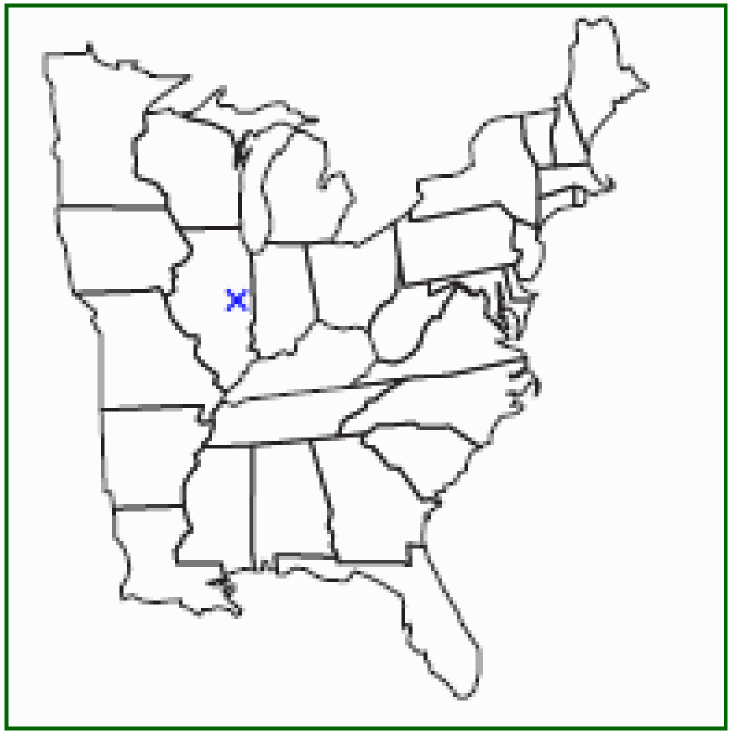
Quadtrees – Hierarchical Grids

Seminar in Algorithms, Spring 2012

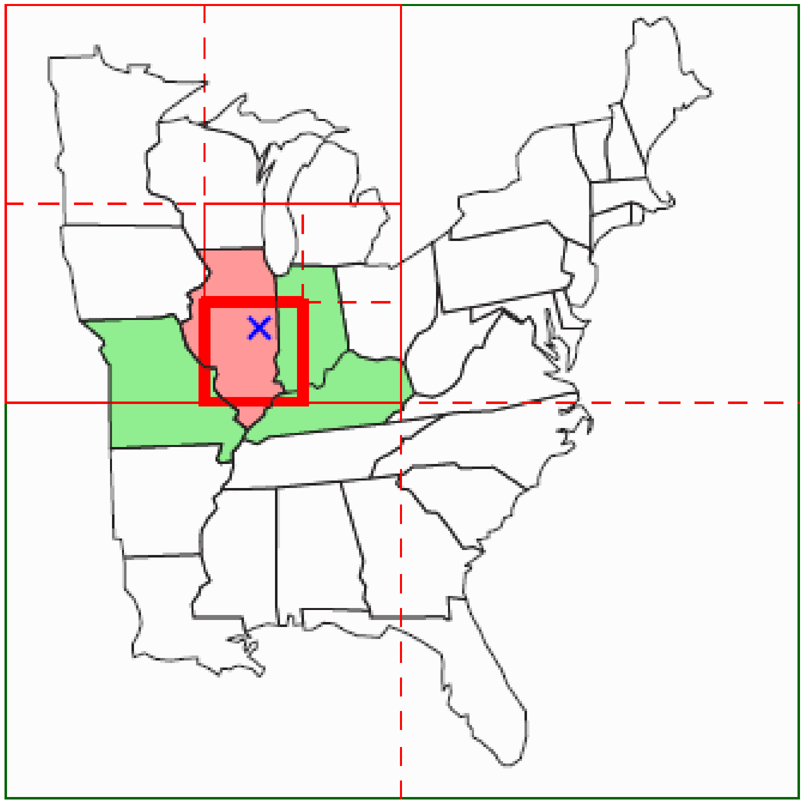
Point Location

Given a partition of the space into disjoint regions, preprocess it so that given a query Point, we can determine in which region it lies



