On Updating and Querying Submatrices

Jason Yang
Mentor: Jun Wan

MIT PRIMES Computer Science

October 19, 2020

Range update-query problem

- A is an array of N numbers
- A range $R = [l, r]$ is the set of indices $\{i | l \leq i \leq r\}$
- $update(R, v)$: for all $i \in R$, set $A[i]$ to $A[i] + v$
- $query(R)$: return $\min_{i \in R} A[i]$

Segment tree + lazy propagation: $O(\log N)$ time updates and queries

Generalizations

Using different operators

- *update*(R, v) : $\forall i \in R, A[i] \leftarrow A[i] \bigtriangledown v$
- *query*(R, v) : return $\bigtriangleup_{i \in R} A[i]$

If $\bigtriangledown$ and $\bigtriangleup$ are associative, segment tree + lazy propagation usually works (but not always)

Ex. $(\bigtriangledown, \bigtriangleup) =$

- $(+, +)$
- $(*, +)$
- $(\leftarrow, \min)$

This problem and variants have applications in

- LCA in a tree
- image retrieval

Generalizations

2 dimensions:

- the array becomes a matrix
- ranges $\{i | l \leq i \leq r\}$ becomes submatrices $[l_0, r_0][l_1, r_1] = \{i | l_0 \leq i \leq r_0\} \times \{j | l_1 \leq j \leq r_1\}$

We call this the **submatrix update-query problem**.

Previous Work

Generalizing segment tree seems to be very difficult

	update	query
$d = 1$		
Segment Tree	$O(\log N)$	$O(\log N)$
$d = 2$		
2D Segment Tree	$O(N \log N)$	$O(\log^2 N)$
Quadtree	$O(N)$	$O(N)$
$d = 2$, special operator pairs $(\bigtriangledown, \bigtriangleup)$		
2D Fenwick Tree (Mishra)	$O(16 \log^2 N)$	$O(16 \log^2 N)$
2D Segment Tree (Ibtehaz)	$O(\log^2 N)$	$O(\log^2 N)$
2D Segment Tree (ours)	$O(\log^2 N)$	$O(\log^2 N)$

Intuition

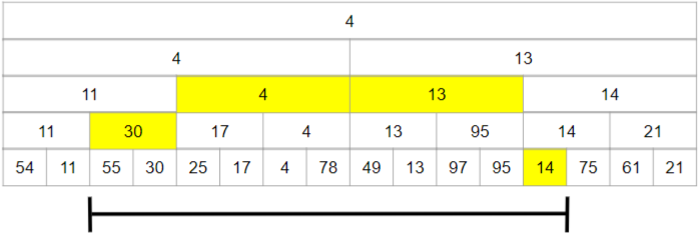
Why is generalizing the segment tree difficult?

Segment Tree: Definition/Preprocessing

A binary tree of nodes:

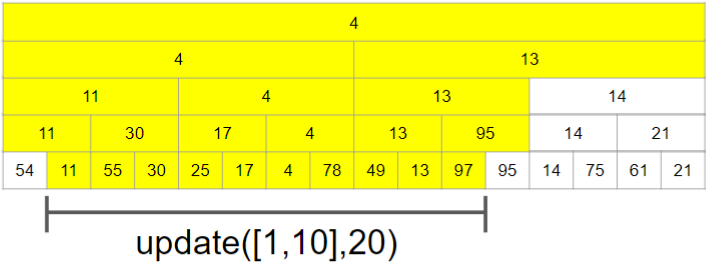
- each node n covers a range n_R and contains a value $n_V = \min_{i \in n_R} A[i]$

When querying any range, we only have to look at $O(\log N)$ nodes



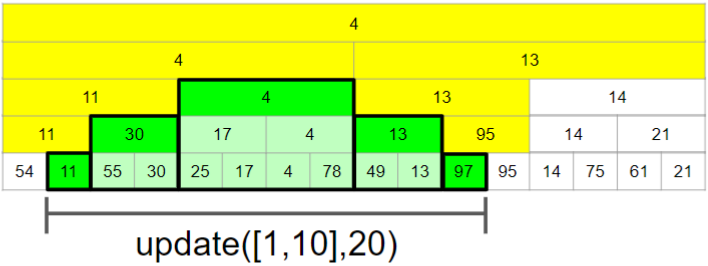
query([2,12])=min(30,4,13,14)=4

update(R, v): change n_V for all n that overlap with R
$\Rightarrow O(N)$ nodes in worst-case



