Sparse Table / Binary Lifting

Arnar Bjarni Arnarson

September 21, 2025

School of Computer Science
Reykjavík University

Range queries

- We have an array A of size n.

Range queries

- We have an array A of size n.
- Given i, j, we want to answer:

Range queries

- We have an array A of size n.
- Given i, j, we want to answer:
    - $\max(A[i], A[i+1], \ldots, A[j-1], A[j])$

Range queries

- We have an array A of size n.
- Given i, j, we want to answer:
  - $\max(A[i], A[i+1], \ldots, A[j-1], A[j])$
  - $\min(A[i], A[i+1], \ldots, A[j-1], A[j])$

Range queries

- We have an array A of size n.
- Given i, j, we want to answer:
  - $\max(A[i], A[i+1], \ldots, A[j-1], A[j])$
  - $\min(A[i], A[i+1], \ldots, A[j-1], A[j])$
  - $\mathrm{sum}(A[i], A[i+1], \ldots, A[j-1], A[j])$

Range queries

- We have an array A of size n.
- Given i, j, we want to answer:
  - $\max(A[i], A[i+1], \ldots, A[j-1], A[j])$
  - $\min(A[i], A[i+1], \ldots, A[j-1], A[j])$
  - $\mathrm{sum}(A[i], A[i+1], \ldots, A[j-1], A[j])$
- We want to answer these queries efficiently, or in other words, without looking through all elements.

Range queries

- We have an array A of size n.
- Given i, j, we want to answer:
  - $\max(A[i], A[i+1], \ldots, A[j-1], A[j])$
  - $\min(A[i], A[i+1], \ldots, A[j-1], A[j])$
  - $\operatorname{sum}(A[i], A[i+1], \ldots, A[j-1], A[j])$
- We want to answer these queries efficiently, or in other words, without looking through all elements.
- Sometimes we also want to update elements.

Another $\log(n)$ idea

- What if we tried something more akin to an array.

Another $\log(n)$ idea

- What if we tried something more akin to an array.
- Could we store $\log(n)$ amounts of data per element somehow?

Another $\log(n)$ idea

- What if we tried something more akin to an array.
- Could we store $\log(n)$ amounts of data per element somehow?
- Yes! For each i we can store the sum on the interval $[i, i + 2^j - 1]$ for $\log$ many j.

Another $\log(n)$ idea

- What if we tried something more akin to an array.
- Could we store $\log(n)$ amounts of data per element somehow?
- Yes! For each i we can store the sum on the interval $[i, i + 2^j - 1]$ for $\log$ many j.
- Then to retrieve a sum from i to j we always take the biggest chunk we can that's stored at i, which will always be at least half.

Another $\log(n)$ idea

- What if we tried something more akin to an array.
- Could we store $\log(n)$ amounts of data per element somehow?
- Yes! For each i we can store the sum on the interval $[i, i + 2^j - 1]$ for $\log$ many j.
- Then to retrieve a sum from i to j we always take the biggest chunk we can that's stored at i, which will always be at least half.
- Then we continue until we reach j, moving i along and collecting the results.

Another $\log(n)$ idea

- What if we tried something more akin to an array.
- Could we store $\log(n)$ amounts of data per element somehow?
- Yes! For each i we can store the sum on the interval $[i, i + 2^j - 1]$ for $\log$ many j.
- Then to retrieve a sum from i to j we always take the biggest chunk we can that's stored at i, which will always be at least half.
- Then we continue until we reach j, moving i along and collecting the results.
- This is what is known as a sparse table.

Sparse tables

- Calculating all of these values takes $\mathcal{O}(n \log(n))$ because we can calculate the values in order of increasing j.

Sparse tables

- Calculating all of these values takes $\mathcal{O}(n\log(n))$ because we can calculate the values in order of increasing j.
- Then when we calculate the sum of $[i, i+2^j-1]$ we just combine the earlier results of $[i, i+2^{j-1}-1]$ and $[i+2^{j-1}, i+2^j-1]$.

Sparse tables

- Calculating all of these values takes $\mathcal{O}(n \log(n))$ because we can calculate the values in order of increasing j.
- Then when we calculate the sum of $[i, i + 2^j - 1]$ we just combine the earlier results of $[i, i + 2^{j-1} - 1]$ and $[i + 2^{j-1}, i + 2^j - 1]$.
- Querying takes $\mathcal{O}(\log(n))$, however updating is slow and difficult.

