeLeft.first + nodeRight.first + tmp;
int o = nodeLeft.second.first + nodeRight.second.first - tmp;
int c = nodeLeft.second.second + nodeRight.second.second - tmp;
return node(ans , pii(o,c));

}

int main(){ std::ios::sync_with_stdio(false); string str; cin>>str; int * ansArr = new
int[4*str.size()+1]; int * OArr = new int[4*str.size()+1]; int * CArr = new
int[4*str.size()+1]; int q; cin>>q; buildTree(ansArr,OArr,CArr ,str ,1 ,0,str.size()-1);

while(q--){
    int l,r;
    cin>>l>>r;
    node ansNode = query(ansArr , OArr,CArr , 1 , 0 , str.size()-1 ,
l-1 , r-1);
    cout<<(2*ansNode.first)<<endl;
}

}`
```

→ Reply