Data Structure (III)

Chin Ka Wang {rina__owo}

2025-07-25

Content

1. Sparse Table
   - Range Maximum/Minimum Query
   - Binary Lifting
2. Fenwick Tree
   - Lowbit function

Sparse Table

What is a Sparse Table?

- A data structure based on **Dynamic Programming** and **Binary Lifting** to answer ***Range Minimum/Maximum Query*** **(RMQ) Problems** in **O(1)** time.
- Works only for **immutable datum** (no updates).
- Preprocessing time: **O(N log N)**

Range Maximum/Minimum Query

Given an array A of N integers and Q queries.

Given L and R for each query, find the maximum value in A[L], A[L + 1], …, A[R].

E.g. A = {3, 4, 1, 5, 2}

L = 1, R = 2 → max value = 4

L = 2, R = 4 → max value = 5

L = 4, R = 5 → max value = 5

Range Maximum/Minimum Query

Naive Solution: Linear Search

For every query, loop from L to R and take max.

Time Complexity: **O(QN)**

TLE when QN is large ☹

Range Maximum/Minimum Query

DP Solution:

Let dp[i][j] = max element from arr[i] to arr[j].

$$dp[i][j] = \max(dp[i][j-1], arr[j])$$

Time Complexity: **$O(N^2 + Q)$**
Space Complexity: **$O(N^2)$**

Still TLE or MLE if N is large ☹.

Range Maximum/Minimum Query

We can use **Sparse Table** instead.

Range Maximum/Minimum Query

We can use **Sparse Table** instead.

It is a data structure based on **DP** and **Binary Lifting**.

Binary Lifting

When we are recursing, if the state space is very large and the usual linear recursion cannot meet the requirements of time and space complexity, then we can use the method of binary lifting to recurse only the values at the integer power position of k (often 2) in the state space as representatives.

When values at other positions are needed, we use the property that **"any integer can be expressed as the sum of several powers of k"** to use the previously calculated representative values to piece together the required values. (Recall the binary grouping method in DP(III))

Sparse Table

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
x=3	8							
x=2	4	4	5	8	8			
x=1	4	4	2	2	5	8	8	
x=0	2	4	2	1	2	5	8	0

Sparse Table

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
x=3	8							
x=2	4	4	5	8	8			
x=1	4	4	2	2	5	8	8	
x=0	2	4	2	1	2	5	8	0

Represented with ST[i][x]

find(1, 8) → max(ST[1][2], ST[5][2])

find(2, 3) → ST[2][1]

find(3, 7) → max(ST[3][1], ST[4][2])

or max(ST[3][2], ST[6][1])

Sparse Table

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
x=3	8							
x=2	4	4	5	8	8			
x=1	4	4	2	2	5	8	8	
x=0	2	4	2	1	2	5	8	0

Represented with ST[i][x]

find(1, 8) → max(ST[1][2], ST[5][2])

find(2, 3) → ST[2][1]

find(3, 7) → max(ST[3][1], ST[4][2])

or max(ST[3][2], ST[6][1])

Overlapping of subinterval doesn't affect max/min result!!

Sparse Table

Let k be the maximum integer such that $R - L + 1 \geq 2^k$

$[L, L + 2^k - 1]$ and $[R - 2^k + 1, R]$ must cover all positions from L to R

$\therefore$ find(L, R) = max(f(L, k), f(R - 2^k + 1, k))

Sparse Table

Pseudocode

```
precompute()
  for i = 1 to N
    ST[i][0] = A[i]
  for x = 0 to ⌊log₂(N)⌋ - 1
    for i = 1 to N - (2^(x + 1) - 1)
      ST[i][x + 1] = min(ST[i][x], ST[i + 2^x][x])
```

Sparse Table

Pseudocode

```
query(L, R)
  k = ⌊log₂(R - L + 1)⌋
  return min(ST[L][k], ST[R - 2^k + 1][k])
```

Sparse Table

The reason why we can have O(1) query is that overlapped ranges does not affect the result of min value.

Therefore, as long as the value of an operation would not be affected by overlapped ranges, we can have O(1) query using sparse table.