Quadtrees

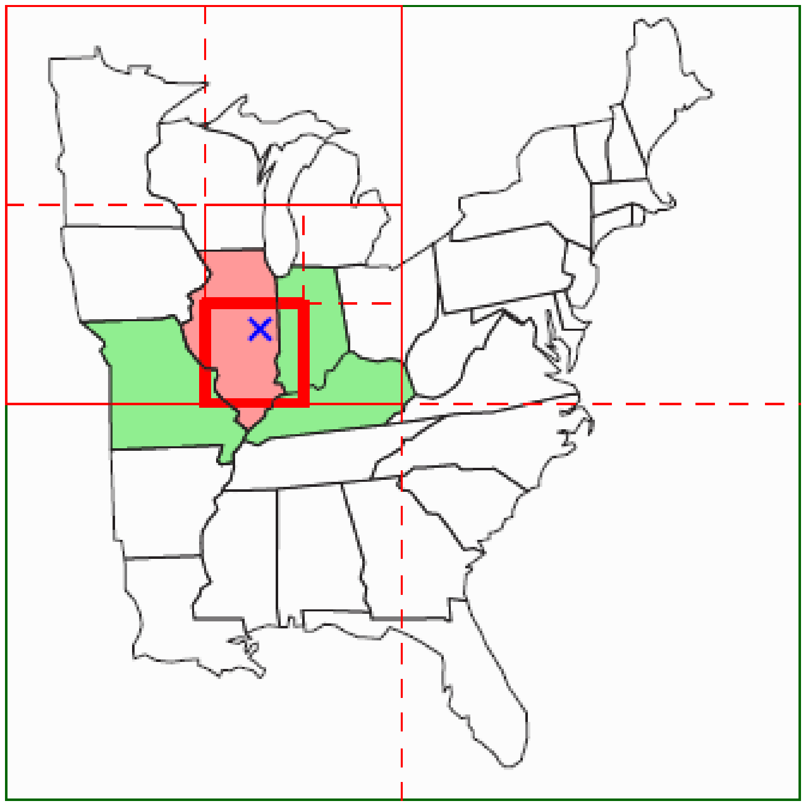
- Each node v corresponds to a cell $\square_v$
- The root node corresponds to the unit square
- Each node has 4 children, represent four equal sized squares inside it



Quadtrees

Complexity:

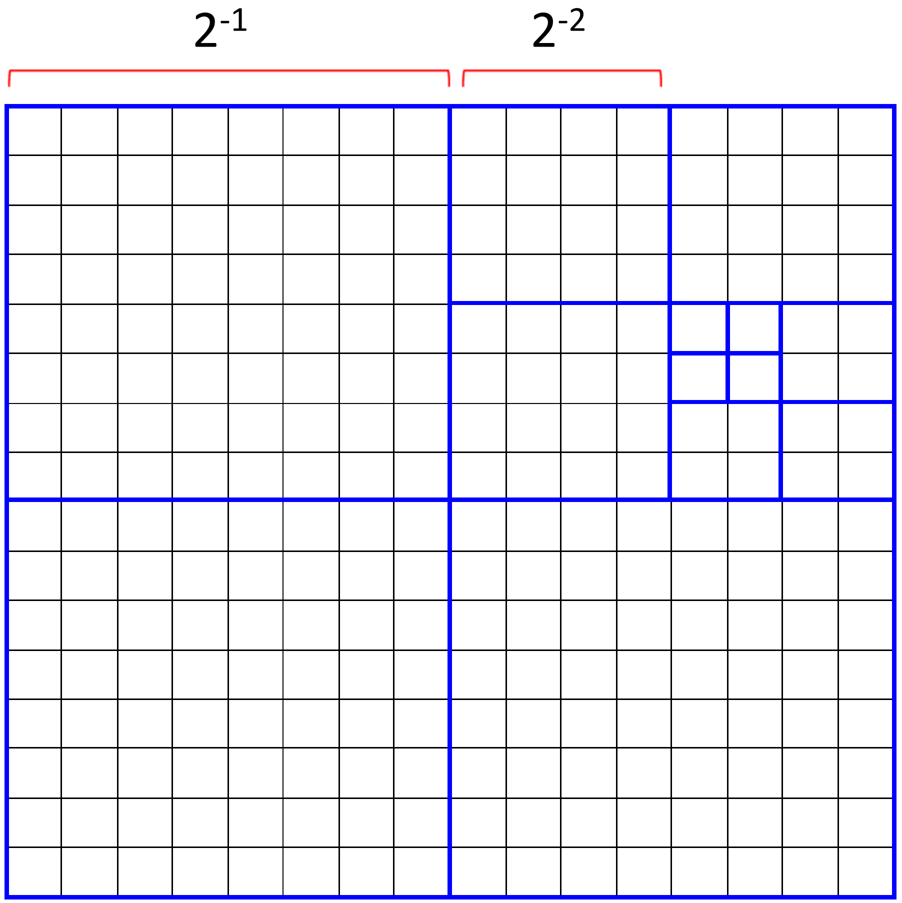
- Unbounded in the worst case
- For reasonable input:
  - Space: **O(n)**
  - Query time: **O(n)** in the worst case



