2-INF-237 Vybrané partie z dátových štruktúr

2-INF-237 Selected Topics in Data Structures

- Instructor: Broňa Brejová
- E-mail: brejova@fmph.uniba.sk
- Office: M163

- Course webpage: http://compbio.fmph.uniba.sk/vyuka/vpds/

Partially persistent data structures

Update only current version, query any old version

versions linearly ordered

Arbitrary pointer machine data structure

with at most $O(1)$ incoming pointers per node

and $f(n)$ update, $g(n)$ query

Partially persistent version with node copying

$O(f(n))$ amortized update, $O(g(n))$ query

Retroactive data structures

Insert updates to the past, delete past updates,

query at any past time relative to the current set of updates

Search problem:

maintain a set S with insert and delete

support query(x, S)

Decomposable search problem

query$(x, A \cup B) =$ query$(x, A) \square$ query(x, B)

Arbitrary data structure

$f(n)$ update, $g(n)$ query

Totally retroactive version

$O(f(n) \log n)$ amortized update, $O(g(n) \log n)$ query

Bentley–Ottmann algorithm

for finding intersections of line segments

n line segments, k intersections

Sweepline algorithm, sweeps from left to right

Maintains priority queue of events: start of a line, end of a line, intersection

Balanced binary search tree of segments at current x-coordinate

$(2n + k)\times$Insert, $(2n + k)\times$ExtractMin

$O((n + k) \log n)$ time

Planar point location

- Plane subdivided into regions by non-intersecting straight lines (planar graph)
- Given point (x, y), which face contains it?
- Examples: regions in a map, GUI elements, ...
- Nearest neighbor via Voronoi diagram
- Static version: preprocess a fixed graph
- Dynamic version: edge added/removed

Vertical ray shooting

- Given set of non-intersecting line segments
- Query: which edge first intersects a vertical ray starting in (x, y)?
- In static case implies planar point location

  (each edge keeps face ID)
- First assume that all line segments horizontal

Vertical ray shooting for horizontal line segments (static)

- Sweep with partially persistent balanced BST

  Left segment endpoint (x_1, y): insert y at time x_1

  Right segment endpoint (x_2, y): delete y at time x_2

- $O(n \log n)$ time preprocessing

- Given ray from (x, y), search for successor of y at time x

- $O(\log n)$ query

Vertical ray shooting for horizontal line segments (dynamic)

- Use retroactive binary search tree
- $O(\log^2 n)$ queries last time, $O(\log n)$ version also exists
- Insert line segment $(x_1, y), (x_2, y)$:

  Insert(x_1,insert(y))

  Insert(x_2,delete(y))
- Delete line segment $(x_1, y), (x_2, y)$:

  Delete(x_1,insert(y))

  Delete(x_2,delete(y))

Vertical ray shooting with arbitrary segments

- Segments do not cross, but any direction
- Static version still with partially persistent BST
- When searching successor of y at time x,
  use comparison function which depends on x
- Dynamic version does not work in $O(\log n)$

Also interesting is ray shooting in arbitrary direction

– no poly-log algorithms known

– motivated by ray tracing

Orthogonal range searching

- Maintain a set of points in R^d
- Query: find points in box $[a_1, b_1] \times \cdots \times [a_d, b_d]$
  existence / count / report k
- Static / dynamic case
- E.g. database queries combining d columns
- Also related to nearest neighbour
- Range trees $O(\log^d n + k)$ query
- Layered range trees $O(\log^{d-1} n + k)$ query
- Updates in range trees $O(\log^d n)$ amortized
- Further improvements exist

