Similarity Searching:

Indexing, Nearest Neighbor Finding, Dimensionality Reduction, and Embedding Methods, for Applications in Multimedia Databases

Hanan Samet[*]

hjs@cs.umd.edu

Department of Computer Science

University of Maryland

College Park, MD 20742, USA

Based on joint work with Gisli R. Hjaltason.

[*] Currently a Science Foundation of Ireland (SFI) Walton Fellow at the Centre for Geocomputation at the National University of Ireland at Maynooth (NUIM)

Outline

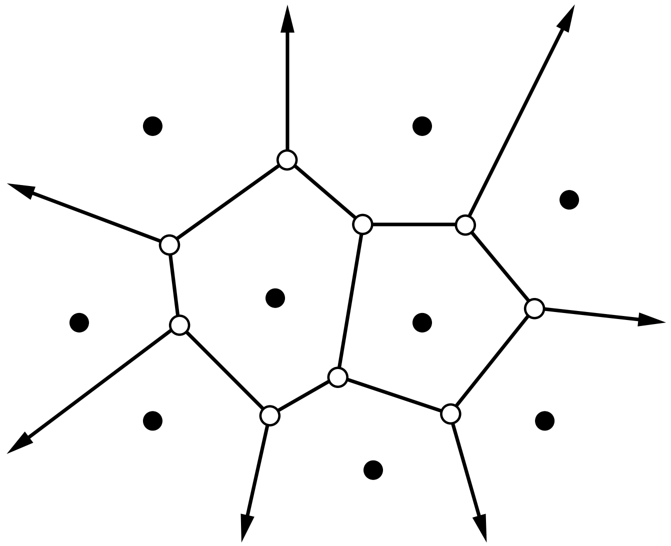
1. Similarity Searching
2. Distance-based indexing
3. Dimension reduction
4. Embedding methods
5. Nearest neighbor finding

Similarity Searching

- Important task when trying to find patterns in applications involving mining different types of data such as images, video, time series, text documents, DNA sequences, etc.
- Similarity searching module is a central component of content-based retrieval in multimedia databases
- Problem: finding objects in a data set S that are similar to a query object q based on some distance measure d which is usually a distance metric
- Sample queries:
  1. point: objects having particular feature values
  2. range: objects whose feature values fall within a given range or where the distance from some query object falls into a certain range
  3. nearest neighbor: objects whose features have values similar to those of a given query object or set of query objects
  4. closest pairs: pairs of objects from the same set or different sets which are sufficiently similar to each other (variant of spatial join)
- Responses invariably use some variant of nearest neighbor finding

Voronoi Diagrams

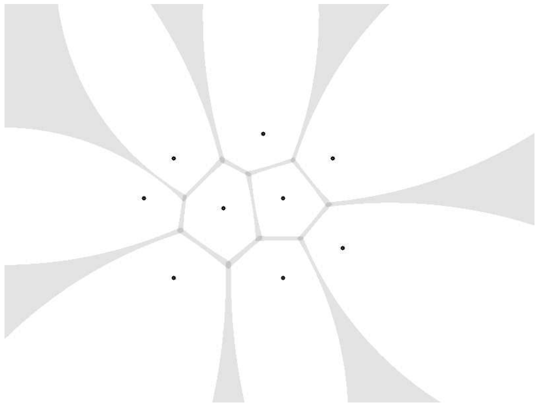
- Apparently straightforward solution:
  1. Partition space into regions where all points in the region are closer to the region's data point than to any other data point
  2. Locate the Voronoi region corresponding to the query point

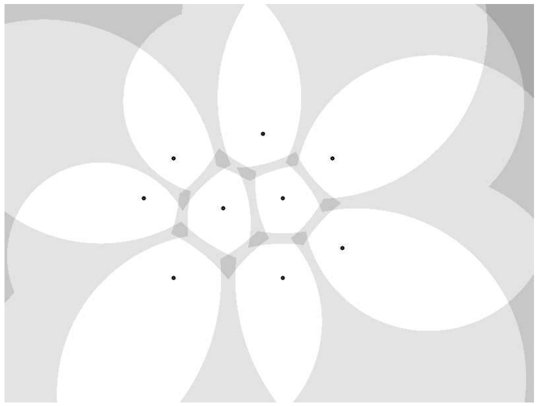


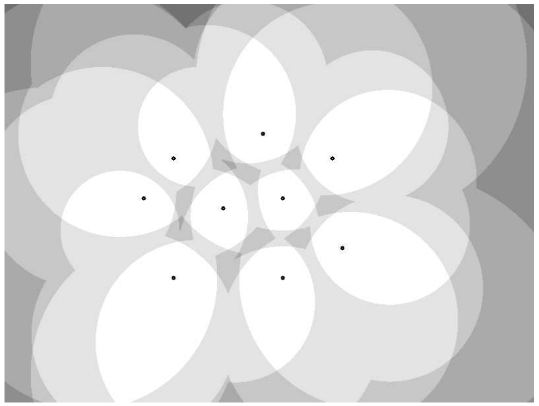
- Problem: storage and construction cost for N d-dimensional points is $\Theta(N^{d/2})$
- Impractical unless resort to some high-dimensional approximation of a Voronoi diagram (e.g., OS-tree) which results in approximate nearest neighbors
- Exponential factor corresponding to the dimension d of the underlying space in the complexity bounds when using approximations of Voronoi diagrams (e.g., (t,ϵ)-AVD) is shifted to be in terms of the error threshold ϵ rather than in terms of the number of objects N in the underlying space
  1. $(1,\epsilon)$-AVD: $O(N/\epsilon^{d-1})$ space and $O(\log(N/\epsilon^{d-1}))$ time for nearest neighbor query
  2. $(1/\epsilon^{(d-1)2},\epsilon)$-AVD: $O(N)$ space and $O(t+\log N)$ time for nearest neighbor query

Approximate Voronoi Diagrams (AVD)

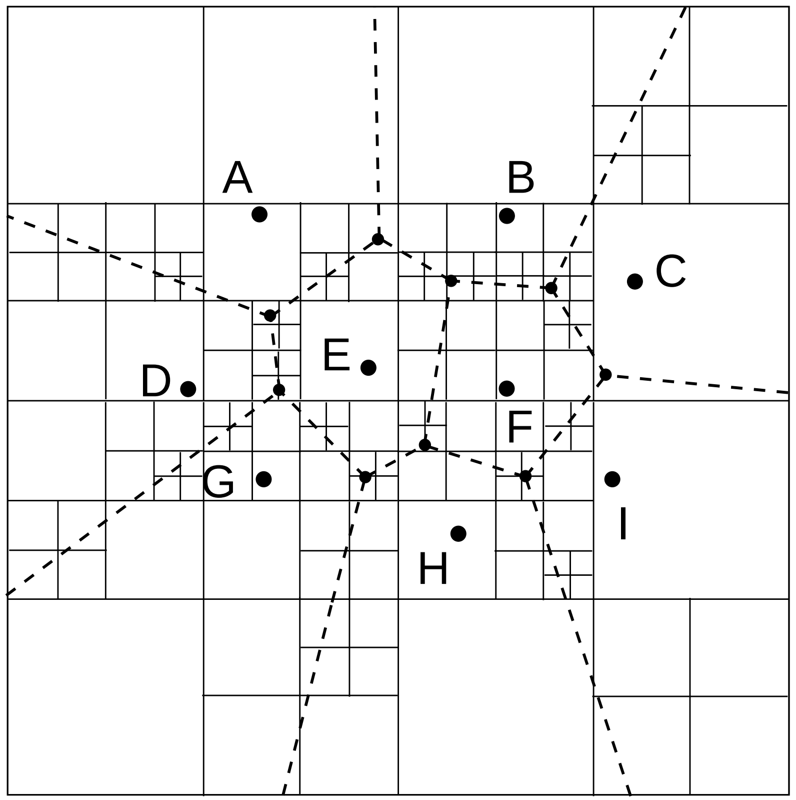
- Example partitions of space induced by ϵ neighbor sets
- Darkness of shading indicates cardinality of nearest neighbor sets with white corresponding to 1

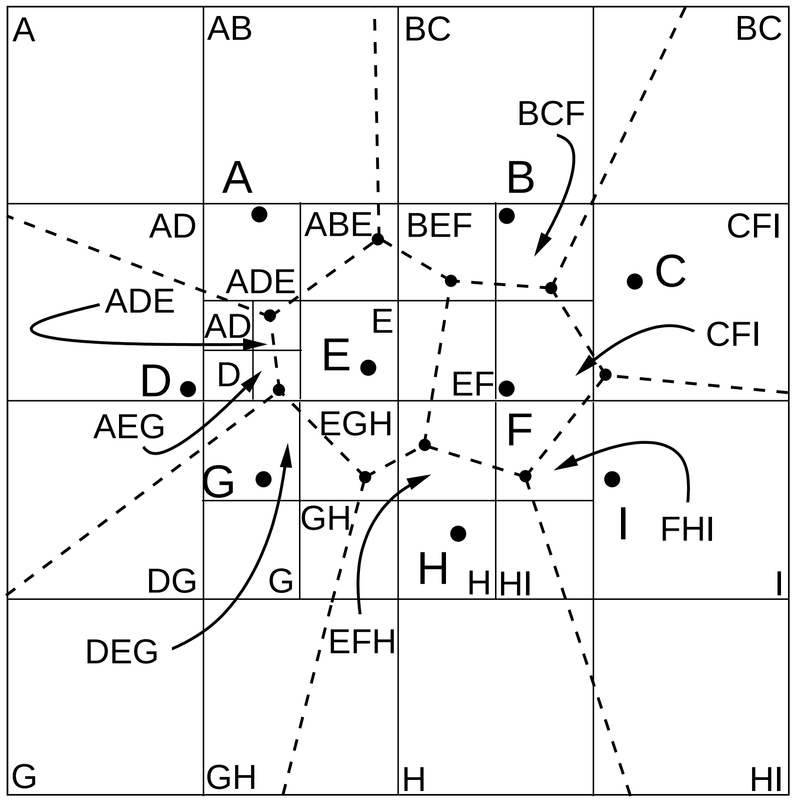

$(\epsilon = 0.10)$


$(\epsilon = 0.30)$


$(\epsilon = 0.50)$

Approximate Voronoi Diagrams (AVD) Representations





- Partition underlying domain so that for $\epsilon \geq 0$, every block b is associated with some element r_b in S such that r_b is an ϵ-nearest neighbor for all of the points in b (e.g., AVD or (1,0.25)-AVD)
- Allow up to $t \geq 1$ elements $r_{ib} (1 \leq i \leq t)$ of S to be associated with each block b for a given ϵ, where each point in b has one of the r_{ib} as its ϵ-nearest neighbor (e.g., (3,0)-AVD)

Problem: Curse of Dimensionality

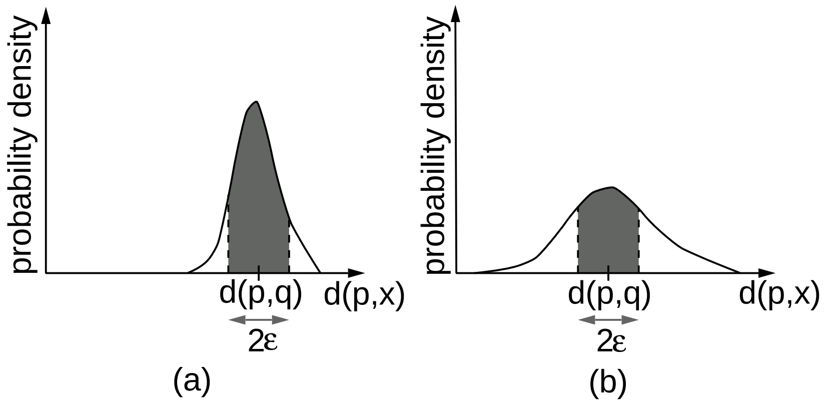
- Number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with the number of variables (i.e., dimensions) that comprise it (Bellman)
- For similarity searching, curse means that the number of points in the data set that need to be examined in deriving the estimate ($\equiv$ nearest neighbor) grows exponentially with the underlying dimension
- Effect on nearest neighbor finding is that the process may not be meaningful in high dimensions
- When ratio of variance of distances and expected distances, between two random points p and q drawn from the data and query distributions, converges to zero as dimension d gets very large (Beyer et al.)

$$\lim_{d\to\infty} \frac{\text{Variance}[dist(p,q)]}{\text{Expected}[dist(p,q)]} = 0$$

  1. distance to the nearest neighbor and distance to the farthest neighbor tend to converge as the dimension increases
  2. implies that nearest neighbor searching is inefficient as difficult to differentiate nearest neighbor from other objects
  3. assumes uniformly distributed data
- Partly alleviated by fact that real-world data is rarely uniformly-distributed

Alternative View of Curse of Dimensionality

- Probability density function (analogous to histogram) of the distances of the objects is more concentrated and has a larger mean value
- Implies similarity search algorithms need to do more work
- Worst case when $d(x,x) = 0$ and $d(x,y) = 1$ for all $y \neq x$
- Implies must compare every object with every other object
  1. can't always use triangle inequality to prune objects from consideration
  2. triangle inequality (i.e., $d(q,p) \leq d(p,x) + d(q,x)$) implies that any x such that $|d(q,p) - d(p,x)| > \epsilon$ cannot be at a distance of ϵ or less from q as $d(q,x) \geq d(q,p) - d(p,x) > \epsilon$
  3. when ϵ is small while probability density function is large at $d(p,q)$, then probability of eliminating an object from consideration via use of triangle inequality is remaining area under curve which is small (see left) in contrast to case when distances are more uniform (see right)



Other Problems

- Point and range queries are less complex than nearest neighbor queries
  1. easy to do with multi-dimensional index as just need comparison tests
  2. nearest neighbor require computation of distance
     - Euclidean distance needs d multiplications and $d-1$ additions
- Often we don't know features describing the objects and thus need aid of domain experts to identify them

Solutions Based on Indexing

1. Map objects to a low-dimensional vector space which is then indexed using one of a number of different data structures such as k-d trees, R-trees, quadtrees, etc.
   - use dimensionality reduction: representative points, SVD, DFT, etc.
2. Directly index the objects based on distances from a subset of the objects making use of data structures such as the vp-tree, M-tree, etc.
   - useful when only have a distance function indicating similarity (or dis-similarity) between all pairs of N objects
   - if change distance metric, then need to rebuild index — not so for multidimensional index
3. If only have distance information available, then embed the data objects in a vector space so that the distances of the embedded objects as measured by the distance metric in the embedding space approximate the actual distances
   - commonly known embedding methods include multidimensional scaling (MDS), Lipschitz embeddings, FastMap, etc.
   - once a satisfactory embedding has been obtained, the actual search is facilitated by making use of conventional indexing methods, perhaps coupled with dimensionality reduction

Outline

Part 1: Indexing Low and High Dimensional Spaces

1. Quadtree variants
2. k-d tree
3. R-tree
4. Bounding sphere methods
5. Hybrid tree
6. Avoiding overlapping all of the leaf blocks
7. Pyramid technique
8. Methods based on a sequential scan

Simple Non-Hierarchical Data Structures

Sequential list

Name	X	Y
Chicago	35	42
Mobile	52	10
Toronto	62	77
Buffalo	82	65
Denver	5	45
Omaha	27	35
Atlanta	85	15
Miami	90	5

Inverted List

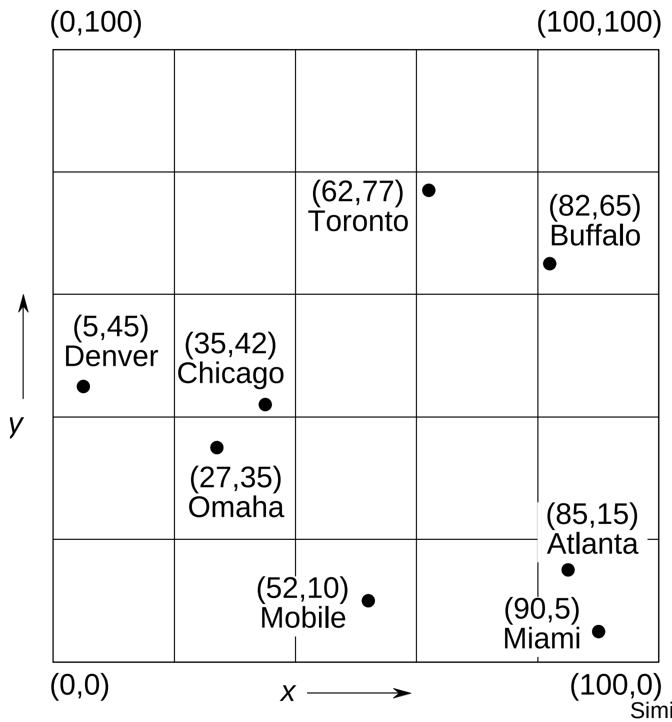
X	Y
Denver	Miami
Omaha	Mobile
Chicago	Atlanta
Mobile	Omaha
Toronto	Chicago
Buffalo	Denver
Atlanta	Buffalo
Miami	Toronto

Inverted lists:

1. 2 sorted lists
2. data is pointers
3. enables pruning the search with respect to one key

