

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

In the name of Allah, Most Gracious, Most Merciful

CSE 4303

Data Structures

Topic: Sparse Table, Fenwick Tree



Asaduzzaman Herok
Lecturer | CSE | IUT
asaduzzaman34@iut-dhaka.edu



Problem Scenario

Think of a university of 10 thousand ex-students. Their results and other informations are sorted by their student IDs. Now it is very frequent that authority want to know from roll x to roll y what is the maximum CGPA.

What to do?

- Linear Search?
- Binary Search?
- Sorting the data based on CGPA?

Is there a way to get the result without changing the relative order of the data?

These types of query / questions are called ***Range Query***.

Example: Range Maximum Query, Range Minimum Query, Range Sum Query, Range GCD / LCM Query

Range Sum Problem

Given array of n elements $A = \{ a_1, a_2, \dots \dots \}$, in each query (L, r) it is asked to give the sum of elements from a_1 to a_r . That is return $\{a_1+a_{1+1}+a_{1+2}+ \dots \dots + a_{r-2}+a_{r-1}+a_r\}$

Range Minimum Problem

Given array of n elements $A = \{ a_1, a_2, \dots \dots \}$, in each query (L, r) it is asked to give the minimum element among a_1 to a_r . That is return minimum $\{a_1, a_{1+1}, a_{1+2}, \dots \dots, a_{r-2}, a_{r-1}, a_r\}$

Generalised Way

Given array of n elements $A = \{ a_1, a_2, \dots \dots \}$, in each query (L, r) it is asked to give the $F(\{a_1, a_{1+1}, a_{1+2}, \dots \dots, a_{r-2}, a_{r-1}, a_r\})$.

Naive Solutions

- Linear Search on every query starting from a_1 to a_r .
 - Complexity on each $O(r-l+1)$ i.e $O(n)$
- Precalculation for once for every pair of (l,r) .
 - Need $O(n^2)$ space and time.
 - But can answer query in constant time i.e $O(1)$

What to do then?

- Sparse Table
- Fenwick Tree
- Segment Tree
- Square Root Decomposition
- Square Root Tree
- Mo's Algorithm

Our Discussion will be restricted upto Segment tree



Sparse Table

A sparse table is a data structure that can answer some range query problems, such as range minimum query problem, in $O(1)$ time. Range sum in $O(\log n)$

Pros:

- Building time complexity $O(n \log n)$.
- Space complexity $O(n \log n)$
- Supported query function:
 - Element in the array supports the associative property i.e $x \circ (y \circ z) = (x \circ y) \circ z$.
[Here \circ is a operator]
 - $F(a, b, c, d) = F(a, b) \circ F(c, d)$, from the non-overlapping segments. Time complexity $O(\log n)$
 - $F(a, b, c, d) = F(a, b, c) \circ F(b, c, d)$, from the overlapping segments. Time complexity $O(1)$ i.e constant time.

Cons:

- Can't work with update i.e data sequence must be immutable
- Update operation would cause $O(n \log n)$ time complexity to rebuild.

Sparse Table

Intuition:

- Any non-negative number can be uniquely represented as a sum of decreasing powers of two.
 - Example: $13 = (1101)_2 = 8 + 4 + 1$
 - Number of powers of two needed is $\log_2(n)$
- So a range of m elements can also be represented as union of continuous sub-segments of size decreasing powers of two
 - Example: $[2, 14] = [2, 9] \cup [10, 13] \cup [14, 14]$
Size: 13 8 4 1
 - If we already know these smaller ranges value then the large range can be computed.

Idea:

- Pre-compute all the answers of range with length equal to some powers of 2
- While querying break the query segments into some segments of powers of 2 and use their value to compute for the query segment.

Sparse Table Pre-Computation

- Declare a 2D array $sparse[n][k+1]$ where $k = \text{floor}(\log_2(n)+1)$
- $sparse[i][j]$ keeps the answer for segment / range starting at i and ending at $i + 2^j - 1$
- The recurrence can be written as:
 - $sparse[i][j] = F(sparse[i][j-1], sparse[i + 2^{j-1}][j-1])$
 - Example: $sparse[4][3] = \min(sparse[4][2], sparse[4+4][2])$
Range: $[4, 11] = \min([4, 7], [8, 11])$
- Base case:
 - $sparse[i][0] = F(a_i)$
 - Example: $sparse[4][0] = \min(a_4) = a_4$
Range: $[4, 4] = a_4$

