Kth minimum in a Range

Given an array of integers and some query operations.
The query operations are of 2 types
1.Update the value of the ith index to x.
2.Given 2 integers find the kth minimum in that range.(Ex if the 2 integers are i and j ,we have to find out the kth minimum between i and j both inclusive).
I can find the Range minimum query using segment tree but could no do so for the kth minimum. Can anyone help me?

algorithm data-structures tree

edited Feb 17 '14 at 15:28
OlivierLi
1,577 9 24

asked Feb 17 '14 at 10:07
user3318603
71 10

1 What complexity are you expecting? O(logN)? O(logN+k)? O(klogN)? – amit Feb 17 '14 at 10:11

2 quick selection algorithm is not fast enough ? – Mhd.Tahawi Feb 17 '14 at 10:21

2 is k constant for this problem? or a query contains 3 integers named i,j and k? – cegprakash Feb 27 '14 at 7:26

3 Answers

Here is a `O(polylog n)` per query solution that does actually not assume a constant `k`, so the `k` can vary between queries. The main idea is to use a segment tree, where every node represents an interval of array indices and contains a multiset (balanced binary search tree) of the values in the represened array segment. The update operation is pretty straightforward:

1. Walk up the segment tree from the leaf (the array index you're updating). You will encounter all nodes that represent an interval of array indices that contain the updated index. At every

node, remove the old value from the multiset and insert the new value into the multiset. Complexity: `O(log^2 n)`

2. Update the array itself.

We notice that every array element will be in `O(log n)` multisets, so the total space usage is `O(n log n)` . With linear-time merging of multisets we can build the initial segment tree in `O(n log n)` as well (there's `O(n)` work per level).

What about queries? We are given a range `[i, j]` and a rank `k` and want to find the k-th smallest element in `a[i..j]` . How do we do that?

1. Find a disjoint coverage of the query range using the standard segment tree query procedure. We get `O(log n)` disjoint nodes, the union of whose multisets is exactly the multiset of values in the query range. Let's call those multisets `s_1, ..., s_m` (with `m <= ceil(log_2 n)`). Finding the `s_i` takes `O(log n)` time.
2. Do a `select(k)` query on the union of `s_1, ..., s_m` . See below.

So how does the selection algorithm work? There is one really simple algorithm to do this.

We have `s_1, ..., s_n` and `k` given and want to find the smallest `x` in `a` , such that `s_1.rank(x) + ... + s_m.rank(x) >= k - 1` , where `rank` returns the number of elements smaller than `x` in the respective BBST (this can be implemented in `O(log n)` if we store subtree sizes). Let's just use binary search to find `x` ! We walk through the BBST of the root, do a couple of rank queries and check whether their sum is larger than or equal to `k` . It's a predicate monotone in `x` , so binary search works. The answer is then the minimum of the successors of `x` in any of the `s_i` .

Complexity: `O(n log n)` preprocessing and `O(log^3 n)` per query.

So in total we get a runtime of `O(n log n + q log^3 n)` for `q` queries. I'm sure we could get it down to `O(q log^2 n)` with a cleverer selection algorithm.

UPDATE: If we are looking for an offline algorithm that can process all queries at once, we can get `O((n + q) * log n * log (q + n))` using the following algorithm:

- Preprocess all queries, create a set of all values that ever occured in the array. The number of those will be at most `q + n` .
- Build a segment tree, but this time not on the array, but on the set of possible values.
- Every node in the segment tree represents an interval of values and maintains a set of positions where these values occurs.

- To answer a query, start at the root of the segment tree. Check how many positions in the left child of the root lie in the query interval (we can do that by doing two searches in the BBST of positions). Let that number be `m`. If `k <= m`, recurse into the left child. Otherwise recurse into the right child, with `k` decremented by `m`.
- For updates, remove the position from the `O(log (q + n))` nodes that cover the old value and insert it into the nodes that cover the new value.

The advantage of this approach is that we don't need subtree sizes, so we can implement this with most standard library implementations of balanced binary search trees (e.g. `set<int>` in C++).

