香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Square Root Decomposition

Fuzen Ng {yfng}

2023-07-02

What is Square Root Decomposition

- A process of separating a size $O(N)$ structure to $O(\sqrt{N})$ "blocks", each with a size $O(\sqrt{N})$.
- Can we do better?
- If we separate it into x blocks, there will be $O(x)$ number of blocks and each block has size $O(N/x)$

Decomposition on array – Range Sum Query

- Given an array A of size N and Q queries
- Each query is either
  - Type 1 - Output the sum of $A[l...r]$
  - Type 2 - Change the value of $A[x]$ to v
- This can easily be solved by segment tree in $O((N+Q)\log N)$
- But we want to solve it using square root decomposition

Decomposition on array – Range Sum Query

- In a naive solution
- For Type 1 query, we calculate the sum in $O(N)$
- For Type 2 query, we update the value in $O(1)$
- The overall time complexity is $O(QN)$

Decomposition on array – Range Sum Query

- First focus on Type 1 query, we represent each query by $[l, r]$.
- Split array A into blocks of size approximately $\sqrt{N}$.
- For each block i, precalculate the sum of elements in $B[i]$.
- We can assume the number of blocks and size of the block are equal to $s = \lceil \sqrt{N} \rceil$
- Then $B[0] = A[0] + A[1] + ... + A[s-1]$, $B[1] = A[s] + A[s+1] + ... + A[2s-1]$, $B[s-1] = A[(s-1) \times s] + ... + A[N-1]$.
- The last block may have fewer elements then s if N is not a multiple of s.

Decomposition on array – Range Sum Query

- If $[l, r]$ is long enough, it will contains multiple whole blocks
- Therefore we can split this query into two ”tails” and the blocks in the middle.
- Suppose the two tails are $[l, (k+1) \times s - 1]$ and $[p \times s, r]$, where k and p are the block index of l and r respectively.
- Blocks in the middle are blocks $k+1$ to $p-1$.
- If $k = p$, the sum should be calculated trivially.
- $\sum_{i=l}^{r} A[i] = \sum_{i=l}^{(k+1)\times s-1} A[i] + \sum_{i=k+1}^{p-1} B[i] + \sum_{i=p\times s}^{r} A[i]$
- $O(\sqrt{N})$

Decomposition on array – Range Sum Query

- Example:
- $A = \{8, 1, 7, 12, 3, 5, 9, 6, 4\}$
  - $B[0] = 8 + 1 + 7 = 16$
  - $B[1] = 12 + 3 + 5 = 20$
  - $B[2] = 9 + 6 + 4 = 19$

Decomposition on array – Range Sum Query

- Example:
- $A = \{8, 1, 7, 12, 3, 5, 9, 6, 4\}$
  - $B[0] = 8 + 1 + 7 = 16$
  - $B[1] = 12 + 3 + 5 = 20$
  - $B[2] = 9 + 6 + 4 = 19$
- $[l, r] = [3, 8] \rightarrow [3, 3] + B[1] + [7, 8] = 7 + 20 + 9 + 6 = 42$
- $[l, r] = [3, 9] \rightarrow [3, 3] + B[1] + B[2] = 7 + 20 + 19 = 46$

Decomposition on array – Range Sum Query

- For Type 2 query, we can easily update the value of the corresponding block as well.
- $A[x] = v$, assume $A[x]$ belongs to block i.
- $B[i]' = B[i] - A[x] + v$
- $O(1)$

Decomposition on array – Range Sum Query

- Overall we precompute in $O(N)$ and answer Q queries in
  - Type 1: $O(\sqrt{N})$
  - Type 2: $O(1)$
- Overall time complexity: $O(N + Q\sqrt{N})$

Decomposition on array – Similar Problems

- Similar techniques can be used on
  - Find min of $A[l...r]$
  - Find max of $A[l...r]$
  - Find number of $A[x] = 0$ where $l \leq x \leq r$
  - Find the first non-zero element in $A[l...r]$

Decomposition on array – Range Update

- We can also handle range update queries in square root decomposition
- Consider the same Range Sum Query problem but add a Type 3 query where we add v to all elements in $A[l...r]$.
- Let $C[i]$ stores the value that has to be added to all elements in the block, initially $C[i] = 0$
- We split the query into two ”tails” and the middle blocks.
- For all the middle block, $C[i]$ += v.
- For the remaining elements in the tail, $A[i]$ += v.