Quadtrees as Grids

Canonical Square

- Contained in the unit square
- A cell in the grid G_r where r is a power of 2



Quadtrees as Grids

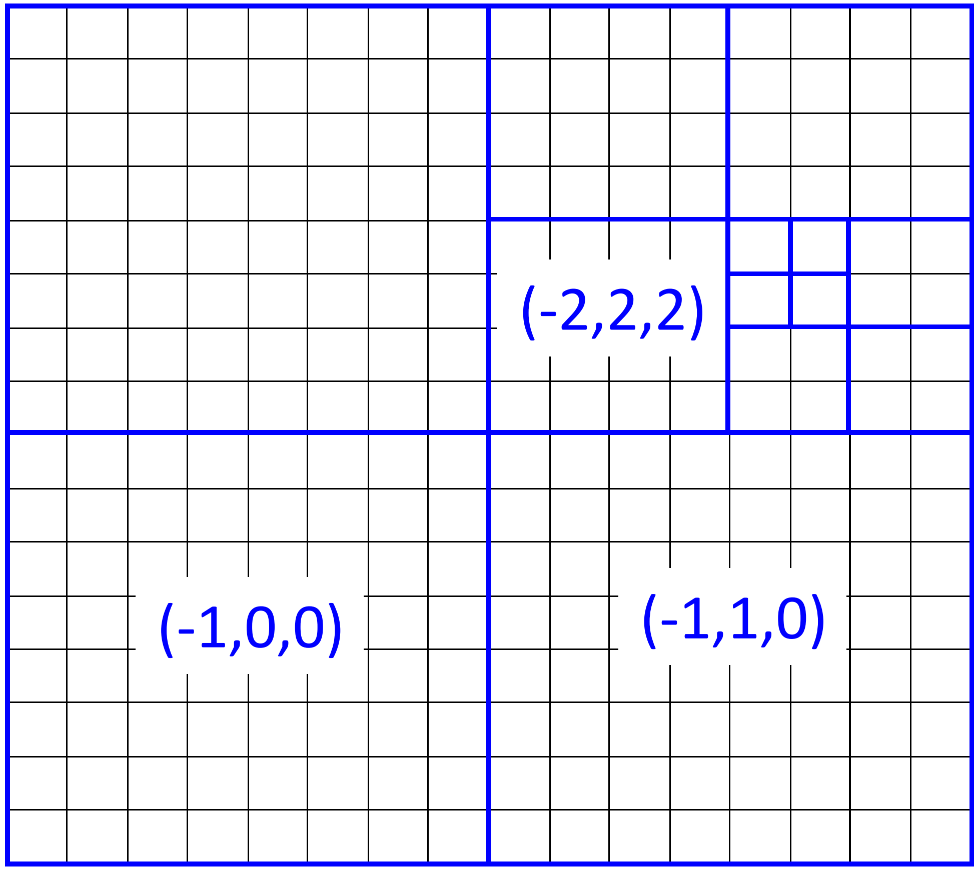
A node v of depth *i* in a quadtree is associated with a square with edge length 2^{-i}, and can be uniquely defined by the triple:

$$\left(l(v), \lfloor x/r \rfloor, \lfloor y/r \rfloor\right)$$

$l(v) = -i$

(x, y) is any point in the square

Quadtrees as Grids



Fast Point-Location

Preprocessing:

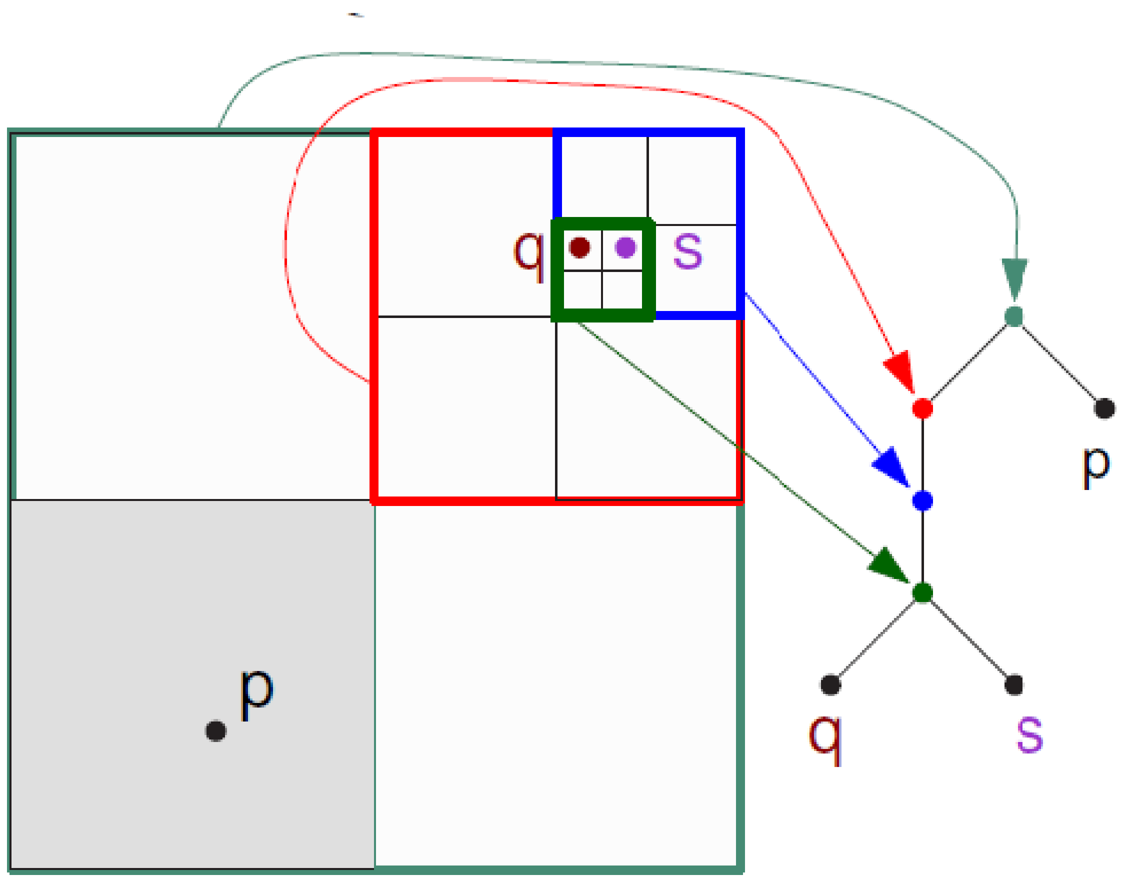
1. Build a quadtree, and store each node in a hash-table, according to their IDs
2. Remember the maximal depth in the tree, *h*

Query:

Given a query point *q*, find the deepest node in which *q* lies, by performing binary search on the depth, each time checking whether the node in depth *i* containing *q* exists in the tree