- max / min / and / or / gcd
- any operation that is idempotent

Sparse Table

Although we cannot have O(1) query for non-idempotent operations like sum / product / xor, we can still have O(log N) query, as we only need at most O(log N) values from the sparse table.

E.g. L = 7, R = 19 → R - L + 1 = 13 → 1101(2)

We need [7, 14], [15, 18] & [19, 19] (i.e. ST[7][3], ST[15][2] & ST[19][0])

Sparse Table

[**Z0103** Range Maximum Query]

[**Z0104** GCD Interval]

[**Z0105** [JRKSJ R1] JFCA]

[**Z0106** [SCOI 2007] 降雨量]

Content

1. Sparse Table
   - Range Maximum/Minimum Query
   - Binary Lifting
2. Fenwick Tree
   - Lowbit function

Fenwick Tree

What is a Fenwick Tree?

- Fenwick Tree is also called Binary Indexed Tree (BIT).
- It supports **point updates** and **range queries**.
- Preprocessing time: **O(N log N)**
- Query time: **O(log N)**

Fenwick Tree

Given an array A of N integers and Q queries.

Given L and R for each query, find the sum of A[L], A[L + 1], …, A[R].

E.g. A = {3, 4, 1, 5, 2}

L = 1, R = 2 → sum = 7

L = 2, R = 4 → sum = 10

L = 4, R = 5 → sum = 7

Fenwick Tree

Easy. Prefix sum.

Fenwick Tree

Easy. Prefix sum.

But what if we have update of data?

Fenwick Tree

A = {3, 4, 1, 5, 2}

L = 2, R = 4 → sum = 10

Update A[3] to 3

L = 2, R = 4 → sum = 12

Fenwick Tree

Naïve Solution:

Directly modify the element in the array and iterate through the required elements for the answer.

Fenwick Tree

Naïve Solution:

Directly modify the element in the array and iterate through the required elements for the answer.

Time Complexity: **O(QN)**

TLE!

Fenwick Tree

2	4	2	1	2	5	8	0

Fenwick Tree

<table>
<tr><td colspan="2">6</td><td colspan="2">3</td><td colspan="2">7</td><td colspan="2">8</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>1</td><td>2</td><td>5</td><td>8</td><td>0</td></tr>
</table>

Fenwick Tree

<table>
<tr><td colspan="4">9</td><td colspan="4">15</td></tr>
<tr><td colspan="2">6</td><td colspan="2">3</td><td colspan="2">7</td><td colspan="2">8</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>1</td><td>2</td><td>5</td><td>8</td><td>0</td></tr>
</table>

Fenwick Tree

<table>
<tr><td colspan="8">26</td></tr>
<tr><td colspan="4">9</td><td colspan="4">15</td></tr>
<tr><td colspan="2">6</td><td colspan="2">3</td><td colspan="2">7</td><td colspan="2">8</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>1</td><td>2</td><td>5</td><td>8</td><td>0</td></tr>
</table>

Fenwick Tree

<table>
<tr><td colspan="8" align="center">26</td></tr>
<tr><td colspan="4" align="center">9</td><td colspan="4" align="center">15</td></tr>
<tr><td colspan="2" align="center">6</td><td colspan="2" align="center">3</td><td colspan="2" align="center">7</td><td colspan="2" align="center">8</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>1</td><td>2</td><td>5</td><td>8</td><td>0</td></tr>
</table>

Any prefix sum can be easily calculated by combinations of subintervals.

Number of subintervals would be at most **lg(N)**.

Fenwick Tree

<table>
<tr><td colspan="8">26</td></tr>
<tr><td colspan="4">9</td><td colspan="4">15</td></tr>
<tr><td colspan="2">6</td><td colspan="2">3</td><td colspan="2">7</td><td colspan="2">8</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>1</td><td>2</td><td>5</td><td>8</td><td>0</td></tr>
</table>

However, some subintervals is never used.