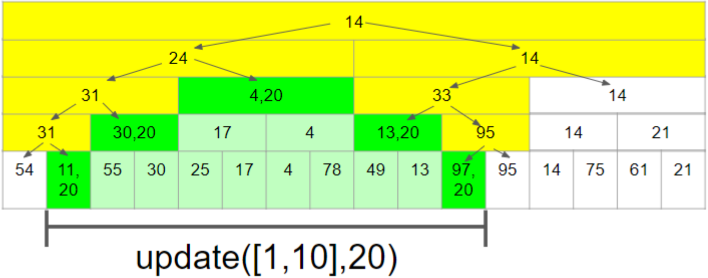
Segment Tree: Updates

- For all n s.t. $n_R \subseteq R$ (shown as green), n_V simply changes to $n_V + v$
- Split green nodes into $O(\log N)$ subtrees
- Attach a "lazy label" t_Z to every node t
  - t_Z represents the command "$n_V \leftarrow n_V + t_Z \ \forall n$ in subtree at t"
- For each subtree, increase its root node's lazy label by v



Segment Trees: Updates

- For each n s.t. $(n_R \cap R \neq \emptyset) \wedge (n_R \not\subseteq R)$ (shown as yellow) in greatest-to-lowest depth, do $n_V \leftarrow \min((n_l)_V + (n_l)_Z, (n_r)_V + (n_r)_Z)$
- Only $O(\log N)$ many such nodes



Segment Trees: Queries revised

- When looking at n_V from n, we must add all lazy values that affect it
  - We must use $n_V + \sum_{m \supseteq n} m_Z$ instead of just n_V
- $\Rightarrow O(\log^2 N)$ time queries (because we look at $O(\log N)$ nodes)
  - Can be improved to $O(\log N)$ time

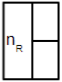
<table>
<tr><td colspan="16">14,0</td></tr>
<tr><td colspan="8">24,0</td><td colspan="8">14</td></tr>
<tr><td colspan="4">31</td><td colspan="4">4,20</td><td colspan="4">33</td><td colspan="4">14</td></tr>
<tr><td colspan="2">31</td><td colspan="2">30,20</td><td colspan="2">17</td><td colspan="2">4</td><td colspan="2">13,20</td><td colspan="2">95</td><td colspan="2">14</td><td colspan="2">21</td></tr>
<tr><td>54</td><td>11, 20</td><td>55</td><td>30</td><td>25</td><td>17</td><td>4</td><td>78</td><td>49</td><td>13</td><td>97, 20</td><td>95</td><td>14</td><td>75</td><td>61</td><td>21</td></tr>
</table>

query([4,5])=17+20+0+0=37

A segment tree of segment trees:

- Construct segment tree across rows of $N \times M$ matrix A
- Each node n in this segment tree contains a segment tree n_T constructed over the array $B = \text{eltwise-min}_{i \in n_R} A[i]$

(∇,△)=(+,min)



33	2	76	42	5	95	32	95	36	98	72	21	46	41	43	37
74	5	42	7	17	40	27	58	9	87	52	92	28	68	25	34

n_T represents the array

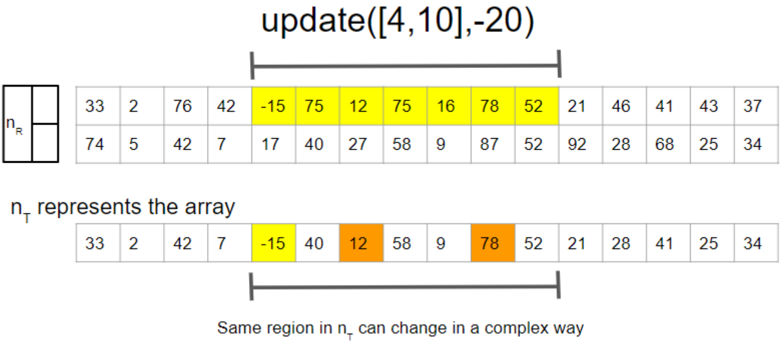
33	2	42	7	5	40	27	58	9	87	52	21	28	41	25	34

We can do queries in $O(\log N \log M)$ time:

- $query(R_X \times R_Y) = \min_{n \in S(R_X)} n_T.query(R_Y)$

2D segment tree

But updates are difficult...



