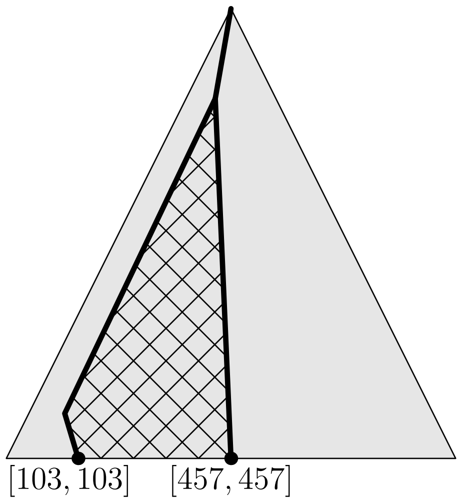
Lecture 4 Overflow

Segment Trees

Key points:

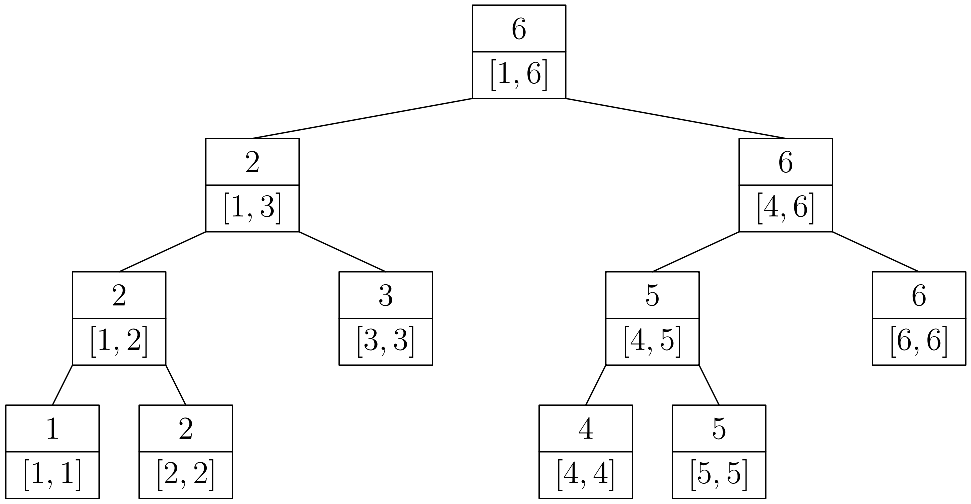
- Used to perform min/max range queries on fixed-length lists whose elements can be updated.
- All operations are $O(\lg n)$.
- Query algorithm:
  1. If query range intersects the range of a child, recurse on that child. Choose min of child results.
  2. If query range contains range of current node's range, return value of current node immediately.
- Update algorithm: Recursively update all nodes on path to leaf corresponding to updated element.

To see why query is $O(\lg n)$, consider a huge Segment Tree, and suppose you are querying the range $[103, 457]$. Consider the two paths from the root to the leaves $[103, 103]$ and $[457, 457]$:



The point where the path forks is the first node on the query path where both left and right children intersect the query range. Every node in the crosshatched region is contained in the query interval, and thus will immediately return when we recurse.

For the (1-based) list $[1, -7, 3, 2, 1, -9]$ we have:



To search for the min of $[2, 5]$ our recursion would terminate at $[2, 2]$, $[3, 3]$, and $[4, 5]$.

Fenwick Trees

Key points:

- Used to perform sum range queries on fixed-length lists whose elements can be updated.
- Allows us to compute sums over ranges $[1, a]$. Compute $[a, b]$ as $[1, b] - [1, a - 1]$.
- All operations are $O(\lg n)$ (except for a separate find operation, which is $O((\lg n)^2)$).
- Query algorithm: Repeatedly sum table indices removing least significant one each time until you hit zero.
- Update algorithm: Repeatedly update table entry adding least significant one each time until you exceed table length.

Suppose you had a large Fenwick Tree and look at table index $x = 101110100000_2$. What is the size of the interval who sum is stored at FT[x], where FT is the Fenwick Tree table? The answer is $2^5 = 32$ as the index ends in 5 zeros. The interval will be $[101110000001_2, 101110100000_2]$.

For the (1-based) list $[1, 2, 1, 3, -1, 1]$ the associated Fenwick Tree table is (with extra info for clarity):

Index_{10}	Index_2	Interval	Value
1	1	$[1,1]$	1
2	10	$[1,2]$	3
3	11	$[3,3]$	1
4	100	$[1,4]$	7
5	101	$[5,5]$	-1
6	110	$[5,6]$	0

What table entries would you look at when computing the sum over $[1, 107]$ where $107_{10} = 1101011_2$? What would you modify when updating the value at entry 107?

For $107_{10} = 1101011_2$ we need to sum the entries with indices 1101011_2, 1101010_2, 1101000_2, 1100000_2, and 1000000_2 corresponding to the ranges

$$[107, 107], [105, 106], [97, 104], [65, 96], [1, 64].$$

When updating $107_{10} = 1101011_2$ we need to update indices 1101011_2, 1101100_2, 1110000_2, 10000000_2, 100000000_2.

Very tight code for Fenwick Tree:

```
class FTree
{
  int[] FT;
  FTree(int size) { FT = new int[size+1]; }
  int LSO(int x) { return x&(-x); }
  int query(int x) { return x == 0 ? 0 : FT[x] + query(x^LSO(x)); }
  int upd(int x, int inc) { for (;x<FT.length;x+=LSO(x)) FT[x]+=inc; }
}
```

Even if you don't fully understand how the Fenwick Tree works, it is important to be able to use it as a black box. That is, you have an object that will answer dynamic sum range queries in $O(\lg n)$ time.