Decomposition on array – CF551E

- CF551E GukiZ and GukiZiana
- `https://codeforces.com/problemset/problem/551/E`
- Given an array A with length N and Q queries.
- Two type of queries:
  - Type 1: Add x to $[l, r]$
  - Type 2: Find the maximum value of $j - i$ s.t. $a[i] = a[j] = y$

Decomposition on array – CF551E

Sample Input:
4 3
1 2 3 4
1 1 2 1
1 1 1 1
2 3
Sample Output:
2

Decomposition on array – CF551E

- Again we first split the array into $\sqrt{N}$ blocks
- Let the array be $\{1, 1, 1, 2, 1, 3, 1, 1\}$
  - Block 0 contains $\{1, 1, 1\}$
  - Block 1 contains $\{2, 1, 3\}$
  - Block 2 contains $\{1, 1\}$

Decomposition on array – CF551E

- For Type 2 queries, we want to find the minimum i and maximum j s.t. $a[i] = a[j] = y$.
- This search can be done by binary search!
- Store a sorted array for each block, where each element is $(a[i], i)$, sorted by $a[i]$ then i in ascending order.
  - Block 0 contains $\{1, 1, 1\} \to \{(1, 0), (1, 1), (1, 2)\}$
  - Block 1 contains $\{2, 1, 3\} \to \{(1, 4), (2, 3), (3, 5)\}$
  - Block 2 contains $\{1, 1\} \to \{(1, 6), (1, 7)\}$
- In each query for y, find the leftmost and rightmost element in blocks with $(a[i], x)$
- Take the smallest x as minimum i, and largest as maximum j.

Decomposition on array – CF551E

- For Type 1 queries, we can also split the query into two ”tails” and the middle blocks.
- For middle blocks, use the range update technique in previous slides.
- For tails blocks, we can reconstruct the sorted block in $O(\sqrt{N} \log \sqrt{N})$

Mo's Algorithm

- Mo's Algorithm is a technique that is based on square root decomposition.
- In normal square root decomposition, we precompute information of each blocks, and merge the queries. But some type of queries cannot be easily merged, e.g. Finding the mode of an interval.
- Mo's Algorithm takes advantages of transition a query in $O(1)$ or $O(\log n)$ from $[l, r]$ to $[l+1, r]$, $[l-1, r]$, $[l, r-1]$ and $[l, r+1]$.
- Then we can avoid extra calculations if we carefully plan how we order the queries and make the transition.

Mo's Algorithm – Template

```
void move(int pos, int sign) {
  // update nowAns
}
void solve() {
  BLOCK_SIZE = int(ceil(pow(n, 0.5)));
  sort(querys, querys + m);
  for (int i = 0; i < m; ++i) {
    const query &q = querys[i];
    while (l > q.l) move(--l, 1);
    while (r < q.r) move(r++, 1);
    while (l < q.l) move(l++, -1);
    while (r > q.r) move(--r, -1);
    ans[q.id] = nowAns;
  }
```

Mo's Algorithm – Sorting

- To achieve $O(N\sqrt{N})$, the queries should be sorted in a special way.
- We will first answer queries with left index in block 0, then queries with left index in block 1, etc.
- We can also define a struct for queries and make it easier to store them and sort them.

```
struct Query {
    int l, r, idx;
    bool operator<(Query other) const {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};
```

Mo's Algorithm – Analysis

- Let the maximum value of leftmost index of each block be $max_1, max_2, \ldots, max_{\lceil\sqrt{N}\rceil}$
- After sorting, we have $max_1 \leq max_2 \leq \ldots \leq max_{\lceil\sqrt{N}\rceil}$
- It takes $O(N)$ to compute the first answer of each block.
- Or $O(\sqrt{N})$ for all blocks except the first one, if we compute the first answer using the first answer of the previous block.
- In worst case of every block, the maximum value of the rightmost index is N, and every modification of the leftmost index is either from $max_{i-1} + 1$ to max_i or from max_i to $max_{i-1} + 1$.

Mo's Algorithm – Analysis

- Since the rightmost index is sorted in each block, answering the queries takes at most $O(N)$ or $O(N \log N)$ time. In total it takes $O(N\sqrt{N})$ or $O(N\sqrt{N} \log N)$ for all blocks.
- Let Q' be the number of queries within the block.
- For the leftmost index, every modification takes $O(max_i - max_{i-1}) = O(\sqrt{N})$, the time complexity within the block is $O(Q' \times \sqrt{N})$.
- The total time complexity for the leftmost index is $O(Q\sqrt{N})$

Mo's Algorithm – Analysis

- The time complexity for the rightmost indexes within a block is $O(N)$
- The total time complexity for the rightmost indexes is $O(N\sqrt{N})$
- The total time complexity is $O((N+Q)\sqrt{N})$

Mo's Algorithm – CF86D

- CF86D - Powerful Array
  `https://codeforces.com/problemset/problem/86/D`
- Given an array of length n and t queries.
- On each query (l, r), output the *power* of $a[l..r]$.
- The power of $a[l..r]$ is $\sum cnt[s]^2 \times s$ for all unique integer s.