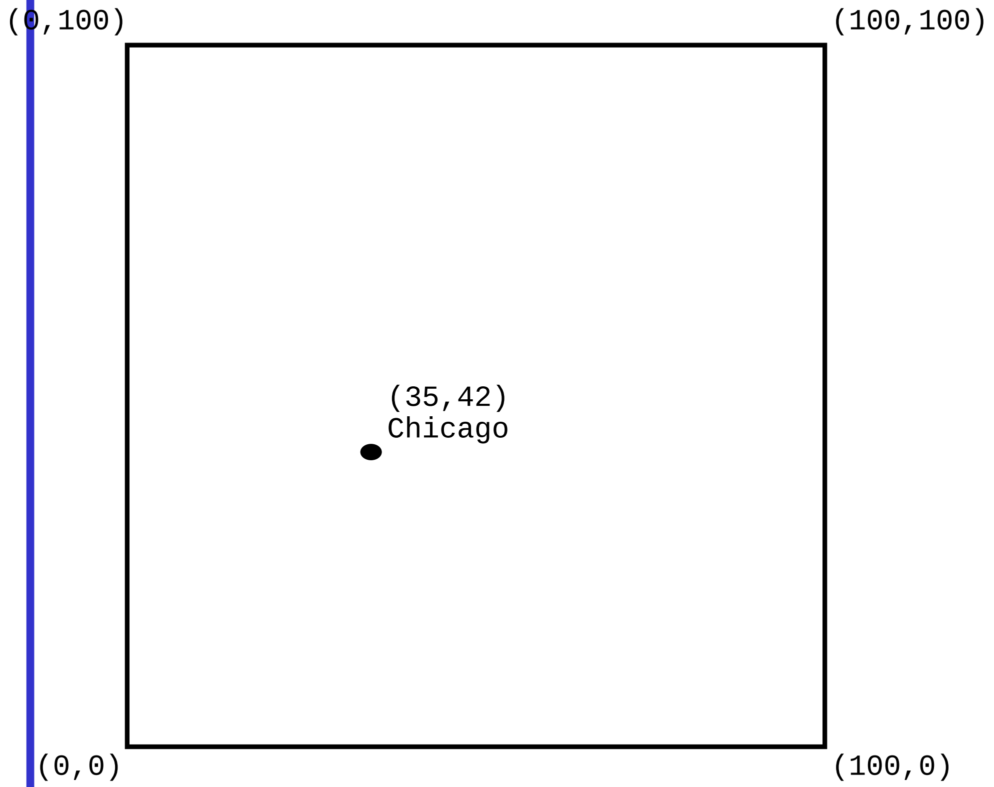
Grid Method

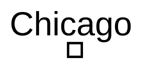
- Divide space into squares of width equal to the search region
- Each cell contains a list of all points within it
- Assume L_∞ distance metric (i.e., Chessboard)
- Assume C = uniform distribution of points per cell
- Average search time for k-dimensional space is $O(F \cdot 2^k)$
  - F = number of records found = C, since query region has the width of a cell
  - 2^k = number of cells examined



Point Quadtree (Finkel/Bentley)

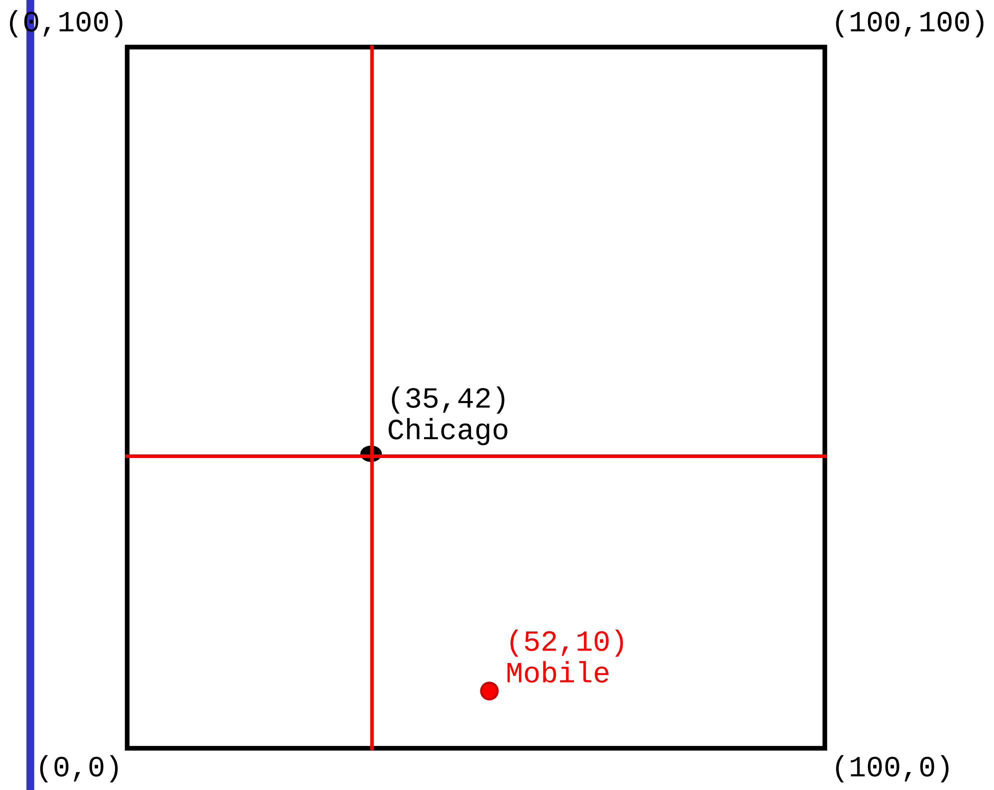
- Marriage between uniform grid and a binary search tree

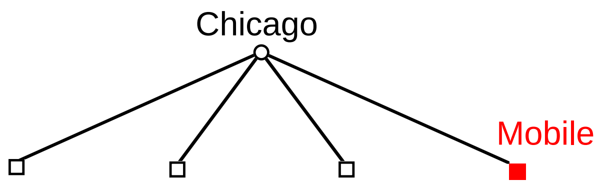




Point Quadtree (Finkel/Bentley)

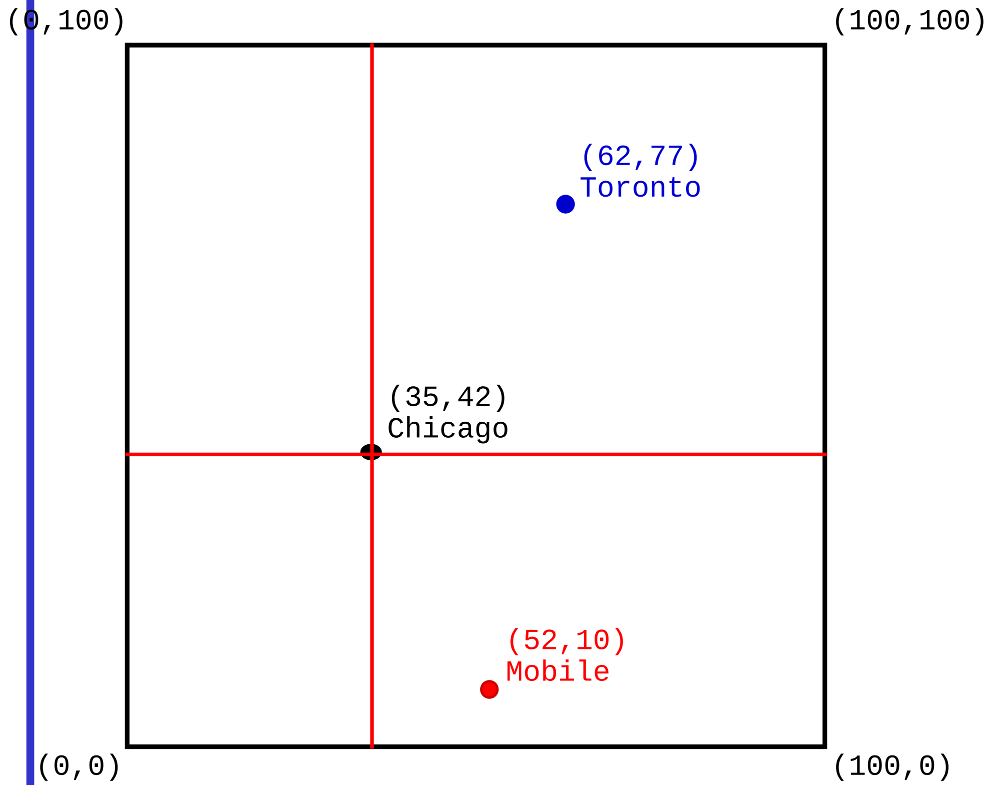
- Marriage between uniform grid and a binary search tree

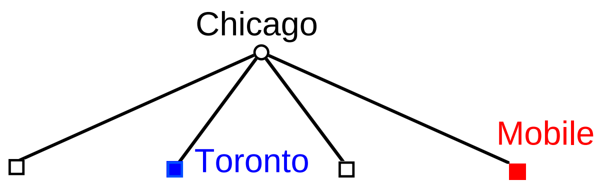




Point Quadtree (Finkel/Bentley)

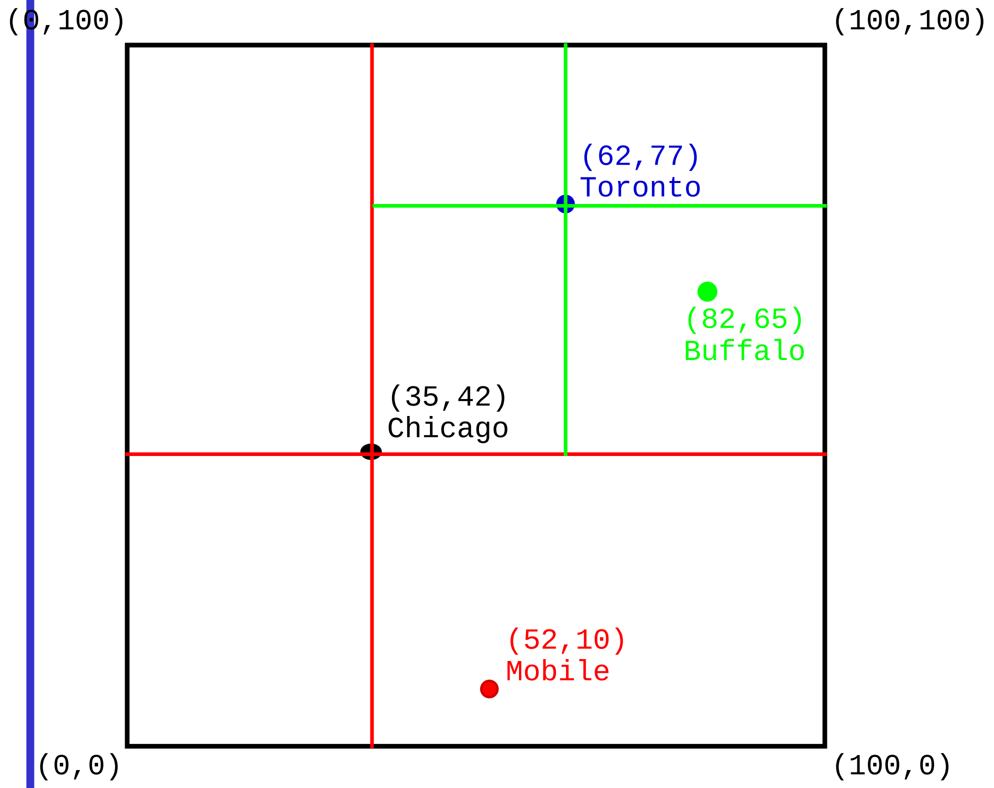
- Marriage between uniform grid and a binary search tree

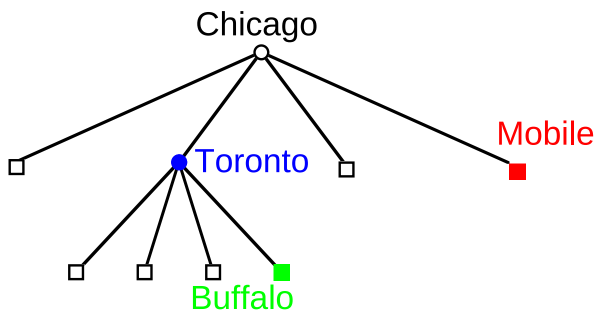




Point Quadtree (Finkel/Bentley)

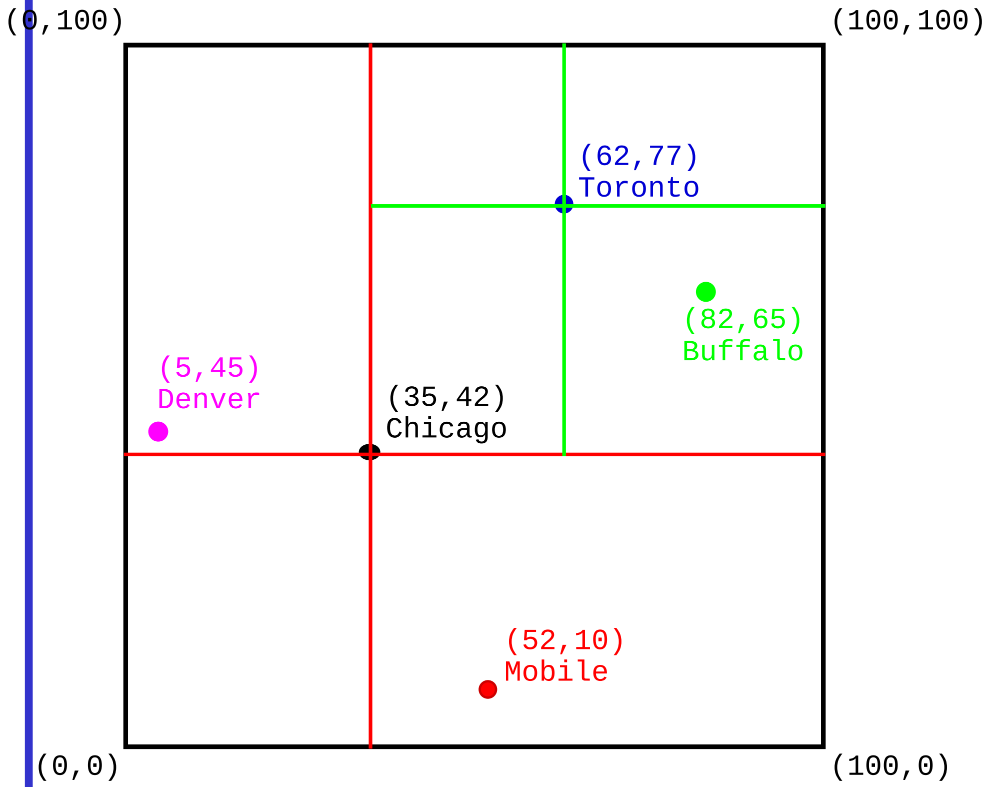
- Marriage between uniform grid and a binary search tree

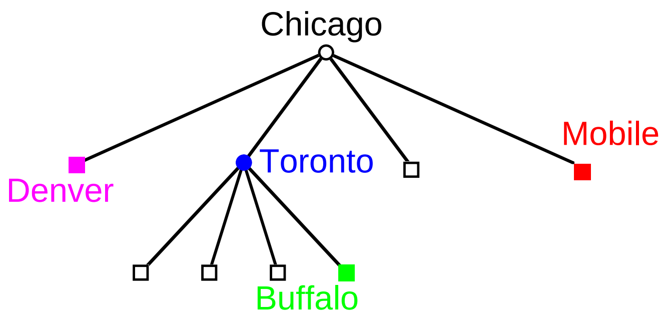




Point Quadtree (Finkel/Bentley)

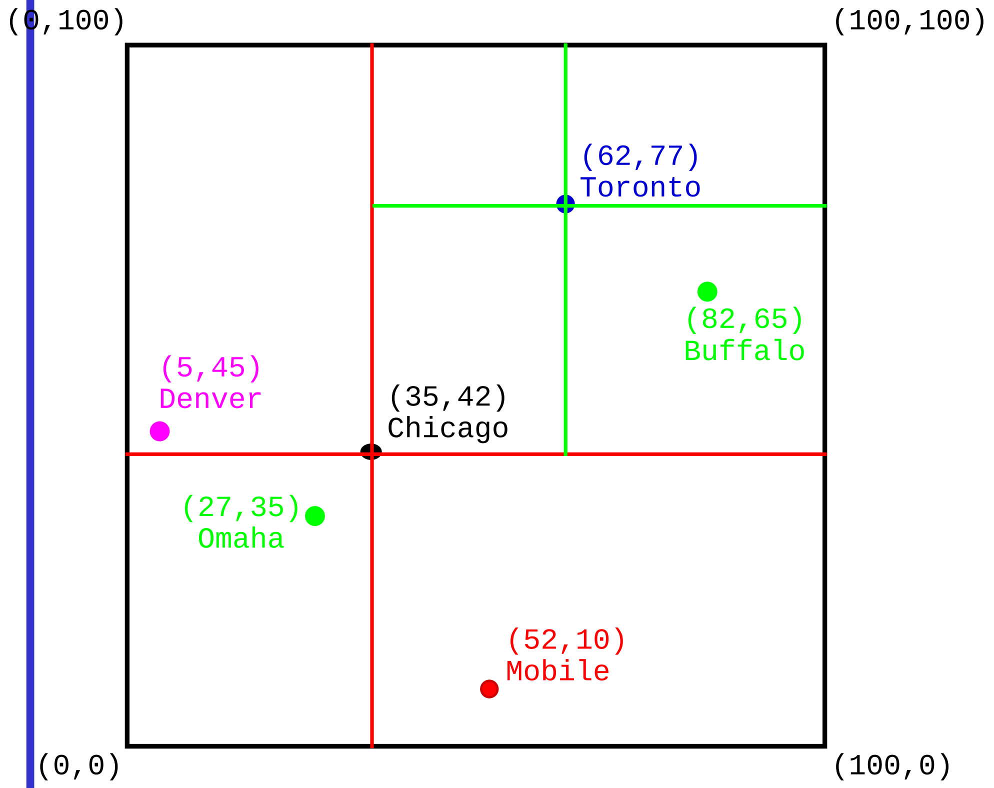
- Marriage between uniform grid and a binary search tree

