Interval Trees

Bruce Merry

IOI Training Mar 2014
Outline

1. Interval Trees
   - A Problem
   - Solution
   - Implementation

2. Fenwick Trees
   - A Problem
   - Solution
   - Implementation

3. More Query/Update Problems
   - Using Transformations
Outline

1. Interval Trees
   - A Problem
   - Solution
   - Implementation

2. Fenwick Trees
   - A Problem
   - Solution
   - Implementation

3. More Query/Update Problems
   - Using Transformations
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 height of the tallest building in the range \([l, r]\)?
Analysis: Naïve Solution

Simply store the height of each building:

- Each update requires $O(1)$ time
- Each query requires $O(N)$ time
Slightly Smarter Solution

- Divide city into “neighbourhoods” of $\sqrt{N}$ buildings
- Maintain the maximum height of each neighbourhood

Running time:
- Each update takes $O(\sqrt{N})$ time
- Each query takes $O(\sqrt{N})$ time (why?)
Outline

1. Interval Trees
   - A Problem
   - Solution
   - Implementation

2. Fenwick Trees
   - A Problem
   - Solution
   - Implementation

3. More Query/Update Problems
   - Using Transformations
Instead of just buildings and neighbourhoods, use a hierarchy:
Walk up the tree, updating ancestors
Walk up the tree, updating ancestors
Walk up the tree, updating ancestors
Walk up the tree, updating ancestors
Pick a set of nodes to cover the range e.g. for $[1, 6)$:
Performance

- Each update touches $O(\log N)$ nodes
- Each query examines $O(\log N)$ nodes (why?)
Outline

1. Interval Trees
   - A Problem
   - Solution
   - Implementation

2. Fenwick Trees
   - A Problem
   - Solution
   - Implementation

3. More Query/Update Problems
   - Using Transformations
Representation

Number nodes in BFS order

- Parent of $i$ is $\lfloor i/2 \rfloor$
- Children of $i$ are $2i$, $2i + 1$
- Round up to a power of 2
void fix(int idx) {
    tree[idx] = max(tree[2 * idx],
                    tree[2 * idx + 1]);
}

vector<int> init(const vector<int> &values) {
    int bias = next_power2(values.size());
    vector<int> tree(2 * bias, 0);
    copy(values.begin(), values.end(),
         tree.begin() + bias);
    for (int i = bias; i > 0; i--)
        fix(tree, i);
    return tree;
}
void update(int pos, int val) {
    pos += tree.size() / 2;
    tree[pos] = val;
    for (pos = pos / 2; pos > 0; pos = pos / 2)
        fix(tree, pos);
int query(int L, int R) {
    int ans = 0;
    L += bias; R += bias;
    while (L < R) {
        if (L & 1) {
            ans = max(ans, tree[L]);
            L++;
        }
        if (R & 1) {
            R--;
            ans = max(ans, tree[R]);
        }
        L /= 2; R /= 2;
    }
    return ans;
}
Outline

1. Interval Trees
   - A Problem
   - Solution
   - Implementation

2. Fenwick Trees
   - A Problem
   - Solution
   - Implementation

3. More Query/Update Problems
   - Using Transformations
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.
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

1. Interval Trees
   - A Problem
   - Solution
   - Implementation

2. Fenwick Trees
   - A Problem
   - Solution
   - Implementation

3. More Query/Update Problems
   - Using Transformations
Interval Tree is Redundant

These nodes are not involved in prefix sum queries.
These nodes are not involved in prefix sum queries.
These nodes are not involved in prefix sum queries.
Representation
Element $i$ is sum of $2^k$ elements, $2^k \mid i$, $k$ is maximum
Finding The Parent

The parent of $i$ is $i + 2^k$ where $2^k | i$, $k$ is maximal.

Example:

$$11001000 + 00001000 = 11010000$$
Finding The Parent

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

\[
\begin{array}{c}
\text{11001000} \\
+ \text{00001000} \\
\hline
\text{11010000}
\end{array}
\]

To find \( 2^k \), we take \( i \) and mask off \( i - 1 \):

\[
\begin{array}{c}
\text{11001000} \\
\& \text{\sim 11000111} \\
\hline
\text{00001000}
\end{array}
\]
void bit_add(int *bit, int p, int v) {
    while (p < size) {
        bit[p] += v;
        p += p & ~(p - 1);
    }
}
To query a prefix sum, we add the current node, then see what is left.

```c
int bit_query(const int *bit, int p) {
    int ans = 0;
    while (p > 0) {
        ans += bit[p];
        p &= p - 1; // same as p -= p & ~(p - 1);
    }
    return ans;
}
```
Outline

1 Interval Trees
   - A Problem
   - Solution
   - Implementation

2 Fenwick Trees
   - A Problem
   - Solution
   - Implementation

3 More Query/Update Problems
   - Using Transformations
Range Update, Point Query

Starting with an array $a$, handle the following queries:

- **Update**: increment by $h$ across a range $[l, r]$
- **Query**: return $a_i$
Operate on array of adjacent differences instead:

\[ b_1 = a_1, \quad b_i = a_i - a_{i-1} \]
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:

\[ b_1 = a_1, \quad b_i = a_i - a_{i-1} \]

Operations become:

**Update**  \( b_l \leftarrow b_l + h, \quad b_{r+1} \leftarrow b_{r+1} - h \)

**Query**  Return  \( a_i = \sum_{j=1}^i b_j \) 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

Interval Trees
Bruce Merry

Interval Trees
A Problem
Solution
Implementation

Fenwick Trees
A Problem
Solution
Implementation

More
Query/Update
Problems
Using
Transformations

Summary
Range Update, Range Query

Solution

Start with the same transformation as before:

\[ b_1 = a_1, \quad b_i = a_i - a_{i-1} \]

Query is

\[ \sum_{i=1}^{r} a_i = \sum_{i=1}^{r} \sum_{j=1}^{i} b_j \]

\[ = \sum_{i=1}^{r} (r - 1 - i) b_i \]

\[ = (r - 1) \left( \sum_{i=1}^{r} b_i \right) - \left( \sum_{i=1}^{r} ib_i \right) \]
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

\[
\begin{align*}
\sum_{i=1}^{r} a_i &= \sum_{i=1}^{r} \sum_{j=1}^{i} b_j \\
&= \sum_{i=1}^{r} (r - 1 - i)b_i \\
&= (r - 1) \left( \sum_{i=1}^{r} b_i \right) - \left( \sum_{i=1}^{r} ib_i \right)
\end{align*}
\]

Let \( c_i = ib_i \). Then we need Fenwick trees for \( b \) and \( c \).
Summary

- Interval trees are a general-purpose tool for accelerating operations on ranges.
- Fenwick trees are less general, but more compact and easier to implement.
- Both have relatively low overhead and simple implementation due to the implicit tree structure.