Anudeep's blog

Persistent segment trees – Explained with spoj problems

13 July 2014

In this post I will introduce the concept of persistent data structures. We shall deal with a couple of problems : COT, MKTHNUM

Consider the following question :

Given a linked list, you need to support 2 operations. 1) Update the first x values. 2) Print the linked list after k'th update (k <= number of update operations made till now)

Simple solution is to create a new linked list after each update operation. print it when needed. Let us optimize it for memory.

We are only updating first x elements of the linked list. That means the new linked list will be almost same as previous linked list if x is small. We shall take advantage of this and avoid allocating memory for some elements. We can allocate memory for x elements, store new values in them, then point the next pointer of x'th node of new linked list to (x+1)'th node of previous linked list. This way we allocated only x units of memory per update operation.

Consider the following example.

Initial linked list : 1 2 3 4 5 6 7 8 9 10.
Update 1 : x = 4, new elements : 12 13 14 15
Update 2 : x = 2, new elements : 20 21
Update 3 : x = 6, new elements : 1 2 3 4 5 6
Update 4 : x = 1, new elements : 100

Initial linked list will look like this

```
head[0]-->[01]-->[02]-->[03]-->[04]-->[05]-->[06]-->[07]-->[08]-->[09]-
```

After first update total data will look like this

```
head[0]-->[01]-->[02]-->[03]-->[04]-->[05]-->[06]-->[07]-->[08]-->[09]-
                                        /
head[1]-->[12]-->[13]-->[14]-->[15]--'
```

We allocated memory for first 4 nodes, stored values in them, then next pointer of 4th node is pointing to 5th node of 1st linked list.
Now if we want to print first linked list, we start with head[0]. If we want to print 2nd linked list we start with head[1].
Data will look like this after all updates

```
 head[0]-->[01]-->[02]-->[03]-->[04]-->[05]-->[06]-->[07]-->[08]-->[09]
                                       /              |
 head[1]-->[12]-->[13]-->[14]-->[15]---'               |
                         /                           /
 head[2]-->[20]-->[21]--'                           /
                                                   /
 head[3]-->[01]-->[02]-->[03]-->[04]-->[05]-->[06]--'
                  /
```

```
    head[4]-->[100]--'
```

We are done with it. Memory used is O(n+x1+x2..+xm), note that also the time complexity is reduced from O(n*n) to O(n+x1+x2+..+xm). This is useful when sum of x is in O(n), in that case total complexity will be O(n) (memory and time).

What we have build above is a Persistent data structure. I am done with introducing you to Persistent data structures, now I will explain Persistent segment trees with this SPOJ problem : COT

10628. Count on a tree – SPOJ – COT – Editorial and solution implementation

You are given a tree with N nodes.The tree nodes are numbered from 1 to N.Each node has an integer weight.
We will ask you to perform the following operation:

> **u v k** : ask for the **k**th minimum weight on the path from node **u** to node **v**

Note : Time limits are very strict. My O(N * log N) solution passed only with fast IO.
Our goal is to come up with O(N * log N) solution.

We can binary search for the answer. Now for a fixed value **x**, we need to answer the number of nodes with value less than **x** on the path from node **u** to node **v**. Let the number of such nodes be **y**. If **y** is less than **k**, we have to consider values greater than **x**. In other case consider values less than **x**.

Now the question is, given **u v x**, we need to answer the number of nodes with value less than **x** on the path from **u** to **v**.
Let f(**u**, **x**) be the number of nodes less than **x** on the path from **root** to **u**.
We can formulate the required answer as follows :
f(**u**, **x**) + f(**v**, **x**) – f(lca(**u**, **v**), **x**) – f(parent(lca(**u**, **v**)), **x**)
where lca(**u**, **v**) is Lowest common ancestor of **u** and **v**. It can be calculated in O(log N) (here and here)

Now the problem is further reduced, we just need to calculate function f. If the function f works in O(log N) time, then we have an O(log N * log N) solution per query. We shall see how function f can work in O(log N)

Let us use coordinate compression on values (here). Then we have all the values in the range [0, N).
Now assume that for each node of the tree, we have a segment tree build and ready to be used. segment tree is build in the following way : For a node **u**, all the nodes from **root** to **u** are considered, their respective compressed values are used to build the segment tree such that it will answer number of elements less than some **x** in O(log N). This is exactly what the function f has to calculate. Hence if we have N segment trees, one per node, we can solve the problem in O(log N * log N) per query. How to build

such segment tree is left out. It is a standard problem.