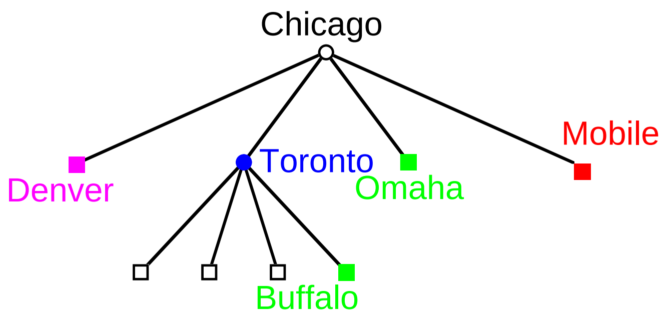




Point Quadtree (Finkel/Bentley)

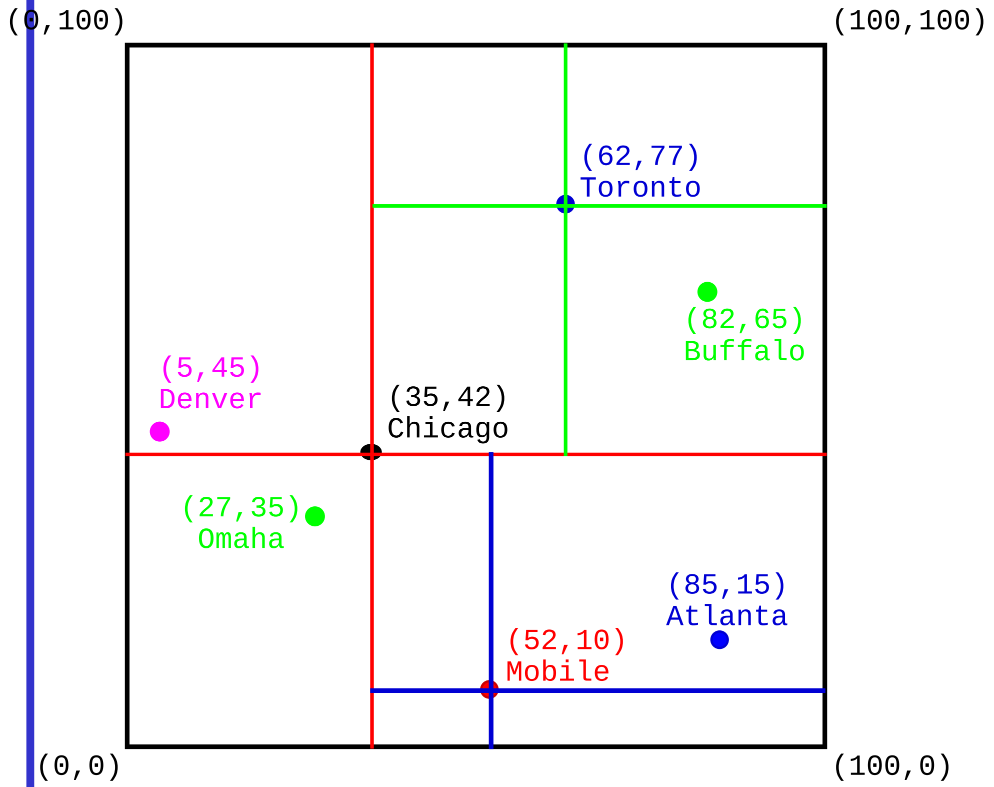
- Marriage between uniform grid and a binary search tree

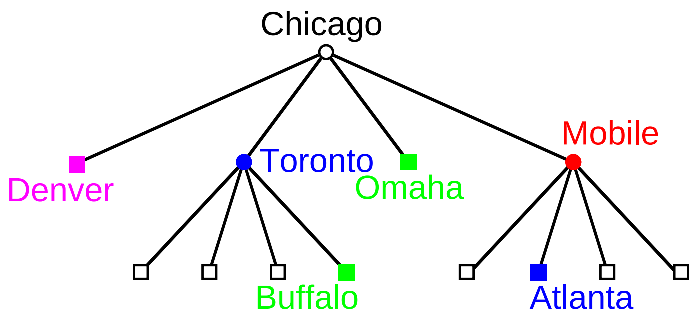




Point Quadtree (Finkel/Bentley)

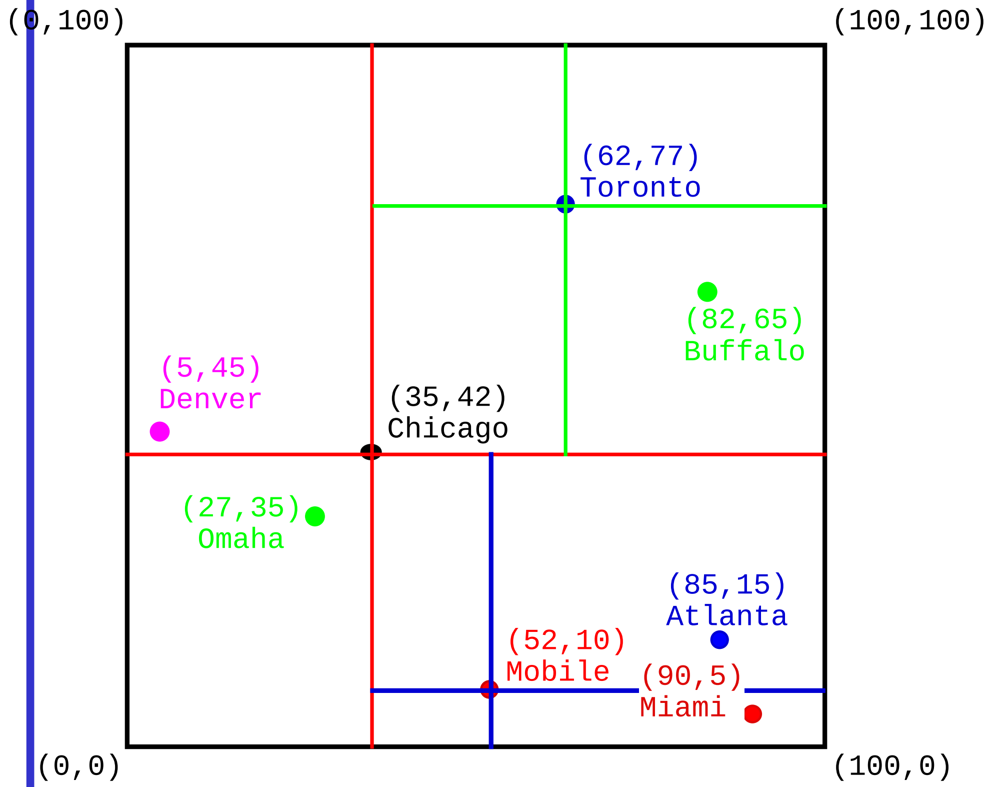
- Marriage between uniform grid and a binary search tree

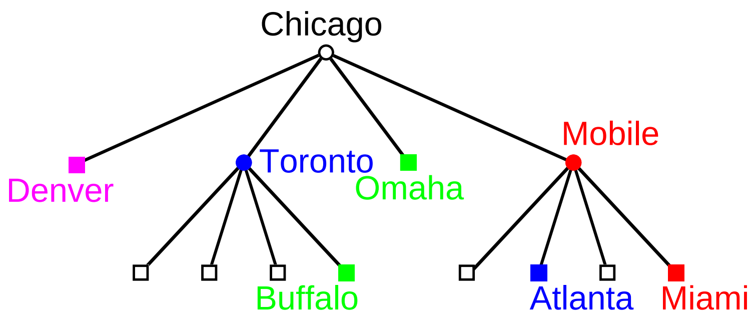




Point Quadtree (Finkel/Bentley)

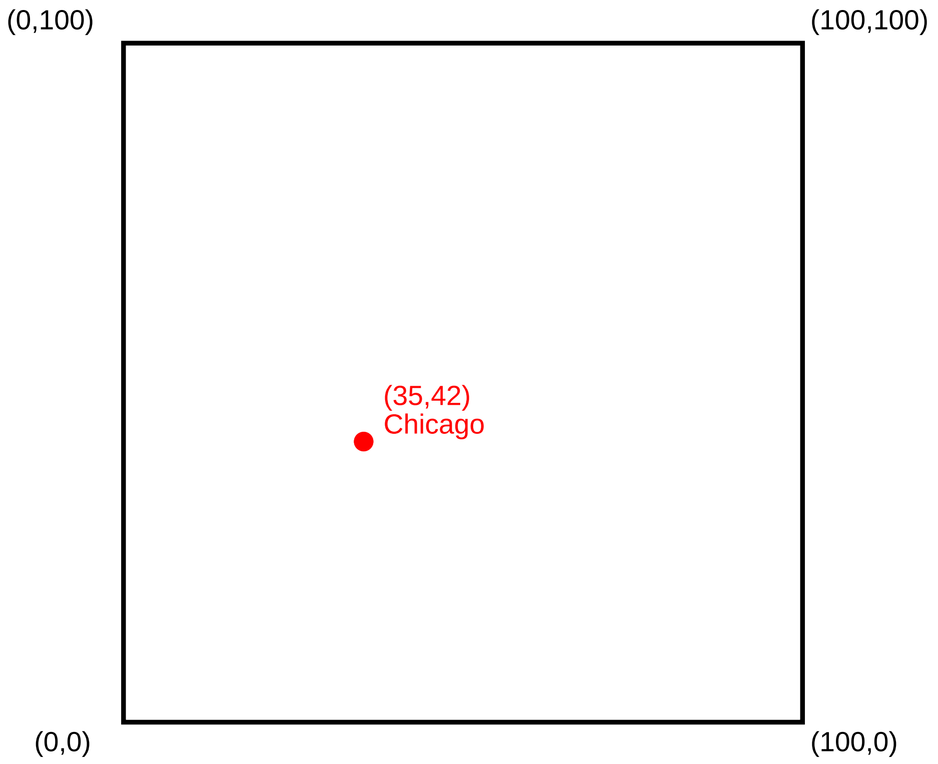
- Marriage between uniform grid and a binary search tree





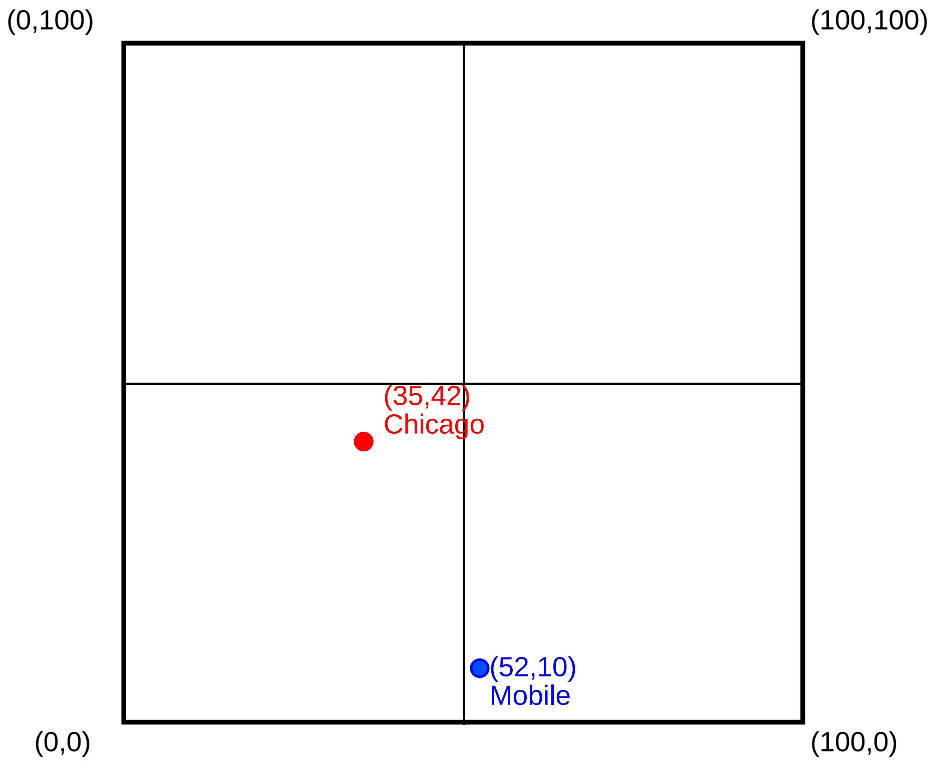
PR Quadtree

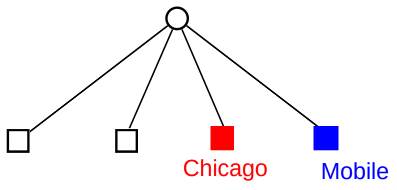
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points



PR Quadtree

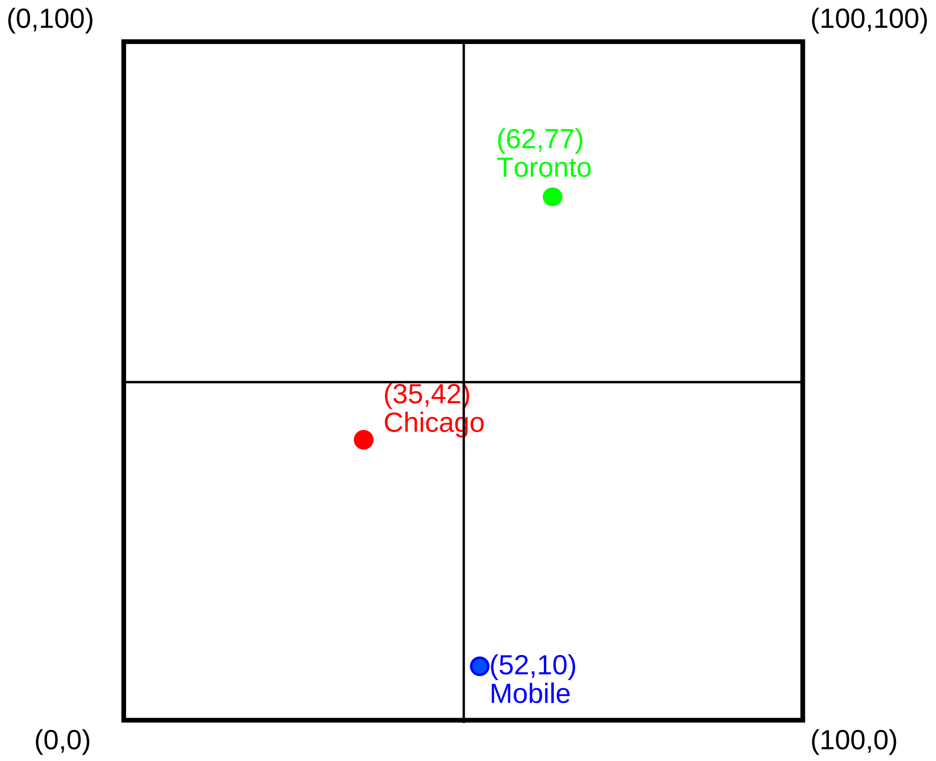
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

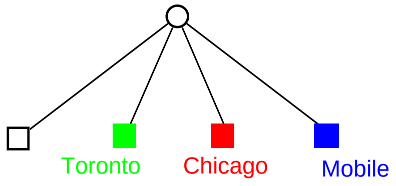




PR Quadtree

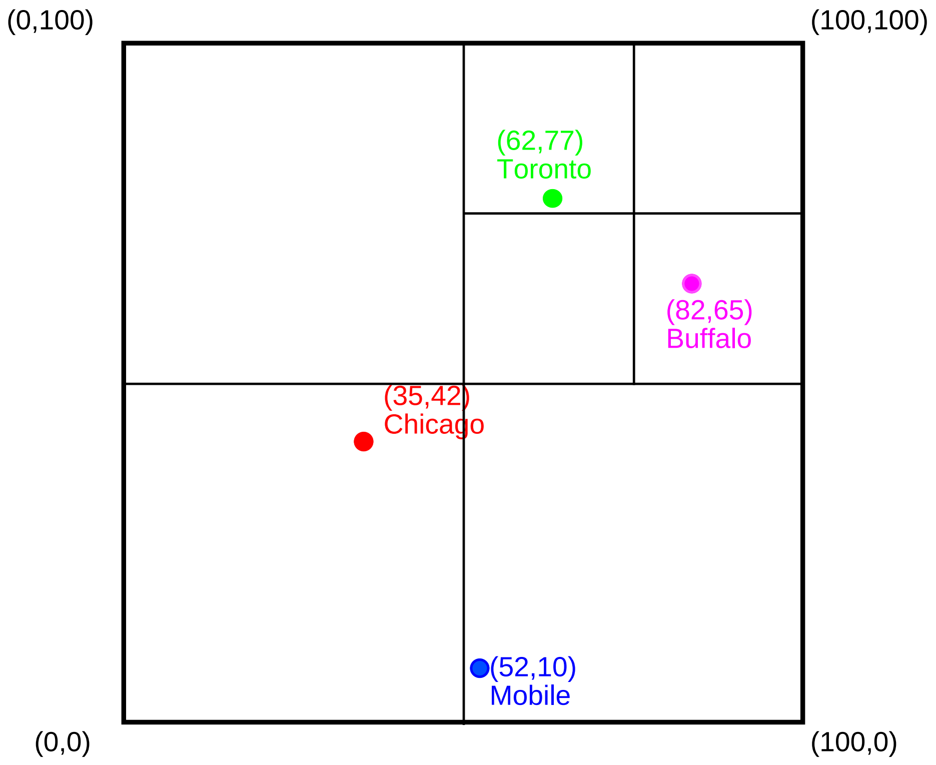
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