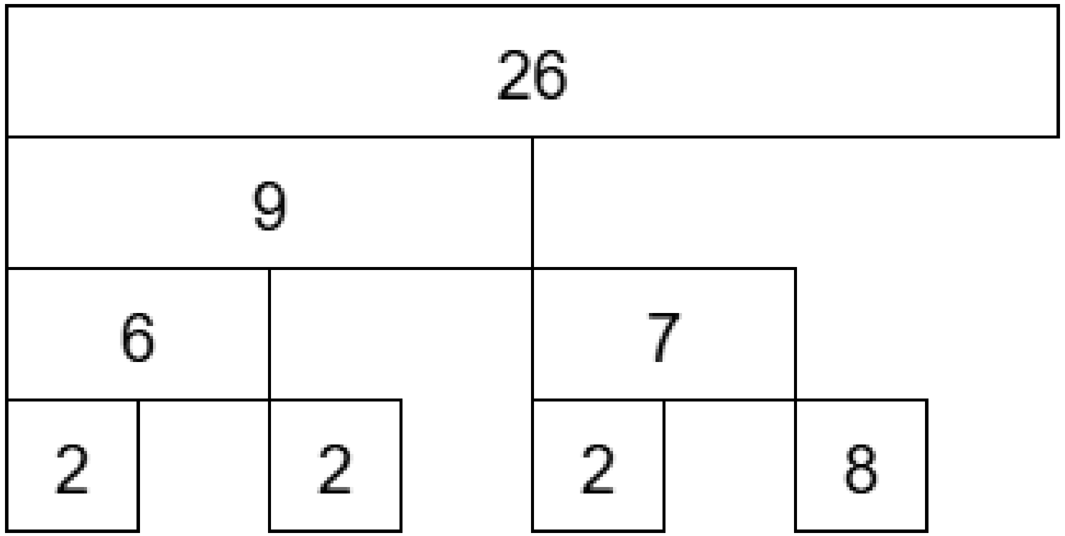
Fenwick Tree

<table>
<tr><td colspan="8">26</td></tr>
<tr><td colspan="4">9</td><td colspan="4">15</td></tr>
<tr><td colspan="2">6</td><td colspan="2">3</td><td colspan="2">7</td><td colspan="2">8</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>1</td><td>2</td><td>5</td><td>8</td><td>0</td></tr>
</table>

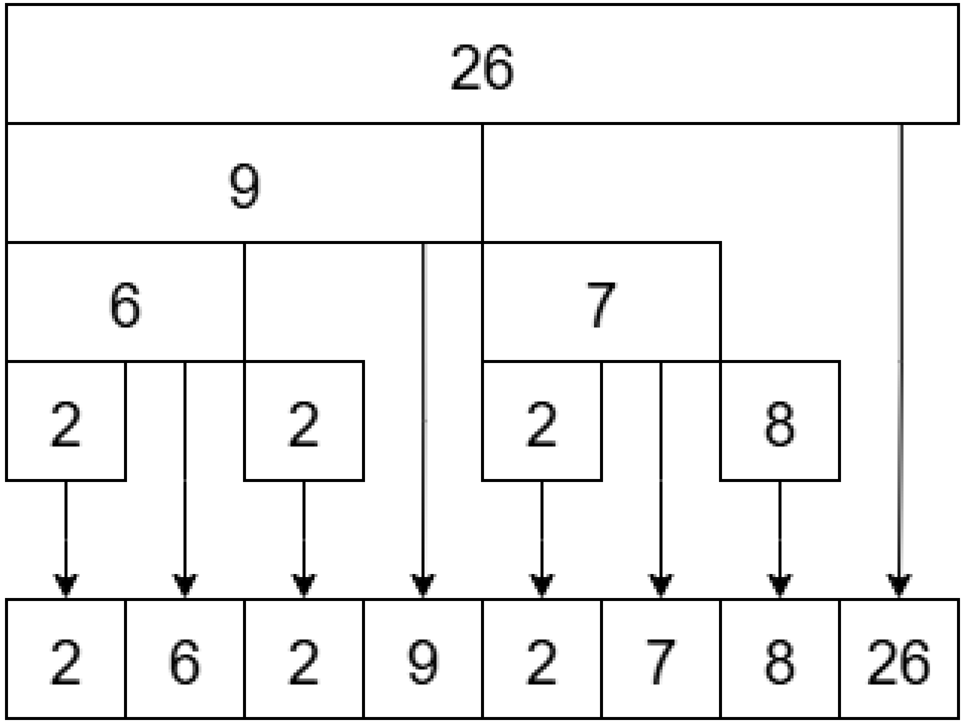
However, some subintervals is never used.

