kAc's blog

Mo's Algorithm

By **kAc**, 2 years ago,

This algorithm can solve some difficult data structure problems. Like this: Give an array a[] of size N, we query Q times. Each time we want to get the mode number(the value that appears most often in a set of data) of subsequence a[l], a[l + 1] .. a[r].

Maybe you want to solve it by segment tree or some other data structures. However, if you know the answer of [l, middle], [middle + 1, r], it's difficult to get the answer of [l, r].

Here is an offline algorithm which can solve this problem in O(N * sqrt(N) * log(N) + Q log Q).

First we need a data structure that can support: insert a number. delete a number. get the mode number. It can be solved easily by binary heap / balanced binary search tree. We call this data structure DS.

Secondly, if we have all the numbers in [l, r] organized in DS, we can transform to [l', r'] in O((|l-l'| + |r-r'|) * log N). We need a good tranform order.

```
S = the max integer number which is less than sqrt(N);
bool cmp(Query A, Query B)
{
  if (A.l / S ! = B.l / S) return A.l / S < B.l / S;
  return A.r > B.r
}
```

We sort all the queries by this comparator, and it can be proofed easily that the total transform costs is N sqrt(N) log N.

+72 kAc 2 years ago 25

Comments (25)

Write comment?

S.HASHEMI 2 years ago, # | 0

Thanks in advance.

→ Reply

dc. 2 years ago, # | ← Rev. 2 +12

....So why does this post keep getting negative feedback...

EDIT : Now it starts to get positive votes

→ Reply

→ Top rated

#	User	Rating
1	tourist	3374
2	Petr	3003
3	vepifanov	2963
4	rng_58	2941
5	subscriber	2912
6	WJMZBMR	2853
7	scott_wu	2841
8	TooSimple	2826
9	qwerty787788	2824
10	KAN	2807

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	Petr	157
1	PrinceOfPersia	157
1	Egor	157
4	Endagorion	153
5	Swistakk	147
6	I_love_Hoang_Yen	144
7	I_love_Tanya_Romanova	142
8	Rubanenko	139
9	Monyura	137
10	marat.snowbear	135

View all →

→ Find user

Handle:

Find

→ Recent actions

shank_punk → Maximum xor subsequence

3013216027 → Test

iamdumb → How many marks can you get ?

Cer → Segment tree lazy propagation ?

Alex-ander007 → Time Limit Exceeded — Why?

Scorpy

2 years ago, # ^ | +21

I think, that it's just nice usage of SQRT-decomposition idea, but not algo that deserves to be named in honor of person.

→ Reply

kAc

2 years ago, # ^ | 0

I'm so sorry but this algorithm is famous as Mo's algorithm in Chinese high school student. Here I just use the ordinary name.

→ Reply

jlcastrillon

2 years ago, # | 0

How can this problem be solved with less time complexity?

→ Reply

becauseofyou

2 years ago, # | ← Rev. 2 0

is there any problems in oj?

→ Reply

ItsLastDay

2 years ago, # ^ | +4

I was told the similar solution to this problem 86D - Powerful array

→ Reply

jlcastrillon

2 years ago, # | ← Rev. 3 0

I got a couple of questions: - Which value would S take if there wasn't any element having a value less than sqrt(N)? - Could you please describe the proof of the overall time complexity? EDIT: When you say the max integer which is less than sqrt(N) you mean floor(sqrt(N)), then it makes sense and the proof is easy as you already said, sorry for the misunderstading.

→ Reply

H3X

2 years ago, # | +6

Why A.r > B.r? I can't see it, shouldn't it be < instead?

→ Reply

kAc

2 years ago, # ^ | +3

Both them are right. let l' = l / S For all the queries whose l' is the same, the movement of right pointer is monotonous, so the total costs is O(N). And there can be at most S kinds of queries whose l' is different, here S = sqrt(N), so the total costs of right pointer movement is at most N * sqrt(N)