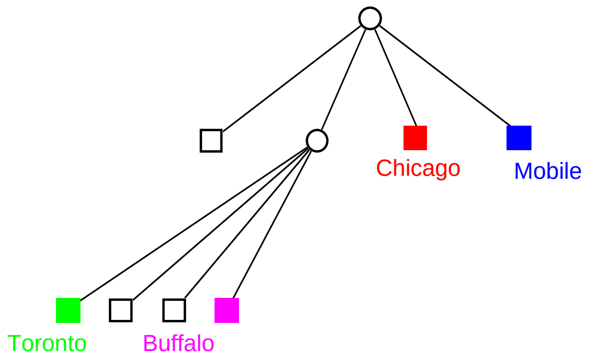




PR Quadtree

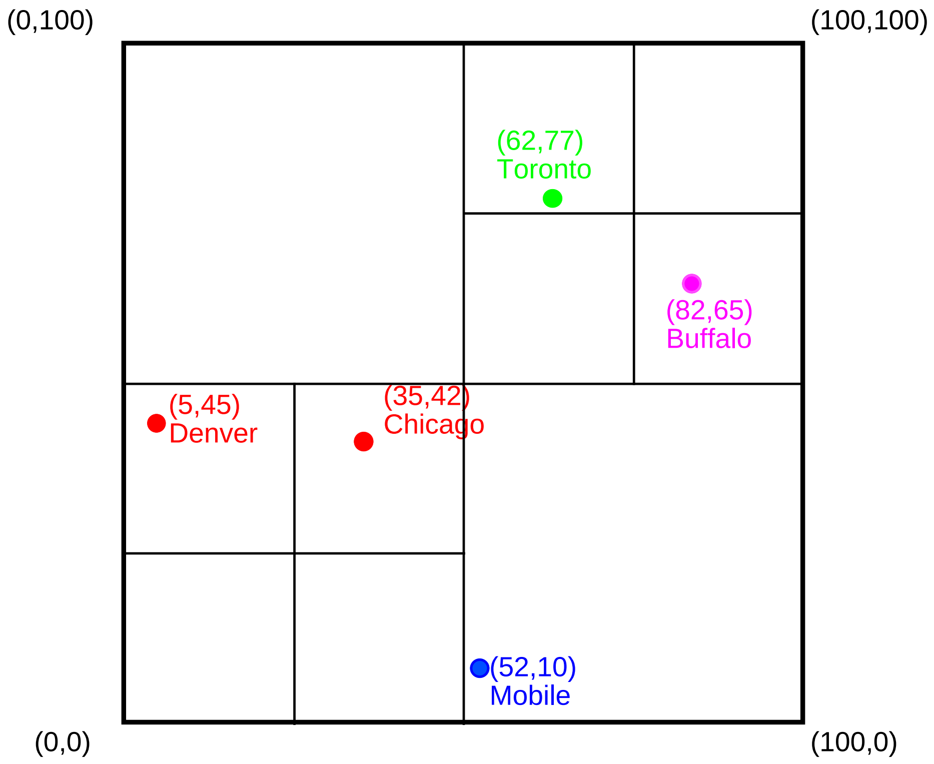
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

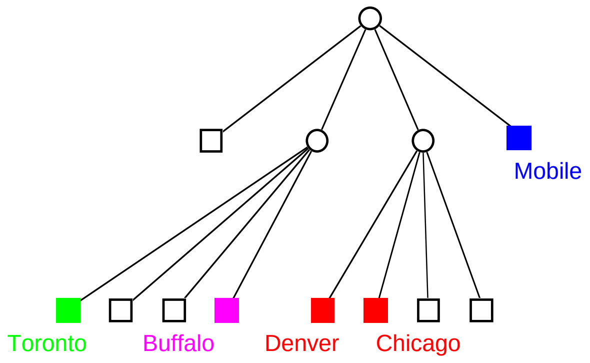




PR Quadtree

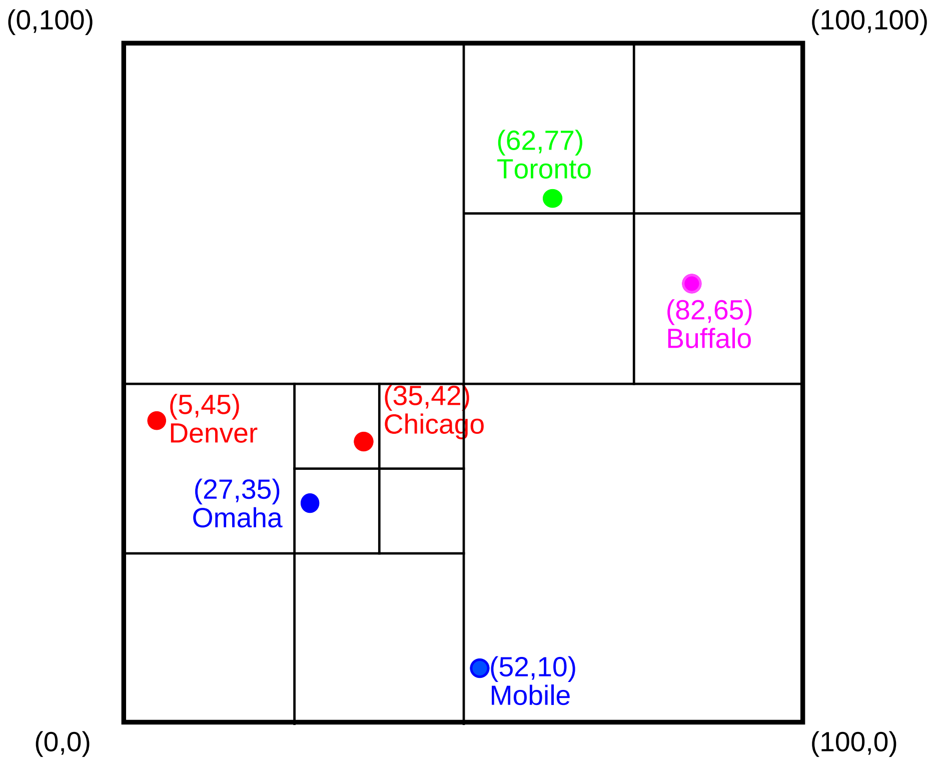
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

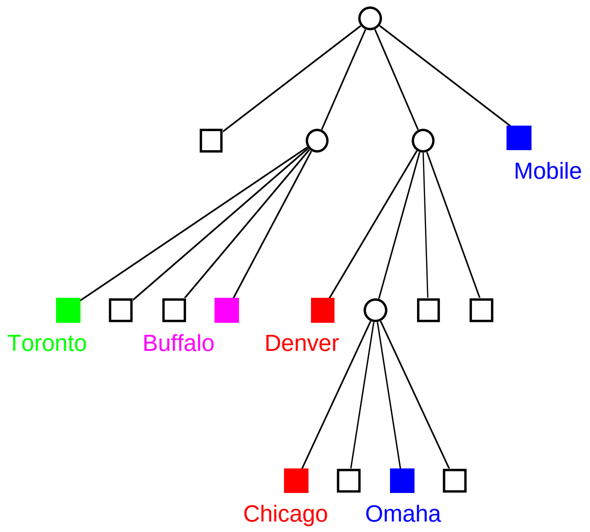




PR Quadtree

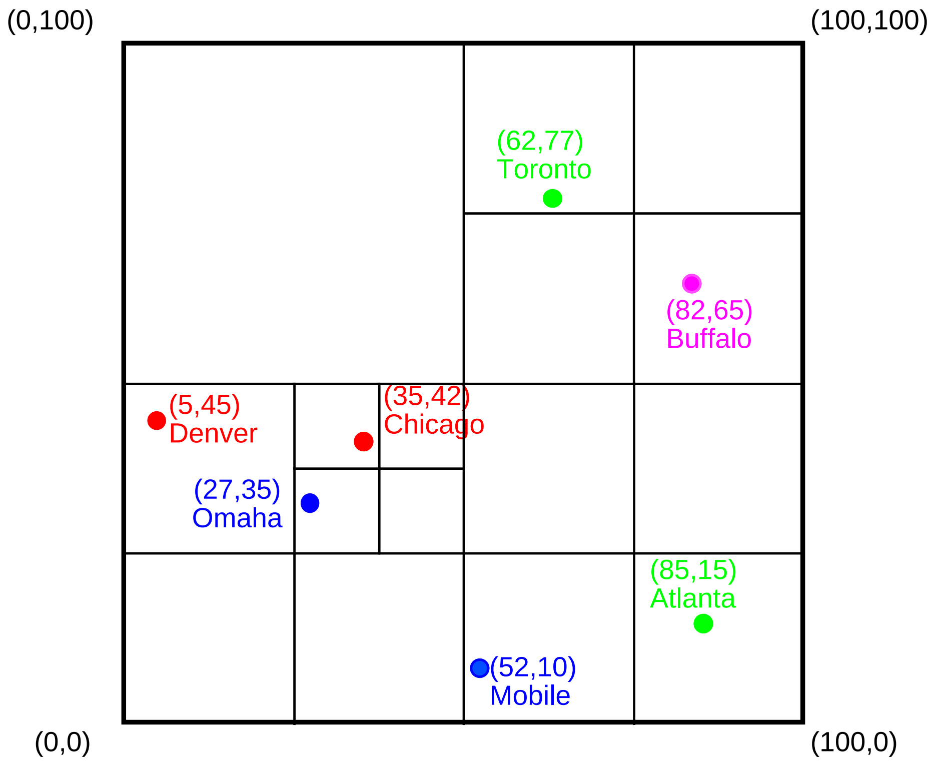
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

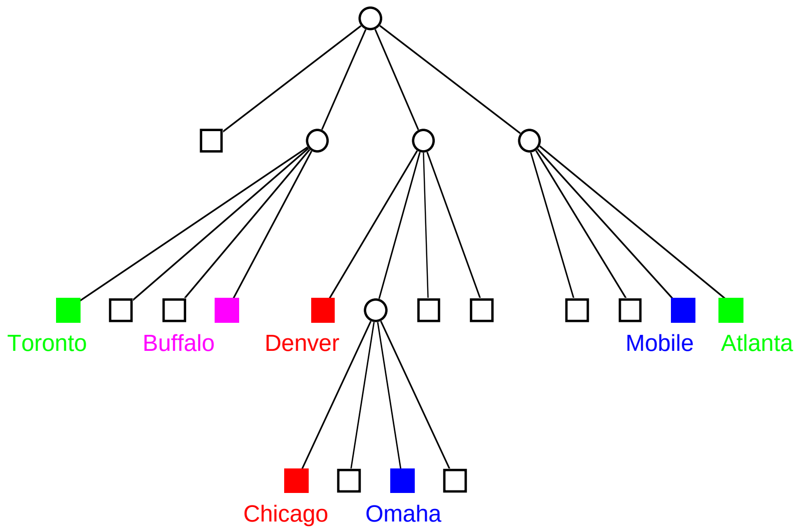




PR Quadtree

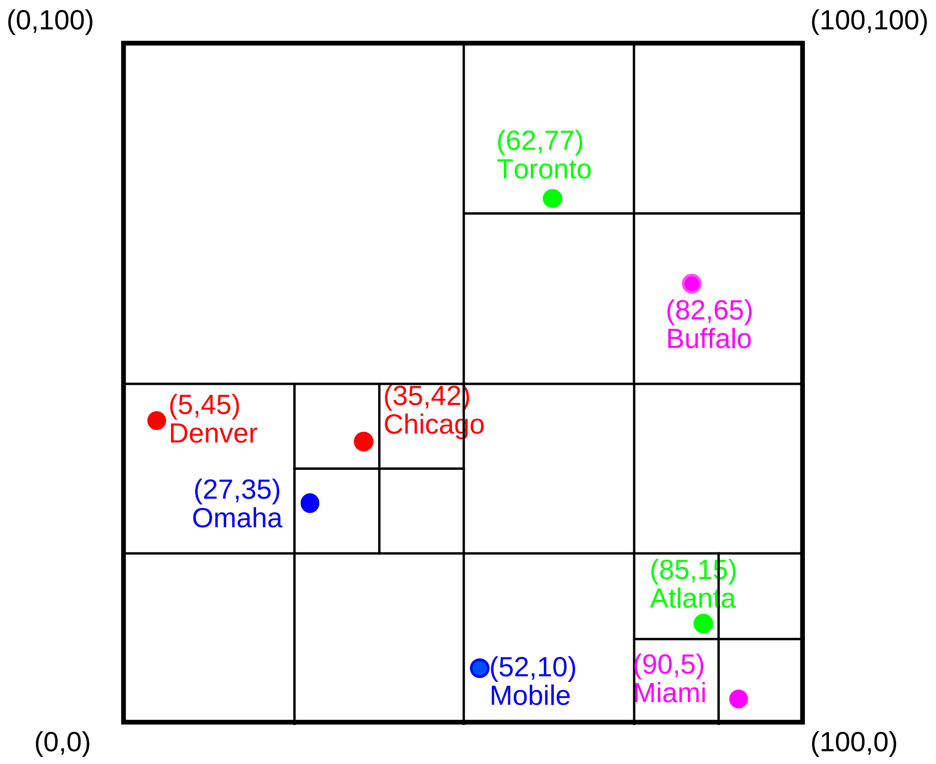
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

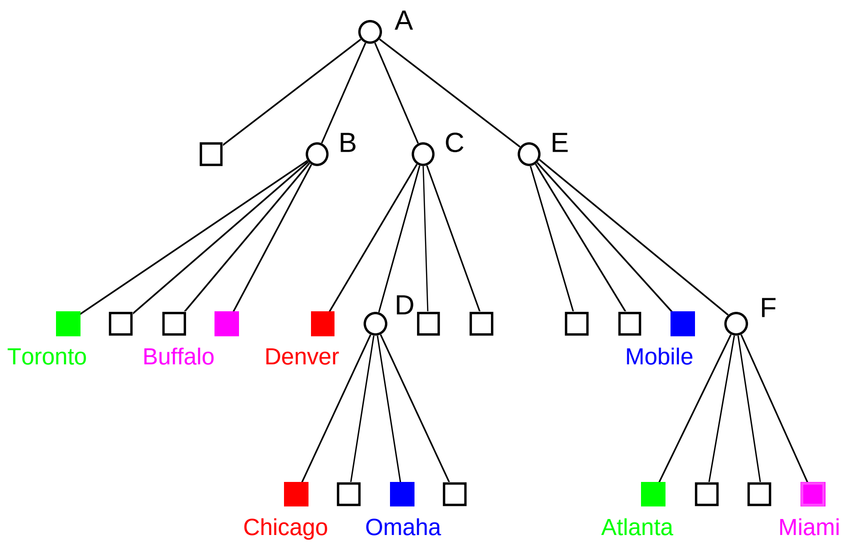




PR Quadtree

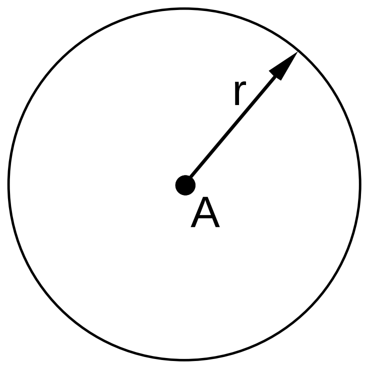
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points





Region Search

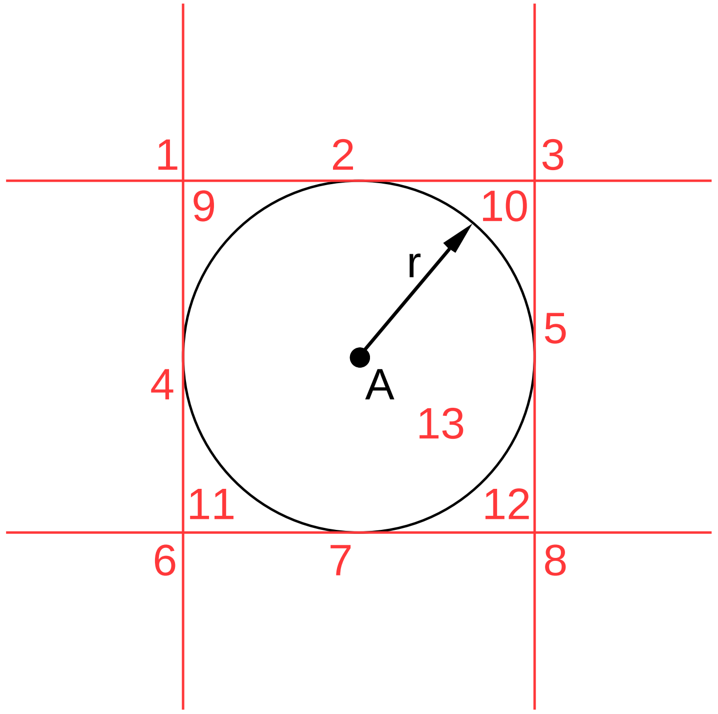
- Ex: Find all points within radius r of point A