Now the only issue left in this solution is building N segment trees, which may take O(N * N) space and time. Let us use the concept which I introduced above to reduce the memory. Note that the segment tree for a node **u** is made with compressed values of nodes from **root** to **u**. Now for any child **c** of **u**, its segment tree of **c** is build with same values of node **u**, except that one new value is added, that is the value of current node **c**. Consider the addition of this new value like an update operation on segment tree of **u**. Then the update operation will change O(log N) nodes from last level to the 1st level. There you go, the new segment tree of node **c** is almost same as segment tree of **u**, except for O(log N) nodes. Hence we can only allocate memory for this new O(log N) nodes and reuse other nodes from old segment tree. We are allocating memory for O(log N) nodes per each node in the tree. Hence the total memory used will be O(N * log N). Note that as the memory is reduced from O(N * N) to O(N * log N), time complexity is also reduced to O(N * log N).

We are done with the O(N * log N + M * log N * log N) solution. We construct the segment trees in O(N * log N), then answer each query in O(log N * log N) (binary search + segment tree query). As I told you the time limits for the problem are very strict, and this solution will not pass. We need to modify it to O((N+M) * log N) to get it accepted.

Do we really need binary search? Can we merge it with the segment tree query? Yes. Here is how.
Start with the segment trees at four nodes (see the formula we derived to understand which four nodes).

Now check the number of elements in left sub tree, if it is greater than **k**, then our answer is present in left sub tree. If not it is present in right sub tree. This way we can travel the 4 segment tress at once and get the answer in O(log N). Don't worry if you did not understand this last step, looking at the code will make it clear.

Click here for code

(You will get TLE if you submit it on SPOJ, adding fast input output will give AC, I removed that module to keep the code small and clean)

3946. K-th Number – MKTHNUM – Editorial and solution implementation

Given an array a[1 … N] of different integer numbers, your program must answer a series of questions Q(i, j, k) in the form: "What would be the k-th number in a[i … j] segment, if this segment was sorted?

There are solutions which work in O((N + M) * log N * log N) or O((N + M * log N) * log N * log N), those solutions will give AC. I will describe O((N+M) * log N) solution.

Let us assume we have a segment tree for all N * N ranges. That is for each i, j such that 1 <= i <= j <= N we have a segment tree ready to be used. In this case we can answer the query in O(log N) which is what we need to do. But building those segment trees will take O(N*N*N) time and memory. Like in previous problem let me concentrate on reducing memory which will in turn reduce time complexity.

Trick 1 : In segment tree for range (i, j) each node can be calculated from respective nodes in segment tree for range (1, i-1) and range (1, j). That is for each node in segment tree for range (i, j) : node for (i, j) = node for (1, j) – node for (1, i-1). Awesome. We only need O(N) segment trees now. One segment tree for each of the prefixes. (This trick works only because all input values are distinct, see why it wont work in other case)

Trick 2 : Segment tree for prefix i is almost same as segment tree for prefix i-1, except some O(log N) nodes that will change. So once again we can reduce the memory to O(N * log N).

Great, we are done with it. From O(N*N*N) memory we reduced to O(N * log N). Time complexity will be O(N * log N). There you go, O((N+M) * log N) solution.

Click here for code

Related Problems

Codechef – QUERY – http://www.codechef.com/problems/QUERY – Editorial
Codechef – SORTING – http://www.codechef.com/problems/SORTING – Editorial
Codeforces – http://codeforces.com/problemset/problem/226/E – Editorial

Comments were not working due to a bug. It is fine now. Also the contact me form was not working and so I lost about 30-40 messages i received in last 2-3 months. Sorry for the inconvenience. You can now click on contact me and mail me.

Do comment problems related to persistent segment trees, I will add them.

Also suggest me a light weight, clean and simple theme.

Share this post. Learn and let learn! 

← Heavy Light Decomposition

When 2 guys talk, its not always about girls/sports ;) →

Abdo • 4 months ago

Hi anudeep

How we can make lazy propagation with persistent segment tree ?

  • Reply • Share ›

John Wu • 4 months ago

Thanks for the great post!

One question, how do you deal with interval update? We could use lazy flag in trivial segment tree, which to delay interval operations to the future. When accessing lazy-flagged node, we simply pass it down to c nodes.

In persistent segment tree, passing down a lazy flag could change the result of previous versions. When eventually in the worst case we need to create 2 * (j – i + 1) new nodes (basically a new segment tree for interval) to prevent damage in previous versions. That is not memory efficient.