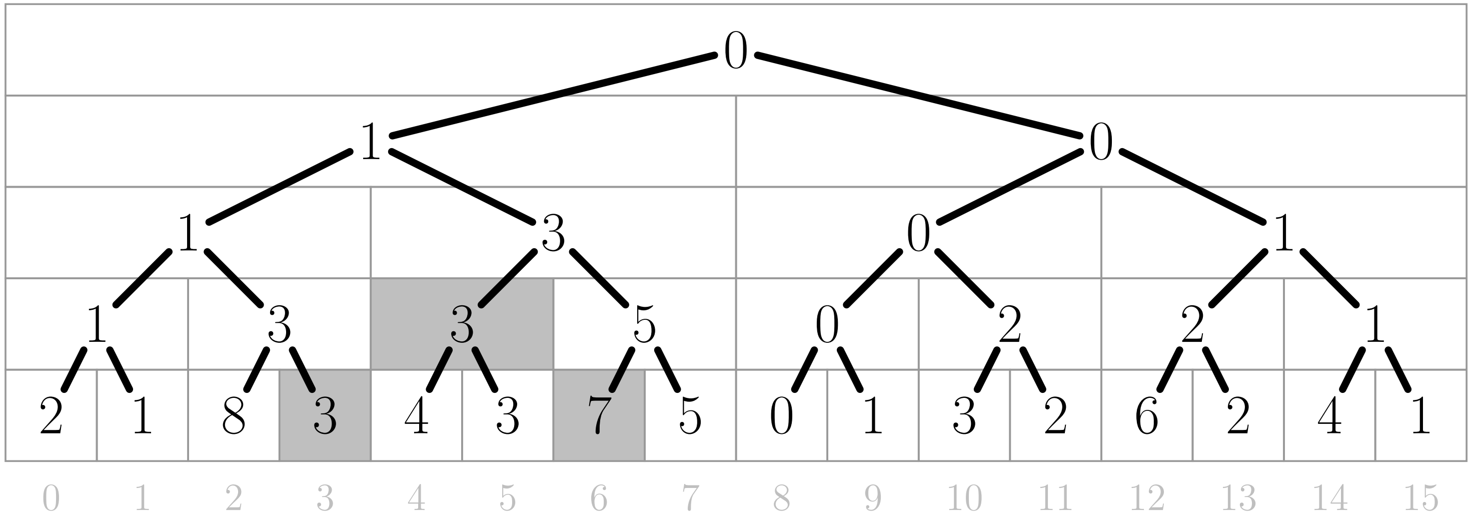
Range trees: 1D case

Report/count points in an interval $[a, b]$

- Balanced binary search tree/segment tree
- Points in the leaves
- Internal nodes store maximum in the left subtree and subtree size for fast counting
- Find predecessor of a, successor of b
- Leaves between form the answer
  $O(\log n)$ subtrees (canonical decomposition)

Canonical decomposition

Decompose query interval $[x, y)$ to a set of disjoint tree intervals



Canonical decomposition

Decompose query interval $[x, y)$ to a set of disjoint tree intervals

- Current node $[i, j)$, and its left child $[i, k)$
  invariant: $[i, j)$ overlaps with $[x, y)$
- If $[i, j) \subseteq [x, y)$, return $\{[i, j)\}$
- $R = \emptyset$
- If $[i, k)$ overlaps with $[x, y)$, recurse on left child, add to R
- If $[k, j)$ overlaps with $[x, y)$, recurse on right child, add to R
- Return R

Range trees: 2D case

- Build BST for x-coordinate
- Consider internal node v
- Build BST tree for subtree rooted at v in y-coordinate
- Each point in $O(\log n)$ y-coord trees
- Search for $[a_1, b_1] \times [a_2, b_2]$:
  – find $O(\log n)$ subtrees for $[a_1, b_1]$ according to x
  – search in each according to y

d dimensions:

- Every node in dimension i has a range tree for remaining dimensions
- Query $O(\log^d n)$, space and preprocessing $O(n \log^{d-1} n)$

Layered range trees: $O(\log^{d-1} n)$ for $d \geq 2$

a.k.a fractional cascading

- Replace y BSTs by sorted arrays
- Root of x BSTs has all points sorted by y in array
- Array in a child a subset of parent's

  link from parent array to successors in child array
- At the root find $[a_2, b_2]$ in the array
- Follow array links as traversing the tree
- In higher dimensions use this at the last dimension
- Saves $\log n$ factor

