Range queries and Fenwick Trees

Version 1.1

Yaseen Mowzer

2nd IOI Training Camp 2017 (3 February 2018)

Preliminaries

- All ranges will be half open ranges $e \in [a, b) \iff a \leq e < b$
- Occasionally 1 is a more convenient starting index than 0

Susie has questions

Problem

Susie has $1 < N < 10^6$ model ships arranged in a sequence numbered $0, \ldots, N-1$. The ith boat has a size of s_i ($1 < s_i < 10^9$). At any given time Susie may replace a boat with another boat of a different size. Given two integers a and b, report the sizes of all the ships between a and b.

Susie has questions

Problem

Susie has $1 < N < 10^6$ model ships arranged in a sequence numbered $0, \ldots, N-1$. The ith boat has a size of s_i ($1 < s_i < 10^9$). At any given time Susie may replace a boat with another boat of a different size. Given two integers a and b, report the sizes of all the ships between a and b.

In summary

- $\sim 10^6$ model ships of different sizes $\sim 10^9$.
- Susie can change the size of a ship.
- Report all sizes of ships between a and b.

Susie's questions are easy to answer

Solution

Store an an array of all the ships.

Time Complexity

- Let $m = b - a$. m is the width of the query.
- O(N) construction
- O(m) query
- $O(1)$ update

Susie wants the size of the smallest ship

Problem

Susie also wants to know the minimum of all the ship sizes between a and b.

Susie wants the size of the smallest ship

Problem

Susie also wants to know the minimum of all the ship sizes between a and b.

Observations

- The min function is associative i.e. $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- In other words, min function forms a semigroup with the integers

Susie wants the size of the smallest ship

Problem

Susie also wants to know the minimum of all the ship sizes between a and b.

Observations

- The min function is associative i.e. $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- In other words, min function forms a semigroup with the integers
- It is unnecessary to iterate over m since

  $$\min(x_1, x_2, \ldots, x_{2n}) = \min(\min(x_1, \ldots, x_n), \min(x_{n+1}, \ldots, x_{2n}))$$

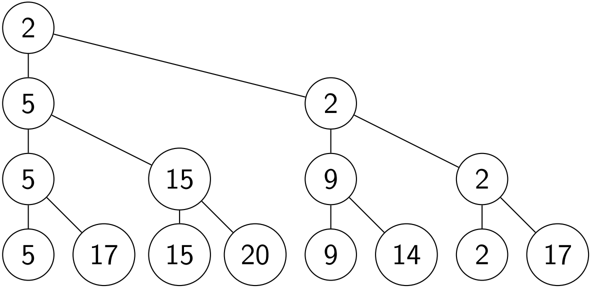
  allows us to "cache" queries.
- We can query in better than $O(m)$ time.

Segment tree

- Perfectly balanced binary tree.
- The leaf nodes correspond with s_i.
- A parent is the minimum of it's children.

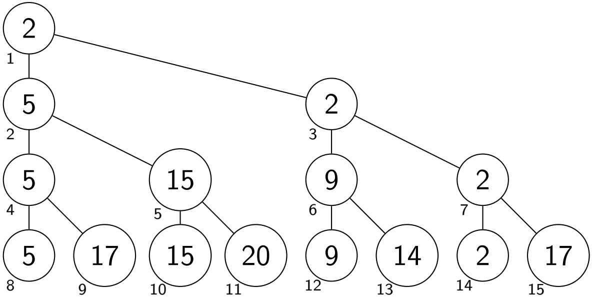
Segment tree

- Perfectly balanced binary tree.
- The leaf nodes correspond with s_i.
- A parent is the minimum of it's children.



Representing a Perfectly Balanced Binary Tree

- Represent the tree as an array indexed from 1
- For every index i the
  - left child is $2i$
  - right child is $2i + 1$



Update by walking up the tree

```
def update(index, value):
    index += N
    seg_tree[index] = value
    index /= 2
    while index > 0:
    seg_tree[index] = min(
            seg_tree[2 * index],
            seg_tree[2 * index + 1]
        )
        index /= 2
```

Query by walking up the tree

```
def query(a, b):
    a += N
    b += N
    ans = ∞
    while a < b:
        if a % 2 == 1:
            ans = min(seg_tree[a], ans)
            a += 1
        if (b - 1) % 2 == 0:
            ans = min(seg_tree[b - 1], ans)
        a /= 2
        b /= 2
```

Time complexity

- $O(N)$ construction
- $O(\log N)$ updates
- $O(\log N)$ query

Susie updates ranges

Problem

Susie can replaces all ships between a and b with many ships of the same size.