→ Reply

2 years ago, # ^ | +1

Yeah i just realized that it doesn't matter on my way

Tparsa → TLE D - Mr. Kitayuta's Colorful Graph

KrK → Rebirth of competitive programming group

DuX → Bubble Cup 8 — Finals

ADJA → Codeforces Round #306 (Div. 2) Editorial

hepic1 → Super Deal Game - Codeforces Api

PrinceOfPersia → Codeforces Round #305 Editorial

AFGhazy → Any help with Beta 1 (C) Ancient Berland Circus

ttrkaya → Denial of judgement at 311/A

Errichto → Codeforces Round #318 [RussianCodeCup Thanks-Round] Editorial

Rupesh_25127 → Looking for Problem B solution in Codeforces Round #201 (Div. 1)

shashank21j → [HackerRank] Week of Code 17

PrinceOfPersia → Algorithm Gym :: Data structures

Sammarize → Codeforces Round 313 — Extended editoral

Gamoool → a problem with Dijkstra (visited boolean array)

A.K.Goharshady → Codeforces Beta Round #40 Mini-Tutorial

cgy4ever → Codeforces Round #270

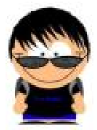
zeliboba → Codeforces Round #317

Petr → A week with SSE

kingofnumbers → September Easy Challenge on HackerEarth

KrK → KTU Programming Camp 2015

Detailed →


H3X

Yeah I just realized that it doesn't matter on my way back to home. Thanks for the reply.

→ Reply

Slayer_

2 years ago, # | ← Rev. 2 -17

thanks for good algorithm.

→ Reply

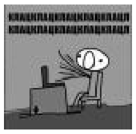
dalex

2 years ago, # | +1

I think it can be solved in $O(n\sqrt{n} + Q\log Q)$.

1. Sort all queries with your comparator except `A.r < B.r` in the last line.
2. Make array `cnt[]` where `cnt[x]` is how many times number x was met. Also we will maintain the number `maxNum` which was met maximal number of times (`cnt[maxNum]` is this number of times).
3. Process the queries group by group. Queries A and B are in the same group if `A.l == B.l` . So there are $\sqrt{n}$ such groups. Clear array `cnt` , then...
4. Start with `r = k*S` $(k = 1, 2, ...)$ (it's the first element of the next block in sqrt-decomposition) and move it to the right, updating `cnt` and `maxNum` . If some query in this block has the right border equal to `r` , then...
5. Store number `maxNum` in the temp variable and move pointer `l` from position `k*S` to the left while it is not equal to current query's left border. It will move to distance no more than $\sqrt{n}$. Of course, we updating array `cnt` and variable `maxNum` during this process.
6. If `l` is equal to the query's left border we can answer the query — it is `maxNum` . Then move pointer `l` back to the right, to its initial place `k*S` , updating `cnt` with subtractions, not additions. Finally, variable `maxNum` — the most appearing number — is now wrong — we cannot maintain it when we subtract from `cnt` . But we stored its actual value (for segment $[kS, r]$) at the beginning of step 5. Our data structure became consistent and we can continue processing the queries.

→ Reply


Fcdkbear

2 years ago, # | +9

Here is $O(N\sqrt{N} + Q\sqrt{N})$ idea.

First of all, lets sort all queries using your comparator.

Let array $S[N]$ be an array, which consist of N hashsets. $S[i]$ contains all numbers, which appear in our current segment exactly i times. Let $A[n]$ be an array(or hashmap), where $A[i]$ contains the number of appearances of number i in our current segment. Let max be the biggest index of nonempty set in S

Using this structures, we can add or delete number in $O(1)$ time:

1) Add some number v: delete v from $S[A[v]]$, increase $A[v]$ by 1, add v to $S[A[v]]$, if $A[v] > max$ — increase max by 1.

2) Delete some number v: delete v from $S[A[v]]$, decrease $A[v]$ by 1, add v to $S[A[v]]$, if $S[max].size() == 0$ — decrease max by 1.

To get answer for the query we just take any number from $S[max]$

To get answer for the query we just take any number from $S[max]$
→ Reply

2 years ago, # | 0

jlcastrillon

The total transformation cost is (N sqrt(N) + Q sqrt(N))log N instead of N sqrt(N) log N, because for each query in the worst case you will have to go through a whole block of length sqrt(N).
→ Reply

18 months ago, # | ← Rev. 23 +33

niklasb

There's one cool adaption of that: Say you have a data structure that supports `insert` trivially but not `delete`. There are lots of examples where this is the case.

