Square Root Techniques
Princeton Competitive Programming
Basic Premise

- Separate something of size $O(n)$ into blocks of size $O(\sqrt{n})$
- Precompute something per block and then combine answers
Problem 1: Range Queries

Problem Description:

- You are given an array $A$ with $n$ integers and $Q$ queries
- Each query asks for the maximum of the elements in interval $[l_i, r_i]$
- (Note: this works with any associative operation, max, min, sum, gcd, etc)

Problem Solution:

- Split $A$ into $x$ blocks of size $n/x$ each (the last block might be smaller if $n$ isn’t divisible by $x$)
- Compute maximum of values in block $i$ and store in an array $B$

\[ B[i] = \max\{A[xi], A[xi+1], \ldots, A[x(i+1)-1]\} \]

- Given an interval $[l, r]$, decompose into blocks: there will be at most 2 partial blocks and $n/x$ full blocks
- Compute maximum in partial blocks (costs $O(x)$) and use precomputed value for full blocks (costs $O(n/x)$)
- Total cost per query is $O(x + n/x)$ so pick $x = \sqrt{n}$ and we get a cost per query of $O(\sqrt{n})$
Problem 1: Range Queries
Problem 2: Range Queries with Updates

Problem Description:

- You are given an array \( A \) with \( n \) integers and \( Q \) queries of two types
- First query type asks for the maximum of the elements in interval \([l_i, r_i]\)
- Second query type asks to update the value of \( A[i] \) to \( v \)

Problem Solution:

- Same block decomposition as before
- To update value of \( A[i] \) recalculate maximum of block containing value \( i \)
- The update cost is \( O(\sqrt{n}) \)
Problem 3: Range Queries with Range Updates

Problem Description:
- You are given an array \( A \) with \( n \) integers and \( Q \) queries of two types:
  - First query type asks for the sum of the elements in interval \([l_i, r_i]\)
  - Second query type asks to add \( v \) to each \( A[l_i \ldots r_i] \)

Problem Solution:
- Same solution as before, but need two extra changes:
  - To update partial blocks, update each \( A[i] \) individually and recalculate the value of \( B \)
  - To update full blocks, add \( \text{block}_\text{size} \times v \) to the corresponding block \( B[i] \)
  - You also need to store a “lazy” value per block, so you can update the \( A[i] \) if you need to access any one individual value
Problem 4: Range Counting

Problem Description:

- You are given an array $A$ with $n$ integers and $Q$ queries of two types
- First query type asks for the number of the elements in interval $[l_i, r_i]$ that are equal to $y$
- Second query type asks to update the value of $A[i]$ to $v$

Problem Solution:

- Divide into blocks as before
- Store a map/dictionary/hash table per block, storing the frequency of each element in the block
- To answer the first query, go through the partial blocks element by element and for full blocks query the map to determine the frequency of $v$
- Updating is the same, just update the map and the individual $A[i]$
- This takes time either $O(n*\sqrt{n})*\log(n))$ or $O(n*\sqrt{n})$ with a hash map
**Problem 5: Tree Updates**

**Problem Description:**
- You are given a tree with $n$ vertices each with a value $v_i$, and $Q$ queries of two types
- First query type asks for the value of vertex $i$
- Second query type asks to add $y$ to all neighbors of vertex $i$

**Problem Solution:**
- Partition each node into one of two categories: heavy, if degree is $> \sqrt{n}$; light, if degree is $< \sqrt{n}$
- Note that there are at most $2\sqrt{n}$ heavy nodes (since number of edges is $< n$)
- To do a query of the second type, if the node is light just go through all neighbors and add one by one. If the node is heavy, store a “lazy” extra value
- To do a query of the first type, add the extra values of all the heavy neighbors of $i$ to its own value
- This takes time $O(Q\sqrt{n})$
Problem 6: Grid Painting

Problem Description:

- Consider a square grid with \( n \) cells that are originally unpainted
- Process \( n \) queries each of which is a cell \( c \) of the grid
- For each query, first compute the distance to the closest painted cell and then paint that cell
- (Note: you can extend this problem to a tree instead of a grid, but it’s more technical)

Problem Solution:

- Given a certain state of the grid, with one BFS running in \( O(n) \) time we can determine the distance from each unpainted cell to the closest painted cell
- So now we can batch the queries into blocks of \( \sqrt{n} \)
- At the start of each batch, use the BFS algorithm to compute distances
- For each query, go through each cell in the batch that came before (so at most \( \sqrt{n} \) of them), calculate the distance to \( c \) and update if it is lower than the precomputed one
- In total we run \( \sqrt{n} \) BFSs, and for each query we go through a list of length at most \( \sqrt{n} \), so runtime is \( O(n \times \sqrt{n}) \)
Problem 7: Balanced BST

Problem Description:

- Consider an array of integers that is initially empty and process n operations:
  - Add x to the array
  - Remove x from the array
  - Find if x is in the array

Problem Solution:

- We can solve this with a balanced BST, but suppose you don’t know how to implement a balanced BST, but you know how to implement a normal BST
- Use an unbalanced BST and insert into it naively
- After sqrt(n) operations, reconstruct the BST so that it is now balanced
- Each reconstruction takes O(n) time, but we only do sqrt(n) of them, so this takes O(sqrt(n)) amortized time per operation
Problem 8: Offline Dynamic Connectivity

Problem Description:

- You are given an undirected graph $G$ and you should process operations of three types:
  - Add an edge to the graph
  - Remove an edge from the graph
  - Check if nodes $u$ and $v$ are connected

Problem Solution:

- Group queries into batches of $\sqrt{n}$
- First, fix all the edges that are contained in all $\sqrt{n}$ graphs of one batch of queries
- Collapse the graph into connected components using the fixed edges (using a DFS)
- Now for each query in one batch, add/remove the edge to the collapsed graph and run one DFS per check
- Since the collapsed graph has at most $O(\sqrt{n})$ edges, each check runs in $O(\sqrt{n})$ time
Further reading

Links:

https://assets.hkoi.org/training2023/sqrt.pdf


https://assets.hkoi.org/training2019/sqrt.pdf

https://codeforces.com/blog/entry/23005

https://codeforces.com/blog/entry/83248

https://usaco.guide/CPH.pdf#page=263