Any idea?

  • Reply • Share ›

Dipjal Chhetri • 5 months ago

@anudeep2011 What exactly is 'count' storing here?

  • Reply • Share ›

Aatmaram • 5 months ago

Thanks for Sharing.

• Reply • Share ›

Xyleth • 7 months ago

I dont undertand how do you build your segment tree and how it works. Do yo have some resource to lea that specific segment tree?

Reply • Share ›

sunil • a year ago

@Anudeep Can you please explain MKTHNUM with example how code is working...

Reply • Share ›

abhishekbind2013 • a year ago

First of all thanks for such a great explanation. I wanted to another problem which can be done using Per segment tree.
https://www.hackerrank.com/con...

Reply • Share ›

phqb • a year ago

Thank you for this great post. :D
There is one point I haven't gotten yet. Can you describe more the segment tree you use to find the numb with value less than x on the path from root to u?

Reply • Share ›

ANKIT KUMAR • a year ago

Can u elaborate the mkthnum lil more
i know only basic segment tree and lazy propagation

Reply • Share ›

parijat • 2 years ago

https://upload.wikimedia.org/w...

^ | v • Reply • Share ›

Rodolfo Miquilarena • 2 years ago

hello , i have a question, i have been trying to solve this problem

http://www.spoj.com/problems/T...

i think its this method (which is amazing) however i don't know how to do a proper update on the tree bec how it is built, do you have any idea on how to do updates on the persistent segment tree DS? :) i bet you in advance.

^ | v • Reply • Share ›

jugal kishor sahu • 2 years ago

In your code

```
struct node
{
int count;
node *left, *right;
node(int count, node *left, node *right):
count(count), left(left), right(right) {}
node* insert(int l, int r, int w);
--------
};
```

how this insert works....can you provide some link so that i can get more...

Reply • Share ›

shaheen13 • 2 years ago

For problem "COT",
you don't need to use faster IO.
just avoid "map".
instead of this , use binary search for data compression .
i just edited your , and got AC.
http://paste.ubuntu.com/100318...

Reply • Share ›

Petar • 2 years ago

I can't undertand what do you mean by having segment tree for every interval,
how would that segment tree look like , can you please elaborate on that?

Reply • Share ›

Aditya • 2 years ago

A very powerful and detailed explanation. Thanks so much! Are there any other problems on SPOJ that r
persistent segment trees so that I can practice the logic myself?

Reply • Share ›

Rahul Kumar • 2 years ago

i has basic understanding of segment tree.my doubt is,
we have to take array size of 2^n for make segment tree of n element array,then how to make segment tr
greater elements,because 2^30 = 1073741824

then how to take array of this larger size for make segment tree,how to implement ?

Reply • Share ›

anudeep2011 ➚ Rahul Kumar • 2 years ago

You will not need 2^n space.
Number of nodes in last level = N (Assume N is of the form 2^m for simplicity)
Number of nodes in one level above = N/2
..
Number of nodes in 2nd level = 2
Number of nodes in 1st level = 1

Total nodes = N + N/2 + N/4 + ... N/(N/2) + N/N
= N (1 + 1/2 + 1/4 + 1/8 + ...)
= 2 * N (Why 2*N?)

Reply • Share ›

Rahul Kumar ➚ anudeep2011 • 2 years ago

ok,now it is clear,thanks :)

Reply • Share ›

niloy • 2 years ago

what to do when we have to update the values, such that, changing value of a index? then how to handle

Reply • Share ›

anudeep2011 ➚ niloy • 2 years ago

It would still be the same. You will have a new node (with new values) in the last level of segment

Reply • Share ›

Dharmendra • 2 years ago

i dont know how is this working
f(u, x) + f(v, x) – f(lca(u, v), x) – f(parent(lca(u, v)), x)
what will happen if we do this
f(u, x) + f(v, x) – 2*f(lca(u, v), x)

Reply • Share ›

anudeep2011 ➚ Dharmendra • 2 years ago

f(u, x) + f(v, x) – 2*f(lca(u, v), x) - If this is done then the "lca(u, v)" node is not counted at all. It is twice in initial 2 functions and again removed twice.

Draw a simple tree. Consider random u, v nodes and check what happens for f(u) + f(v) - 2 * f(lc Where f(u) means traveling from root to u and putting a dot on the node. -f(u) means traveling fror and erasing a dot on the node.

At the end of your formula there should be exactly one dot on each node from u to v path.