Another problem: lazy propagation

2D Segment tree

By only using lazy propagation in inner segment trees, we do updates in $O(N \log M + M \log N)$ time and queries in $O(\log N \log M)$ time.

Impossible?

Perhaps it is impossible to get $O(\text{polylog}(N))$ time updates and queries?

Min-plus matrix multiplication

Given $N \times N$ matrices A, B, min-plus product is
$C_{i,j} = \min_{0 \leq k < N}(A_{i,k} + B_{k,j})$

Min-plus matrix multiplication is known to be equivalent to all-pairs shortest paths

Reducing min-plus matrix multiplication to submatrix update-query

$$C_{0,0} = \min(A_{0,0} + B_{0,0}, A_{0,1} + B_{1,0}, \cdots, A_{0,N-1} + B_{N-1,0})$$
$$C_{1,0} = \min(A_{1,0} + B_{0,0}, A_{1,1} + B_{1,0}, \cdots, A_{1,N-1} + B_{N-1,0})$$
$$\cdots$$
$$C_{N-1,0} = \min(A_{N-1,0} + B_{0,0}, A_{N-1,1} + B_{1,0}, \cdots, A_{N-1,N-1} + B_{N-1,0})$$

Reducing min-plus matrix multiplication to submatrix update-query

$C_{0,0}$	$A_{0,0} + B_{0,0}$	$A_{0,1} + B_{1,0}$	$\cdots$	$A_{0,N-1} + B_{N-1,0}$
$C_{1,0}$	$A_{1,0} + B_{0,0}$	$A_{1,1} + B_{1,0}$	$\cdots$	$A_{1,N-1} + B_{N-1,0}$
		$\vdots$		
$C_{N-1,0}$	$A_{N-1,0} + B_{0,0}$	$A_{N-1,1} + B_{1,0}$	$\cdots$	$A_{N-1,N-1} + B_{N-1,0}$

Reducing min-plus matrix multiplication to submatrix update-query

$C_{0,0}$	$A_{0,0}+B_{0,0}$	$A_{0,1}+B_{1,0}$	$\cdots$	$A_{0,N-1}+B_{N-1,0}$
$C_{1,0}$	$A_{1,0}+B_{0,0}$	$A_{1,1}+B_{1,0}$	$\cdots$	$A_{1,N-1}+B_{N-1,0}$
		$\vdots$		
$C_{N-1,0}$	$A_{N-1,0}+B_{0,0}$	$A_{N-1,1}+B_{1,0}$	$\cdots$	$A_{N-1,N-1}+B_{N-1,0}$

Reducing min-plus matrix multiplication to submatrix update-query

	$+B_{0,0}$	$+B_{1,0}$	$\cdots$	$+B_{N-1,0}$
$C_{0,0}$	$A_{0,0}$	$A_{0,1}$		$A_{0,N-1}$
$C_{1,0}$	$A_{1,0}$	$A_{1,1}$		$A_{1,N-1}$
		$\vdots$		
$C_{N-1,0}$	$A_{N-1,0}$	$A_{N-1,1}$		$A_{N-1,N-1}$

- N elements of C can be found with N submatrix updates and N submatrix queries.
- We can then undo all updates and use different elements of B to get N other elements of C, and then repeat this.

Reducing min-plus matrix multiplication to submatrix update-query

1: Initialize $(+, \min)$ update-query DS with A
2: **for** $j = 0$ to $N - 1$ **do**
3: $\quad update([0, N-1][k,k], B[k][j])\ \forall 0 \le k < N$
4: $\quad C[i][j] \leftarrow query([i,i][0, N-1])\ \forall 0 \le i < N$
5: $\quad update([0, N-1][k,k], -B[k][j])\ \forall 0 \le k < N$

Runs in $O(P(N) + N^2(U(N) + Q(N))$ time, where $P(N), U(N), Q(N)$ are worst-case preprocessing, update, and query times resp. over a $N \times N$ matrix

Lower bounds

- We can replace matrix of an update-query data structure to A_1 by doing $update([i,i][j,j], -Q([i,i][j,j]) + A_1[i][j])\ \forall 0 \leq i,j < N$
  - $\Rightarrow$ We can find many matrix multiplications while initializing only once