Sparse Table Pre-Computation

Algorithm 4: Sparse Table Construction for Range Minimum Query Problem

Data: A number array A with size n

Result: The minimum values of all possible power-of-two ranges of A

Function preprocessSparse(A, n):

 Initialize an $n \times (\log_2 n + 1)$ 2D array *sparse*;

for $i = 1$ **to** n **do**

 | $\text{sparse}[i][0] = A[i]$;

end

for $j = 1$ **to** $\log_2 n$ **do**

for $i = 1$ **to** n **do**


 | $\text{sparse}[i][j] = \min(\text{sparse}[i][j-1], \text{sparse}[i + 2^{j-1}][j-1])$;

end

end

return *sparse*;

	2^0	2^1	2^2	2^3
0				
1				
2			★	
3				
4				
5				
6				
7				

							
31	41	59	26	53	58	97	93
0	1	2	3	4	5	6	7

Sparse Table Query

Range Sum Query: (l, r)

Non-Overlapping Case

- Compute the power $k = \text{floor}(\log_2 (r-l+1))$
 - Example: Range $[4, 9]$, $k = \text{floor}(\log_2 (9-4+1 = 6)) = 2$
- While $(l \leq r)$ do $\text{ans} += \text{sparse}[l][k]$, $l += 2^k$, $k = \text{floor}(\log_2 (r-l+1))$
 - Example: Range $[4, 9]$
 $\text{ans} += \text{sparse}[4][2] \quad // [4, 7]$
 $l = 4 + 2^2 = 8$
 $k = \log_2(9-8+1 = 2) = 1$
 $\text{ans} += \text{sparse}[8][1] \quad // [8, 9]$
 $l = 8 + 2^1 = 9$

Range Min Query: (l, r)

Overlapping Case

- Compute the power $k = \text{floor}(\log_2 (r-l+1))$
 - Example: Range $[4, 9]$, $k = \text{floor}(\log_2 (9-4+1 = 6)) = 2$
- $\text{ans} = \min(\text{sparse}[l][k], \text{sparse}[R-2^k+1][k])$
 - Example: Range $[4, 9]$
 $\text{ans} = \min(\text{sparse}[4][2], \text{sparse}[9-2^2+1 = 6][2])$
 $\quad \quad \quad [4, 7] \quad \quad \quad [6, 9]$

Fenwick Tree

Fenwick tree is a data structure which:

- Calculates the value of function F in the given range $[1, r]$ (i.e. $F(\{a_1, a_{1+1}, \dots, a_{r-1}, a_r\})$ in $O(\log_2 n)$ time;
- Updates the value of an element of A in $O(\log_2 n)$ time;
- Requires $O(n)$ memory, or in other words, exactly the same memory required for A
- Build time is also $O(n \log_2 n)$
- Also called Binary Indexed Tree, or just BIT abbreviated.

The most common application of Fenwick tree is calculating the sum of a range.

$$F(\{a_1, a_{1+1}, \dots, a_{r-1}, a_r\}) = a_1 + a_{1+1} + \dots + a_{r-1} + a_r$$

F should support both $F(a, b, c, d) = F(a, b) \circ F(c, d)$ and $F(c, d) = F(a, b, c, d) \circ F(a, b)$,
 \circ denotes some kind of operators.

Example: $\text{Sum}(a, b, c, d) = \text{Sum}(a, b) + \text{Sum}(c, d)$ and $\text{Sum}(c, d) = \text{Sum}(a, b, c, d) - \text{Sum}(a, b)$
 $\text{Xor}(a, b, c, d) = \text{Xor}(a, b) \text{ xor } \text{Xor}(c, d)$ and $\text{Xor}(c, d) = \text{Xor}(a, b, c, d) \text{ xor } \text{Xor}(a, b)$

Fenwick Tree

Basic Idea:

- Each integer can be represented as a sum of powers of two.
 $13 = (1101)_2 = 2^3 + 2^2 + 2^0 = 8+4+1$
- The answer of segment[1, 13] can be found by combining the range $[1,8] \cup [9,12] \cup [13,13]$
- That is the same way, a cumulative values can be represented as a sum of sets of sub segments value .
- A 1D array of size n is enough to store these sub-segment values.
 $\text{BIT}[1, 2, \dots, n]$
- Let's i be an index of BIT and k be the position last non zero bit of the binary representation of i .
 - Example: $i = 13 = (1101)_2, k = 0$
 $i = 6 = (110)_2, k = 1$
 $i = 8 = (1000)_2, k = 3$
- $\text{BIT}[i]$ will keep the segment value of segment $[i-2^k+1, i]$
 - Example: $\text{BIT}[13] = [13, 13]$
 $\text{BIT}[6] = [5, 6]$
 $\text{BIT}[8] = [1, 8]$
- So to get the segment value of $[1, 13]$ we have to union $\text{BIT}[8] \cup \text{BIT}[10] \cup \text{BIT}[13]$
 $\text{BIT}[(1000)_2] \cup \text{BIT}[(1100)_2] \cup \text{BIT}[(1101)_2]$

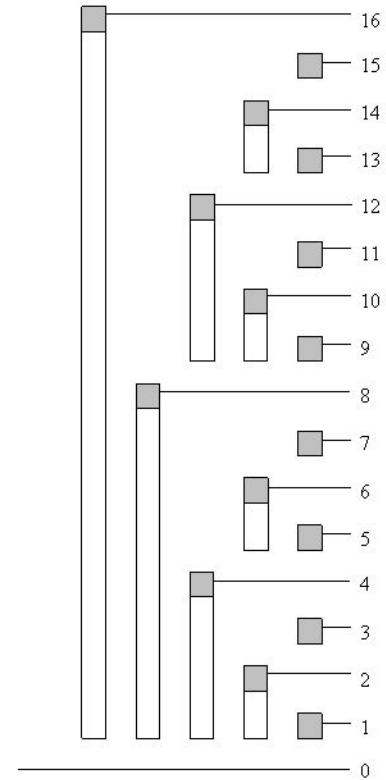


Image 1.3 – tree of responsibility for indices (bar shows range of frequencies accumulated in top element)

Fenwick Tree

Basic Idea:

- We need to find a way to jump from these segments to segments. Like from 13 to 12 to 8 by setting the least significant bit to 0.

Isolating the last BIT:

- Let i be $a1b$ as a representation of i in binary where a is the prefix from the last significant bit and b is all zeros.
 - Example: $20 = (10100)_2$, $a = 10$, $b = 00$
 $a1b$
- $(a1b)^-$ is the inversion of bits.
 - $(20)^- = (10100)_2^- = (01011)_2$
- $-num = (a1b)^- + 1 = a^-0b^- + 1 = a^-0(0...0)^- + 1$
 $= a^-0(1...1) + 1 = a^-1(0...0) = a^-1b$.
- If we do bitwise **and** of $a1b$ with a^-1b we get $(0..010..0)_2$

Example: $20 \& -20 = 4$

$$(10100)_2 \& (01100)_2 = (100)_2$$

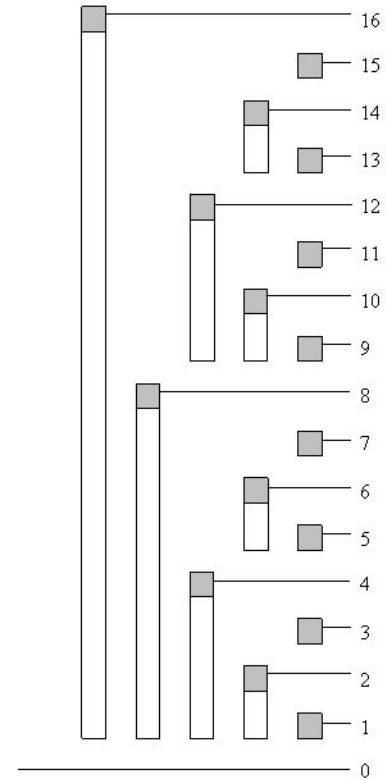
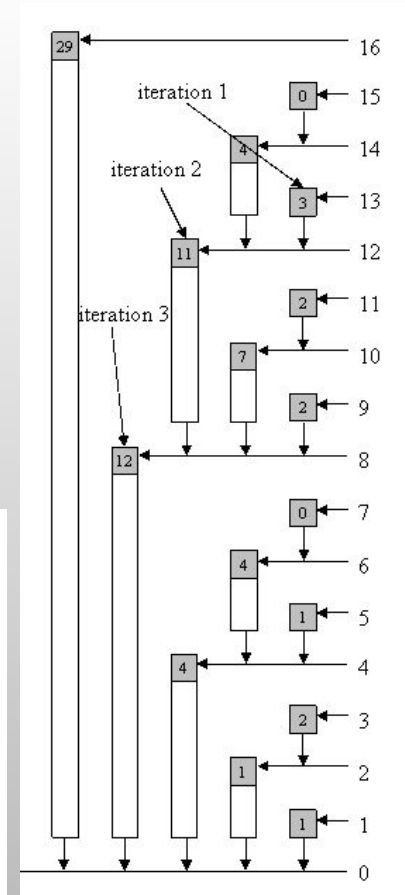