Query time: **O(log*h*)**

Quadtree of a set of points



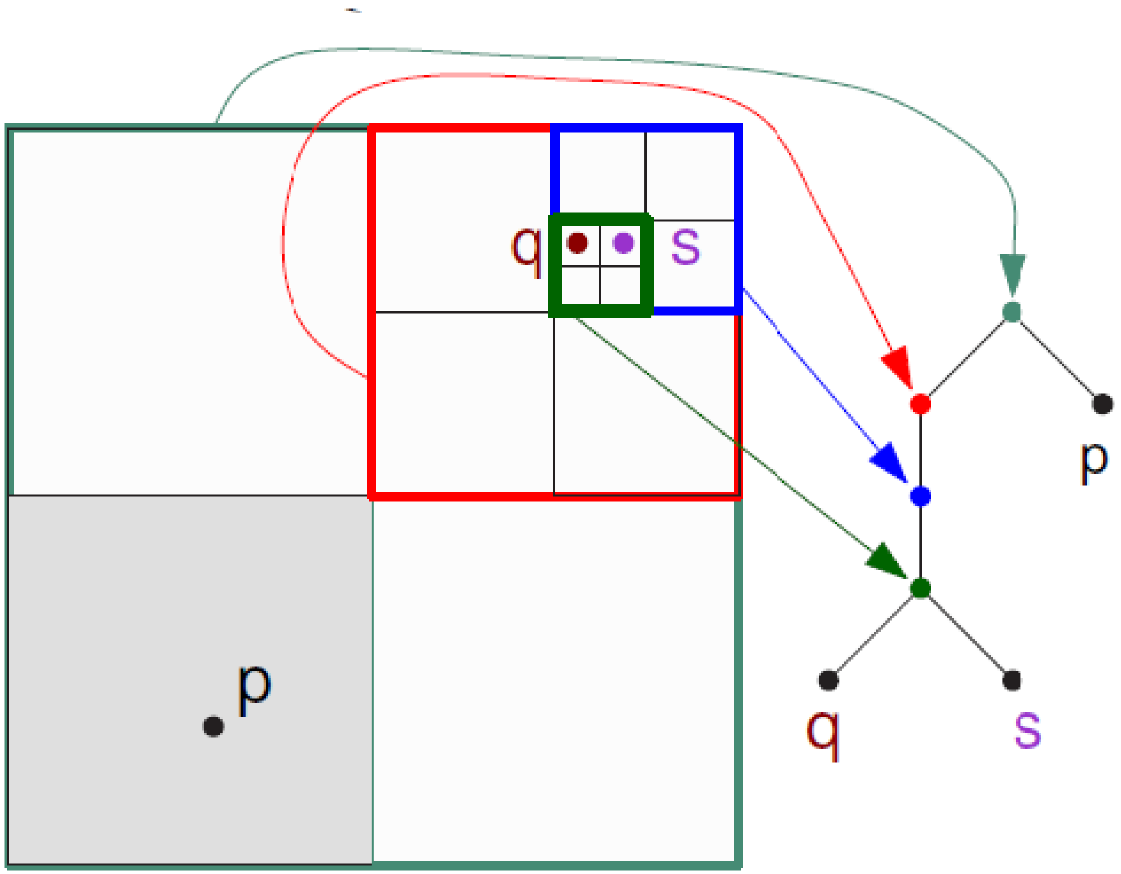
Quadtree of a set of points

Given a set P of n points, the spread of P is defined as:

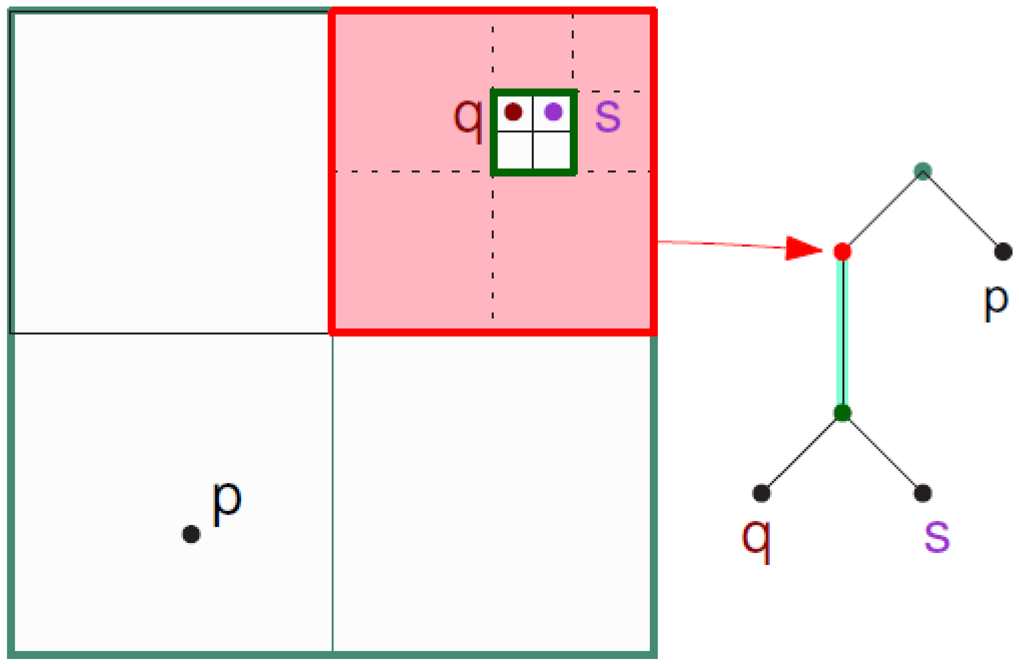
$$\Phi(P) = \frac{\max_{p,q \in P} \|p - q\|}{\min_{p,q \in P,\, p \neq q} \|p - q\|}$$

If $diam(P) \geq \frac{1}{2}$, a quadtree that stores the points of P in the leaves can be constructed in $O(n \log \Phi(P))$ time and will have depth of $O(\log \Phi(P))$

Quadtree of a set of points



Compressed quadtrees



Compressed quadtrees

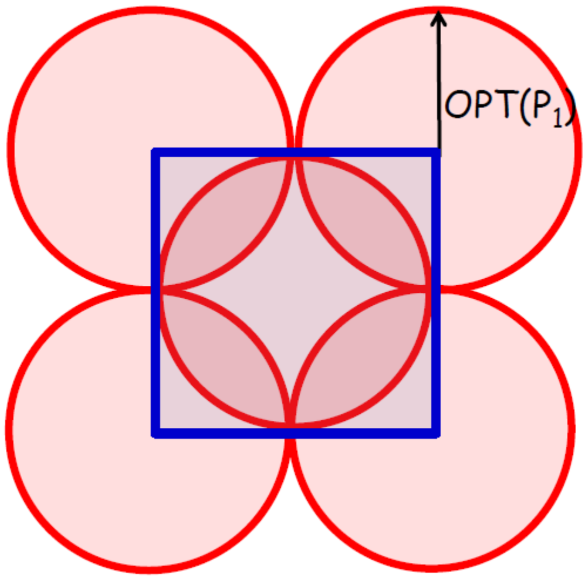
- A compressed node has a single child and it represents the region around the square represented by its child
- uncompressed nodes are either leaves or split the point set in their square into two or more subsets
- The Compressed Quadtree has linear size

Constructing a compressed quadtree

P is a set of *n* points in the unit square

- Compute a disk *D* of radius *r*, which contains at least $\frac{n}{10}$ of the points, such that $r \leq 2r_{opt}$
- Compute the grid G$_\alpha$ where $\alpha = 2^{\lfloor \log r \rfloor}$

1. There exist a cell with at least $\frac{\left(\frac{n}{10}\right)}{25}$ points
2. No cell contains more than $5\left(\frac{n}{10}\right)$ points



Constructing a compressed quadtree

- Find the cell □ in G_α containing the largest number of points
- Split P into P_{in} and P_{out} , we know that:

$$|P_{in}| \geq \frac{n}{250} \qquad |P_{out}| \geq \frac{n}{2}$$

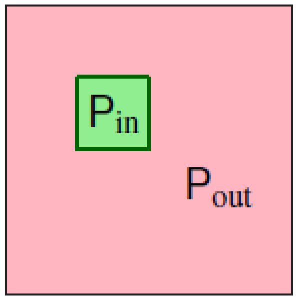
- Compute recursively the quadtrees T_{in} and T_{out}

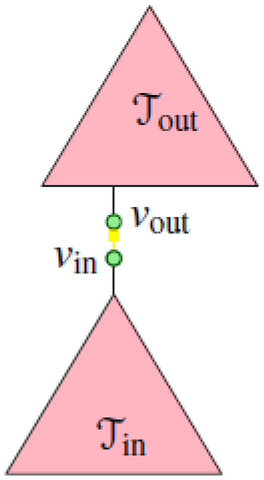
Constructing a compressed quadtree

- Create in both trees a node that corresponds to □:

  In T_{in} it would be the root node v_{in}

  In T_{out} it would be a new leaf, v_{out}





- Hang v_{out} on v_{in}.

