Square Root Decomposition

Special class

Tanuj Khattar • Nov 21, 2020

Square Root Decomposition

Course: https://unacademy.com/a/i-p-c-intermediate-track

tanujkhattar@

Objective

- MO’s Algorithm
    - Introduction and Problem Discussion
- MO’s with Updates
    - Introduction and Problem Discussion
- Other MO’s Variants
    - MO’s with Hilbert Curves
    - MO’s On Trees
- Overview of other Types of Square Root Decomposition
    - Array Square Root Decomposition
    - Query Square Root Decomposition
    - Heavy Set / Light Set Based Square Root Decomposition
- Conclusion

MO’s Algorithm

- Find a way to quickly “add” and “remove” an element to a range.
    - Given some DS and an answer for the range [L, R], we should be able to quickly “add”/”remove” an element s.t. we have updated DS and updated answer for range [L, R+1] / [L, R - 1].

MO’s Algorithm

- Notice that it takes $|L_1 - L_2| + |R_1 - R_2|$ operations to go from $[L_1, R_1]$ to $[L_2, R_2]$.
    - Here an “operation” refers to the add or remove operation.

MO's Algorithm

- Sort the queries offline such that $\sum (|L_i - L_{i+1}| + |R_i - R_{i+1}|)$ is minimized.
  - Reduces to TSP - NP Hard.
  - Can sort the queries smartly such that this summation is O((N + Q) * Sqrt(Q))
  - bool cmp(Query a, Query b) {
    - return (a.lb < b.lb) || (a.lb == b.lb && a.r < b.r);
  - }

MO's Algorithm

- Find a way to quickly "add" and "remove" an element to a range.
  - Given some DS and an answer for the range [L, R], we should be able to quickly "add"/"remove" an element s.t. we have updated DS and updated answer for range [L, R+1] / [L, R - 1].
- Notice that it takes $|L_1 - L_2| + |R_1 - R_2|$ operations to go from $[L_1, R_1]$ to $[L_2, R_2]$.
  - Here an "operation" refers to the add or remove operation.
- Sort the queries offline such that $\sum (|L_i - L_{i+1}| + |R_i - R_{i+1}|)$ is minimized.
  - Reduces to TSP - NP Hard.
  - Can sort the queries smartly such that this summation is O((N + Q) * Sqrt(Q))
  - bool cmp(Query a, Query b) {
    - return (a.lb < b.lb) || (a.lb == b.lb && a.r < b.r);
  - }

MO's Algorithm - Sorting Approach 1

- Two queries with L in the same block are sorted as per increasing R.
- Two queries with L in different blocks are sorted as per increasing LB (L Block)

```
bool cmp(Query a, Query b) {
    return (a.lb < b.lb) ||
           (a.lb == b.lb && a.r < b.r);
}
```

MO’s Algorithm - Sorting Approach 1

- We add/remove at most O(B) elements on the left side for every query -- O(B * Q).
- For every block, we add at-most O(N) elements on the right side -- O(N * N / B)
- For B = Sqrt(N), we get O((N + Q) * Sqrt(B)).

```
bool cmp(Query a, Query b) {
    return (a.lb < b.lb) ||
           (a.lb == b.lb && a.r < b.r);
}
```

MO's Algorithm - Sorting Approach 2

- We can (slightly) optimize the previous approach by sorting the R in reverse order for even blocks.

```
bool cmp(Query a, Query b) {
    return (a.lb < b.lb) ||
           (a.lb == b.lb &&
           (a.lb & 1 ? a.r < b.r : a.r > b.r));
}
```

Practice Problem - 1 (spoj DQUERY)

- Given an array A of N integers, there are Q queries of the form:
    - Range Query: Given **L, R -** Find number of distinct elements in [L, R]

Practice Problem - 2 (spoj XXXXXXXX)

- Given an array A of N integers, there are Q queries of the form:
    - Range Query: Given **L, R -** Find number of distinct elements in [L, R]
    - Point Update: Given **i, x** - set **A[i] = x**

MO's with Updates

- Given the DS maintained for MO's without updates, find a way to quickly "apply" and "undo" the update on the DS.
    - Eg: For point updates, store the previous value so that the point update can be "undone".

