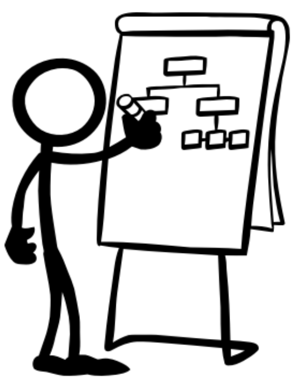
Fractional Cascading and Range Trees



ZHENG Yufei
郑羽霏
יופיי ז'נג

Dec. 12, 2016

Fractional Cascading – Warmup

- **Problem** – predecessor/successor search for x among k sorted lists each of length n

- **Trivial solution** - $O(k \log n)$ time
  - Binary search in each list separately
  - Each search can be done in $O(\log n)$ time

L_1	6	7	26	54
L_2	2	21	29	60
L_3	9	13	31	45

Better solution

- Time complexity $O(\log(nk) + k)$
- Space complexity $O(nk^2)$

- Search in the union of all lists L
- Each element of L holds the result of the search query for each of the lists

L_1	6	7	26	54
L_2	2	21	29	60
L_3	9	13	31	45

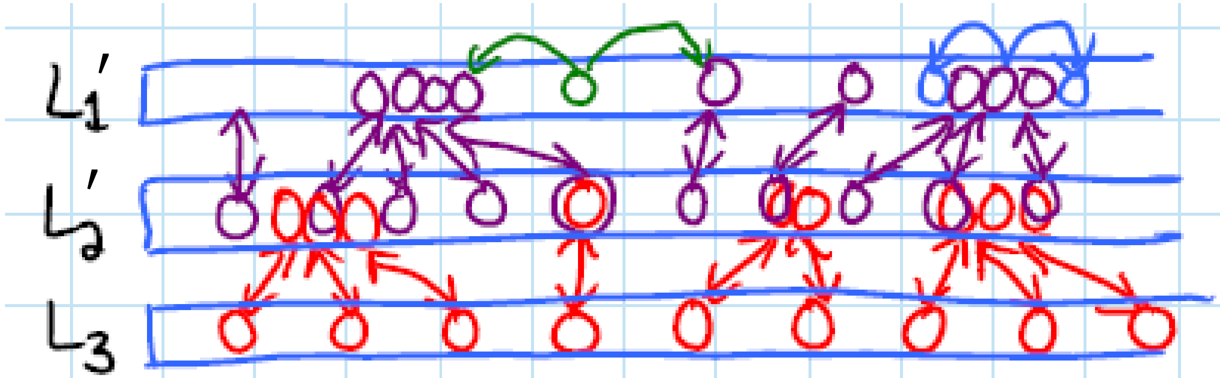
L	2	6	7	9	13	21	26	29	31	45	54	60
	0	0	1	2	2	2	2	3	3	3	3	4
	0	1	1	1	1	1	2	2	3	3	3	3
	0	0	0	1	2	2	2	2	2	3	4	4

Fractional Cascading – Special Case

Data Structure

- Let $L'_i = L_i \cup F(L'_{i+1})$

  $F(L) \triangleq$ every other element of L
- Link between identical elements in L'_i and L'_{i+1}
- Each element in L_i stores pointer to previous/next element in $L'_i - L_i$
- Each element in $L'_i - L_i$ stores pointer to previous/next element in L_i

