Advanced and dynamic data structures
Segment trees

Jill-Jênn Vie   CP Algorithms

September 26, 2024

Data structure

For a given array a

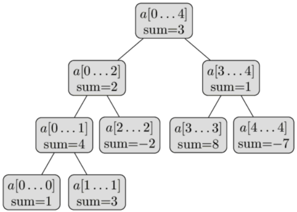
Task	Naively	Prefix sums	Segment tree
Compute range sum $a[\ell \dots r] = \sum_{k=\ell}^{r} a[k]$	$O(n)$	$S_r - S_{\ell-1}$ $O(1)$	$O(\log n)$
Update element $a[i]$	$O(1)$	All S_k $O(n)$	$O(\log n)$

Segment tree

Each node:

- handles segment $[\ell, r]$
- has one or several attributes/values (ex. min or sum of $a[\ell, r]$)
- has children $[\ell, m]$ and $[m+1, r]$ where $m = \lfloor (\ell + r)/2 \rfloor$

Building the tree has complexity $O(n)$



The height is $O(\log n)$, the complexity of queries is $4 \log n = O(\log n)$

Segment trees defined by arrays

Just like heaps, start from $i = 1$

- Childrens of i are $2i$ and $2i + 1$
- Parent of i is $\lfloor i/2 \rfloor$

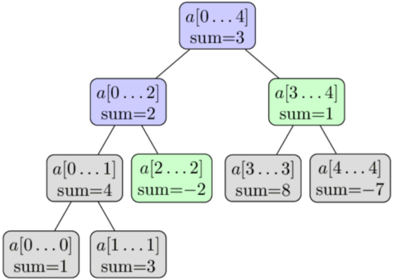
```
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

Sum queries

Requested $I = [\ell, r]$

If current node is $[t\ell, tr]$, three cases:

- $\ell > r$: return 0
- $[\ell, r] = [t\ell, tr]$: return current value
- Otherwise: sum of 2 recursive calls on left $[t\ell, tm] \cap I$ and right $[tm + 1, tr] \cap I$ where $tm = \lfloor (t\ell + tr)/2 \rfloor$




```
int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
           + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}
```

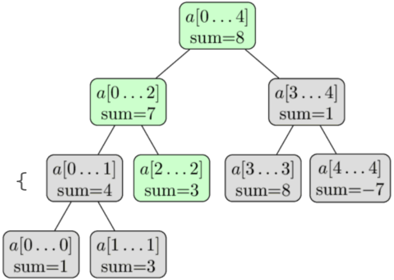
Update queries

Update element at position i

If current node is $[t\ell, tr]$:

- If $i \in [t\ell, tm]$: 1 recursive call to the left
- If $i \in [tm, tr]$: 1 recursive call to the right

Update every node on the branch




```
void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}
```

Variants of segment trees

Should wonder: what attributes at each node, how to merge children info upwards

- Min / Max / GCD / LCM instead of Sum: easy (associative operations)
- Max and number of occurrences of the max
- Count number of zeroes / finding the k-th zero
- Given value x find smallest i such that $a[i] \geq x$ in $O((\log n)^2)$ or $O(\log n)$
- Finding subsegments of maximal sum: slightly harder

Strategy

- What is the space on which we want to perform those $[\ell, r]$ queries?
- What information can be stored and easily computed? (states)
- How to combine child information? (recurrence relation)

Related structures

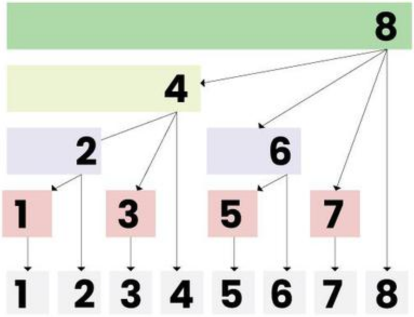
- Fenwick tree: update and query prefix sums (needs commutative operations)
- Range Min Query: cannot update, query min $O(\log n)$
- Sparse table: cannot update, works for idempotent operations (min), query $O(1)$
- Lowest common ancestor: reduction to RMQ
- Square root decomposition: array of cells that consider $O(\sqrt{n})$ consecutive elements

Needs invertible operations for going from prefix sums to range sums

Fenwick tree / BIT binary-indexed tree (1994)

- Point update & Range Query
- Range update & Point Query