Reply • Share ›

sundar • 2 years ago

can you please explain this line "This trick works only because all input values are distinct"?I don't unders doesn't work if input values aren't distinct.

Reply • Share ›

anudeep2011 ➚ sundar • 2 years ago

Let us say we have 2 occurrences of 7. One at index 1 and another at index 10. Now our query is so we take it like [1..12] - [1..2], we are actually removing the 7 but it is still present at index 10.

We can modify the logic, instead of just storing presence of number, we can store count of numb should work then.

 • Reply • Share ›

Nitin • 2 years ago

Hey anudeep, can MKTHNUM be solved using fractional cascading and merge sort trees?
This is what I've tried-
1. Created a merge sort tree M[LOGN][N] to keep a track of sorted values in different levels. Level 0 conta complete sorted array.
2. For any query [i,j] got disjoint intervals(at max logN number of them).
3. Binary search for X for which number of elements < X in all lists = k using fractional cascading.(log(maximumNumber-minimumNumber)*log(N))

Doubts-
What will be the complexity of creating a merged list out of these k sorted lists. Will it be k.log(N) ?
Secondly, the numbers can be upto 10^9 . I am not able to understand how will a binary search on [minin maximumNumber] will work in time.

 • Reply • Share ›

anudeep2011 ➚ Nitin • 2 years ago

1) I have no idea about fractional cascading and merge sort trees.

Merging K sorted lists is not k * log(N) because for 2 lists of length N/2. You will have 2 * log(N) v true.

Simple upper bound is N * log(N), Proof? Just join those K lists and sort them.

Yes the numbers are up to 10^9, one method is you can use coordinate-compression.

  • Reply • Share ›

ma5termind • 2 years ago

here is one more problem which can be solved using persistent segment tree.
http://www.codechef.com/proble...
Thanx for such a great help

  • Reply • Share ›

avinashmailAvinash • 2 years ago

could u explain this line "f(u, x) + f(v, x) – f(lca(u, v), x) – f(parent(lca(u, v)), x) ". Why r u subtracting f(lca(u, v)), x" ??

  • Reply • Share ›

anudeep2011 avinashmailAvinash • 2 years ago

Take some tree image (Google for it). Now f(u, x) means root to u.
Now terms f(u, x) and f(v, x) are additions, so consider rounding each node on the path from root root to v.
terms f(lca(u, v), x) and f(parent(lca(u, v)), x) are deletions, so consider erasing one rounding node on the path from root to lca(u,v) and root to parent(lca(u,v)). You will notice that you will be rounds on path from u to v, which is exactly what you need.

Try this with different u, v values if needed.

  • Reply • Share ›

yogi ➚ anudeep2011 • 2 years ago

Hi anudeep

I tried the above but i feel it should be f(u, x) + f(v, x) – 2 * f(lca(u, v), x) . Could you pleas this.

^ | v • Reply • Share ›

anudeep2011 ➚ yogi • 2 years ago

f(u, x) + f(v, x) – 2*f(lca(u, v), x) - If this is done then the "lca(u, v)" node is not cc It is added twice in initial 2 functions and again removed twice.

Draw a simple tree. Consider random u, v nodes and check what happens for f(u) lca(u, v)). Where f(u) means traveling from root to u and putting a dot on the node. means traveling from root to u and erasing a dot on the node.

At the end of your formula there should be exactly one dot on each node from u to

^ | v • Reply • Share ›

Sreenivasan AC • 2 years ago

Nice explanation!

^ | v • Reply • Share ›

Subscribe to our Newsletter

Email*

SUBSCRIBE!

Categories

Algorithms (2)

Data Structures (1)

Machine Learning (1)

Segment trees (2)

SPOJ (1)

Uncategorized (2)

Recent Posts

20 by 25
September 26, 2016

Machine learning everywhere, why not in Competitive programming?
April 18, 2016

MO’s Algorithm (Query square root decomposition)
December 28, 2014

When 2 guys talk, its not always about girls/sports ;)
July 18, 2014

Persistent segment trees – Explained with spoj problems
July 13, 2014

Heavy Light Decomposition
April 11, 2014

Recent Comments

Vineeth Kanaparthi on 20 by 25

nabil1997 on MO’s Algorithm (Query square root decomposition)

Arpit Agrawal on MO’s Algorithm (Query square root decomposition)

Arpit Agrawal on MO’s Algorithm (Query square root decomposition)

shahidul_brur on Heavy Light Decomposition

Contact Me

Ask about something or report me a bug or just say Hi. I will be happy to get back.