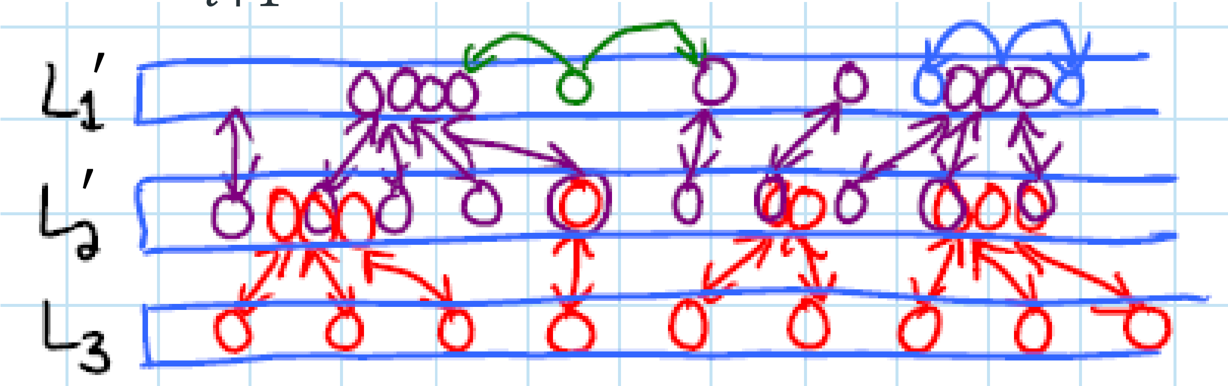


Fractional Cascading – Special Case

Search Algorithm

Search(x):

- Binary search in L'_1
- From $i = 1$ to $k - 1$
  - If amid $L'_i - L_i$, follow pointers to neighbors in L_1 to solve the query problem in L_i.
  - If amid L_i, follow pointers to neighbors in $L'_i - L_i$ (Else stay)
  - Walk down to L'_{i+1}



Fractional Cascading – Special Case

- **Time Complexity** - $O(\log n + k)$ Only 1 binary search in L'_1
- **Space Complexity** - $O(nk)$

$$
\begin{aligned}
|L'_i| &= |L_i| + \frac{1}{2}|L'_{i+1}| \\
&= |L_i| + \frac{1}{2}\left(|L_{i+1}| + \frac{1}{2}|L'_{i+1}|\right) \\
&= |L_i| + \frac{1}{2}|L_{i+1}| + \frac{1}{2^2}|L_{i+2}| + \cdots + \frac{1}{2^{k-i}}|L_k| \\
&= |L_i| \sum_{j=0}^{k-i} \frac{1}{2^j} \leq 2n
\end{aligned}
$$

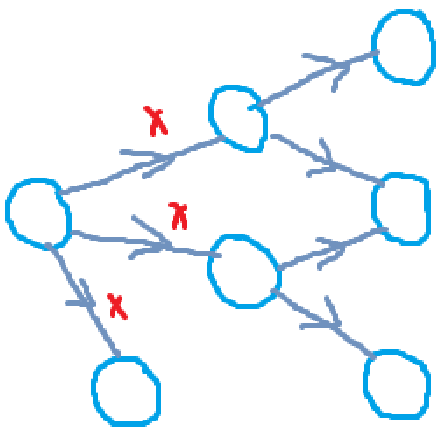
Each element in the list has **constant** number of pointers

$$
\sum_{i=1}^{k} |L'_i| = \sum_{i=1}^{k}\left(|L_i| \sum_{j=0}^{k-i} \frac{1}{2^j}\right) \leq 2kn
$$

Fractional Cascading – General Case

- A **directed graph** where each
  - vertex contains a set of sorted elements
  - edge labeled with range $[a, b]$
  - locally bounded degree:

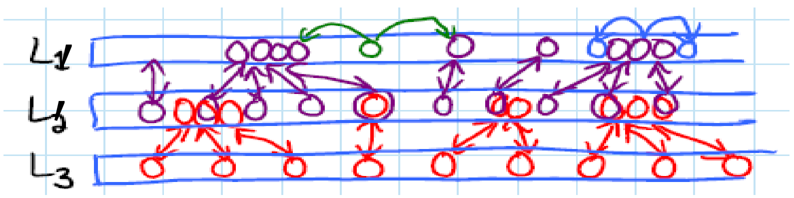
    # incoming edges whose labels $\ni x$
    is less or equal to c



