Range Queries and Segment Trees
Duke COMPSCI 309s

Siyang Chen

Spring 2014
Introduction

*Range query problems* have the following structure:
Introduction

Range query problems have the following structure:

Given list $x_0, \ldots x_{n-1}$, efficiently implement the following operations:
Introduction

Range query problems have the following structure:

Given list \( x_0, \ldots, x_{n-1} \), efficiently implement the following operations:

- For some \( i \) and \( j \), compute \( f(x_i, x_{i+1}, x_{i+2}, \ldots, x_j) \).
Introduction

Range query problems have the following structure:

Given list $x_0, \ldots x_{n-1}$, efficiently implement the following operations:

- For some $i$ and $j$, compute $f(x_i, x_{i+1}, x_{i+2}, \ldots, x_j)$.
- Update $x_i$. 

Assuming $f$ is well-behaved, we can implement both operations with a segment tree in $O(\log n)$ time. 

We can also create segment trees which handle range updates and point queries, or even range updates and range queries, but for now we’ll focus on the simpler case.
Range query problems have the following structure:

Given list $x_0, \ldots x_{n-1}$, efficiently implement the following operations:

- For some $i$ and $j$, compute $f(x_i, x_{i+1}, x_{i+2}, \ldots, x_j)$.
- Update $x_i$.

Assuming $f$ is well-behaved, we can implement both operations with a segment tree in $O(\log n)$ time.
Introduction

Range query problems have the following structure:

Given list \( x_0, \ldots x_{n-1} \), efficiently implement the following operations:

- For some \( i \) and \( j \), compute \( f(x_i, x_{i+1}, x_{i+2}, \ldots, x_j) \).
- Update \( x_i \).

Assuming \( f \) is well-behaved, we can implement both operations with a segment tree in \( O(\log n) \) time.

We can also create segment trees which handle range updates and point queries, or even range updates and range queries, but for now we’ll focus on the simpler case.
Given list $x_0, \ldots, x_{n-1}$, efficiently implement the following operations:
Range Sums

Given list $x_0, \ldots, x_{n-1}$, efficiently implement the following operations:

- For some $i$ and $j$, compute $\sum_{k=i}^{j} x_k$.  

There are obvious solutions which are $O(1)$ in one operation and $O(n)$ in the other, but we want to do better.
Range Sums

Given list $x_0, \ldots x_{n-1}$, efficiently implement the following operations:

- For some $i$ and $j$, compute $\sum_{k=i}^{j} x_k$.
- For some $i$ and $\delta$, update $x_i \leftarrow x_i + \delta$. 

There are obvious solutions which are $O(1)$ in one operation and $O(n)$ in the other, but we want to do better.
Range Sums

Given list $x_0, \ldots, x_{n-1}$, efficiently implement the following operations:

- For some $i$ and $j$, compute $\sum_{k=i}^{j} x_k$.
- For some $i$ and $\delta$, update $x_i \leftarrow x_i + \delta$.

There are obvious solutions which are $O(1)$ in one operation and $O(n)$ in the other, but we want to do better.
Range Sums via Partitioning

Tangent: Here’s one possible solution via \textit{range partitioning}:
Range Sums via Partitioning

Tangent: Here’s one possible solution via range partitioning:

Let $y_i = \sum_{j=im}^{(i+1)m-1} x_k$, where $m = \left\lfloor \sqrt{n} \right\rfloor$. 
Range Sums via Partitioning

Tangent: Here's one possible solution via range partitioning:

- Let \( y_i = \sum_{j=im}^{(i+1)m-1} x_k \), where \( m = \left\lfloor \sqrt{n} \right\rfloor \).
- Cost to compute sum?
Tangent: Here’s one possible solution via *range partitioning*:

- Let \( y_i = \sum_{j=im}^{(i+1)m-1} x_k \), where \( m = \lfloor \sqrt{n} \rfloor \).
- Cost to compute sum? \( O(\sqrt{N}) \)
Tangent: Here’s one possible solution via *range partitioning*:

- Let $y_i = \sum_{j=im}^{(i+1)m-1} x_k$, where $m = \left\lfloor \sqrt{n} \right\rfloor$.
- Cost to compute sum? $O(\sqrt{N})$
- Cost to update?
Tangent: Here’s one possible solution via range partitioning:

- Let \( y_i = \sum_{j=im}^{(i+1)m-1} x_k \), where \( m = \lfloor \sqrt{n} \rfloor \).
- Cost to compute sum? \( O(\sqrt{N}) \)
- Cost to update? \( O(1) \)
Tangent: Here’s one possible solution via *range partitioning*:

- Let $y_i = \sum_{j=im}^{(i+1)m-1} x_k$, where $m = \left\lfloor \sqrt{n} \right\rfloor$.
- Cost to compute sum? $O(\sqrt{N})$
- Cost to update? $O(1)$

Range partitioning is in general a pretty useful technique, but in this case we have something better.
Range Sums via Segment Trees

We say a node *governs* the range \([l, r)\) if it stores the value \(\sum_{i=l}^{r-1} x_i\). Then we can define a segment tree on \(x_0, \ldots, x_{n-1}\) in the following manner:
Range Sums via Segment Trees

We say a node *governs* the range \([l, r)\) if it stores the value \(\sum_{i=l}^{r-1} x_i\). Then we can define a segment tree on \(x_0, \ldots, x_{n-1}\) in the following manner:

- The root governs the range \([0, n)\).
Range Sums via Segment Trees

We say a node *governs* the range \([l, r)\) if it stores the value 
\[\sum_{i=l}^{r-1} x_i.\] Then we can define a segment tree on \(x_0, \ldots, x_{n-1}\) in the following manner:

- The root governs the range \([0, n)\).
- If a node governs \([l, r)\) and \(r - l > 1\), then its children govern \([l, m)\) and \([m, r)\), where \(m = \lfloor(l + r)/2\rfloor\).
We say a node \textit{governs} the range \([l, r)\) if it stores the value 
\(\sum_{i=l}^{r-1} x_i\). Then we can define a segment tree on \(x_0, \ldots, x_{n-1}\) in the following manner:

- The root governs the range \([0, n)\).
- If a node governs \([l, r)\) and \(r - l > 1\), then its children govern \([l, m)\) and \([m, r)\), where \(m = \lfloor (l + r)/2 \rfloor\).
- Leaves govern ranges of the form \([l, l + 1)\).
Range Sums via Segment Trees

Example: A segment tree on the list $x_0, x_1, \ldots, x_4$ has the following structure:
Range Sums via Segment Trees

How do we query and update the segment tree in $O(\log n)$ time?
Range Sums via Segment Trees

How do we query and update the segment tree in $O(\log n)$ time?

- $update(node, i, \delta)$:
  - If $i \in [node.l, node.r)$:
    - $node.value \leftarrow node.value + \delta$
    - $update(node.leftChild, i, \delta)$
    - $update(node.rightChild, i, \delta)$

Efficiency?

$O(\log n)$ time for both operations and $O(n)$ memory
How do we query and update the segment tree in $O(\log n)$ time?

- **update**(node, $i$, $\delta$):
  - If $i \in [\text{node}.l, \text{node}.r)$:
    - $\text{node}.value \leftarrow \text{node}.value + \delta$
    - update(node.leftChild, $i$, $\delta$)
    - update(node.rightChild, $i$, $\delta$)

- **query**(node, $l$, $r$):
  - If $l \geq \text{node}.r$ or $\text{node}.l \geq r$:
    - Return 0
  - If $l \leq \text{node}.l$ and $\text{node}.r \leq r$:
    - Return $\text{node}.value$
  - Otherwise return
    $\text{query}(\text{node}.leftChild, l, r) + \text{query}(\text{node}.rightChild, l, r)$
How do we query and update the segment tree in $O(\log n)$ time?

- **update**(*node*, *i*, *δ*):
  - If $i \in [\text{node}.l, \text{node}.r)$:
    - *node*.value $\leftarrow$ *node*.value $+$ *δ*
    - **update**(*node*.leftChild, *i*, *δ*)
    - **update**(*node*.rightChild, *i*, *δ*)

- **query**(*node*, *l*, *r*):
  - If $l \geq \text{node}.r$ or $\text{node}.l \geq r$:
    - Return 0
  - If $l \leq \text{node}.l$ and $\text{node}.r \leq r$:
    - Return *node*.value
  - Otherwise return
    $$\text{query(} \text{node}.\text{leftChild, } l, r \text{)} + \text{query(} \text{node}.\text{rightChild, } l, r \text{)}$$

Efficiency?
Range Sums via Segment Trees

How do we query and update the segment tree in $O(\log n)$ time?

- **update(node, i, δ):**
  - If $i \in [node.l, node.r)$:
    - $node.value \leftarrow node.value + δ$
    - $update(node.leftChild, i, δ)$
    - $update(node.rightChild, i, δ)$

- **query(node, l, r):**
  - If $l \geq node.r$ or $node.l \geq r$:
    - Return 0
  - If $l \leq node.l$ and $node.r \leq r$:
    - Return $node.value$
  - Otherwise return
    $query(node.leftChild, l, r) + query(node.rightChild, l, r)$

Efficiency? $O(\log n)$ time for both operations and $O(n)$ memory
Codeforces: Sereja and Brackets

Time for a slightly harder problem:

[Codeforces Link]

See [here] and [here] for hints on how we solved the problem.
Time for a slightly harder problem:

Codeforces: Sereja and Brackets

Time for a slightly harder problem:

▶ http://codeforces.com/contest/380/problem/C

See [here] and [here] for hints on how we solved the problem.
Lazy Segment Trees

A technique called *lazy propagation* or *lazy updating* is used to implement segment trees which support both *range queries* and *range updates*. 

[See repository for reference solution using lazy updates.](http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf)

Lazy Segment Trees

A technique called *lazy propagation* or *lazy updating* is used to implement segment trees which support both *range queries* and *
range updates*.

▶ [http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf](http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf)

Lazy Segment Trees

A technique called *lazy propagation* or *lazy updating* is used to implement segment trees which support both *range queries* and *range updates*.

▶ [http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf](http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf)

Example problem:
Lazy Segment Trees

A technique called lazy propagation or lazy updating is used to implement segment trees which support both range queries and range updates.

- http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf

Example problem:

Lazy Segment Trees

A technique called lazy propagation or lazy updating is used to implement segment trees which support both range queries and range updates.

▶ [http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf](http://activities.tjhsst.edu/sct/lectures/1112/rquery102811.pdf)

Example problem:

(See repository for reference solution using lazy updates.)
Variations

The *Fenwick tree* is another tree structure that has similar properties, but is much easier to implement. On the other hand, it’s also more difficult to understand.

- [http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees](http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees)
- [http://petr-mitrichev.blogspot.ru/2013/05/fenwick-tree-range-updates.html](http://petr-mitrichev.blogspot.ru/2013/05/fenwick-tree-range-updates.html)