Training III

2021. Sept 26. YeCao

Fenwick Tree & Range Sum Query

Importance of This topic in CP:

"I mean RSQ (range sum query), RMQ (range minimum / maximum query), or Rwhatever Q is a topic in cp that cannot be emphasized enough!" — Charlie Liu.

What's this data structure (Fenwick Tree) used for?

Consider this RSQ problem:
There are n boxes and 2 types of queries
1. add a marbles to box i
2. sum marbles from box k to box r.

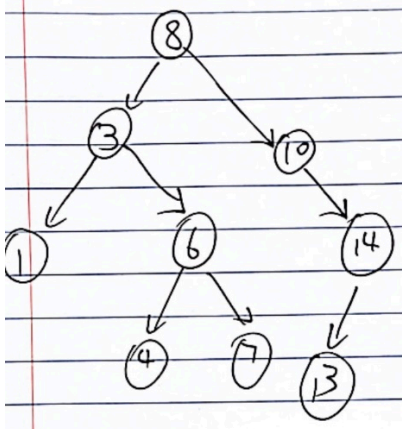
0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	4	5	4	10	13	3	7	9	10	1	2	3	4

SUM(0, 3) = 10 Query 2
Update(0, +3) : Query 1
Sum(0, 3) = 13 Query 2

We will discuss / go over 3 possible ways to solve this problem! The worst case complexity are

$O(n)$ $O(\sqrt{n})$ $O(\log(N))$
$O(N)$ construction $O(N \cdot \log(N))$ Construction

Review of Binary Search tree



std:: set

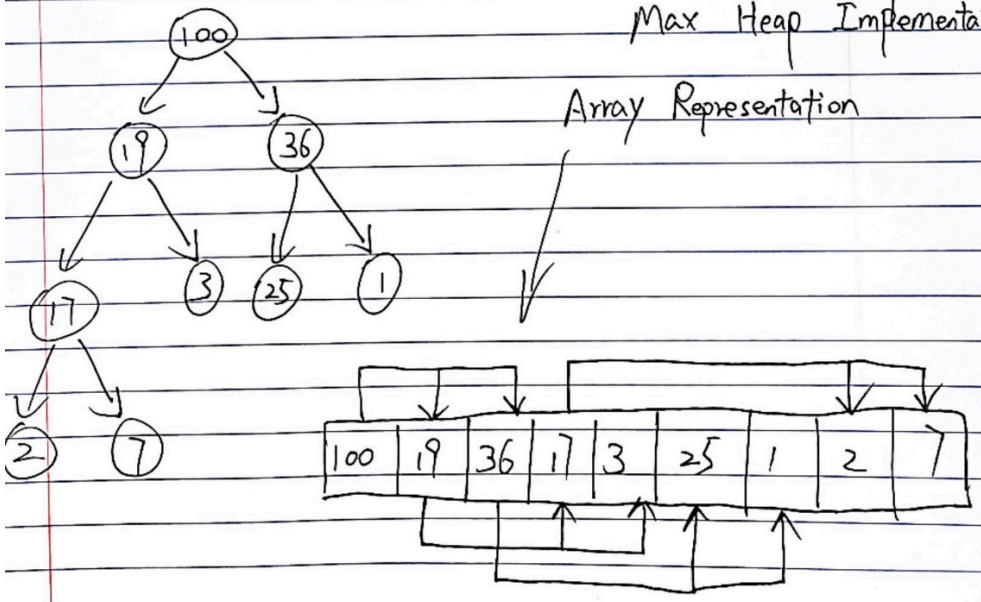
Red Black Tree Implementation

Review of (Max) Heap

std:: priority_queue

Max Heap Implementation

Array Representation



1. Accumulative Sum List

Define A where $a_i = \sum_{j=0}^{i} B[j]$ $0 \le i \le n$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
B:	1	4	5	4	10	13	3	7	4	9	10	1	2	3	4	7	9	13	14

	0	1	2	3	
A:	1	5	10	14	.....

$a_1 = b_1$

$a_2 = b_1 + b_2$

$a_3 = b_1 + b_2 + b_3$

$\vdots$

$a_n = b_1 + b_2 + b_3 + \cdots + b_n$

Sum Query · $Sum(i,j) = A[j] - A[i-1]$ for $i \ge 1$

$= A[j]$ $i = 0$

Complexity ~~O(N)~~ O(1)

Update Query Update (i, m)

$A[k] += m$ $i \le k \le n$

Complexity O(N)

Worst time complexity : O(N)

Good to use if no updates happen.

Two dimensional Accumulative Sum Matrix

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24

1	3	6	. . . . .
10	22	36	. . . .
⋮	⋮	⋮	⋮ ⋱
⋮	⋮	⋱	⋮

$$A_{ij} = \sum_{n=1}^{i} \sum_{m=1}^{j} B_{mn}$$

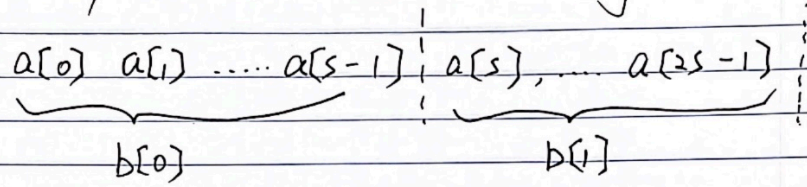
Sum Query $O(1)$

Update $O(n^2)$

2. Square Root Decomposition

$s = \lceil \sqrt{n} \rceil$

array a is divided into blocks of size s.



Last block may have fewer than s elements (if n is not a multiple of s)

$$b[k] = \sum_{i=ks}^{\min(n-1,\,(k+1)s-1)} a[i]$$

To calculate Sum:

Sum of two tails $[l \dots (k+1)\cdot s - 1]$ + $[p \cdot s \dots r]$ + $b[i]$ $k+1 \le i \le p-1$

Complexity: $O(\sqrt{n})$

Update Query

$b[k] \mathrel{+}= a_{new}[i] - a_{old}[i]$

Complexity: $O(1)$

Note that sqrt decomposition not only can be used in RMQ (range minimum Query) or RSQ (range sum Query)

But some problems can only be solved easily by sqrt decomposition rather than fenwick tree / segment tree

"sqrt decomp isn't just a datastructure. It's a problem solving strategy. If I am doing something with an array of length N, does it help if I break it into SQRT(N) blocks of subarray of length SQRT(N) ?" — Charlie Liu

Complexity : $O(\log(N))$

Update $O(\log(N))$

Sum

3. Fenwick Tree

First a trick from number theory: how to get the last bit of a number?

e.g. Lastbit (10010) = 10

(11000) = 1000

(10001) = 1

(10000) = 10000

Before last bit

a|b

↓ ↘ zeros

By definition, this is last bit

$$(a|b)^{-} = a^{-} \circ b^{-} + 1$$
$$= a^{-1} \circ (111\ldots1) + 1$$
$$= a^{-1} | b$$

$$a|b \;\&\; (a|b)^{-} = a|b \;\&\; a^{-1}|b = 1b = \text{Lastbit}$$

Update

1 → 10 → 100 → 1000

10 → 100 → 1000

11 → 100 → 1000

100 → 1000

```
while (idx <= n)
    idx += idx & (-idx)
```

Get sum

Start at idx + 1 (1-based index) continue update to child index

(13) (9-12) (1-8)

1101 → 1100 → 1000

```
while (idx > 0)
    idx -= idx & (-idx)
```