paladin8's blog

Segment Tree

By **paladin8**, 5 years ago,

This blog post is motivated by an interesting problem I solved today: Timus 1846. I knew about segment trees previously for solving range-minimum query, but was not fully aware of the obvious generalizations of this nice data structure. Here I will describe the basics of a segment tree and some applications.

Problem

The problem that a segment tree can solve is the following. We are given an array of values $a[0], a[1], ..., a[N - 1]$. Assume without loss of generality that $N = 2^n$; we can generally pad the computations accordingly. Also, consider some associative binary function f. Examples of f include sum, min, max, or gcd (as in the Timus problem). Segment trees allow for each of the following two operations on $O(logN)$ time:

- compute $f(a[i], a[i + 1], ..., a[j])$ for $i \le j$; and
- update $a[x] = v$.

Description

So how does a segment tree work? It's quite simple, in fact. We build a single two-dimensional array $t[0...N - 1][0..n]$ where $t[x][y] = f(a[x], a[x + 1], ..., a[x + 2^y - 1])$ where x is divisible by 2^y (i.e. most of the entries are ignored, but it is easiest to represent this way). We can initialize the entries by the following procedure:

- $t[x][0] = a[x]$; and
- $t[x][y] = f(t[x][y - 1], t[x + 2^{y-1}][y - 1])$ for $y = 1, ..., n$,

where the second step uses the associative property of f. The logic is that $t[x][y - 1]$ computes f over the range $[x, x + 2^{y-1})$ and $t[x + 2^{y-1}][y - 1]$ computes f over the range $[x + 2^{y-1}, x + 2^y)$ so when we combine them we get the corresponding computation of f for $t[x][y]$. Now we can describe how each of the two operations is implemented (at a high level).

We'll start with computing $f(a[i], a[i + 1], ..., a[j])$. It's pretty easy to see that this amounts to representing the interval $[i, j]$ as disjoint intervals $[i_1, j_1];[i_2, j_2];...;[i_k, j_k]$ where each of these is of the form $[x, x + 2^y - 1]$ where x is divisible by 2^y (i.e. it shows up in the table t somewhere). Then we can just combine them with f (again using the associative property) to get the result. It is easy to show that $k = O(logN)$ as well.

Now for updating $a[x] = v$, notice that there are only a few terms in the segment tree which get affected by this change. Obviously, $t[x][0] = v$. Also, for $y = 1, ..., n$, only the entry $t[x - (x\&(2^y - 1))][y]$ will update. This is because $x - (x\&(2^y - 1))$ is the only value divisible by 2^y (notice the last y bits are zero) that also covers x at level y. Then it suffices to use the same formula as in the initialization to recompute this entry in the tree.

Code

Here's a somewhat crude implementation of a segment tree. Note that the value of IDENTITY should be such that $f(IDENTITY, x) = x$, e.g. 0 for sum, $+\infty$ for min, $-\infty$ for max, and 0 for gcd.

```
void init() {
  for (int x = 0; x < N; x++)
    t[x][0] = a[x];
  for (int y = 1; y <= n; y++)
    for (int x = 0; x < N; x+=(1<<y))
      t[x][y] = f(t[x][y-1], t[x+(1<<(y-1))][y-1]);
}

void set(int x, int v) {
  t[x][0] = a[x] = v;
  for (int y = 1; y <= n; y++) {
    int xx = x-(x&((1<<y)-1));
    t[xx][y] = f(t[xx][y-1], t[xx+(1<<(y-1))][y-1]);
  }
}

int get(int i, int j) {
  int res = IDENTITY, h = 0; j++;
  while (i+(1<<h) <= j) {
    while ((i&((1<<(h+1))-1)) == 0 && i+(1<<(h+1)) <= j) h++;
    res = f(res, t[i][h]);
    i += (1<<h);
  }
  while (i < j) {
    while (i+(1<<h) > j) h--;
    res = f(res, t[i][h]);
    i += (1<<h);
  }
  return res;
}
```

Application

One possible application is letting f just be the sum of all arguments. This leads to a data structure that allows you to compute sums of ranges and do updates, but is slightly less efficient than a Binary Indexed Tree. The most well-known application seems to be the range-minimum query problem, where f is the minimum of all arguments. This also allows you to compute the least-common ancestor of nodes in a tree.

