Sqrt Decomposition

Patrick Moore

Australian Mathematics Trust

April 12, 2022

Motivating SQRT Decomposition

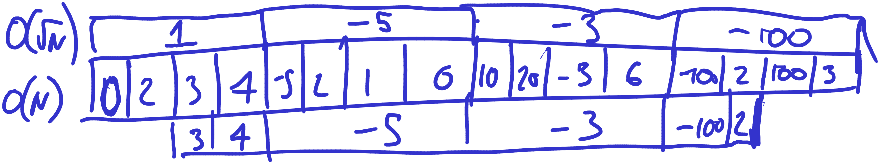
We are given an array of size N with integers from 1 to 1,000,000. We are given Q queries of the form (L_i, R_i). For each query, output the minimum of the range $[Li, Ri]$. You must also support updates of the form (X_i, Y_i), which corresponds to setting the X_i-th element to Y_i. This is vanilla RMQ.

segtree

N $\leqslant$ 100,000
Q $\leqslant$ 100,000

Bucketing the array

When we receive lots of long queries, it is inefficient to check every single element from L to R. We can place contiguous elements into a "bucket" and precompute the minimum of that bucket. When we answer queries, instead of checking every element in that bucket, we just check the one precomputed value.



What size bucket?

Let's assume we have an array of length N, and buckets of size B. Derive the following quantities in terms of N and B:

a) The number of buckets $\frac{N}{B}$
b) The number of operations to update a point B
(WC) c) The number of buckets to check in a query $\frac{N}{B}$
(WC) d) The number of elements to check at query endpoints B
e) The total number of values to check per query

$$\text{Ops} \rightarrow B + \frac{N}{B} \Rightarrow \sqrt{N} \qquad \sqrt{N} + \sqrt{N} \simeq \sqrt{N}$$

Thus we can see that the optimal bucket size is: $B = \sqrt{N}$

Complexities of SQRT Decomposition

a) Precomp: $O(N)$

b) Update: $O(B) = O(\sqrt{N})$ per Query

c) Range Query: $O(\sqrt{N})$

d) Additional Memory: $O(\sqrt{N})$

Segtree
→ recursive $\Rightarrow 4 \cdot N$
→ iterative $\Rightarrow 2 \cdot N$

Implementation notes

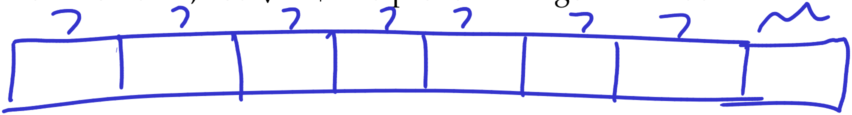
int

sqrt() is a slow operation - precompute $\sqrt{N}$. $\Rightarrow$ sq / block

$B = \sqrt{N}$ $\quad \sqrt{N} \approx x.9999999 = 7.9999$

$\sqrt{N}$ rounds down, use $\sqrt{N}+1$ to prevent a large final block.

$N = 63$



The block of a given index i is i/B $\quad$ 49 $\quad$ 63-49

14

Batching Queries

Sqrt Decomposition can also be used to batch queries together as a form of precomputation that we update every so often.

$Q = 16 \Rightarrow \sqrt{Q} = 4$

$O(1)$

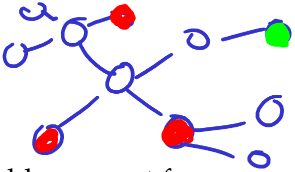
$| \; U \; U \; Q \; Q \; | \; U \; Q \; U \; U \; | \; U \; U \; Q \; Q \; | \; U \; U \; Q \; U \; |$

$\sqrt{Q} \quad \sqrt{Q} \quad \sqrt{Q} \quad \sqrt{Q}$

Precomp: $\sqrt{Q} \cdot N$ or Q

$Q: O(1) + \sqrt{Q}$

Batching Queries - Xenia and Tree



Xenia has a tree with N nodes, all nodes are blue except for node 1, which is red. Support the following operations:

1. Paint a node red.
2. Output the minimum distance for a node, v, to a red node.

Xenia and Tree solution

| U U Q Q | U Q V Q | Q Q U Q | Q U U Q |

$Q = 16$

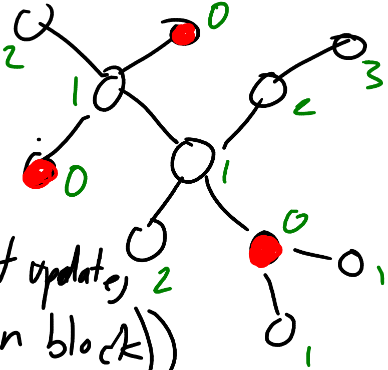
$\sqrt{Q} = 4$

PL: $O(\log(N))$

BFS: $O(1)$

$\mathrm{Min}(bfs[v], PL(v, \forall \text{ update in block}))$

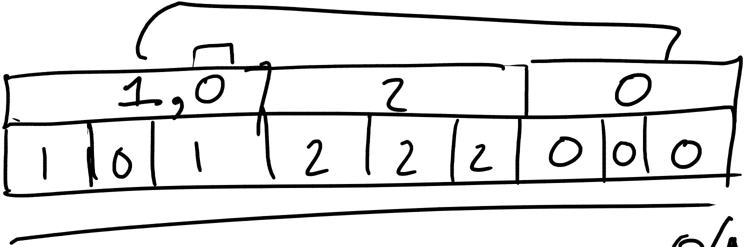
$O((N+Q)\sqrt{Q}\log(N))$



Mo's Algorithm

RMQ → min(O(1))