They are the even indices of each layer.

Fenwick Tree



Fenwick Tree



Fenwick Tree

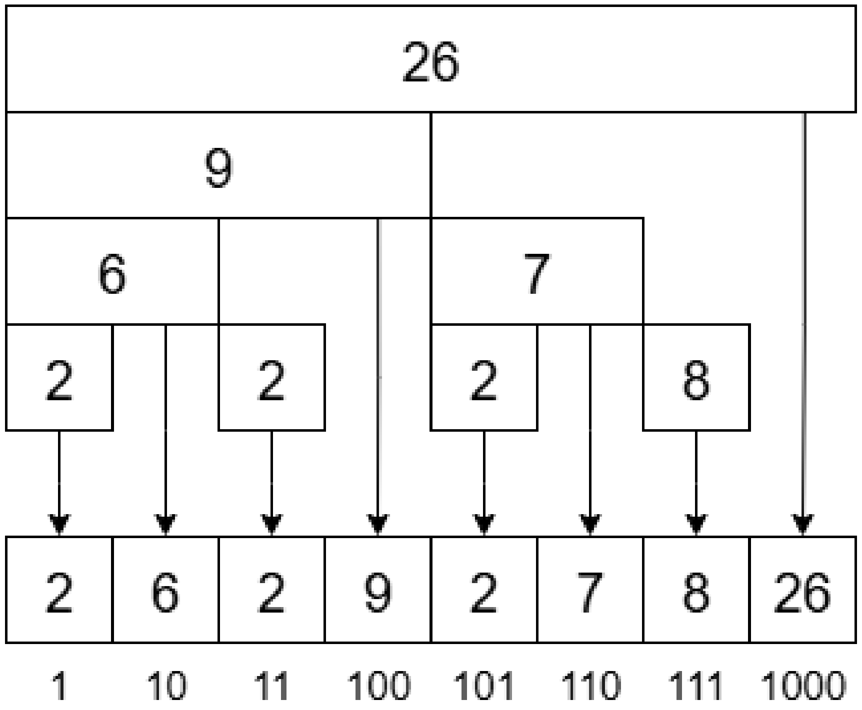
The size of the array is actually unchanged!

Fenwick Tree

The size of the array is actually unchanged!

But how can we actually index them?

Fenwick Tree



Lowbit function

Let lowbit(x) be the value of the rightmost “1” in binary representation of x.

E.g. x = 22 = 10110(2) , lowbit(x) = 00010(2) = 2

Lowbit function

How to compute lowbit(x)?

Lowbit function

How to compute lowbit(x)?