Susie updates ranges

Problem

Susie can replaces all ships between a and b with many ships of the same size.

Solution

When updating a range, if a node is completely within the range, mark it as overridden and don't update the children.

Update code

```
def rec_update(i, l, r, v):
    a = segment_start(i)
    b = segment_end(i)
    if l <= a and b <= r:
        # Completely contained in the interval
        overide[i] = True
        seg_tree[i] = v
    elif l < b and a < r:
        # Intersects, thus update children
        push_down_overide(i)
        rec_update(2 * i, l, r, v)
        rec_update(2 * i + 1, l, r, v)
        seg_tree[i] = min(seg_tree[2 * i], seg_tree[2 * i + 1])

def push_down_overide(i):
    l = 2 * i
    r = l + 1
    if overide[i]:
        overide[i] = False
        overide[l] = overide[r] = True
        seg_tree[l] = seg_tree[r] = seg_tree[i]
```

Query

```
def query(i, l, r):
  a = segment_start(i)
  b = segment_end(i)
  if l <= a and b <= r:
      # Completely contained in the interval
      return seg_tree[i]
  elif b <= l or r <= a:
      # Don't intersect do nothing
      return ∞ # Return identity
  else:
      push_down_overide(i)
      return min(query(2 * i, l, r), query(2 * i + 1, l, r))
```

Susie asks for the sum

Problem

Find the sum of the sizes of the boats between a and b. (Only updating single points at a time).

Susie asks for the sum

Problem

Find the sum of the sizes of the boats between a and b. (Only updating single points at a time).

Observation

- Addition has an identity (0)
- and an inverse operation ($-$)
- Addition forms a group with the integers

Susie asks for the sum

Problem

Find the sum of the sizes of the boats between a and b. (Only updating single points at a time).

Observation

- Addition has an identity (0)
- and an inverse operation ($-$)
- Addition forms a group with the integers

We can subtract!

Prefix sums

```
prefix_sum = [0]
for i in range(N):
    prefix_sum.append(ships[i] + prefix_sum[-1])

def query(l, r):
    return prefix_sum[r] - prefix_sum[l]
```

"Subtraction" is required

Prefix sums

```
prefix_sum = [0]
for i in range(N):
    prefix_sum.append(ships[i] + prefix_sum[-1])

def query(l, r):
    return prefix_sum[r] - prefix_sum[l]
```

"Subtraction" is required

Time Complexity

- O(N) construction
- O(1) query
- O(N) update

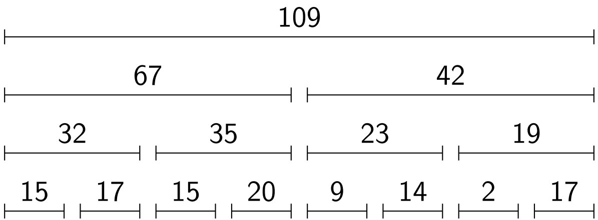
Update is too slow!

Fenwick trees

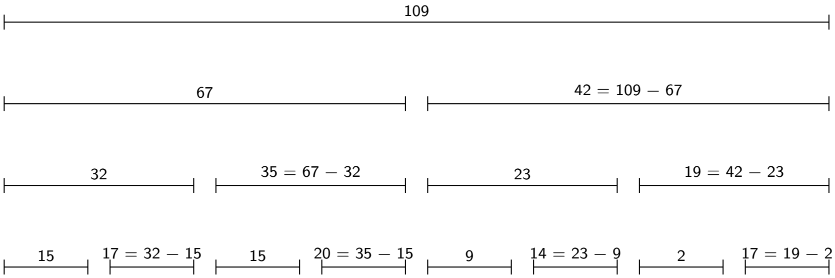
Ideas

- We can use a segment tree, but we can do better

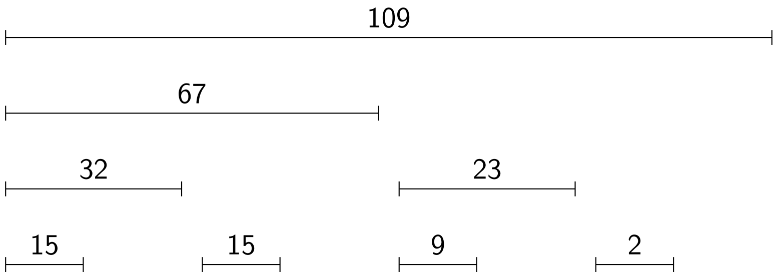
Combine the prefix sum with the segment tree