Lastly, in Timus 1846, we can use it with f as the GCD function. Every time we remove a number we should update the corresponding entry to zero (effectively removing it from the GCD computation). Then the answer at each step is $f(a[0], a[1], ..., a[k])$ which can be computed directly from the segment tree.

algorithm, **tutorial**, **segment tree**, **timus**

+22 paladin8 5 years ago 33

Comments (33)

Write comment?

niyaznigmatul

5 years ago, # | +14

Actually, it's known that you need only $O(N)$ memory to store Segment Tree with N leaves. As you see, your array has many elements, that are not beeing updated or used at all. You can store segment tree in one-dimensional array.
Root is being number 1. Two children of node v are $2v$ and $2v + 1$.

9	csacademy	145
10	amd	142

View all →

→ Find user

Handle:

Find

→ Recent actions

vepifanov → Intel Code Challenge Elimination Round (div.1 + div.2 combined)

P1kachu → Codeforces Round #374 (Div. 2)

wrick → Bug in CodeForces in Intel Code Challenge Elimination Round

ErzhanDS → guess who's back

rng_58 → AtCoder Grand Contest 005

ifsmirnov → DSU with randomized linking is as fast as with rank heuristic

ahmed_aly → More new and hopefully helpful ladders on A2OJ

ahmed_aly → A2 Online Judge is back!

rivudas → Invitation to October Easy 2016 on Hackerearth

XuMuk → Codeforces Round #373 — Editorial

isat1729 → Inclusion - Exclusion problem from Light OJ

MikeMirzayanov → 2016-2017 CT S03E04: Codeforces Trainings Season 3 Episode 4

fadi_yns1 → Need help!!

KADR → Codeforces Beta Round #13 editorial

adurysk → ICPC Preparatory Series — Final Round — Contest by Team Akatsuki

dusht1408 → Float Mod ,Not Working?

r3gz3n → Invitation to HackerEarth Collegiate Cup

r3gz3n → HackerEarth Collegiate Cup First Elimination Results

sahilarora.535 → Getting Runtime error in Round 374 Div2 B.

gKseni → ACM-ICPC World Finals Reunion Dinner 2016

Endagorion → Codeforces Round #300 Editorial (+challenges)

I_love_Tanya_Romanova → Invitation to HackerEarth Collegiate Cup Wild Card Round

Tanmoy1228 → Help Needed for Graph + Segment Tree Problem

slow_coder4 → why WA help

So the parent of node v is $v / 2$. One can easily prove that it contains only $O(N)$ nodes.
→ Reply

5 years ago, # ^ | ☆ ← Rev. 2 0

paladin8

Yes, you are right. But I think it is easier to think about it as a 2-dimensional array and only use the 1-dimensional compression when you have to.

→ Reply

4 years ago, # ^ | ☆ 0


erreze

you actually need 2*2^(log(N)+1) for memory, don't you? It contains O(N) leaves, not nodes. But our colors suggest i'm wrong and you're right, can you share your prof?
→ Reply

4 years ago, # ^ | ☆ +3


AlexanderBolshakov

2 * 2 ^ (log(N) + 1) = 4 * N = O(N)
→ Reply

2 years ago, # ^ | ☆ -19


yashar_sb_sb

- O(N) is a set. And 4 * N isn't.
- So you can't say that they are equal. Instead you can say 4 * N is a member of O(N).

→ Reply

2 years ago, # ^ | ☆ 0


AlexanderBolshakov

I don't really say they are equal. Please read this.
→ Reply

5 years ago, # | ☆ +8


goo.gl_SsAhv

Look at this cool implementation of Range Sum Query (RSQ). This implementation needs 2 * N memory, there is some implementations with only N memory, but as for me, this implementation is very simple, and it can be easily extended with some non trivial actions, (for example, we can build an array with all indexes we need to update, and run for them a[i] = F(a[i * 2], a[i * 2 + 1]) in order)
→ Reply

5 years ago, # ^ | ☆ ← Rev. 3 0


goo.gl_SsAhv

BTW, it can be easily extended to 2D or 3D arrays, but in this case it takes (2 * n)^k memory, where k is 1, 2, 3 for 1d, 2d, 3d, ...

upd
→ Reply

5 years ago, # ^ | ☆ 0

paladin8

Right, and the operation cost becomes $O((logN)^k)$.
→ Reply

ahmed_aly → Please support A2 Online Judge to come back

Detailed →


saikrishna17394

4 years ago, # ^ | ☆ 0

@Alias I would like to know how to update and query in 2d segment tree??

→ Reply


goo.gl_SsAhv

4 years ago, # ^ | ☆ 0

there is two ways. look at implementation.

→ Reply


danhnguyen0902

3 years ago, # ^ | ☆ 0

Do you know how to implement Lazy Propagation on 2D Segment Tree? I'm just being curious. Ideas or codes would be really useful!