Sparse tables

- Calculating all of these values takes $\mathcal{O}(n \log(n))$ because we can calculate the values in order of increasing j.
- Then when we calculate the sum of $[i, i + 2^j - 1]$ we just combine the earlier results of $[i, i + 2^{j-1} - 1]$ and $[i + 2^{j-1}, i + 2^j - 1]$.
- Querying takes $\mathcal{O}(\log(n))$, however updating is slow and difficult.
- Why would we then ever use this instead of segment trees?

Binary lifting

- The reason might be is that with sparse tables we can do many things that segment trees can not because of how the results are combined.

Binary lifting

- The reason might be is that with sparse tables we can do many things that segment trees can not because of how the results are combined.
- Let us consider binary lifting in particular.

Binary lifting

- The reason might be is that with sparse tables we can do many things that segment trees can not because of how the results are combined.
- Let us consider binary lifting in particular.
- Suppose we have some function f that rearranges the values $\{0, 1, \ldots, n-1\}$ and we get q queries asking what happens to x if we apply f exactly m times to x.

Binary lifting

- The reason might be is that with sparse tables we can do many things that segment trees can not because of how the results are combined.
- Let us consider binary lifting in particular.
- Suppose we have some function f that rearranges the values $\{0, 1, \ldots, n-1\}$ and we get q queries asking what happens to x if we apply f exactly m times to x.
- The naïve solution is to calculate it every time, giving a time complexity of $\mathcal{O}(qm\mathcal{O}(f))$.

Binary lifting

- The reason might be is that with sparse tables we can do many things that segment trees can not because of how the results are combined.
- Let us consider binary lifting in particular.
- Suppose we have some function f that rearranges the values $\{0, 1, \ldots, n-1\}$ and we get q queries asking what happens to x if we apply f exactly m times to x.
- The naïve solution is to calculate it every time, giving a time complexity of $\mathcal{O}(qm\mathcal{O}(f))$.
- How might we use sparse tables to do better?

Binary lifting ctd.

- Let $f^{[y]}(x)$ denote the result of applying f exactly y times to x

Binary lifting ctd.

- Let $f^{[y]}(x)$ denote the result of applying f exactly y times to x
- For each i we store $f^{[2^j]}(i)$ as a sparse table

Binary lifting ctd.

- Let $f^{[y]}(x)$ denote the result of applying f exactly y times to x
- For each i we store $f^{[2^j]}(i)$ as a sparse table
- Then we can compute these in increasing order of j, calculating $j = 1$ using f itself and then for larger j letting $f^{[2^j]}(x) = f^{[2^{j-1}]}(f^{[2^{j-1}]}(x))$

Binary lifting ctd.

- Let $f^{[y]}(x)$ denote the result of applying f exactly y times to x
- For each i we store $f^{[2^j]}(i)$ as a sparse table
- Then we can compute these in increasing order of j, calculating $j = 1$ using f itself and then for larger j letting $f^{[2^j]}(x) = f^{[2^{j-1}]}(f^{[2^{j-1}]}(x))$
- Thus we can precompute the table in $\mathcal{O}(n(\mathcal{O}(f) + \log(n)))$ and each query takes $\mathcal{O}(\log(m))$, a much better time complexity

Sparse table example

7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

												$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8												$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7											$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10										$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12									$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8								$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8	9							$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8	9	11						$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8	9	11	4					$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8	9	11	4	9				$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8	9	11	4	9	8			$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8	9	11	4	9	8	7		$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

8	7	10	12	8	9	11	4	9	8	7	6	$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

18	19	18	21	19	13	20	12	16	14	7	6	$j = 2$
8	7	10	12	8	9	11	4	9	8	7	6	$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

37	32	38	33	35	27	27	18	16	14	7	6	$j = 3$
18	19	18	21	19	13	20	12	16	14	7	6	$j = 2$
8	7	10	12	8	9	11	4	9	8	7	6	$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Sparse table example

`query(1, 8) = 19 + 9 + 2`

37	32	38	33	35	27	27	18	16	14	7	6	$j = 3$
18	$\boxed{19}$	18	21	19	13	20	12	16	14	7	6	$j = 2$
8	7	10	12	8	$\boxed{9}$	11	4	9	8	7	6	$j = 1$
7	1	6	4	8	0	9	$\boxed{2}$	2	7	1	6	$j = 0$

Sparse table example

`query(0, 9) = 37 + 9`

37	32	38	33	35	27	27	18	16	14	7	6	$j = 3$
18	19	18	21	19	13	20	12	16	14	7	6	$j = 2$
8	7	10	12	8	9	11	4	9	8	7	6	$j = 1$
7	1	6	4	8	0	9	2	2	7	1	6	$j = 0$

Example problem: Stikl

- https://open.kattis.com/problems/stikl