Lower bounds

- Product of two $KN \times KN$ matrices
  - $\Rightarrow$ block matrix product of two $K \times K$ matrices where each element is a $N \times N$ matrix instead of a number
  - $\Rightarrow O(K^3)$ many $N \times N$ matrix multiplications using schoolbook algorithm
- $\Rightarrow KN \times KN$ min-plus matrix product in $O(P(N) + K^3N^2(U(N) + Q(N)))$ time

Main theorem

- $N \times N$ min-plus matrix multiplication widely believed to not have $O(N^{3-\varepsilon})$ time solution
- If true, then $O(P(N) + K^3N^2(U(N) + Q(N))) > O((KN)^{3-\varepsilon})\ \forall\varepsilon > 0$

Theorem

If min-plus matrix multiplication cannot be done in $O(N^{3-\varepsilon})$ time, then either $U(N)$ or $Q(N) > O(N^{1-\varepsilon})$ for any $\varepsilon > 0$, or $P(N)$ is superpolynomial

- A quadtree has $O(N)$ time updates and queries and $O(N^2)$ time preprocessing.
- Thus, our lower bound is tight up to $o(N^{\varepsilon})$ factors.

Next steps/open questions

For submatrix updates and queries:

- Is sublinear (ex. $O(\frac{N}{\log N})$) update and query time w/ polynomial preprocessing time possible for $(\bigtriangledown, \bigtriangleup) = (+, \min)$?
- Are $O(\log N \log M)$ time updates and queries possible for more operator pairs?
  - i.e. beyond cases where $\bigtriangledown = \bigtriangleup$ and $\bigtriangledown$ is commutative and associative (ex. min, $+$, *, AND)
- If $\bigtriangledown$ is noncommutative, are $O(\text{poly}(N, M))$ updates and queries possible at all?
  - 1D case solved with segment tree $+$ lazy propagation (but lazy part is more complex)

Acknowledgments

I would like to thank

- Jun Wan for his mentorship
- Dr. Gerovitch and MIT PRIMES for making this project possible
- My parents for their support
- You for listening

References

- Timothy M. Chan, Kasper Green Larsen, Mihai Patrascu. Orthogonal Range Searching on the RAM, Revisited. arXiv:1103.5510, 2011.
- Minakshi Banerjee, Sanghamitra Bandyopadhyay, and Sankar K. Pal. A Clustering Approach to Image Retrieval Using Range Based Query and Mahalanobis Distance. In *Rough Sets and Intelligent Systems - Professor Zdzisław Pawlak in Memoriam*, pages 79-91. Springer, 2013.
- Michael A Bender and Martin Farach-Colton. The lca problem revisited. In Latin American Symposium on Theoretical Informatics, pages 88–94. Springer, 2000.
- Changxi Zheng, Guobin Shen, Shipeng Li, and Scott Shenker. Distributed segment tree: Support of range query and cover query over dht. In IPTPS, 2006.
- Stabbing Queries.
  `http://www.cs.nthu.edu.tw/~wkhon/ds/ds10/tutorial/tutorial6.pdf`

References

- Peter M Fenwick. A new data structure for cumulative frequency tables. Software: Practice and Experience, 24(3):327–336, 1994.
- Pushkar Mishra. A new algorithm for updating and querying sub-arrays of multidimensional arrays. arXiv:1311.6093, 2016.
- Nabil Ibtehaz, M. Kaykobad, and M. Sohel Rahman. Multidimensional segment trees can do range queries and updates in logarithmic time. arXiv:1811.01226, 2018.
- Segment Tree - Competitive Programming Algorithms https://cp-algorithms.com/data_structures/segment_tree.html
- Animesh Fatehpuria. 2DRangeSumQuerywithUpdates.cpp https://github.com/animeshf/Competitive_Programming/blob/master/Algorithms/DataStructures/2DSegmentTree

References

François Le Gall. Powers of Tensors and Fast Matrix Multiplication. arXiv:1401.7714, 2014.

Ryan Williams. Faster all-pairs shortest paths via circuit complexity. arXiv:1312.6680v2, 2014.

Virginia Vassilevska Williams and Yinzhan Xu. Truly Subcubic Min-Plus Product for Less Structured Matrices, with Applications. arXiv:1910.04911, 2018.