- Use of quadtree results in pruning the search space

Region Search

- Ex: Find all points within radius r of point A

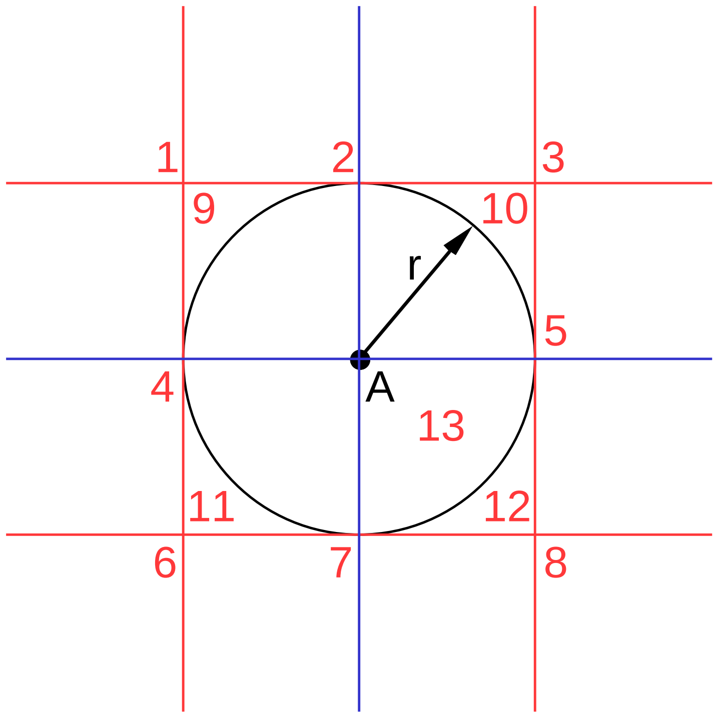


- Use of quadtree results in pruning the search space
- If a quadrant subdivision point p lies in a region l, then search the quadrants of p specified by l

1.	SE	5.	SW, NW	9.	All but NW	13.	All
2.	SE, SW	6.	NE	10.	All but NE		
3.	SW	7.	NE, NW	11.	All but SW		
4.	SE, NE	8.	NW	12.	All but SE		

Region Search

- Ex: Find all points within radius r of point A

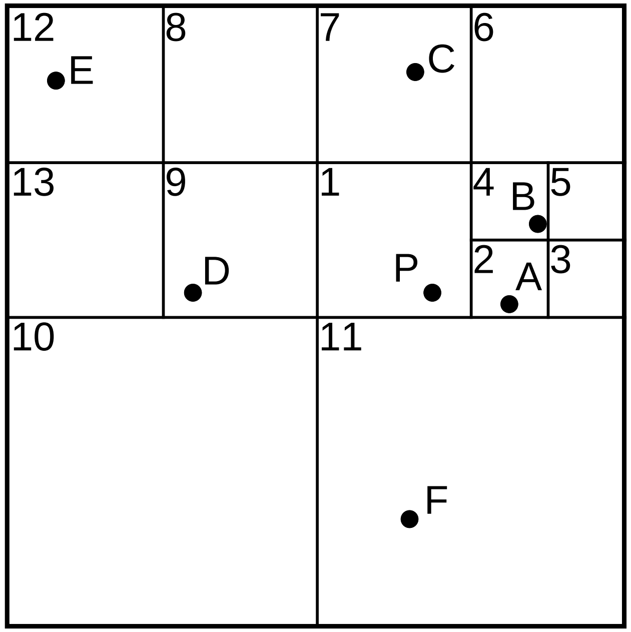


- Use of quadtree results in pruning the search space
- If a quadrant subdivision point p lies in a region l, then search the quadrants of p specified by l

1.	SE	5.	SW, NW	9.	All but NW	13.	All
2.	SE, SW	6.	NE	10.	All but NE		
3.	SW	7.	NE, NW	11.	All but SW		
4.	SE, NE	8.	NW	12.	All but SE		

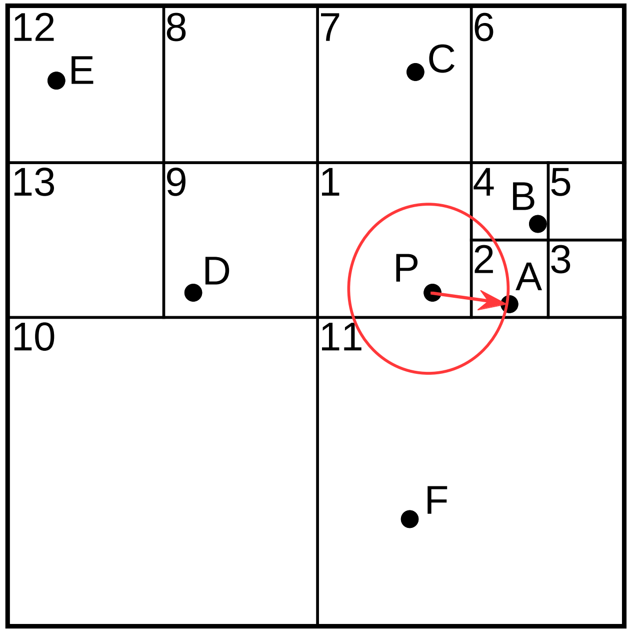
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:



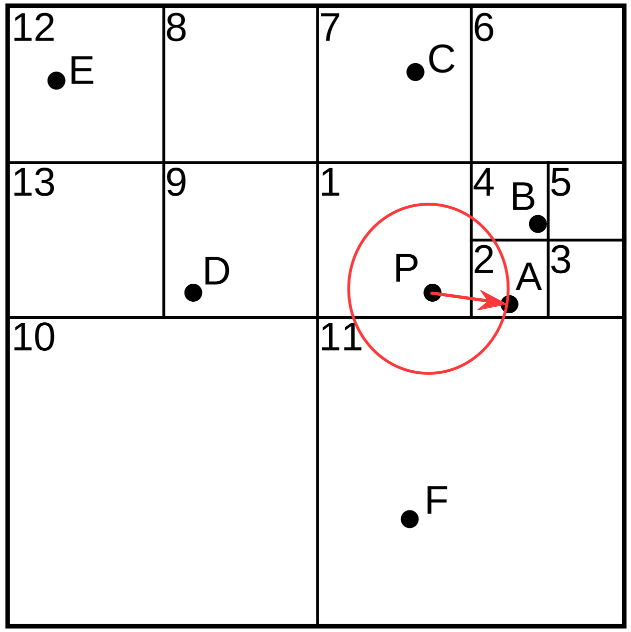
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
  1. start at block 2 and compute distance to P from A



Finding Nearest Object

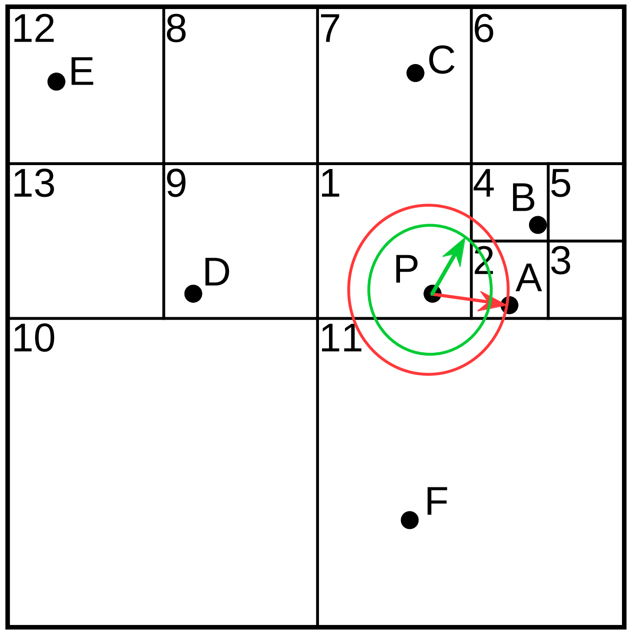
- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:



1. start at block 2 and compute distance to P from A
2. ignore block 3, even if nonempty, as A is closer to P than any point in 3

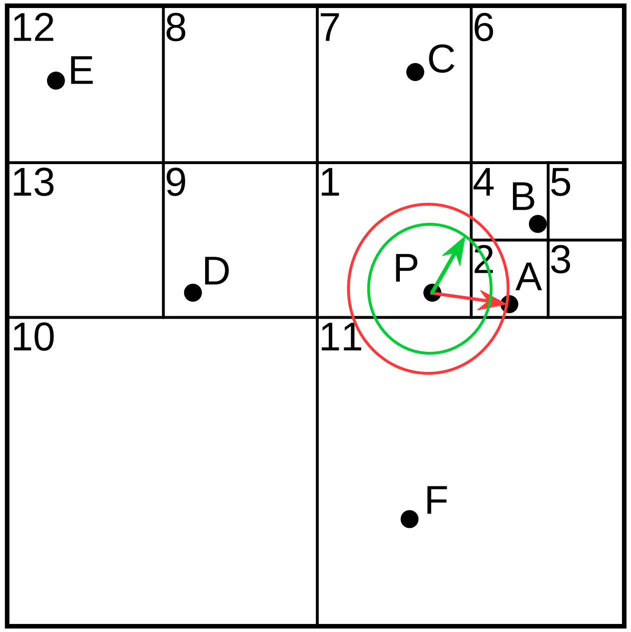
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
  1. start at block 2 and compute distance to P from A
  2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
  3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A



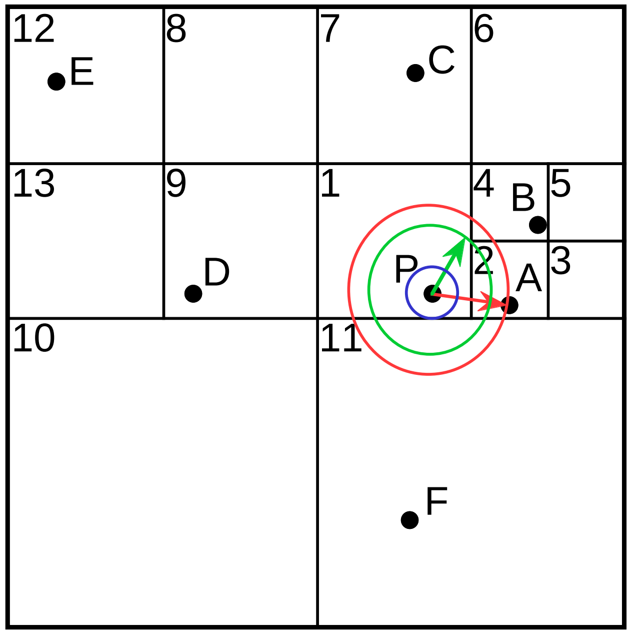
Finding Nearest Object

- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:
  1. start at block 2 and compute distance to P from A
  2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
  3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A
  4. ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A



Finding Nearest Object

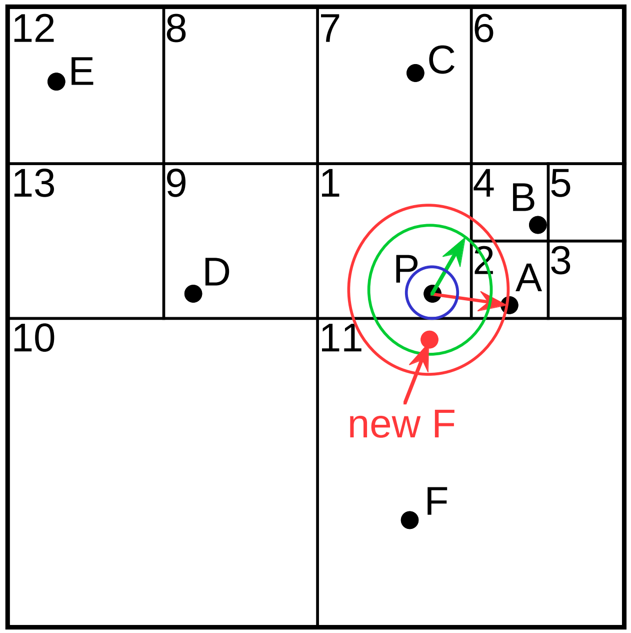
- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space
- Algorithm:



1. start at block 2 and compute distance to P from A
2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A
4. ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A
5. examine block 11 as the distance from P to the S border of 1 is shorter than distance from P to A; but, reject F as it is further from P than A

Finding Nearest Object

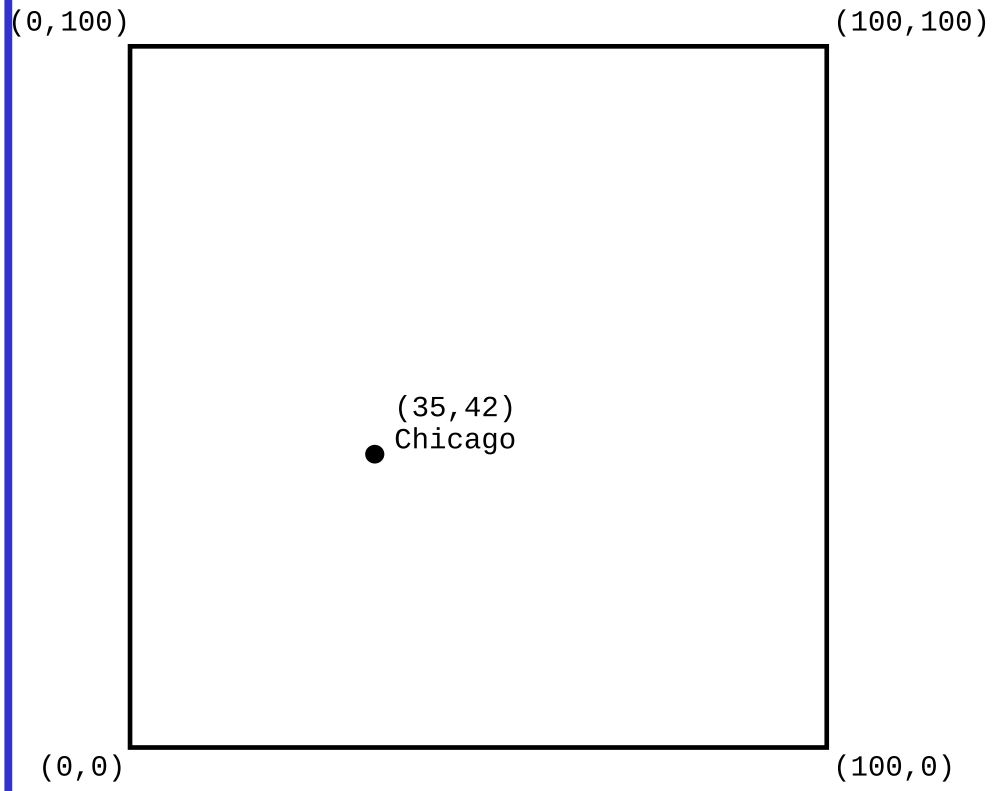
- Ex: find the nearest object to P
- Assume PR quadtree for points (i.e., at most one point per block)
- Search neighbors of block 1 in counterclockwise order
- Points are sorted with respect to the space they occupy which enables pruning the search space

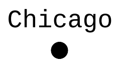


- Algorithm:
    1. start at block 2 and compute distance to P from A
    2. ignore block 3, even if nonempty, as A is closer to P than any point in 3
    3. examine block 4 as distance to SW corner is shorter than the distance from P to A; however, reject B as it is further from P than A
    4. ignore blocks 6, 7, 8, 9, and 10 as the minimum distance to them from P is greater than the distance from P to A
    5. examine block 11 as the distance from P to the S border of 1 is shorter than distance from P to A; but, reject F as it is further from P than A
- If F was moved, a better order would have started with block 11, the southern neighbor of 1, as it is closest to the new F

k-d tree (Bentley)

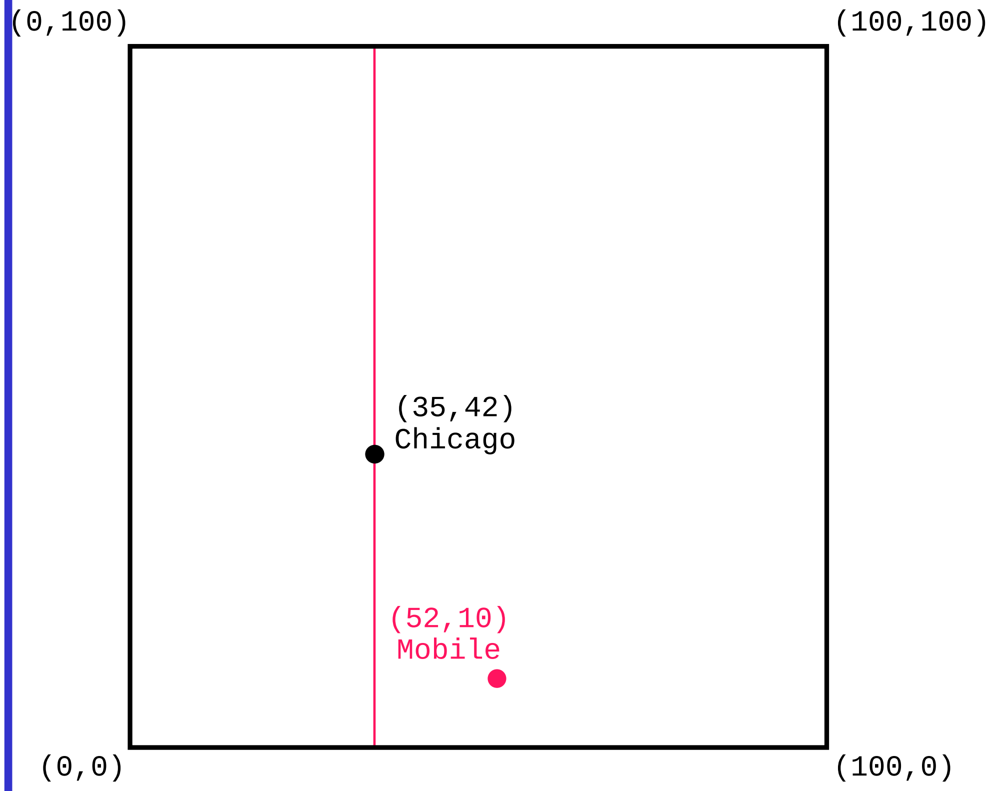
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