→ Reply


caustique

3 years ago, # ^ | ☆ +12

Who can help me with practice problems on 2d-, persistent and compressed segment tree? I appreciate links to OJ problems.

→ Reply


Manish-Kumar

5 years ago, # | ☆ 0

Thanks,I will take this code as a supplement in regionals. :)
Must be okay with you. :)

→ Reply


asif_iut

5 years ago, # | ☆ +3

Can you please explain the method for updating values over a range of indexes. I looked up in the internet and saw a solution called lazy propagation, but could not understand clearly. It would be great if you can explain a little. Thanks a lot ! .

→ Reply


Manish-Kumar

5 years ago, # ^ | ☆ 0

Lets start with my explanation, hope that somebody will give better explanation.

Lazy means "Update value only when you need it" .

That is you keep a flag at the "parent node" , and update the children only when you encounter them while querying.

→ Reply


Betlista

5 years ago, # | ☆ 0

great, thanks ;-)

→ Reply


megalodon

4 years ago, # | ☆ 0

Another tutorial about it http://prodeportiva.wordpress.com/2013/02/08/segment-trees/

→ Reply


Empty

3 years ago, # | ☆ 0

Could you please give me an idea of how to implement a 2d-segment tree?. For example, to solve the problem 11297 — Census. Thanks in advance.

→ Reply

3 years ago, # ^ | ☆ ← Rev. 3 +1

The idea behind 2D segment tree is that you first build segment tree for every row, then you build segment tree for every column. Its easier to operate with NxM array if N and M are degrees of 2, if not, you can fill the array with 0s, but if memory is more important, you can do without it. For example, you have the following 4x4 array:

1 2 3 4

4 3 2 1

5 6 7 8

8 7 6 5

Build a segment tree for every row. It will look like this:

10 3 7 1 2 3 4

10 7 3 4 3 2 1

26 11 15 5 6 7 8

26 15 11 8 7 6 5


sparik

(the first element of the row is the sum of elements (1...n), the second is the sum of (1...n/2), the third is the sum of (n/2+1...n) ...

Then build a segment tree for every column of the above matrix in the same way:

72 36 36 18 18 18 18

20 10 10 5 5 5 5

52 26 26 13 13 13 13

10 3 7 1 2 3 4

10 7 3 4 3 2 1

26 11 15 5 6 7 8

26 15 11 8 7 6 5

The tree contains 2N-1 rows and 2M-1 columns, and our initial matrix starts with element (N,M)

If you want to update element (x,y), you update it, then all its parents in its column ((x,y/2),(x,y/4),...(x,1)), then divide x by 2, and do the same thing while x>0.

For finding the sum of the rectangle with opposite vertices (R1,C1) (R2,C2) (where R1<R2 and C1<C2) recursively, start from the first

row(of the tree), If the range of the row R has no intersection with (R1...R2) then return 0, if the range is not completely into (R1...R2) but has an intersection, then call the function for its children ((R*2) and (R*2+1)). Finally, if the range is completely into (R1...R2) we do the same thing with the tree in the Rth row, but this time we return tree[R][C] if the range of the Cth element of the Rth row is completely into (C1...C2).

The complexity of queries is log(N)*log(M).
→ Reply


Empty

3 years ago, # ^ | ☆ 0

Very nice explanation, thank you.
→ Reply


danhnguyen0902

3 years ago, # ^ | ☆ ← Rev. 2 0

What if I want to update a rectangle, say (R1,C1) (R2,C2)? Is the complexity still log(N)*log(M)? Could we now use the Lazy Propagation technique here, like the one in the 1D Segment Tree?
→ Reply


faiyaz26

3 years ago, # ^ | ☆ +5

here is a tutorial about 2D segment tree. Link

You might need google translate if you are not Russian. :)
→ Reply


HAL9000

3 years ago, # | ☆ 0

Do you know about other problems where segment trees should be used with some other associative binary functions?
→ Reply


k790alex

3 years ago, # ^ | ☆ 0

You could solve these:

http://coj.uci.cu/24h/problem.xhtml?abb=2125

http://codeforces.com/problemset/problem/339/
→ Reply


HAL9000

3 years ago, # ^ | ☆ 0

Thanks for your reply. I already solved the first one long time ago, and as I remember it was not about using some associative operation, but storing the progression terms in the nodes with lazy propagation. As for the second one, the link points to nowhere.
→ Reply


k790alex

3 years ago, # ^ | ☆ 0

this one is more interesting: http://coj.uci.cu /24h/problem.xhtml?abb=2511

this one its only a sum: http://coj.uci.cu /24h/problem.xhtml?abb=1850

this is only a sum but it's in 2D: http://coj.uci.cu/24h/problem.xhtml?abb=1904

and for the Interval Product problem, you dont need lazy propagation, but you need it for this one: http://www.spoj.com/problems/HORRIBLE/
→ Reply


ymondelo20

4 months ago, # ^ | ☆ 0

Links to coj.uci.cu problems actually uses pid instead of abb.
→ Reply


k790alex

4 months ago, # ^ | ☆ 0

I wrote that long time ago, today the links are broken.
→ Reply

2 years ago, # | ☆ ← Rev. 18 +17

We can generalize that a bit more: Let (M, f) be a semigroup (that is, a set M together with an associative binary operator $f : M \times M \to M$).