  If the new node is redundant, compress it

  If v_{out} is a compressed node, compute the new compressed node

Constructing a compressed quadtree

Computing a compressed node of two points is Not trivial. The following problem is equivalent:

Given $\alpha, \beta \in [0.1)$, in base two, what is the index of the first bit after the period in which they differ?

Assuming this is done in constant time, the overall time of building the compressed tree is:

$$T(n) = O(n) + T(|P_{in}|) + T(|P_{out}|) = O(n \log n)$$

Point Location in a compressed quadtree

Separator of a tree with n nodes:

a node *v* in *T* such that if we remove *v* from *T* we remain with a forest, in which every tree has at most $\left\lceil \frac{n}{2} \right\rceil$ vertices

Every tree has a separator and it can be computed in linear time

Point Location in a compressed quadtree

Pre-processing:

- Find a separator *v* in *T*
- Create a finger tree T', in its root f_v keep a pointer to *v*
- Recursively build finger trees from each tree in the forest T\{*v*} and hang them on f_v

The depth of the finger tree T' is:

$$D(n) \leq 1 + D\left(\left\lceil \frac{n}{2} \right\rceil\right) = O(\log n)$$

Point Location in a compressed quadtree

Query:

- Given a point *q*, at each node f_v of *T'* check whether $q \in \square_v$

  if $q \notin \square_v$, search into the child of f_v which corresponds to the connected components outside $\square_v$

  if $q \in \square_v$, search into the child that contains $\square_v$

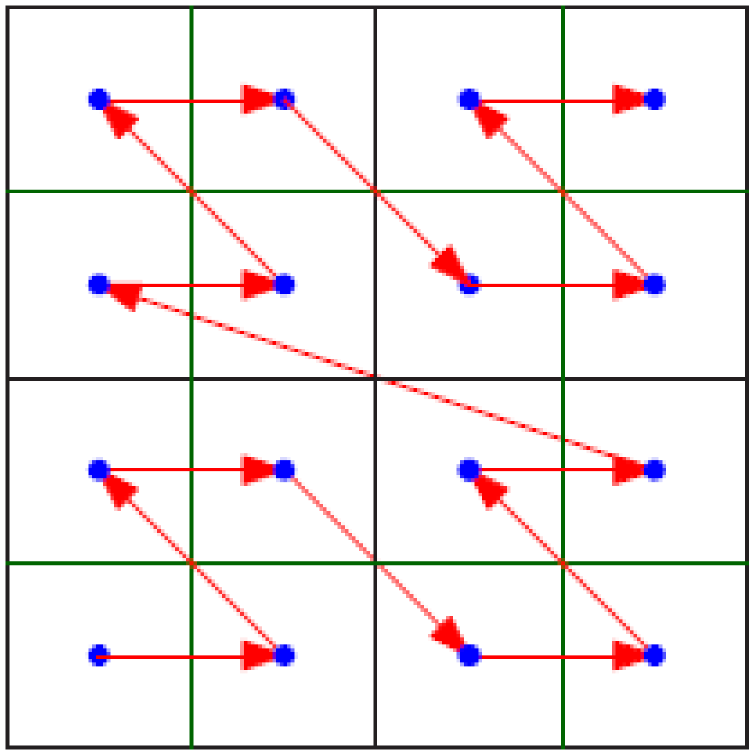
The running time is $O(\log n)$

Dynamic Quadtrees

- Define an order on the quadtrees nodes
- Store the nodes in a data structure for ordered sets
    - Easy to implement
    - Allow performing basic operations efficiently (insertion, deletion, point location)

Dynamic Quadtrees

DFS traversal of the tree:



Dynamic Quadtrees

Q-order:

- $\square_1 \prec \square_2$ iff a DFS traversal visits $\square_1$ before $\square_2$
- If $p \in \square_1$ then $\square_1 \prec p$