We can turn this into an online algorithm by changing the segment tree out for a weight-balanced tree such as a BB[α] tree. It has logarithmic operations like other balanced binary search trees, but allows us to rebuild an entire subtree from scratch when it becomes unbalanced by charging the rebuilding cost to the operations that must have caused the imbalance.

edited Mar 5 '14 at 13:09

community wiki
15 revs
Niklas B.

Note that, to be provided efficiently, `.rank()` needs the BBST to be augmented, e.g., each node stores the number of children in its left child. – David Eisenstat Feb 27 '14 at 17:05

@David Eisenstat: Yes, the BBST must have `O(log n)` rank/insert/delete and must support multiple equal nodes. It should also support linear time merge so that we get `O(n log n)` time to build the initial segment tree (but that's not too important) – Niklas B. Feb 27 '14 at 17:07

One can work around equal values by pairing with the array index, but rank alone disqualifies most standard library implementations. – David Eisenstat Feb 27 '14 at 17:15

@David: Sure enough. Byebye `multiset<int>` or `TreeSet`, hello own BBST implementation. Good thing treaps can be implemented in < 60 LoC. Not sure what your point is, though, since BBSTs with subtree sizes are a pretty common primitive in algorithm engineering. I kept the answer at a pseudocode level, but obviously the implementation would require a bit of work. – Niklas B. Feb 27 '14 at 17:20

My point is that it's a very useful decoration, which more library designers should see the value of. – David Eisenstat Feb 27 '14 at 17:23

If this is a programming contest problem, then you might be able to get away with the following O(n log(n) + q n^0.5 log(n)^1.5)-time algorithm. It is set up to use the C++ STL well and has a much better big-O constant than Niklas's (previous?) answer on account of using much less space and indirection.

Divide the array into k chunks of length n/k. Copy each chunk into the corresponding locations of a second array and sort it. To update: copy the chunk that changed into the second array and sort it again (time O((n/k) log(n/k)). To query: copy to a scratch array the at most 2 (n/k - 1) elements that belong to a chunk partially overlapping the query interval. Sort them. Use one of the answers to this question to select the element of the requested rank out of the union of the sorted scratch array and fully overlapping chunks, in time O(k log(n/k)^2). The optimum setting of k in theory is (n/log(n))^0.5. It's possible to shave another log(n)^0.5 using the complicated algorithm of Frederickson and Johnson.

edited Mar 5 '14 at 12:18

answered Feb 27 '14 at 21:57
David Eisenstat
20.1k 5 11 39

In a programming contest we can get away with a simpler `O((q + n) * log^2 (q + n))` offline approach that I outlined in an update of my answer now. It also doesn't require subtree size augmentation :) Also may I ask what Bentley's algorithm is? Is it the "naive" range query implementation? What would be a thriftier alternative? Using fractional cascading in a weight-balanced range tree to support updates? – Niklas B. Mar 5 '14 at 6:50

@NiklasB. Yes, it's the naive range query implementation. Chazelle reduced the space requirement to linear, and several authors since have improved the query time slightly. – David Eisenstat Mar 5 '14 at 12:02

I feel that this solution is easier to understand and easier to implement rather than building a segement tree of BST.. – cegprakash Mar 5 '14 at 12:14

@cegprakash The segment tree is less work than it sounds like, but perhaps. – David Eisenstat Mar 5 '14 at 12:19

I haven't worked much on Segment Tree. But of course I've tried RMQ problem. Niklas' solution is too difficult to understand for beginners like me. He is more into space complexity and Time complexity rather than explaining what and why he does stuffs. – cegprakash Mar 5 '14 at 12:29

perform a modification of the bucket sort: create a bucket that contains the numbers in the range you want and then sort this bucket only and find the kth minimum.

answered Feb 17 '14 at 10:42
Muhammad Adel
348 2 14

1 Not sure how you want to implement bucket sort here - the range is unknown. Anyway, this will resut at best case in linear time performance and linear space, where selection algroithm gets linear time performance with very little extra space. I believe the OP is after a sub-linear time performance algorithm, at the cost of sublinear operations during update. – amit Feb 17 '14 at 10:49