Mo's Algorithm – CF86D

Example

- $a = [1, 1, 2, 2, 1, 3, 1, 1]$
- Query $(2, 7)$
- Answer = $3^2 \times 1 + 2^2 \times 2 + 1^2 \times 3 = 9 + 8 + 3 = 20$

Mo's Algorithm – CF86D

- Naive solution
- For each query, loop over (l_i, r_i) and calculate the answer.
- $O(tn)$
- Notice that these queries can be processed offline, i.e. the order of processing the queries does not matter!
- We can sort and change the order of the queries.
- Maybe we can reuse the information of the previous queries.

Mo's Algorithm – CF86D

- Assume you have the answer for (l, r).
- We should consider how the answer changes if we add an extra element or remove an element.
- When we add an element s, $cnt[s] = cnt[s] + 1$, the contribution of it to answer is from $cnt[s]^2$ to $(cnt[s] + 1)^2$, therefore the answer should change by $2 \times cnt[s] + 1$.
- it is similar when we remove an element, therefore this transition is $O(1)$.
- By the analysis of previous slides, we can guarantee that this takes at most $O((t + n)\sqrt{n})$ steps.

Mo's Algorithm – CF617E

- CF617E - XOR and Favorite Number
  `https://codeforces.com/problemset/problem/617/E`
- Given an array a of length n, and m queries and an integer k.
- For each queries (l, r), count the number of pairs (i, j) s.t. $a[i] \oplus a[i+1] \oplus \ldots \oplus a[j] = k$ and $l \leq i \leq j \leq r$.

Mo's Algorithm – CF617E

- First think of how to handle one query quickly.
- We can calculate the prefix xor sum $pre[i] = a[1] \oplus a[2] \oplus \ldots \oplus a[i]$.
- For a pair (i, j), $a[i] \oplus a[i+1] \oplus \ldots \oplus a[j] = k$ can be replaced by $pre[j] \oplus pre[i-1] = k$.
- For a query (l, r), for all distinct prefix sum v in this range we keep track of their count $cnt[v]$ as well as the answer to the query.

Mo's Algorithm – CF617E

- For some prefix sum value v, we have $u = v \oplus k$, and the contribution of one v to the answer equals $cnt[u]$.
- Therefore when we add a new element i, with $v = pre[i]$, we can find the desired value $u = v \oplus k$.
- Since $cnt[v]$ will increase by 1, $cnt[u]$ will be added to the answer.
- The same for removing an element.
- With Mo's Algorithm, we can calculate the answers to the queries in $O((n+m)\sqrt{n})$.

Avoiding $n\sqrt{n}\log n$

- evil time complexity
- Try your best to avoid this

Avoiding $n\sqrt{n}\log n$

- Given an array A of length N, and N queries of the form ”given l and r, find the MEX of $A[l], ..., A[r]$”, $N \leq 10^5$.

Avoiding $n\sqrt{n}\log n$

- Given an array A of length N, and N queries of the form "given l and r, find the MEX of $A[l], ..., A[r]$", $N \leq 10^5$.
- Try to solve it using Mo's Algorithm.
- Maintain a counting array cnt, and a set S containing all numbers v that has $cnt[v] = 0$.
- Inserting an element: $O(\log n)$ (calls $O(n\sqrt{n})$ times)
- Removing an element: $O(\log n)$ (calls $O(n\sqrt{n})$ times)
- Querying the minimum element: $O(\log n)$ (calls $O(n)$ times)

Avoiding $n\sqrt{n}\log n$

- Inserting an element: $O(\log n)$ (calls $O(n\sqrt{n})$ times)
- Removing an element: $O(\log n)$ (calls $O(n\sqrt{n})$ times)
- Querying the minimum element: $O(\log n)$ (calls $O(n)$ times)
- Total time complexity: $O(n\sqrt{n}\log n)$
- Query part only requires $O(n\log n)$
- Can we balance the time?

Avoiding $n\sqrt{n}\log n$

- Still Mo's Algorithm for the queries
- Instead of using a set S, we use a boolean array B to store whether each number i has $cnt[i] = 0$
- we then split B into $\sqrt{n}$ blocks
- For each block, we maintain the number of elements presented in this block

Avoiding $n\sqrt{n}\log n$

- Inserting and removing an element can be done in $O(1)$
- Update the boolean array and the block
- querying can be done in $O(\sqrt{n})$
- Find the first block that isn't empty
- Then find the first existing number as the answer
- Total time complexity is $O(n\sqrt{n})$