Assume however that we can implement a `snapshot()` and `rollback()` function for our data structure, so that a `rollback()` brings us back to the state of the latest snapshot in $O(k \cdot \text{insert time})$, where k is the number of inserts since the snapshot. Every persistent data structure makes this possible trivially, but that's actually more than we need and often we can find tricks to implement these operations in a much simpler way, possibly even by using `memcpy`

We can still use the technique presented here with runtime $O((Q+n)\sqrt{n} \cdot \text{insert time})$. `init()` can have complexity $O(n \cdot \text{insert time})$.

In the following code, we assume zero-based indexing of the queries and inclusive right borders:

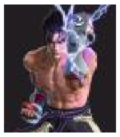
```
rt = sqrt(n)
init()  // this initializes our data structure (clears it)
snapshot()
sort queries (l, r) by (l / rt, r) ascending
for all queries q
    if q.r - q.l + 1 <= rt + 1 // we process light queries
        for j := q.l to q.r
            insert(j)
        store answer for query q
        rollback()
last_bucket = -1
for all queries q
    if q.r - q.l + 1 <= rt: continue
    bucket = q.l / rt

    if bucket != last_bucket
        init()
        l = (bucket + 1) * rt // right border of the bucket
        r = q.r
        for j := l to r
            insert(j)
    last_bucket = bucket

    while r < q.r
        insert(++r)
    snapshot()
    for j := q.l to l - 1
        insert(j)
    store answer for query q
    rollback()
```

→ Reply

→ Reply



ng420

18 months ago, # ^ | 0

Will you please elaborate a bit? Will you do rollback, if you need O(n) or O(nlgn) to do it?

→ Reply

niklasb

18 months ago, # ^ | ← Rev. 4 0

You do that $\Omega(Q)$ times, so to stay within the time bound, we have to be able to charge it the inserts that happened since the last snapshot. That's why I said it must be bounded by $O(k \cdot \text{insert time})$

→ Reply

gojira

17 months ago, # ^ | +16

Now that the contest is finally over, would you mind answering why you posted the solution to this problem while the contest was midway?

→ Reply

niklasb

17 months ago, # ^ | ← Rev. 17 +20

Someone asked a question about the problem on Stack Overflow (obviously without reference to the source). I came up the idea (or better reinvented it, since it seems to be quite well-known), and because it seems like a generally useful construction I posted it here. When I was notified by a fellow user that it is from a running contest, I removed the references to DSU, which seems to be the core idea, but well you can always see the old revisions. As you might know, there is no option to delete your posts on Codeforces.

Unfortunately it often happens that we get questions about running contests on Stack Overflow. It is not against SO's rules to post them there, so they cannot be deleted and often get good answers as well. I have seen at least 20 questions about the MIKE3 and SSTORY problem, some had good answers, and I have also unknowingly described a solution for STREETTA there towards the beginning of the contest. Even if you realize this later, you cannot delete your answers once they are marked as accepted. After you see the same question 3 times in a row you can usually figure that something is probably up, but then it's often too late. What can I say. Just one more reason to have more fellow competitive programmers on Stack Overflow to spot these types of questions and mark them in the comments.

The same goes for cs.stackexchange.com I suppose, although it is a bit less frequented.

I guess in the end this is all about fun, so whoever used this approach blindly without thinking about it further has missed a good opportunity to come up with the much more elegant approach using Link-cut trees... It will probably haunt you in the future :)

→ Reply

17 months ago, # ^ | +8

gojira

What surprised me initially was that you had submissions to the problems of this contest before you posted this solution. Now I see that all the initial submissions were to the problem ANUGCD, so I assume you only had that one problem read at the time of this post.

→ Reply

17 months ago, ← Rev. 5 # ^ | +8

niklasb

Yes indeed, the reason I looked at ANUGCD was yet another Stack Overflow post that made me interested in that particular problem.

→ Reply

10 months ago, # ^ | 0

trungpham90

Hmm, you can always edit your answer, make it become blank for example!

→ Reply

10 months ago, # ^ | 0

niklasb

And you can just look at the old revisions.

→ Reply

10 months ago, # | -8

darkshadows

"First we need a data structure that can support: insert a number. delete a number. get the mode number. It can be solved easily by binary heap / balanced binary search tree."
Another easy way would be to keep a map of (number, frequency) and a set of pair (frequency,number). Insertion/Deletion are O(log N). Getting mode is O(1).

→ Reply