- **Search Algorithm** –

  find x in k vertices' sets by navigating from some vertex, along edges whose labels $\ni x$
  - **Time Complexity** - $O(k + logn)$

Fractional Cascading, Chazelle and Guibas, 1986

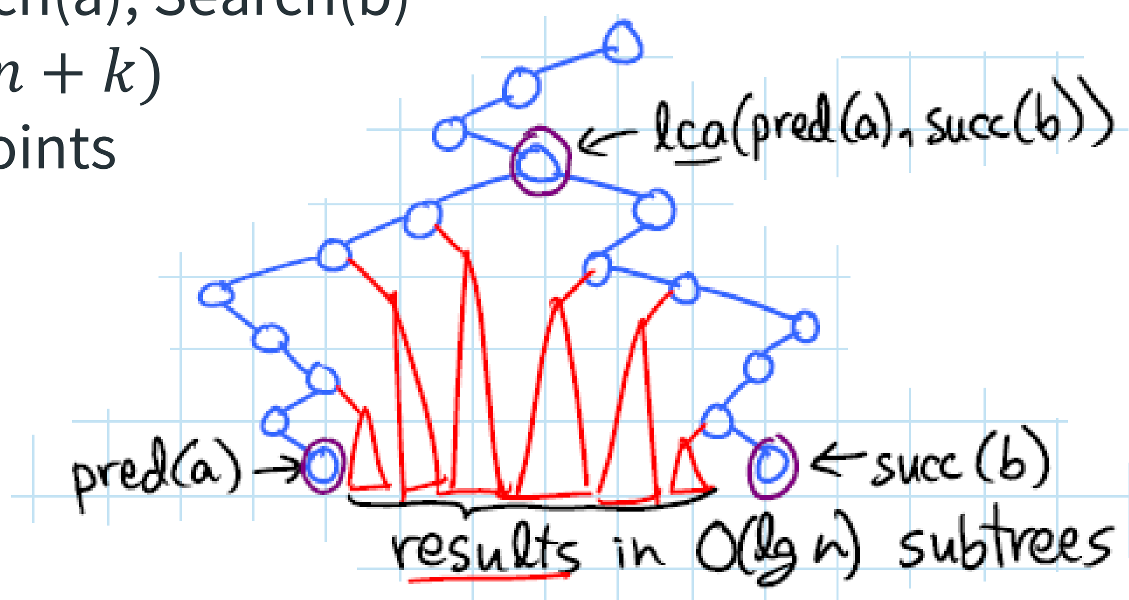


Range Trees – Recall

- **1D**:
  - **Data Structure**: balanced BST on leaves
  - Internal node key = maximum of left subtree
  - Leaves = points

