Multi-level range search
Example: Rectangular range counting

Data: 2d points represented as \((x, y)\) coordinate pairs

Query: How many points are inside a given rectangle?

Answer = 5
Binary search tree on $x$-coordinates

Query range: left and right $x$-coordinates of rectangle

Decomposes the points whose $x$-coordinate is in range into

- $O(\log n)$ individual points
- $O(\log n)$ larger sets of points
Multi-level structure

Binary search tree of points sorted by $x$-coordinates

Each node stores a 1D range search structure for intervals of $y$-coordinates, for points in its subtree (e.g. a sorted array)
Using a multi-level structure

To count points in a query rectangle:

- Perform query on $x$-range of rectangle
- For each individual point $(x, y)$ found by query:
  - Test whether $y$ is in range
- For each subtree identified by query:
  - Use 1d structure at subtree root to count descendants whose $y$ coordinate is in range
- Add the results and return the total
Multi-level analysis

If $x$-tree is balanced $\Rightarrow$ each point contributes to $y$-structures in $O(\log n)$ ancestors $\Rightarrow$ total space is $O(n \log n)$

Each rectangle query makes $O(\log n)$ calls to one-dimensional $y$-structures $\Rightarrow$ query time is $O(\log^2 n)$
Making it dynamic

Suppose we want to insert or delete points?

- Use a dynamic binary search tree on \( x \)-coordinates
- Replace 1D sorted arrays by dynamic binary search trees on \( y \)-coordinates

*We cannot rotate quickly* because that would cause big changes to the 1D structures

Instead, use weight-balanced binary search tree on \( x \)-coordinates, and when we rebuild a subtree we also rebuild the recursive structures stored in its nodes
Fractional cascading
Related binary searches

In the multi-level structure for rectangular range counting, each query does $O(\log n)$ binary searches:

- In one-dimensional structures stored at certain tree nodes
- All searching for the same $y$-coordinates (top and bottom coordinates of query rectangle)
- In a related sequence of nodes (children of the nodes on a tree path)

Goal of fractional cascading: Speed up multiple related binary searches without paying too big a penalty in space
A simpler multi-binary-search problem

Data: $k$ sorted lists of numbers $S_0, S_1, \ldots S_{k-1}$

Total length: $n = |S_0| + |S_1| + \cdots + |S_{k-1}|$

No repeated values, even in different lists

Query: find the successors of a given number $q$ in each list

($s_i =$ successor of $q$ in list $S_i$)
Example

Data:

- $S_0 = [0, 10, 20, 30, 40, 50, 60, 70]$
- $S_1 = [1, 2, 13, 25, 27, 51, 57]$
- $S_2 = [21, 22, 31, 32, 33, 41, 99]$
- $S_3 = [67, 68, 69]$

Total length $n = 8 + 7 + 7 + 3 = 25$

Query for $q = 24$ would find

$s_0 = 30 \quad s_1 = 25 \quad s_2 = 31 \quad s_3 = 67$
Naïve solutions

Do the binary searches separately

Space = \(O(n)\) for storing each \(S_i\) as a sorted list
Query time = \(O(k \log n)\) for \(k\) binary searches

Merge into one list

For each value \(x\), store \(k\)-tuple of successors for queries that return \(x\) as their smallest value

0: (0, 1, 21, 67), 1: (10, 1, 21, 67), 2: (10, 2, 21, 67), 10: (10, 13, 21, 67), 13: (20, 13, 21, 67), 20: (20, 25, 21, 67), 21: (30, 25, 21, 67), ...