```python
def add(b, idx, x):
    while idx <= N:
        b[idx] += x
        idx += idx & -idx

def sum(b, idx):
    total = 0
    while idx > 0:
        total += b[idx]
        idx -= idx & -idx
    return total
```

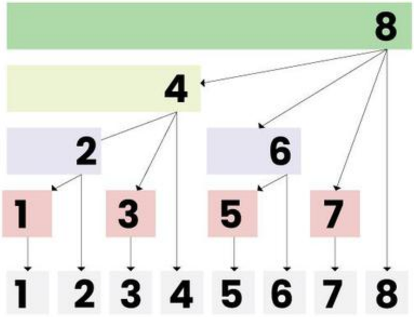


`i -= i & -i` removes the last set bit of the binary representation: 7 (111) $\mapsto$ 6 (110) $\mapsto$ 4 (100)

Fenwick tree / BIT binary-indexed tree (1994)

- Point update & Range Query
- Range update & Point Query

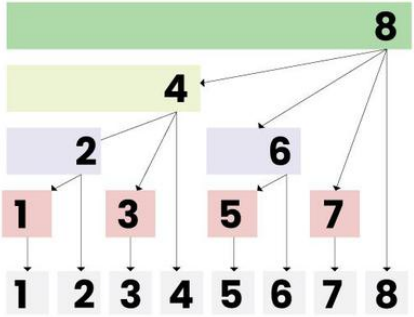
```
def add(b, idx, x):
    while idx <= N:
        b[idx] += x
        idx += idx & -idx

def sum(b, idx):
    total = 0
    while idx > 0:
        total += b[idx]
        idx -= idx & -idx
    return total
```



`i -= i & -i` removes the last set bit of the binary representation:
7 (111) $\mapsto$ 6 (110) $\mapsto$ 4 (100)
M A G I C

Fenwick tree / BIT binary-indexed tree (1994)

- Point update & Range Query
- Range update & Point Query
- Range update & Range Query




```python
def range_add(l, r, x):
    add(B1, l, x)
    add(B1, r+1, -x)
    add(B2, l, x*(l-1))
    add(B2, r+1, -x*r)

def prefix_sum(idx):
    return sum(B1, idx) * idx - sum(B2, idx)

def range_sum(l, r):
    return prefix_sum(r) - prefix_sum(l - 1)
```

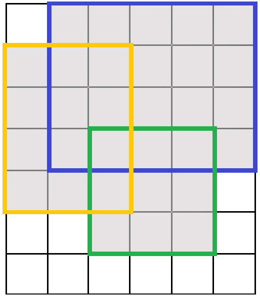
Union of rectangles

Actually the inventor of segment trees is Jon Bentley (born in 1953) who solved, while he was a master's student, Klee's measure problem (1977):

Given a set of n rectangles, find the area of their union.

Online Judges

- Kattis – Unreal Estate
  $n = 5000$ $\quad -1000 \leq x, y \leq 1000$
- SPOJ – NKMARS – Mars Map
  $n = 10000$ $\quad 0 \leq x, y \leq 30000$
- LightOJ – Rectangle Union
  $n = 30000$ $\quad 0 \leq x, y \leq 10^9$



More variants: lazy

- Adding x to all cells in a range $[\ell, r]$
- Get $a[i]$

Attribute is "how much is added to this segment".
Then we will compute the actual value of a cell only if requested, in $O(\log n)$ (sum over the branch)

- Assign x to all cells in a range $[\ell, r]$
- Get $a[i]$

Attribute is: "is this segment constant?" and if the yes "what is the value"

Or:

- Adding x to all cells in a range $[\ell, r]$
- Query for max in a range $[\ell, r]$

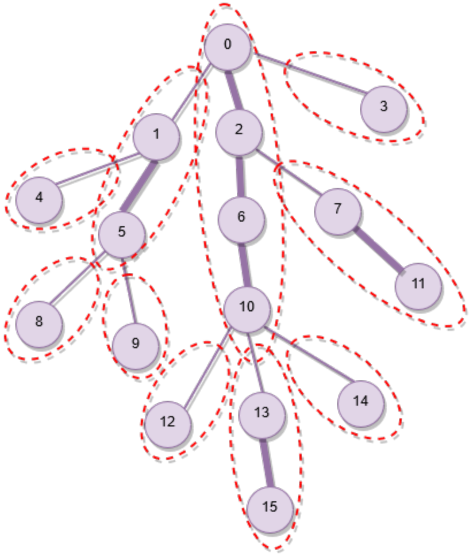
One can also lazy build the segment tree (grow node only if needed)

What's in the notebook

- 4.5 Binary Tree is a sum segment tree
  - update point
  - query sum range
- 4.6 Binary Tree with Lazy Propagation
  - update point
  - assign range
  - query sum range
- 4.7 Persistent Binary Tree == 4.8 Persistent Segment Tree
  - application: what is the kth smallest element in range $a[\ell : r]$
- 4.10 Heavy-Light decomposition
  - decomposition of trees into heavy/light paths
  - can use segment trees for heavy paths
- 4.11 Range min query
  - uses sparse table: queries $O(1)$

Heavy-light decomposition

- Maximum / sum on the path between two vertices
- Repainting the edges on the path between two vertices



Cartesian tree / treap (Siedel and Aragon, 1989)

Binary search tree for X and binary heap for Y

- Insert $O(\log N)$
- Search $O(\log N)$
- Erase $O(\log N)$
- Build $O(\log N)$
- Union $O(M \log(N/M))$
- Intersect $O(M \log(N/M))$