Square root tricks

Ivan Smirnov
ifsmirnov@yandex.ru

September 28, 2018

1 Periodic rebuilding

Let us have some static data structure that allows answering some queries. For example, we have a set of numbers and ask queries of a kind "how many numbers are between L and R". It is easy to solve if the array is known in advance: sort it and do a couple of binary searches. And what if we can add the numbers to the set?

Let us fix some parameter K. Each K iterations we rebuild our data structure from scratch. To answer the query we must look at the sorted array, as before, and check each of (at most K) numbers added since last rebuild. If we have n insertions and m queries, the running time, depending on K, is $O(n/K \cdot n \log n) + O(mlogn) + O(mK)$.

If $m \approx n$, it is easy to show that the optimal choice for K is around $\sqrt{n \log n}$ and that it gives $O(n\sqrt{n \log n})$ complexity.

Summing up:

- Maintain a static data structure, rebuild it each $\sqrt{n}$ iterations.
- Store all elements added since last rebuild separately.
- For each query, make a request to the static data structure and consider all recent elements one by one.

Exercise Given a tree, perform two kinds of queries: paint vertex v black and return the closest black vertex to vertex v. Initially the tree is white.

2 Range queries

Consider a classical range-sum problem: given an array a, answer queries about the sum of elements between l_i and r_i and set $a_i = x_i$.

Fix some parameter K. Split the array into n/K parts: $[0, K)$, $[K, 2K)$, ... Store the sum for each part.

- Updating $a_i = x_i$ Locate the corresponding block (it has index i/K) and update its sum.
- Querying sum between l_i and r_i First, note that there are three kinds of interesting blocks: one block contains l_i, one block contains r_i, and several blocks are fully contained in the range. We process elements from the first two blocks in linear time ($O(K)$) and take the sum value the intermediate blocks (in $O(n/K)$ time).

Once again, optimal choice for K is $\sqrt{n}$ and we have $O(n\sqrt{n}$ complexity per n queries.

2.1 Lazy propagation

What if we want to do range updates, like "add x to all numbers between l_i and r_i"? It is possible as well. Now we store a value *inc* for each block. It means that we would like to add *inc* to all elements of this block. How do we do range update?

- For left and right boundary blocks add x to each element of the array explicitly.
- For intermediate blocks, do $inc += x$ for each block.

Clearly, it works in $O(n\sqrt{n})$ as well.

When computing range sum, take into account *inc*. For boundary blocks it is easy: just add blocks's *inc* to each element you consider. For intermediate block, note that its sum increases by $inc \times block\_size$, where $block\_size$ is usually equal to K (except for the last block).

Push It may be convenient to push lazy propagated changes sometimes. It is possible to push the *inc* value of the block to all its elements (adding *inc* to each of them) and set blocks's *inc* to zero. It may be convenient to push every time when you want to consider the block as a boundary block.

3 Small and large

We have an undirected graph and want to count the number of triangles in it (that is, triples of vertices u, v, w such that there are edges (u, v), (v, w) and (w, u) in the graph.

Let $deg(v)$ be the degree of vertex v. We say that v is *large* if $deg(v) \geq \sqrt{n}$, and *small* otherwise. Now consider all vertices one by one.

- If the vertex v is small, consider all pairs u, w of its neighbors. If there is an edge (u, w) in the graph, we found the triangle. It works in $\sum\limits_{v \text{ is small}} deg(v) \cdot deg(v) \leq \sum\limits_{v \text{ is small}} deg(v) \cdot \sqrt{n} \leq \sum\limits_{v} deg(v) \cdot \sqrt{n} = 2n \cdot \sqrt{n}$.
- If the vertex v is large, consider all edges (u, w) in the graph. If there are edges (v, u) and (v, w) as well, we found the triangle. We spend $O(n)$ time per each large vertex, so $O(n\sqrt{n})$ for all large vertices in total.