Binary search in merged sorted array + look up \(k\)-tuple

Space \(O(kn)\), query time \(O(k + \log n)\)
Fractional cascading

Working backwards through the sequence of lists $S_i$, construct $T_i$: merged structure for $(S_i + \text{half the elements of } T_{i+1})$

Choosing the half of the elements that are in odd-numbered positions e.g. if $T = 1, 2, 3, 5, 7, 11, 20$ then $\frac{1}{2} T = 2, 5, 11$

So $T_i$ consists of:

- A sorted array of the merged items from $S_i + \frac{1}{2} T_{i+1}$
- A dictionary mapping each merged item $x$ to a pair $(a, b)$ where one of $a$ or $b$ is $x$, and the other one is the successor of $x$ in the other merged list
- When there is no successor in the other list, use $+\infty$
Example

- \( S_3 = 67, 68, 69 \)  \( T_3 = S_3 \) (nothing to merge)  Half elements: 68
- \( S_2 = 21, 22, 31, 32, 33, 41, 99 \)
- \( T_2 = 21:(21,68), 22:(22,68), 31:(31,68), 32:(32,68), 33:(33,68), 
  41:(41,68), 68:(99,68), 99:(99,\infty) \)
- Half the elements of \( T_2 \): 22, 32, 41, 99
- \( S_1 = 1, 2, 13, 25, 27, 51, 57 \)
- \( T_1 = 1:(1,22), 2:(2,22), 13:(13,22), 22:(25,22), 25:(25,32), 
  27:(27,32), 32:(51,32), 41:(51,41), 51:(51,99), 57:(57,99), 
  99:(\infty,99) \)
- Half the elements of \( T_1 \): 2, 22, 27, 41, 57
- \( S_0 = 0, 10, 20, 30, 40, 50, 60, 70 \)
- \( T_0 = 0:(0,2), 2:(10,2), 10:(10,22), 20:(20,22), 22:(30,22), 
  27:(30,27), 30:(30,41), 40:(40,41), 41:(50,41), 50:(50,57), 
  57:(60,57), 60:(60,\infty), 70:(70,\infty) \)
Searching fractionally cascaded lists

To find the successors of $q$:

- Binary search for successor $t_0$ in merged list $T_0$
- Set $i = 0$
- Then, repeat:
  - Use dictionary for $T_i$ to find the pair $(a, b)$ where $a = s_i =$ successor in $S_i$ and $b$ is successor in $\frac{1}{2} T_{i+1}$
  - Output $s_i$
  - Let $c$ be the (skipped) element of $T_{i+1}$ just before $b$
  - If $q < c$ then $t_{i+1} = c$ else $t_{i+1} = b$
  - Set $i = i + 1$
Example (continued)

To search for the successor of $q = 24$:

- Binary search in $T_0$ finds successor $t_0$: 27:(30,27)
- Output $s_0 = 30$, successor in $S_0$
- Successor in $T_1$ might be either 27 or previous item, 25
- Because $q < 25$, successor in $T_1$ is 25:(25,32)
- Output $s_1 = 25$, successor in $S_1$
- Successor in $T_2$ might be either 32 or previous item, 31
- Because $q < 31$, successor in $T_2$ is 31:(31,68)
- Output $s_2 = 31$, successor in $S_2$
- Successor in $T_3$ might be either 68 or previous item, 67
- Because $q < 67$, successor in $T_3$ is 67
- Output $s_3 = 67$, successor in $S_3$
Fractional cascading analysis

Query time

One binary search + $O(1)$ for each list after the first

Total $O(k + \log n)$

Space and set-up time

Each element of $S_i$ contributes 1 to the length of $T_i$, $\frac{1}{2}$ to the length of $T_{i-1}$, $\frac{1}{4}$ to the length of $T_{i-2}$, ...  

So the total space and total set-up time is $O(n)$

Best combination of time and space from naïve solutions

Also works for multi-level search trees, for example rectangular range counting with $O(n \log n)$ space and $O(\log n)$ query time
Summary
Summary

- Ranking and unranking operations; efficient dynamic implementation by augmenting search tree with relative ranks
- Types of range searching problems including range counting, range reporting, range minimum, and range sum; decomposable problems using associative binary operation
- Dynamic range searching by augmenting search tree with value of its subtree and decomposing range into a logarithmic number of subtrees and individual nodes
- Cell probe model of computing and lower bound on dynamic prefix sums
- Multi-level range search and multi-level augmented binary search trees
- Fractional cascading