Image 1.3 – tree of responsibility for indices (bar shows range of frequencies accumulated in top element)

Fenwick Tree Read Operation on Segment Sum

```
int read(int idx) {  
    int sum = 0;  
    while (idx > 0) {  
        sum += tree[idx];  
        idx -= (idx & -idx);  
    }  
    return sum;  
}
```

ITERATION	IDX	POSITION OF THE LAST BIT	IDX & -IDX	SUM
1	13 = 1101	0	1 (2^0)	3
2	12 = 1100	2	4 (2^2)	14
3	8 = 1000	3	8 (2^3)	26
4	0 = 0	—	—	—

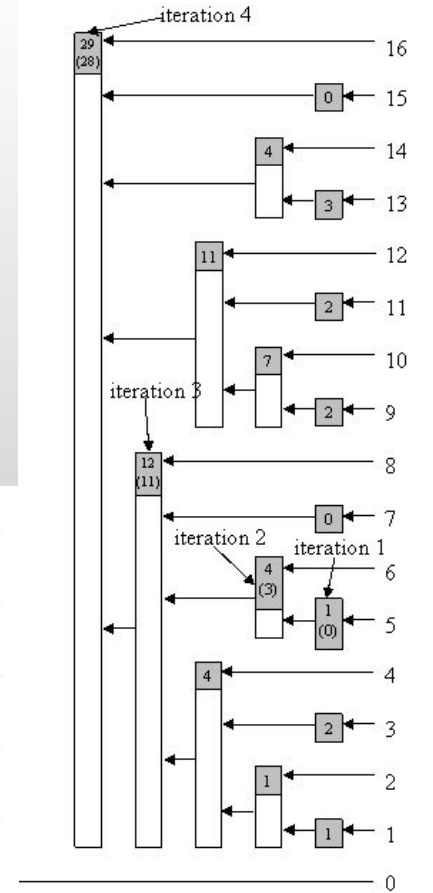


Fenwick Tree Update Operation on Segment Sum

- The update operation is to increase or decrease the certain value at index i of the given array A .
- Increase or decrease should be reflected in the affected range sums in the BIT.
- To find the affected segments we have to add the last bit of i to itself; and, repeat while i is less than or equal to n

```
void update(int idx, int val)
{
    while (idx <= MaxIdx) {
        tree[idx] += val;
        idx += (idx & -idx);
    }
}
```

ITERATION	IDX	POSITION OF THE LAST BIT	IDX & -IDX
1	5 = 101	0	1 (2^0)
2	6 = 110	1	2 (2^1)
3	8 = 1000	3	8 (2^3)
4	16 = 10000	4	16 (2^4)
5	32 = 100000	—	—



Fenwick Tree Range Query (L, r)

Recall the property:

$$F(c, d) = F(a, b, c, d) \circ F(a, b)$$

- $Read(r)$ gives the segment value of $[1, r]$
 $Read(L-1)$ gives the segment value of $[1, L-1]$
- $Query(L, r) = Read(r) \circ Read(L-1)$
 $[1, r] \circ [1, L-1]$

Range / Segment Sum $Query(L, r)$

- $Read(r)$ gives the segment value of $[1, r]$
 $Read(L-1)$ gives the segment value of $[1, L-1]$
- $Query(L, r) = Read(r) - Read(L-1)$
 $[1, r] - [1, L-1]$



Acknowledgements

[Sparse Table - Algorithms for Competitive Programming \(cp-algorithms.com\)](https://cp-algorithms.com)

[Fenwick Tree - Algorithms for Competitive Programming \(cp-algorithms.com\)](https://cp-algorithms.com)

[Binary Indexed Trees \(topcoder.com\)](https://topcoder.com)