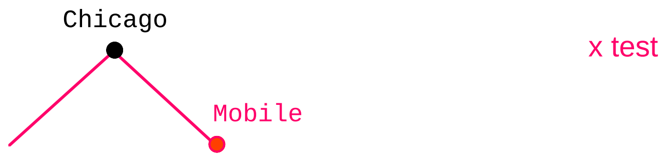




k-d tree (Bentley)

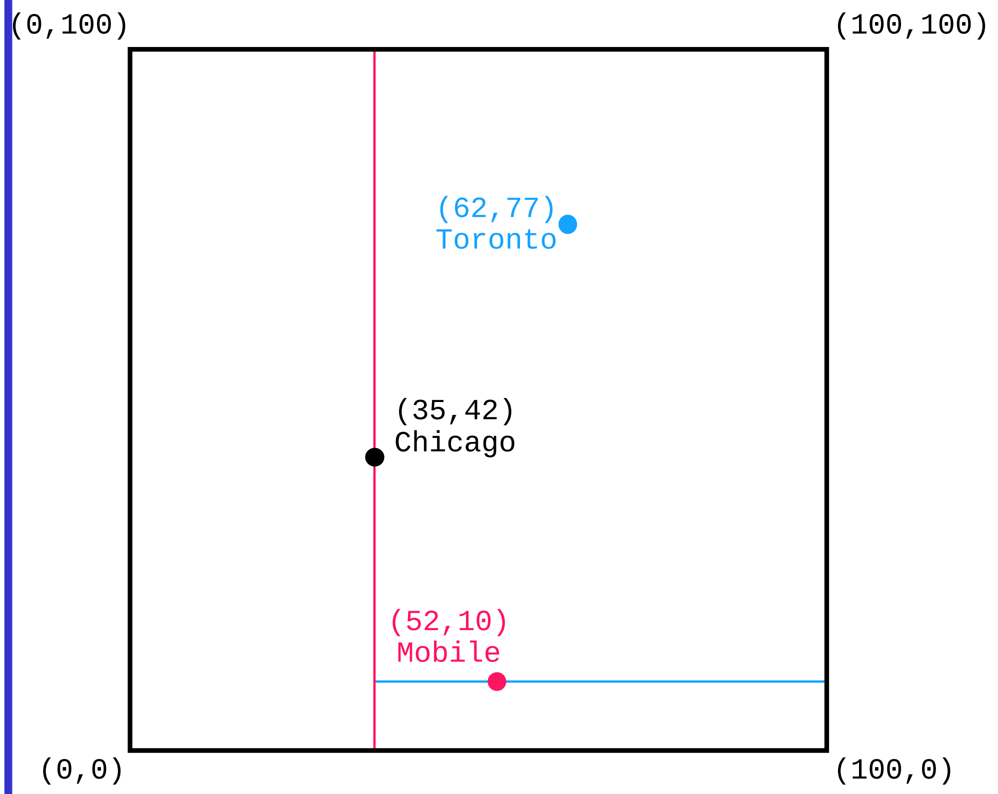
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

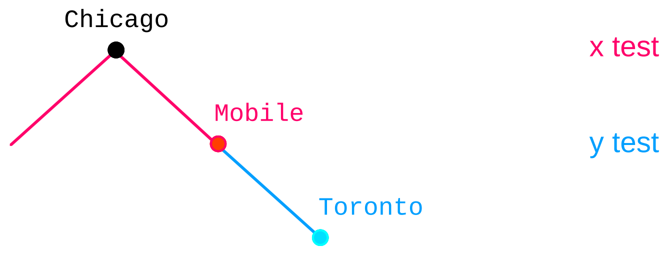




k-d tree (Bentley)

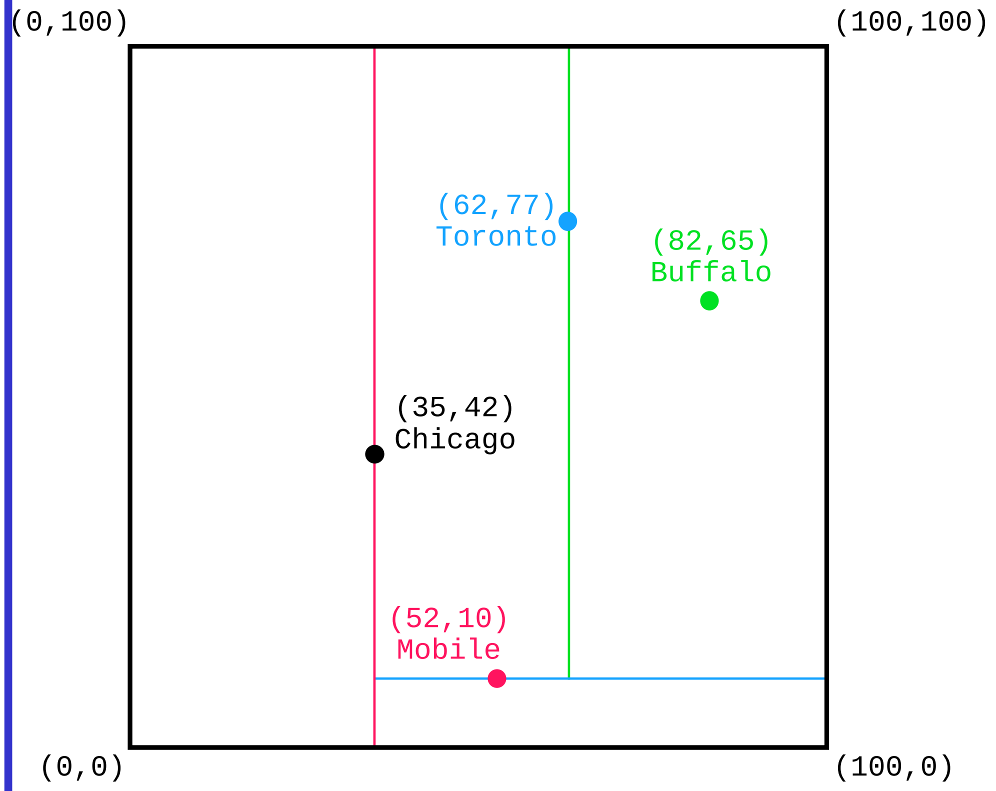
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

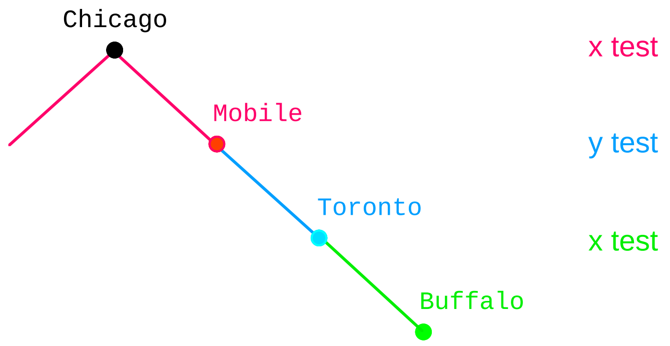




k-d tree (Bentley)

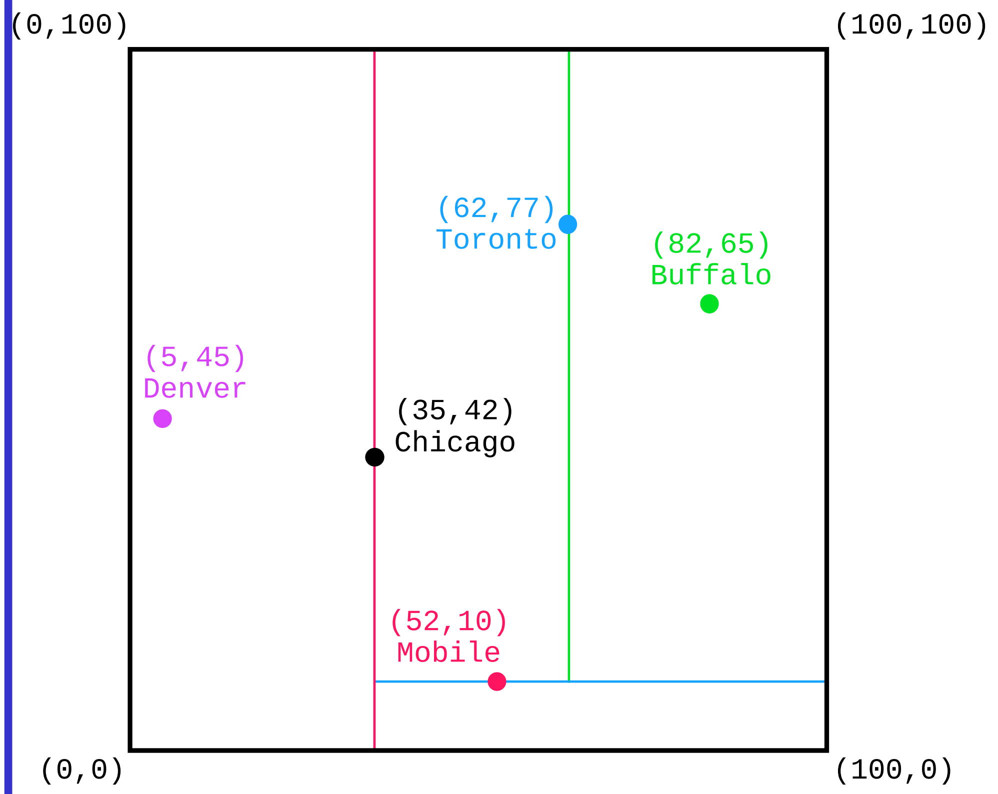
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

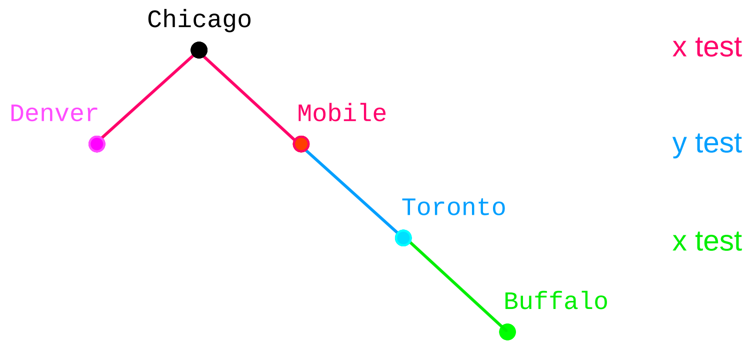




k-d tree (Bentley)

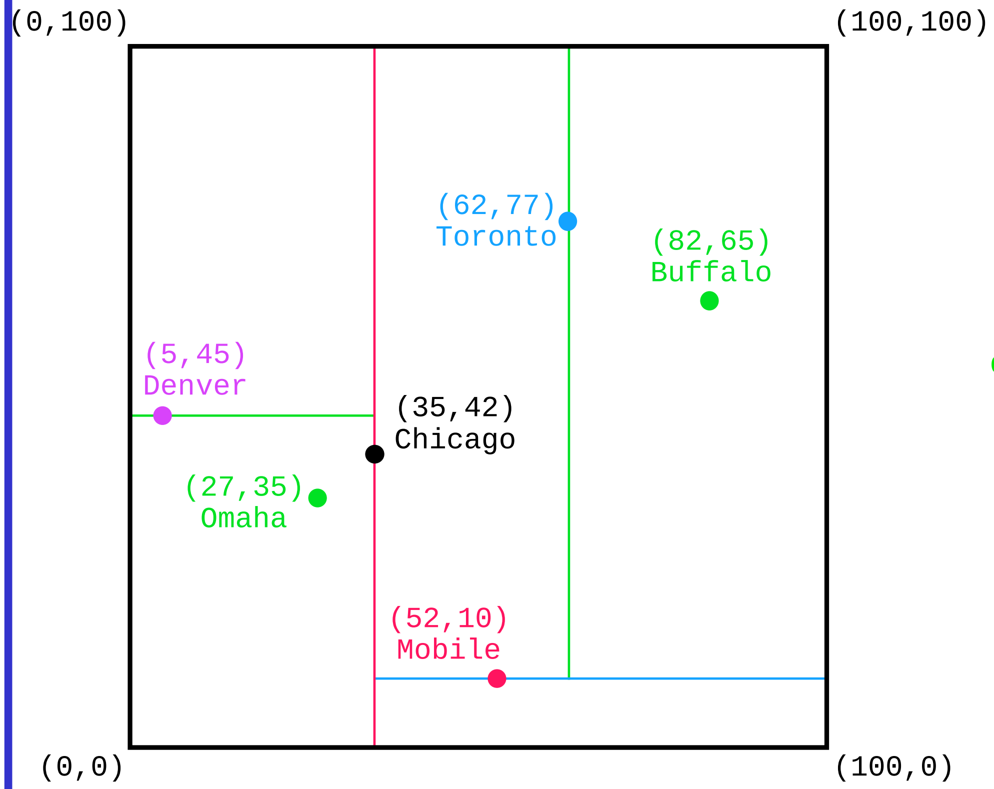
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

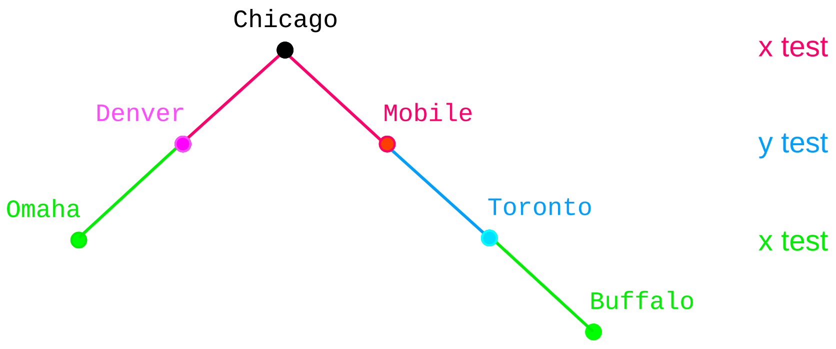




k-d tree (Bentley)

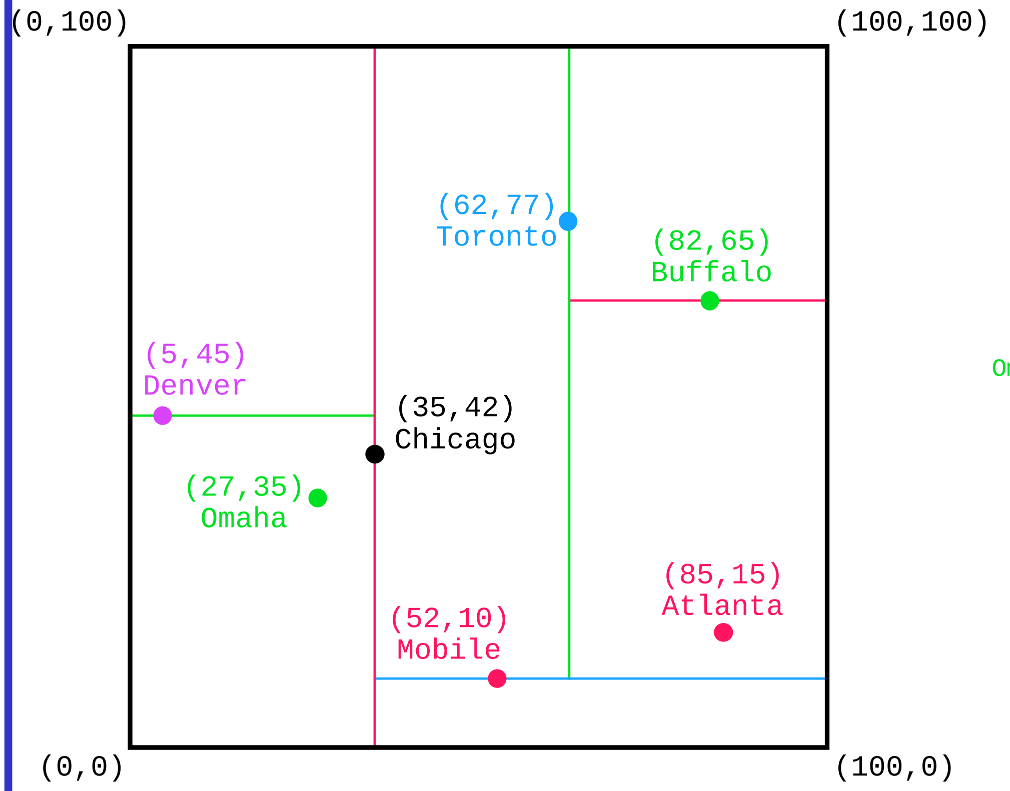
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

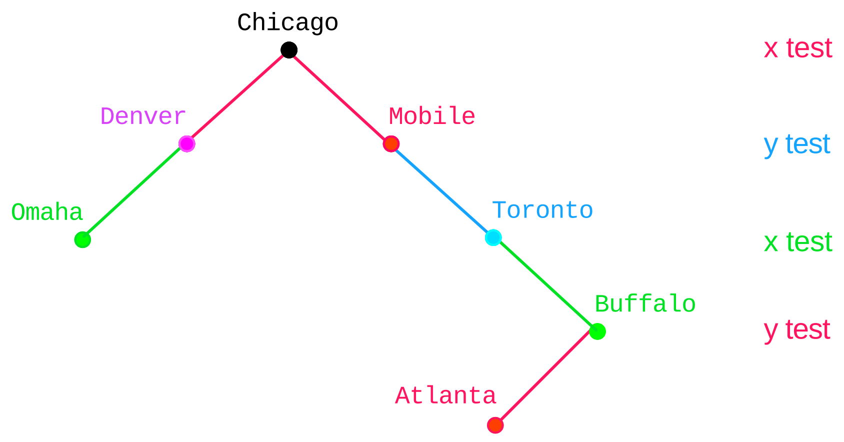




k-d tree (Bentley)

