Fenwick Trees

Bruce Merry

IOI Training Feb 2020

Outline

Outline

An Example Problem

A city has N buildings in a row, numbered from 1 to N. Initially, every building has height 0. Accept a sequence of queries and updates of the form

- Building i now has height h.
- What is the sum of the building heights in the range $[l, r]$?

An Example Problem

A city has N buildings in a row, numbered from 1 to N. Initially, every building has height 0. Accept a sequence of queries and updates of the form

- Building i now has height h.
- What is the sum of the building heights in the range $[l, r]$?

You only have enough memory for $N + \epsilon$ integers.

A Non-Obvious Solution

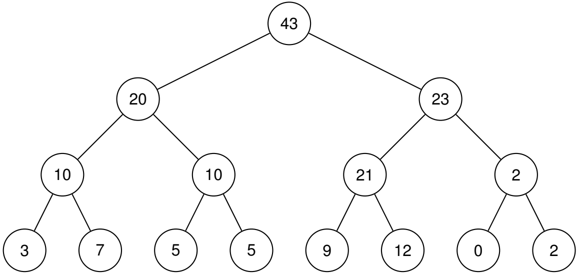
Store a prefix sum of the heights: sum of the first i heights for every i.

Query Take the difference between two prefix sums: $O(1)$

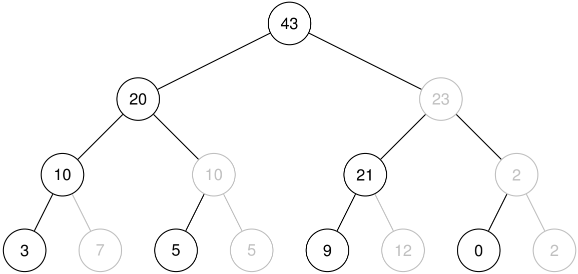
Update Modify all prefix sums that include this element: $O(N)$

Outline

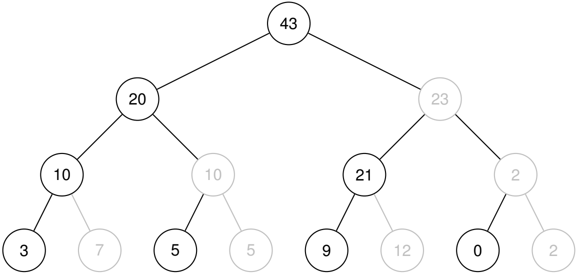
Segment Tree is Redundant



Segment Tree is Redundant

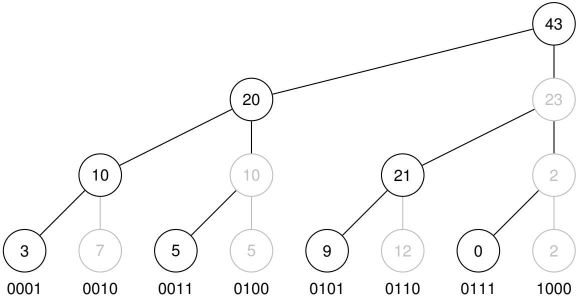


Segment Tree is Redundant



These nodes are not involved in prefix sum queries.

Representation



Outline

Finding The Parent

The parent of i is $i + 2^k$ where $2^k \mid i$, k is maximal.
Example:

$$\begin{array}{r} 11001000 \\ +\ 00001000 \\ =\ 11010000 \end{array}$$

Finding The Parent

The parent of i is $i + 2^k$ where $2^k \mid i$, k is maximal.
Example:

$$\begin{array}{r} \mathtt{11001000} \\ +\ \mathtt{00001000} \\ =\ \mathtt{11010000} \end{array}$$

To find 2^k, we take i and mask off $i - 1$:

$$\begin{array}{r} \mathtt{11001000} \\ \&\ \tilde{}\mathtt{11000111} \\ =\ \mathtt{00001000} \end{array}$$

Update

```
void fenwick_add(
       vector<int> &data, int p, int v) {
  int size = data.size();
  while (p < size) {
    data[p] += v;
    p += p & ~(p - 1);
  }
}
```

Query

To query a prefix sum, we add the current node, then see what is left.

```
int fenwick_query(
      const vector<int> &data, int p) {
  int ans = 0;
  while (p > 0) {
    ans += data[p];
    p &= p - 1; // same as p -= p & ~(p - 1);
  }
  return ans;
}
```

Indexing

- Code above uses 1-based indexing.
- Can be modified to present 0-based interface.

Outline

Range Update, Point Query

Starting with an array a, handle the following queries

- Update: increment by h across a range $[l, r]$
- Query: return a_i

Range Update, Point Query

Solution

Operate on array of adjacent differences instead:

$$b_1 = a_1, b_i = a_i - a_{i-1}$$

Range Update, Point Query

Solution

Operate on array of adjacent differences instead:

$$b_1 = a_1, b_i = a_i - a_{i-1}$$

Operations become:

Update $b_l \leftarrow b_l + h$, $b_{r+1} \leftarrow b_{r+1} - h$

Range Update, Point Query

Solution

Operate on array of adjacent differences instead:

$$b_1 = a_1, b_i = a_i - a_{i-1}$$

Operations become:

Update $b_l \leftarrow b_l + h$, $b_{r+1} \leftarrow b_{r+1} - h$

Query Return $a_i = \sum_1^i b_j$ using Fenwick tree.

Range Update, Range Query

Starting with an array a, handle the following queries

- Update: increment by h across a range $[l, r]$
- Query: return the sum $\sum_{i=l}^{r} a_i$

Note: sufficient to be able to answer $\sum_{i=1}^{r} a_i$.

Range Update, Range Query

Solution

Start with the same transformation as before:

$$b_1 = a_1, b_i = a_i - a_{i-1}$$

Query is

$$\begin{aligned}\sum_{i=1}^{r} a_i &= \sum_{i=1}^{r}\sum_{j=1}^{i} b_j \\ &= \sum_{i=1}^{r}(r-1-i)b_i \\ &= (r-1)\left(\sum_{i=1}^{r} b_i\right) - \left(\sum_{i=1}^{r} ib_i\right)\end{aligned}$$

Start with the same transformation as before:

$$b_1 = a_1, b_i = a_i - a_{i-1}$$

Query is

$$\begin{aligned}\sum_{i=1}^{r} a_i &= \sum_{i=1}^{r}\sum_{j=1}^{i} b_j \\ &= \sum_{i=1}^{r}(r - 1 - i)b_i \\ &= (r-1)\left(\sum_{i=1}^{r} b_i\right) - \left(\sum_{i=1}^{r} ib_i\right)\end{aligned}$$

Let $c_i = ib_i$. Then we need Fenwick trees for b and c.