Let (T, c) be a monoid with identity element t_{id} and $g_k : T \times M^k \to M^k$ a family of transformation functions (range updates), that take as their arguments a parameter $t \in T$ and a sequence $a \in M^k$ and output a sequence $g(t, a) \in M^k$ of the same length. g_k should respect the monoid structure of t, i.e. for $t_1, t_2 \in T$, $g_k(c(t_1, t_2), \cdot)$ should be equivalent to the composition $g_k(t_2, \cdot) \circ g_k(t_1, \cdot)$ and $g_k(t_{id}, \cdot)$ should be the identity function. Intuitively, this allows us to compress a series of updates into one single update.

Let $I = [1, n] \cap \mathbb{N}$ be the set of integers between 1 and n.

A segment tree is able to maintain a dynamic sequence $< a_1, \ldots, a_n > \in M^n$. It allows the following operations:


niklasb

- range query: Given two integers $l \leq r \in I$, return $f(a_l, ..., a_r)$ (the notation is just for convenience, the function f is in fact applied $r - l$ times).
- range update: Given two integers $l \leq r \in I$ and a parameter $t \in T$, replace the subsequence $< a_l, ..., a_r >$ by $g(t, < a_l, ..., a_r >)$.

It is able to do so in $\mathcal{O}(\log n)$ time per operation if the following conditions are met:

- f and c can be computed in $\mathcal{O}(1)$.
- There is a function $h : T \times M \to M$, that, given an update $t \in T$ and the value $f(a)$ for a sequence a, can compute $f(g_t(a))$ in $\mathcal{O}(1)$. Intuitively, we need to be able to compute the new aggregated value of an interval when we only know the old aggregated value and the update that happened since.

The implementation works by assigning nodes to ranges in a binary tree-like fashion and storing $f(a_l, ..., a_r)$ and a lazy update t inside the node responsible for the range $[l, r]$.

The tree can be implemented without any further knowledge about the internal structure. It just needs implementations of f, c, h and it needs to know t_{id}.

This characterization is general enough to directly allow a scenario like:

- allow range queries for min, max and sum of an interval
- allow range updates "set all to v" and "add v to each value"

Here the values in M are tuples $(min, max, sum, length)$ and the values in T are tuples $(action\_type, v)$. The functions f (combine range data), c (combine updates) and h (apply update to range data) look like this (in pseudo-Haskell pattern matching syntax):

```
f (min1, max1, sum1, length1) (min2, max2, sum2, length2)
=
       (min min1 min2, max max1, max2, sum1 + sum2,
length1 + length2)

c _ (Set, v) = (Set, v)
c (Add, x) (Add, y) = (Add, x + y)
c (Set, x) (Add, y) = (Set, x + y)

h (Set, v) (min, max, sum, length) = (v, v, v, length)
h (Add, v) (min, max, sum, length) = (min + v, max + v,
sum + v * length, length)
```

We also need to supply $t_{id} = (add, 0)$.

For problems like http://www.spoj.com/problems/SEGSQRSS/ or http://www.spoj.com/problems/GSS3/ we need to get a bit more creative and store additional values in the M tuples that are not actually needed to answer queries directly, but to implement the function h efficiently.

→ Reply

androidrxample

2 years ago, # | ☆ 0

Hi I have found one good example here Segment Tree — Data Structure

→ Reply

kaushal02

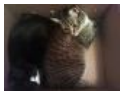
7 months ago, # | ☆ 0

Correct me if I'm wrong, but an `O(nlogn)` is getting TLE on the problem you stated initially. Here is the problem: Timus 1846

Unfortunately, I can't provide a link to my solution since the solutions are private. But I'd be very happy to hear a solution using Segment tree since any segment tree must use `O(nlogn)` . Correct me if i am wrong again!

Thanks in advance

→ Reply



winger

7 months ago, # ^ | ☆ +3

My solution is O(n*log n) using Segment tree and gets AC in 0.39 execution time (in Java).

→ Reply