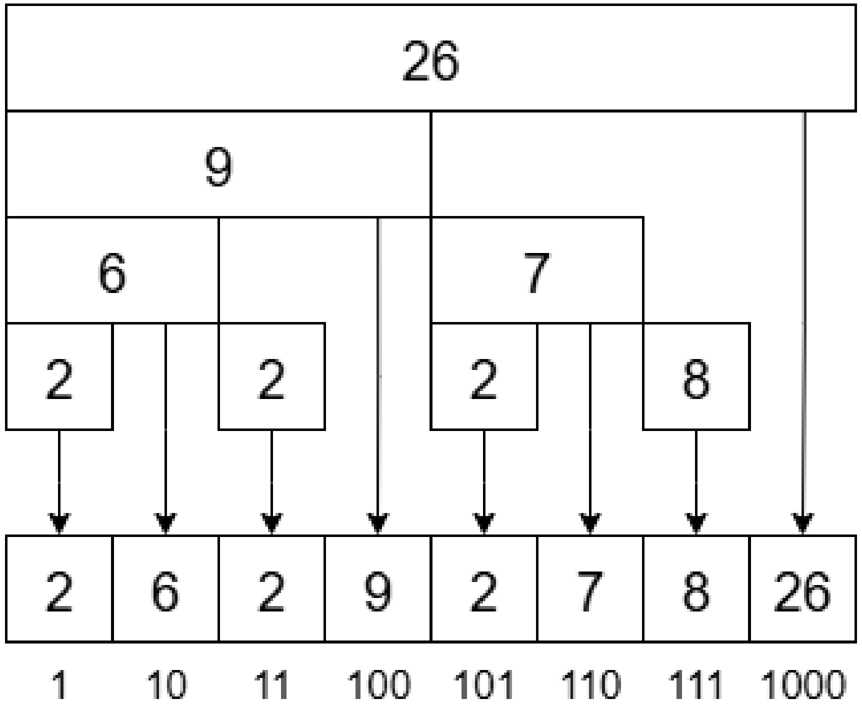
lowbit(x) = x & -x

Time Complexity: **O(1)** !

Lowbit function

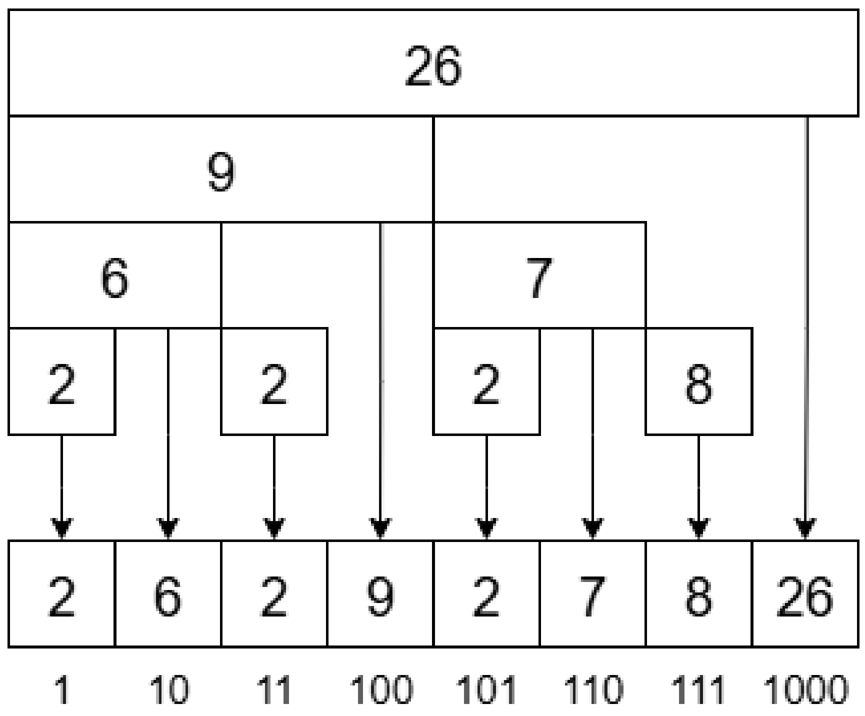
```
int lowbit(int x)
{
    return x&(-x);
}
```

