SoC 2023: Competitive Programming
Week-6: Range Queries

Mentor: Virendra Kabra

Summer 2023

Contents

1 Range Queries

Given an array `a` of size n, answer multiple queries of the form (l, r). Common examples are sum and minimum of `a[l...r]`. Some problems may also have updates to the array.

2 Prefix arrays

For an operation f, $p[i] = f(a[0, \ldots, i])$. Example: sum

a	1	3	0	-2	5
p	1	4	4	2	7

Can be used for

- Any operation with $(0, r)$ queries. Examples: max, min of $a[0, \ldots, r]$.
- Invertible operations with (l, r) queries. Examples: $\sum_{i=l}^{r} a[i] = \sum_{i=0}^{r} a[i] - \sum_{i=0}^{l-1} a[i]$, product.

Above queries are $O(1)$. Updates are costly ($O(n)$). Good with immutable arrays.
Code.

3 Sparse Table

Reference.

i=4 -> 0
3
2^0

<table>
<tr><td>1</td><td>3</td><td>0</td><td>-2</td><td>5</td></tr>
<tr><td colspan="2">4</td><td colspan="2">-2</td><td>5</td></tr>
<tr><td colspan="4">2</td><td>5</td></tr>
</table>

- Any number can be uniquely expressed as sum of distinct powers of two. This follows from binary representation of the number. For example, $22 = (10110)_2 = 2^4 + 2^2 + 2^1$.
- Similarly, any interval $[l, r]$ can be expressed as union of intervals with lengths being distinct powers of two. For example, $[5, 26] = [5, 20] \cup [21, 25] \cup [25, 26]$.
- Idea of sparse tables is to precompute all range queries with length being powers of two.
- A 2-D array st is used, with $st[i][j]$ holding the result for $[j, j + 2^i - 1]$ (length 2^i).
  - For an array with n elements, $2^i - 1 < j + 2^i - 1 < n$, so the first dimension is $\lfloor \log_2 n \rfloor$
  - $[j, j + 2^i - 1] = [j, j + 2^{i-1} - 1] \cup [j + 2^{i-1}, j + 2^i - 1]$. Recurrence:

(j+2^i - 1) - j + 1 = 2^i

$$st[i][j] = f(st[i-1][j], st[i-1][j + 2^{i-1}])$$

Above queries are $O(\log n)$. Updates would require recomputation of st. Again, good with immutable arrays. Works with non-invertible functions.
Code.

[2, 5] = 5 - 2 + 1

[2,5] = [2,3] U [4,5]

r-j+1 = 2^{i-1}
r = j+2^{i-1}-1

R - (j+2^{i-1}) +1 = 2^{i-1}
R = j+2^i - 1

4 Fenwick Tree

Aka Binary Indexed Tree (BIT). References - cp-algos, gfg.

- We saw that precomputing results of certain intervals helps in answering queries faster.
- Suppose these are computed into an array BIT. Let the interval be $[g(i), i]$. Then, with sum as an example, $BIT[i] = \sum_{j=g(i)}^{i} a[j]$.
- $g(i) = i$ makes $BIT = a$. Costly queries.
- $g(i) = 0$ makes $BIT = p$. Costly updates.
- Pseudocode for updates and queries looks like

[0, r]

```
query(int r):
    res = 0
    while (r >= 0):      // [g(r), r] U [g(g(r)-1), g(r)-1] U ...
        res += t[r]
        r = g(r) - 1
    return res
```

[...., g(g(r)-1) - 1]

a[i] += delta

i in [g(j), j]

```
increase(int i, int delta):
    for all j with g(j) <= i <= j:
        t[j] += delta
```

- As we saw, every number can be uniquely represented as a sum of distinct powers of two. Here, length of $[g(i), i]$ is the smallest power of two in this unique representation. So, $g(i) = i - (i\& - i) + 1$, where $i\& - i$ gives the least significant set bit of i:

$$\begin{aligned} i &= (0\ldots0)10110 \\ -i &= (1\ldots1)01010 \\ i\& - i &= (0\ldots0)00010 \end{aligned}$$

i=22
i= 10110
i'=01001
i'+1 = 01010

[g(i),i]
i-g(i)+1 = i&-i
g(i) = i - (i&-i) + 1

We use 1-indexed implementation. Example:

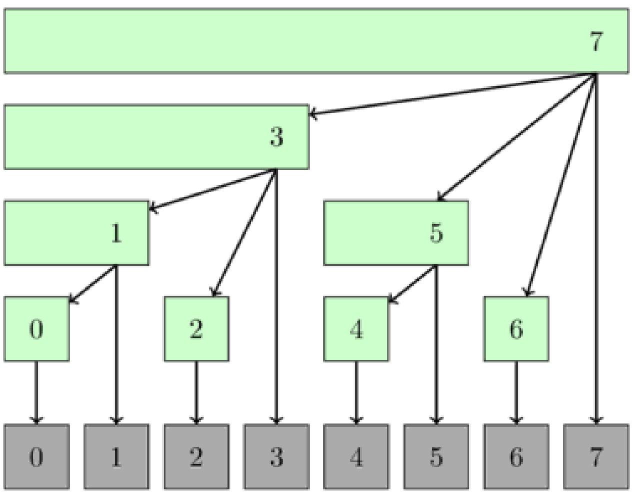
$$a[1\ldots13] = a[1\ldots(1101)_2] = a[13\ldots13] \cup a[9\ldots12] \cup a[1\ldots8]$$

14 - 2 + 1 = 13

- Updates: We want to increase $a[13]$ by 5. Then, we need to add 5 to all $BIT[i]$ that have $a[13]$ as part of their sums. For example, $[g(14), 14] = [13, 14]$ but $[g(15), 15] = [15, 15]$. So, we add 5 to $BIT[14]$. The idea is to repeatedly add the least significant set bit:

$$13 \xrightarrow{[1101,?]} 14 \xrightarrow{[1110,?]} 16\ldots$$

[13,r] = [g(j),j] => get j

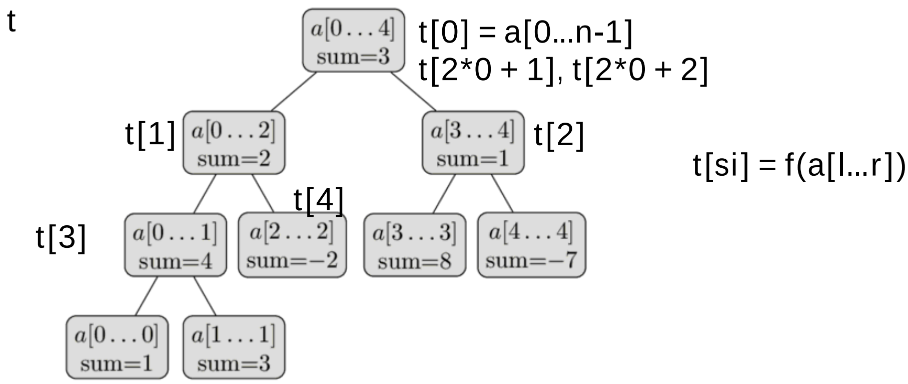


This choice of $g(i)$ allows both queries $[1, r]$ and updates in $O(\log n)$. However, for queries $[l, r]$, invertible operations are required.
Code.

5 Segment Tree

Divide-and-conquer approach. References - cp-algos, gfg, CF.

Example with $a = [1, 3, -2, 8, -7]$



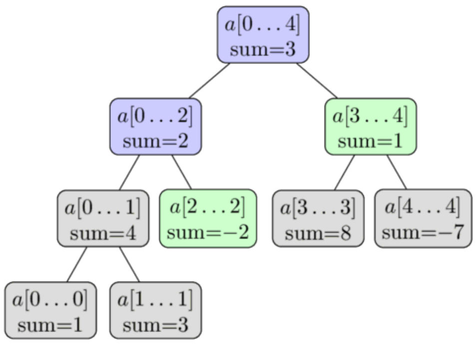


Figure 1: Sum $a[2 \ldots 4]$

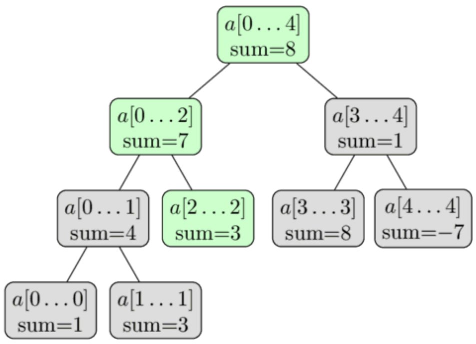


Figure 2: Update $a[2]$

- Underlying data structure is an array.
- Queries and updates in $O(\log n)$. Larger constants than BIT.
- Works with non-invertible functions as well.

6 Todos

Check sheet.