Mo's algorithm is very useful when answering queries **OFFLINE**, where merging is slow. The classic Mo's problem is answering range mode queries offline. Merging the results of two buckets is very slow and inefficient.

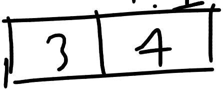


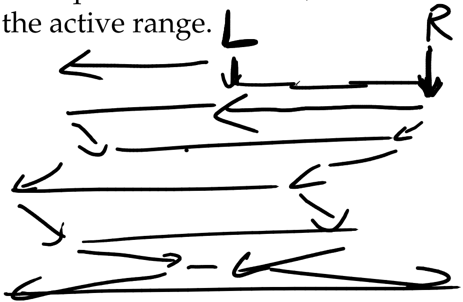
O(N)

Mo's Algorithm - The Active Range

Let's maintain an "active range", and store the data about that range as it changes. Observe that it is easy to take baby steps (remove or add a single element from the start or end of the active range). Since we know all the queries in advance, we can try and minimise movement of the active range.

3: 1
4: 1

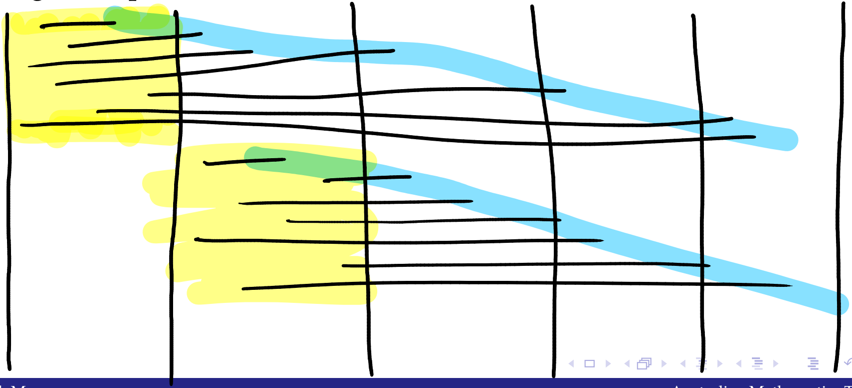




Mo's Algorithm - Ordering the queries

$N=25$

Order the queries by the $\overbrace{\text{bucket}}^{\sqrt{N}}$ of the left endpoint, then by the right endpoint.

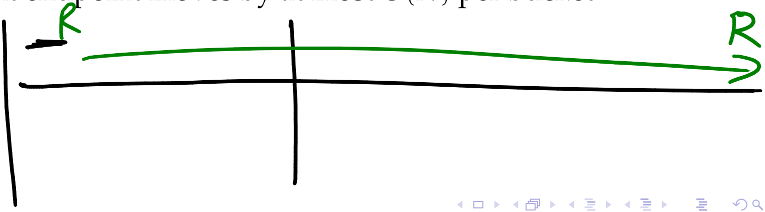


Mo's Algorithm - Movement of the endpoints

B=size

The left endpoint moves by at most $O(B)$ per query

The right endpoint moves by at most $O(N)$ per bucket



Mo's Algorithm - Complexity Analysis

N elements. Q queries. Block size of B.

Sorting the queries: $Q \times \log(Q)$

Movement of the left endpoint: $Q \times B$

Movement of the right endpoint: $N \times \frac{N}{B} = \frac{N^2}{B}$

Total complexity:

$QB + N \times \frac{N}{B}$

$NB + \frac{N^2}{B} = N\left(B + \frac{N}{B}\right)$

→ If $Q \approx N$, the optimal block size is: $B = \sqrt{N}$

This gives a total complexity of: $(Q+N)\sqrt{N}$

Mo's Algorithm - Implementation Notes

Create functions add(index) and remove(index) to make things easy.

Between blocks, don't bother recalculating the range from scratch, just move the right pointer back.

if(b[a] == b[b])

→ REP

custom comparator

Else b[a] < b[b]

Problems

- Sqrt Decomposition
  - 1 XOR Array (SQRT)
  - 3 Elephants (IOI)
  - 2 Xenia and Tree
  - 4 Coloured Walkway
  - 5 Regions
- Mo's Algorithm
  - 1 OMO
  - 2 Common Ground (Subtask 3)