MO’s with Updates

- Let every query be represented as (L, R, T) where T = number of updates before this query.
  - Now to go from [L1, R1, T1] to [L2, R2, T2], we need |L1 - L2| + |R1 - R2| + |T1 - T2| operations.

MO's with Updates

- Sort the queries such that the $\sum (|L_i - L_{i+1}| + |R_i - R_{i+1}| + |T_i - T_{i+1}|)$ is minimized.
  - Process the queries whose Left and Right blocks are same together.
  - Sort such queries based on T s.t. Every pair of blocks, we spend O(Q) time iterating over increasing T.
  - Therefore, if $B = N^{(2/3)}$, $N / B = N^{(1/3)}$. So total complexity = $O(Q * (N / B)^2) = O(Q * N^{(2/3)})$

MO's with Updates

- Given the DS maintained for MO's without updates, find a way to quickly "apply" and "undo" the update on the DS.
  - Eg: For point updates, store the previous value so that the point update can be "undone".
- Let every query be represented as (L, R, T) where T = number of updates before this query.
  - Now to go from [L1, R1, T1] to [L2, R2, T2], we need |L1 - L2| + |R1 - R2| + |T1 - T2| operations.
- Sort the queries such that the $\sum (|L_i - L_{i+1}| + |R_i - R_{i+1}| + |T_i - T_{i+1}|)$ is minimized.
  - Process the queries whose Left and Right blocks are same together.
  - Sort such queries based on T s.t. Every pair of blocks, we spend O(Q) time iterating over increasing T.
  - Therefore, if B = N^(⅔), N / B = N^(⅓). So total complexity = O(Q * (N / B)^2) = O(Q * N ^ (⅔))

MO’s with updates sorting approach.

```
bool cmp(Query a, Query b) {
  return (a.lb < b.lb) || (a.lb == b.lb && a.rb < b.rb) ||
         (a.lb == b.lb && a.rb == b.rb && a.t < b.t);
}
```

MO’s with updates iteration loop.

```
for (int i = 1, T = 0, L = 1, R = 0; i <= nq; i++) {
  while (T < q[i].t) reflect_update(++T, true);
  while (T > q[i].t) reflect_update(T--, false);
  while (R < q[i].r) add_element(++R);
  while (L > q[i].l) add_element(--L);
  while (R > q[i].r) remove_element(R--);
  while (L < q[i].l) remove_element(L++);
  ans[q[i].idx] = curr_ans;
}
```

Practice Problem - 2

- Given an array A of N integers, there are Q queries of the form:
  - Range Query: Given **L, R -** Find number of distinct elements in [L, R]
  - Point Update: Given **i, x** - set **A[i] = x**

MO’s on Trees

- https://codeforces.com/blog/entry/43230
- Linearize the tree using Euler Tour Traversal.
- A path in the tree reduces to a continuous range in an array
  - Elements on the path occurs once.
  - Elements not on the path occur twice.
- Use standard MOs on linearized array

MO’s with Hilbert Curves

- https://codeforces.com/blog/entry/61203
- Better sorting algorithm based on hilbert curves.

Other Types of Square Root Decomposition

- https://unacademy.com/course/course-on-data-structures-square-root-decomposition/XKRCFDJV
- Array Square Root Decomposition
  - Divide the array into blocks of Size B ~ Sqrt(N) & maintain some information for each block.
  - Divide a query/update into two parts
    - Individual elements in blocks of L & R
    - Complete Blocks which lie between L & R
  - Perform Range Updates Lazily
- Query Square Root Decomposition
  - Divide Q queries into blocks of size B ~Sqrt(Q)
  - Process all updates at the end of each block and maintain a “hard-to-update” DS for queries.
  - Reflect contribution of B updates on a query “quickly”.
- Heavy Set / Light Set based Query Decomposition
  - Identify a property whose sum(count) is bounded and can divided into heavy and light sets.
  - Process the heavy and light sets separately - Eg: O(N * #HeavySets) + O(SizeOf(LightSet)^2).

Conclusion

- Square Root Decomposition is a powerful tool with minimal prerequisites.
- It's usually simple to code and is a great alternative to more complex data structures like Segment Trees etc.