  - **Query([a,b])**: Search(a), Search(b)
  - **Query time**: $O(\log n + k)$

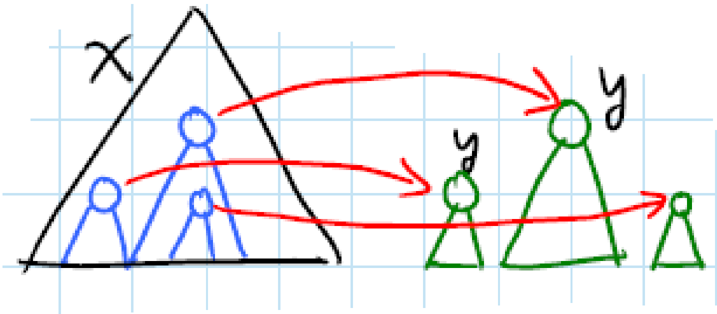
    k - # reported points



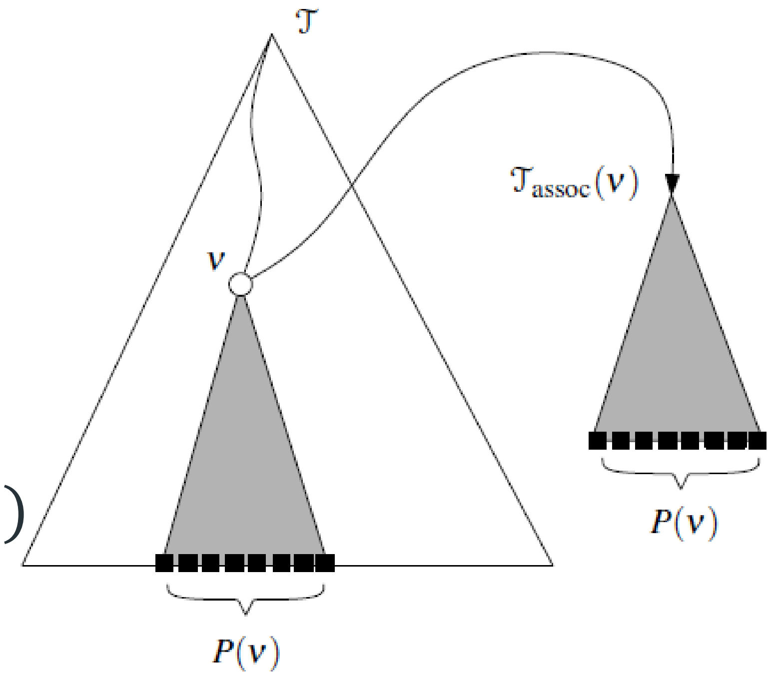
Range Trees – Recall

- **2D**:
  - **Data Structure**:

1D tree on x + each subtree links to 1D tree on y on same points



  - **Query**$([a_1, b_1] \times [a_2, b_2])$:
    - x Query$([a_1, b_1])$
    - Follow pointers
    - y Query$([a_2, b_2])$
  - **Query time**: $O(\log^2 n + k)$

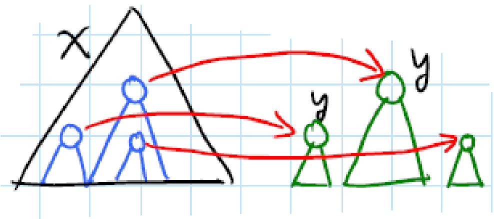


- ***d*D**: **Query time**: $O(\log^d n + k)$

Layered Range Tree

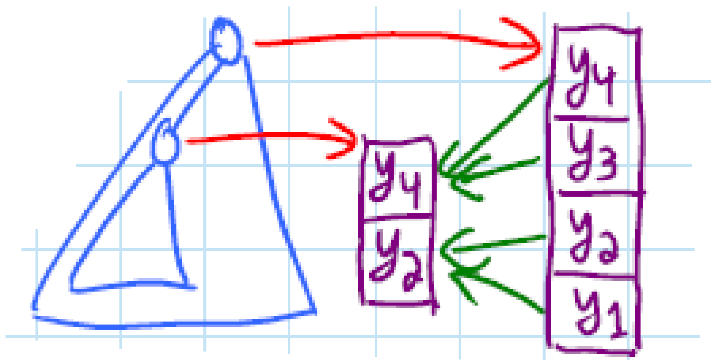
2D:

- **Data Structure**: 1D tree on x + sorted arrays on y and store pointers:
  - from each x subtree to its corresponding y array
  - from y arrays of node v to y arrays of left(v), right(v)
- **Query**$([a_1, b_1] \times [a_2, b_2])$:
  - x Query$([a_1, b_1])$
  - Search once in root y structure
  - Carrry search results down to result subtree roots
- **Query time**: $O(\log n + k)$



dD:

- **Data Structure**:
  same dD range tree + 2D base case
- **Query time**: $O(\log^{d-1} n + k)$



Scaling and Related Techniques for Geometry Problems, Gabow, Bentley and Tarjan, 1984