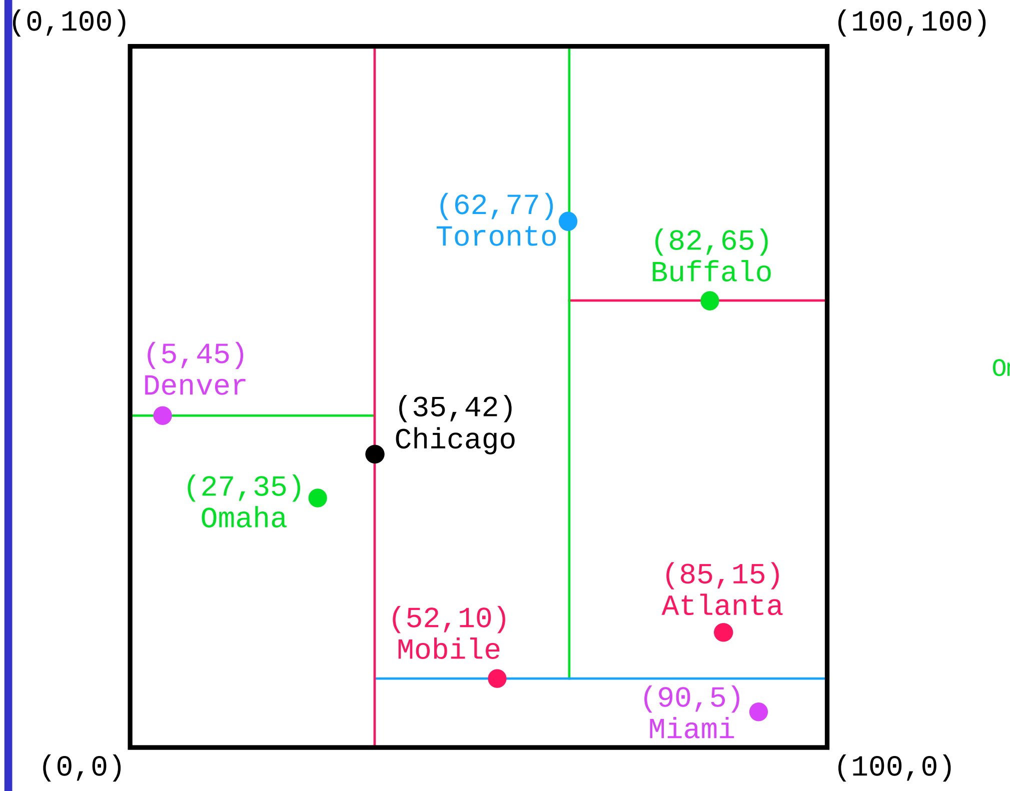
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered

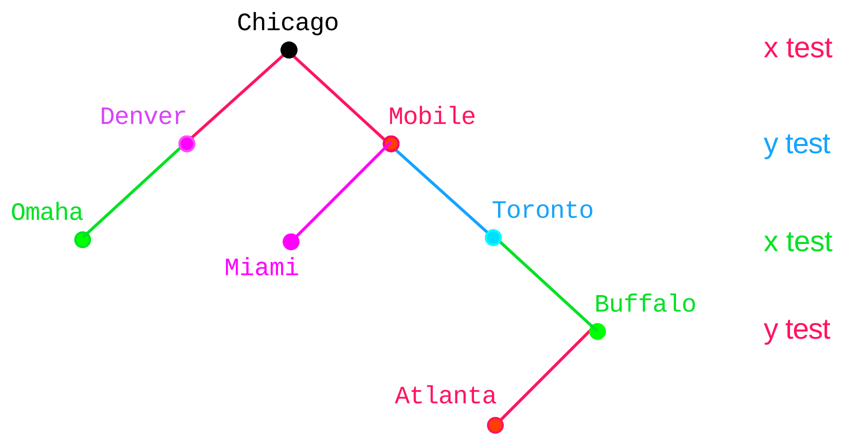




k-d tree (Bentley)

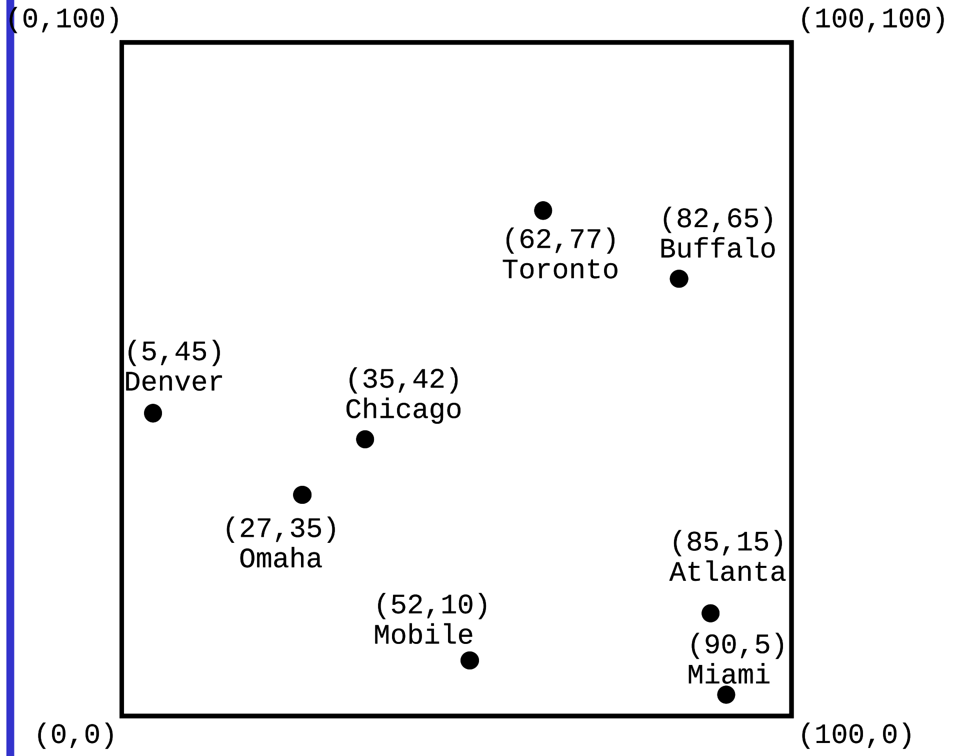
- Test one attribute at a time instead of all simultaneously as in the point quadtree
- Usually cycle through all the attributes
- Shape of the tree depends on the order in which the data is encountered





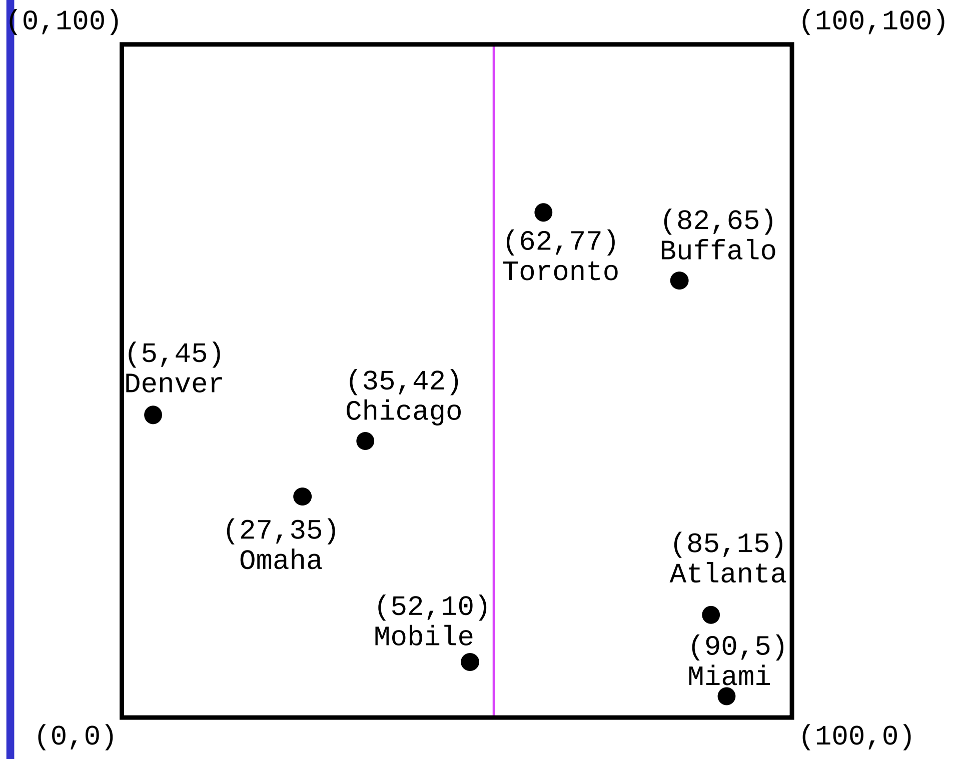
Adaptive k-d tree

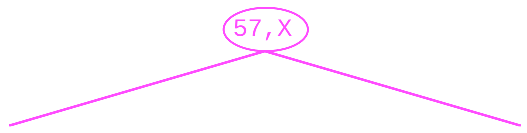
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum



Adaptive k-d tree

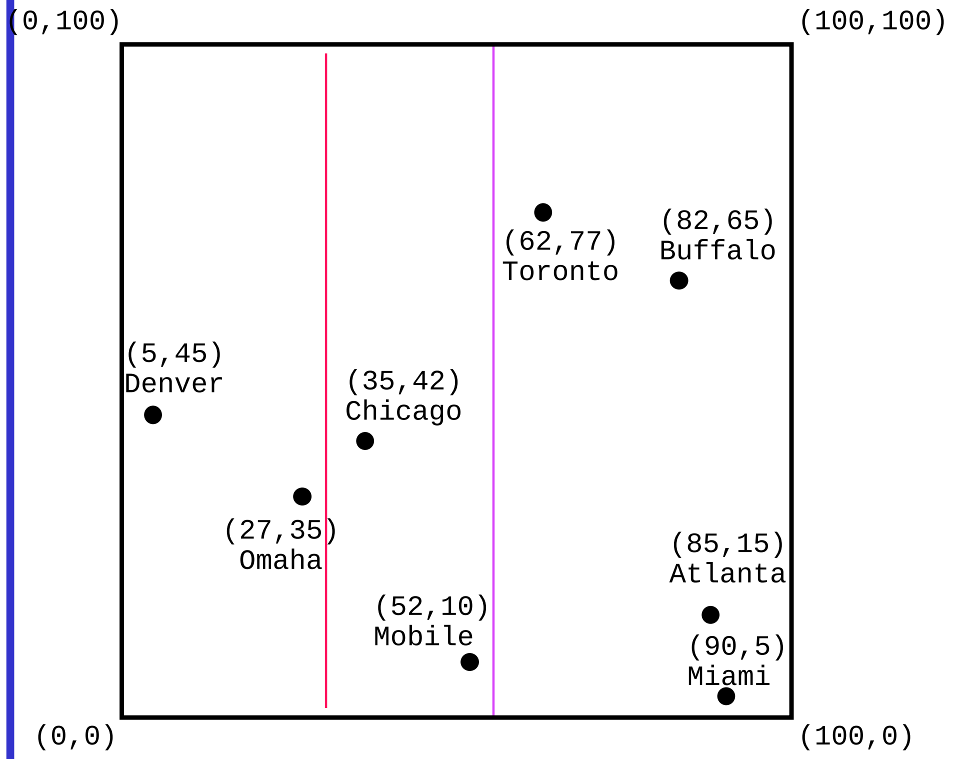
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

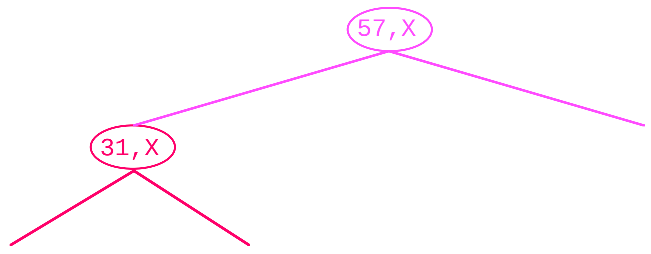




Adaptive k-d tree

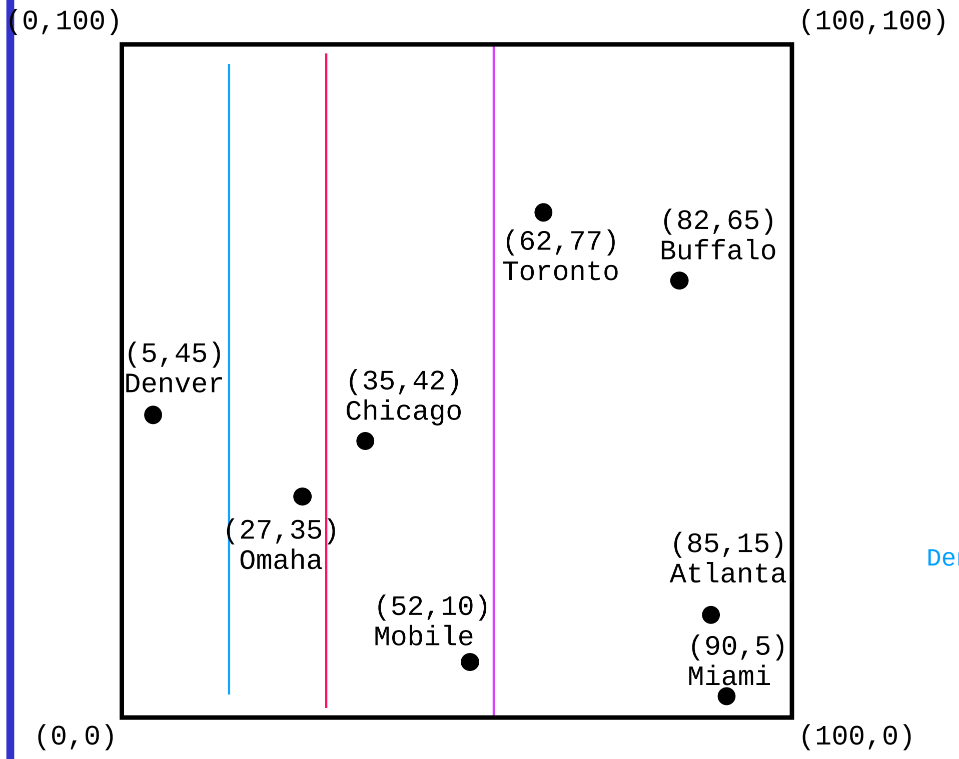
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

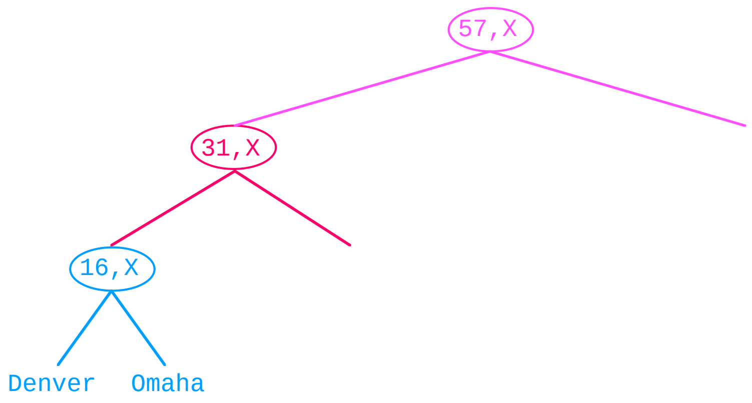




Adaptive k-d tree

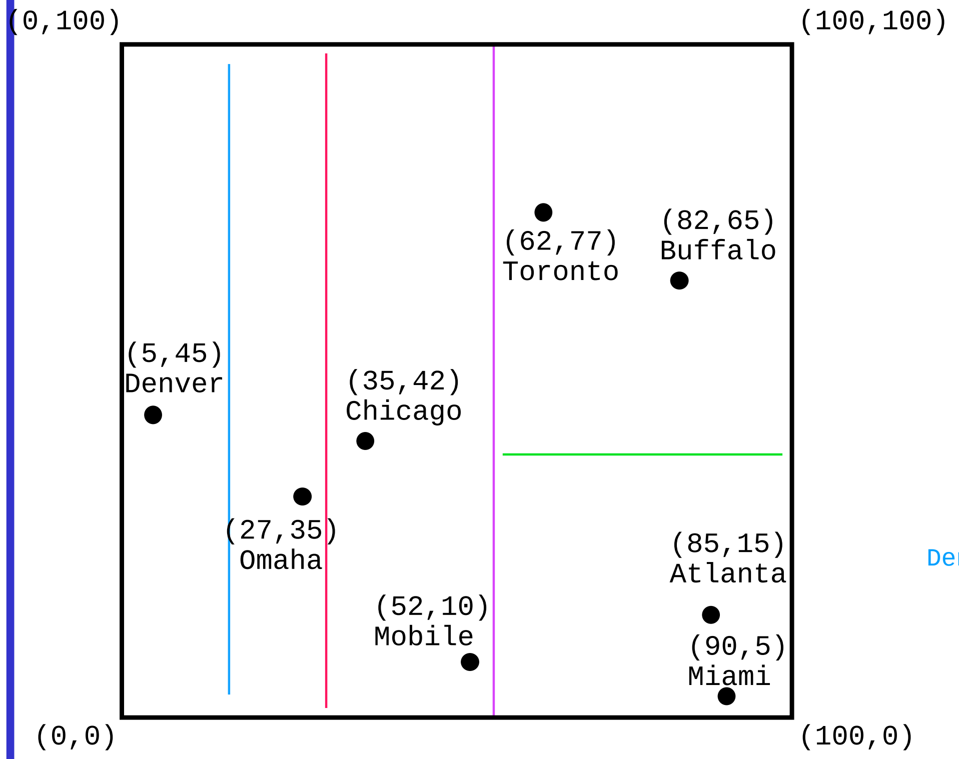
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