Lowbit function



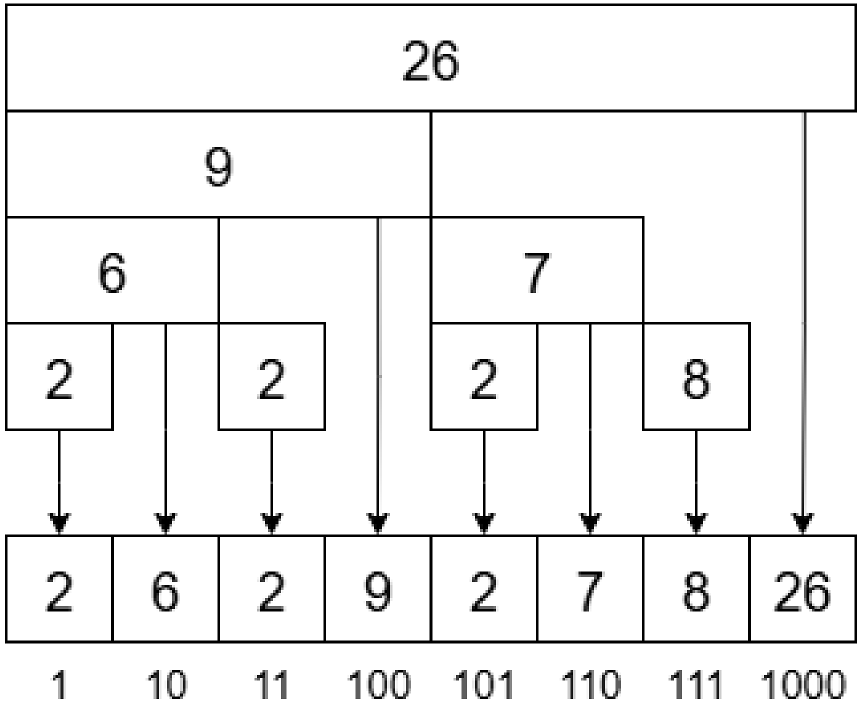
In BIT, node x stores the information of interval [x - lowbit(x) + 1, x]!

Fenwick Tree



How to update the tree?

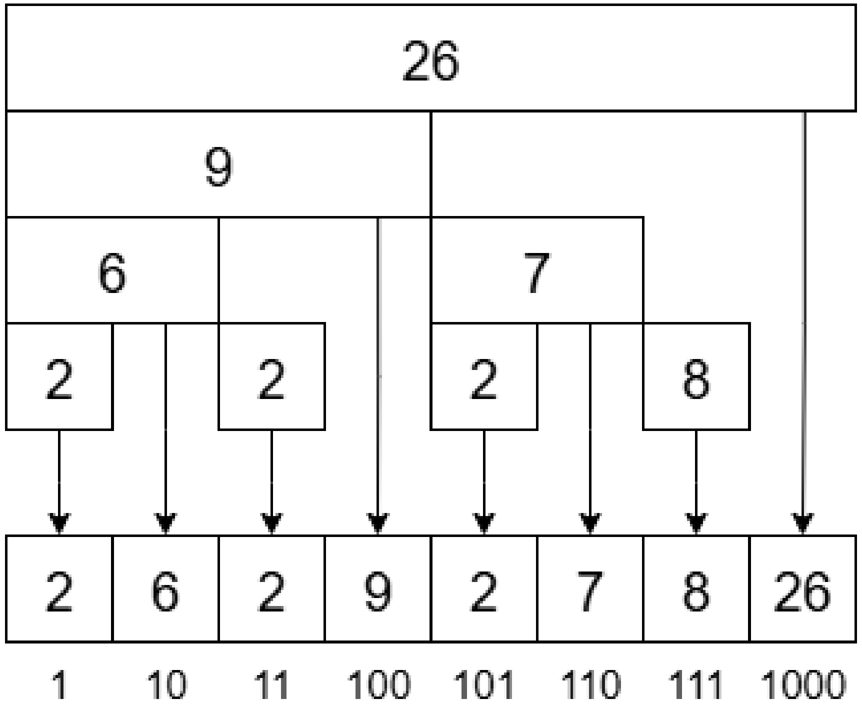
Fenwick Tree



How to update the tree?

By observation, you will find that the length of next BIT[i] we need to update would be double of the current BIT[i].

Fenwick Tree



How to update the tree?

By observation, you will find that the length of next BIT[i] we need to update would be one of current BIT[i] more.

i.e. i += i & -i

Fenwick Tree

Pseudocode

```
add(id, val)
  while id ≤ N
    Node[id] += val
    id += id & -id
```

```
sum(id)
  res = 0
  while id > 0
    res += Node[id]
    id -= id & -id
  return res
```

Lowbit function

Z0107 Fenwick Tree I

Z0108 Fenwice Tree II

Z0109 [GZOI 2017] 配對統計

Z0110 [NOIP-S 2013] 火柴排隊

Z0111 [SHOI 2009] 會場預約

Q&A