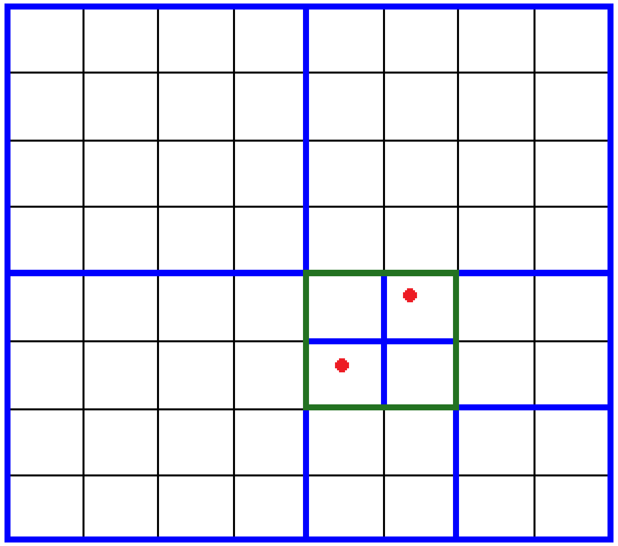
  Otherwise, if $\square_1 \in G_i$, let $\square_2 \in G_i$ be the cell that contains p. Then $\square_1 \prec p$ iff $\square_1 \prec \square_2$
- $p \prec q$ iff $\square_p \prec \square_q$ where $\square_p, \square_q \in G_i$, $p \in \square_p$, $q \in \square_q$ and G_i is a fine enough grid s.t. p and q are in different cells

Dynamic Quadtrees

Lowest Common Ancestor:

For any $p,q \in [0,1]^2$, *lca*(p,q) is the smallest canonical square that contains p and q.

For any two cells, their *lca* will be the *lca* of their centers.



Dynamic Quadtrees

Computing the Q-order between two cells $\square_1$, $\square_2$:

If $\square_1 \subseteq \square_2$ then $\square_2 \prec \square_1$

If $\square_2 \subseteq \square_1$ then $\square_1 \prec \square_2$

Otherwise, compute $\square$=*lca*($\square_1$, $\square_2$). $\square_1 \prec \square_2$ iff $\square_1$ is visited before $\square_2$ in the DFS traversal of $\square$

Point Location in Dynamic Quadtrees

Given a query point *q*, if the tree is regular:

1. Find the two consecutive cells in the list such that *q* lies between them in the Q-order
2. Let □ be the last cell in the list before *q*. Then □ is the quadtree leaf containing *q*

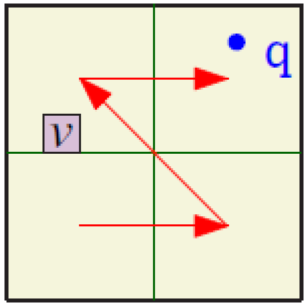
The query time is O(Q(n)), where Q(n) is the time to perform a query in the data structure used

Point Location in Dynamic Quadtrees

If the tree is compressed:

Find the last node *v* in T such that $\square_v \prec q$

- If $q \in \square_v$ return $\square_v$
- Otherwise, *v* is a child of a compressed node *w*. compute $\square = lca(\square_v, q)$. If $\square$ corresponds to the a compressed node in T, return it Otherwise, return the predecessor of $\square$



- Running time: O(Q(*n*))

Operations in Dynamic Quadtrees

T is a compressed dynamic quadtree, *q* is a new point

- Find w, the node in the tree that contains *q* (returned by point location)
  1. w is an empty leaf: store *q* in *w*
  2. w is a non-empty leaf, $p \in \square_w$:

     find □ = *lca(p,q)*, insert □ to the tree under *w*, and create its children, containing *p* and *q*
  3. w is a compressed node, *z* is its child:

     find □ = *lca*($\square_z$,*q*). Insert □ to the tree under w and create its children. Insert *z* and *q* to the appropriate leaves

Operations in Dynamic Quadtrees

Each case requires a constant number of operations, so the overall time is O(Q(n)).

Deletion can be performed in a similar way, in time O(Q(n)).