Dynamic range trees: outline

- Use scapegoat trees
- Rebalancing: rebuild an entire subtree
- If rebuild linear, $O(\log n)$ amortized updates
  – pay $O(1)$ on each level towards future rebuilds
  – linearly many updates between 2 rebuilds of the same node
- If rebuild $O(n \log n)$, $O(\log^2 n)$ amortized updates
  – pay $O(\log n)$ on each level towards future rebuilds
- In layered range trees: $O(\log^d n)$ amortized update

Review: Scapegoat trees

- Lazy amortized binary search trees
- Do not require balancing information stored in nodes
- Insert and delete $O(\log n)$ amortized

  search $O(\log n)$ worst-case
- Invariant: keep the height of the tree at most $\log_{3/2} n$
- When invariant not satisfied, completely rebuild a subtree

Wavelet tree (Grossi, Gupta, Vitter 2003)

$\Sigma_0 = \{\$, ., a\} \quad \Sigma_1 = \{e, m, u\}$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
S[i]	e	m	a	.	m	a	.	m	a	m	u	.	m	a	m	a	.	m	a	.	e	m	u	$
B[i]	1	1	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0	1	1	1	0

S0 a.a.a.aa.a.$ S1 emmmummmemu

$\Sigma_{00} = \{\$\}, \Sigma_{01} = \{., a\}, \quad \Sigma_{010} = \{.\}, \Sigma_{011} = \{a\}$
$\Sigma_{10} = \{e\}, \Sigma_{11} = \{m, u\}, \quad \Sigma_{110} = \{m\}, \Sigma_{111} = \{u\}$

i	0	1	2	3	4	5	6	7	8	9	10	11
S0[i]	a	.	a	.	a	.	a	a	.	a	.	$
B0[i]	1	1	1	1	1	1	1	1	1	1	1	0

S00 a.a.a.aa.a. S01 $

Store

B[i] =110010010110101001001110

B0[i] = 111111111110 B1[i] = 011111111011

B01[i] = 10101011010 B11[i] = 000100001

2D range searching via wavelet trees

- Points $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ s.t. $y_i < y_{i+1}$
- Wavelet tree for ”string” $T = x_0, \ldots, x_{n-1}$
- Wavelet tree similar to BST/segment tree for x-coordinate:

  – each node an interval
- Before: rank(a, i): the number of occurrences of a in $T[0..i]$
- Extend to: rank(a, b, i): the number of occurrences of values from $[a, b]$ in $T[0..i]$
- Canonical decomposition of $[a, b]$ in $O(\log n)$ intervals
- If one of the children $[c, d]$ of the current node is a canonical interval, one binary rank in parent can count rank(c, d, i) in $O(1)$
- Overall rank(a, b, i) in $O(\log n)$ time

2D range searching via wavelet trees (cont.)

- Points $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ s.t. $y_i < y_{i+1}$
- Wavelet tree for $T = x_0, \ldots, x_{n-1}$ + array $y_0, \ldots, y_{n-1}$

Counting points in $[a_1, b_1] \times [a_2, b_2]$:

- Find substring $x_i \ldots x_j$ of T corresponding to $[a_2, b_2]$ (binary search in array y)
- Return rank(a_1, b_1, j) − rank$(a_1, b_1, i - 1)$
- Counting points in $O(\log n)$, small memory, static
- Reporting points takes $O(\log n)$ per point, can be improved

Exercise

- Consider a static set of points in 2D, each with a cost (for example hotels...)
- Find the lowest-cost point in a given rectangle
- How to add to layered range trees and to wavelet trees?

Exercise

- Preprocess text T (e.g. to suffix array plus other structures)
- Query: (P, i, j): find/count occurrences of P in $T[i..j]$
- How can we use range searching for this?