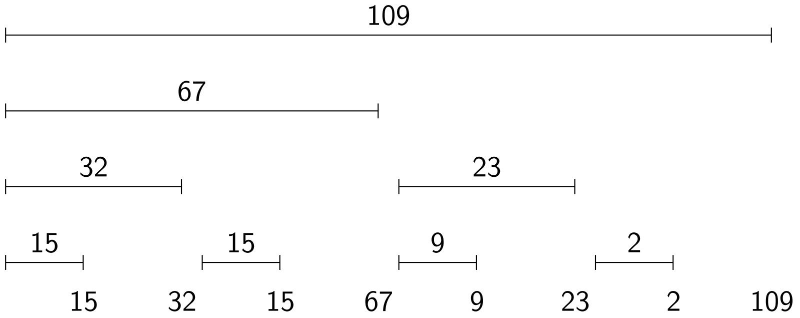
Right nodes are redundant



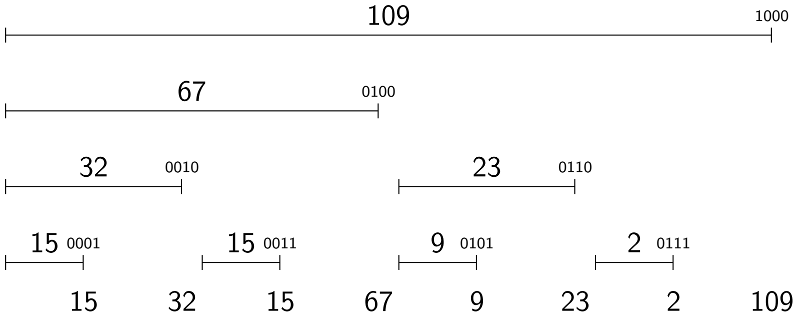
Chop off the right nodes



Chop off the right nodes



We are left with N numbers



Storage

- We only have N nodes (not $2N$)
- We use an array indexed from 1.
- Let s be the greatest power of 2 that divides i
- Index i contains the sum of $[i - r + 1, i + 1)$

Updating

- We update by increasing rather than setting.
- It is easy to compute what to increase
- i is the smallest index that contains s_i
- $i + r$ is the next element that contains i

Computing r

We can compute the largest power of two by using `i & ~(i - 1)`

```
   10101000
-         1
-----------
~  10100111
-----------
   01011000
&  10101000
-----------
   00001000
```

Code for fenwick tree

```python
def update(i, v):
    while i < N:
        fenwick_tree[i] += v
        # Go to parent
        i += (i & ~(i - 1))

def query(i):
    acc = 0 # Identity
    while i > 0:
        acc += fenwick_tree[i]
        # Go to previous
        i -= (i & ~(i - 1))
```

query(a, b) $=$ query(b) - query(a)

Problem

Susie can also increase the size of the boats from a to b by v, but will only ask for the size of one boat.

Problem

Susie can also increase the size of the boats from a to b by v, but will only ask for the size of one boat.

We can apply a tranformation.

- $d_i = s_i - s_{i-1}$
- $d_0 = s_0$

Construct a fenwick tree over d

- We can query a point just by querying `query(point)`
- Update a range by update(a, v) and update(b, -v)

Problem

Susie can also increase the size of the boats from a to b by v, but will only ask for the size of one boat.

We can apply a tranformation.

- $d_i = s_i - s_{i-1}$
- $d_0 = s_0$

Construct a fenwick tree over d

- We can query a point just by querying `query(point)`
- Update a range by update(a, v) and update(b, -v)
- Beware of off-by-one errors

Susie wants to query a range

$$
\begin{aligned}
\sum_{i=0}^{a-1} s_i &= \sum_{i=0}^{a-1} \sum_{j}^{i} d_j \\
&= \sum_{i=0}^{a-1} (a-i) d_j \\
&= a \left(\sum_{i=0}^{a-1} d_i \right) - \left(\sum_{i=0}^{a-1} i d_i \right)
\end{aligned}
$$

Make a Fenwick tree with id_i as well.

Better solution

Use a segment tree instead