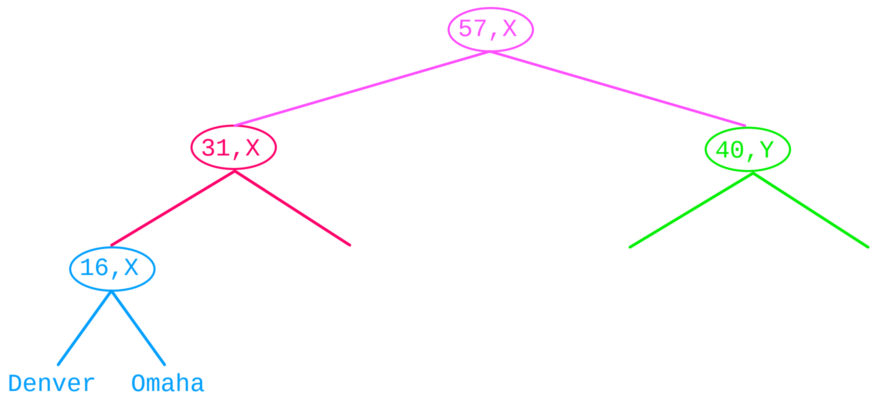




Adaptive k-d tree

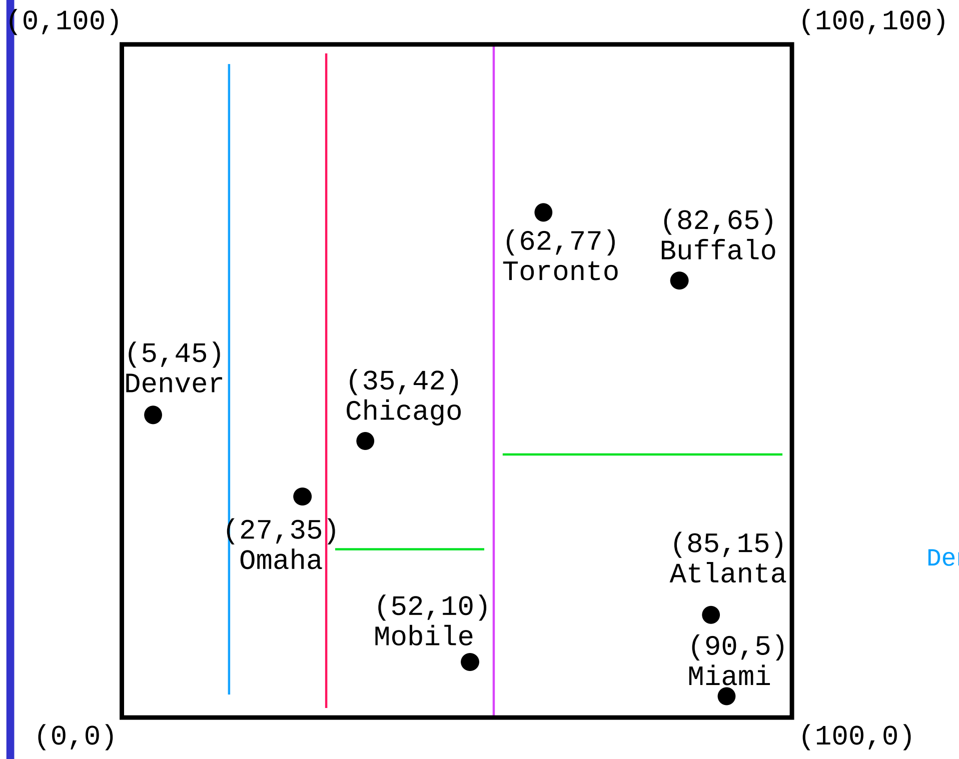
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

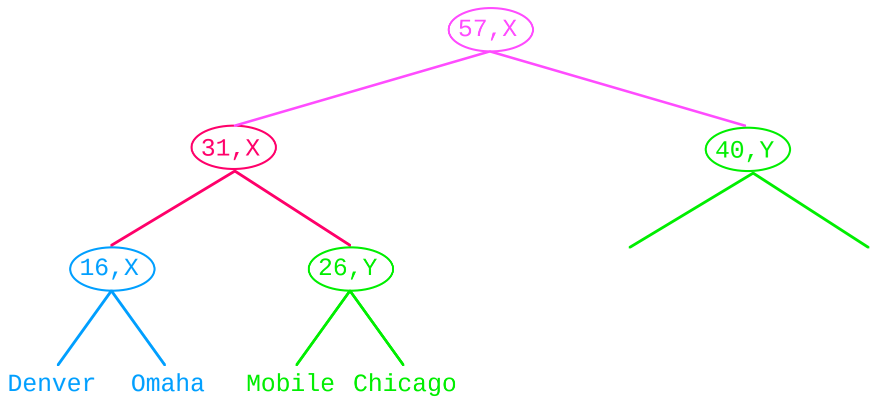




Adaptive k-d tree

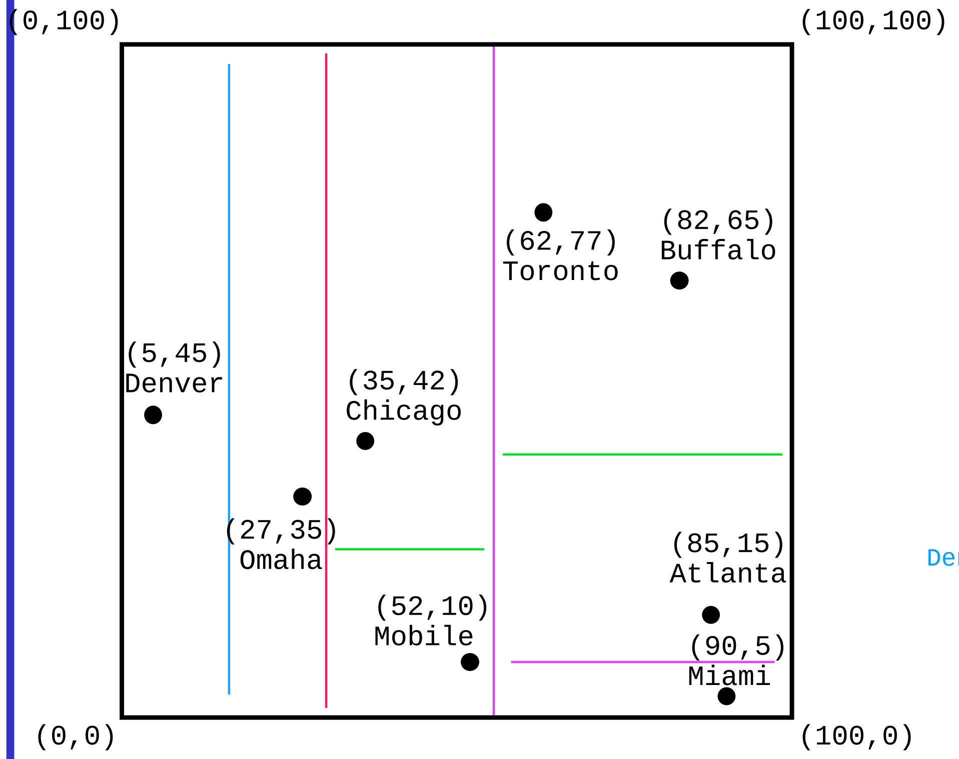
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

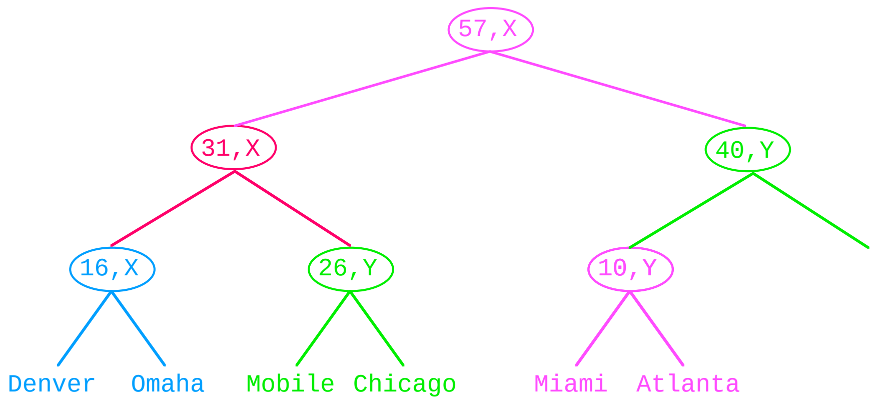




Adaptive k-d tree

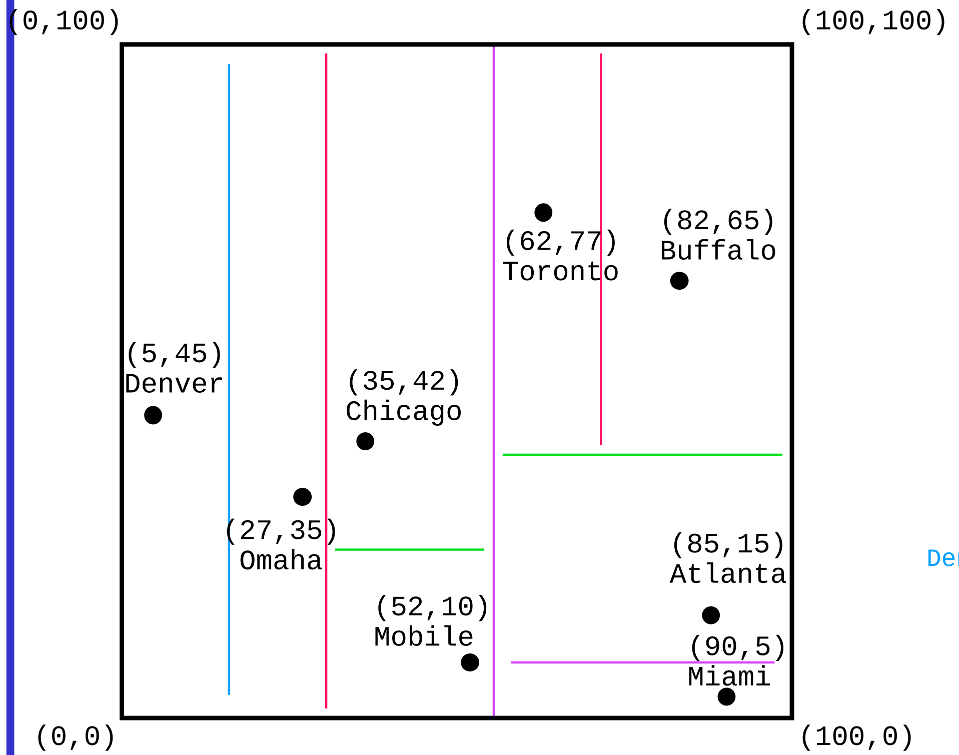
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

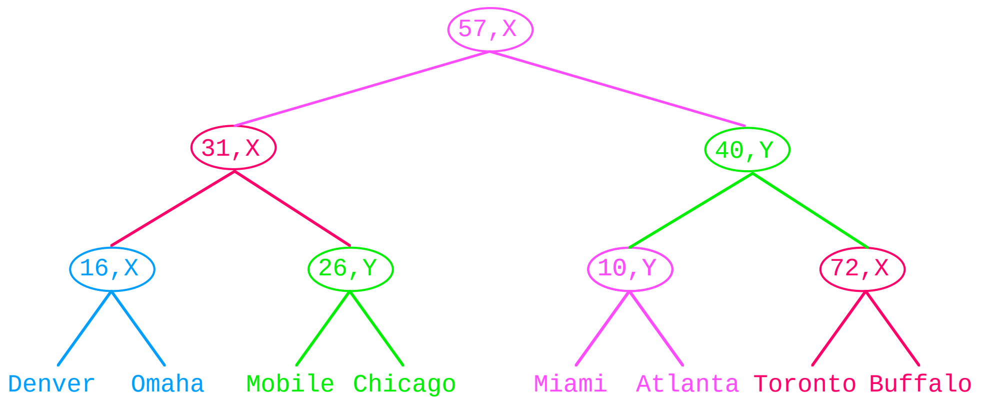




Adaptive k-d tree

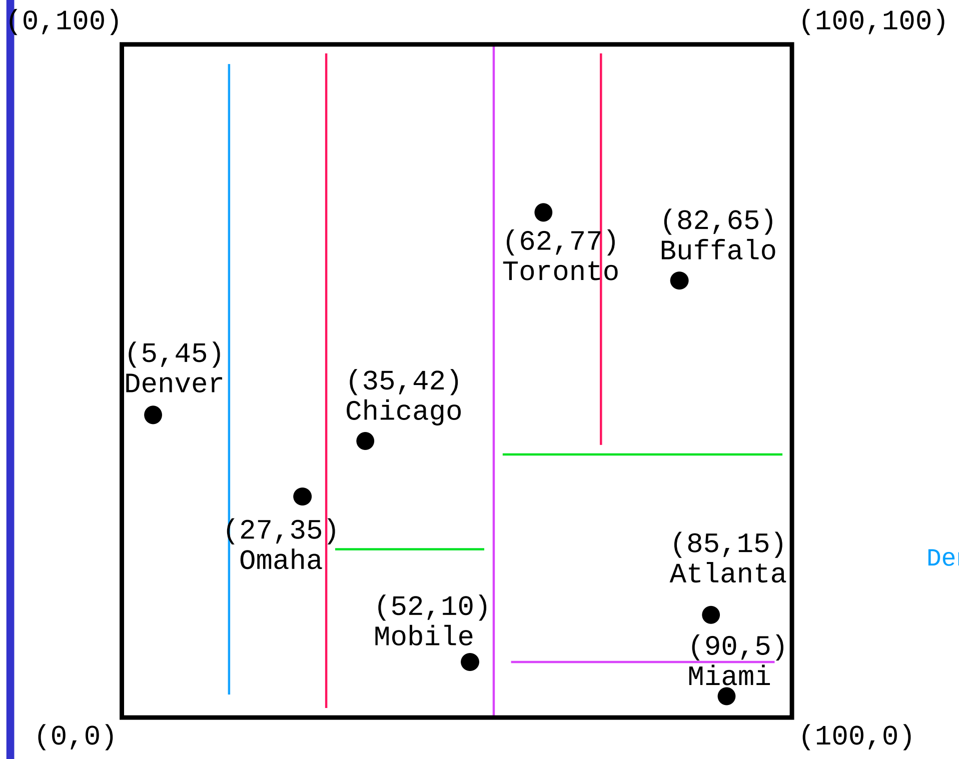
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum

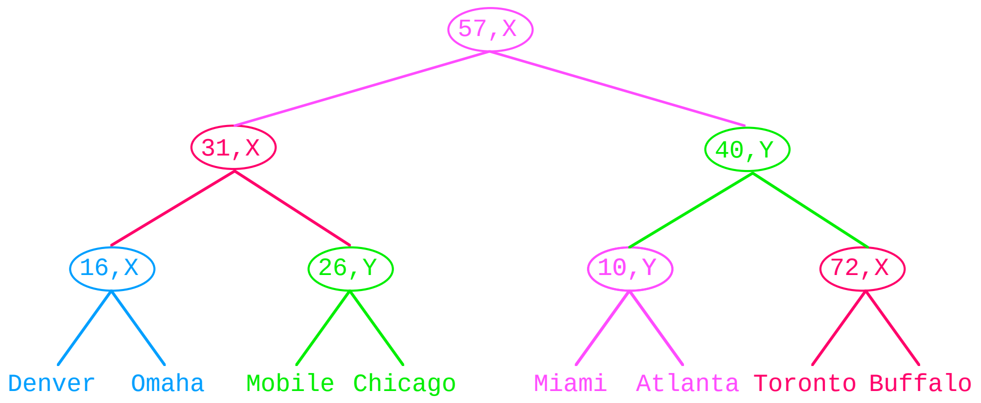




Adaptive k-d tree

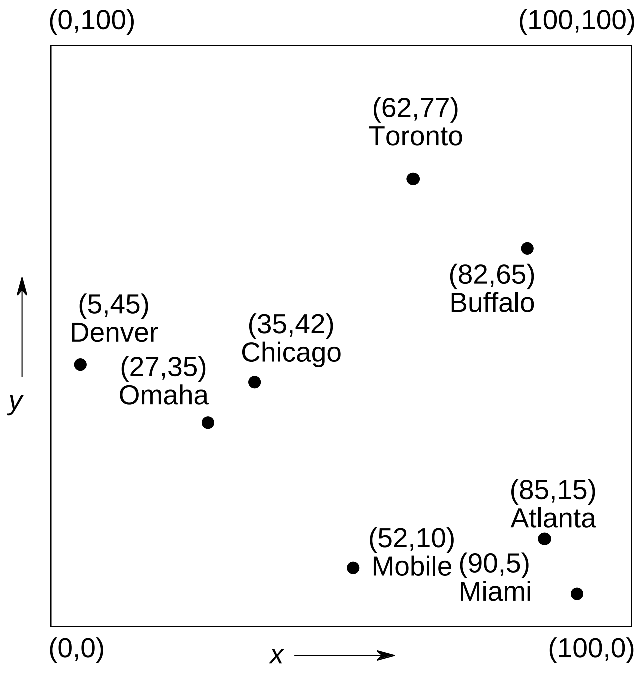
- Data is only stored in terminal nodes
- An interior node contains the median of the set as the discriminator
- The discriminator key is the one for which the spread of the values of the key is a maximum





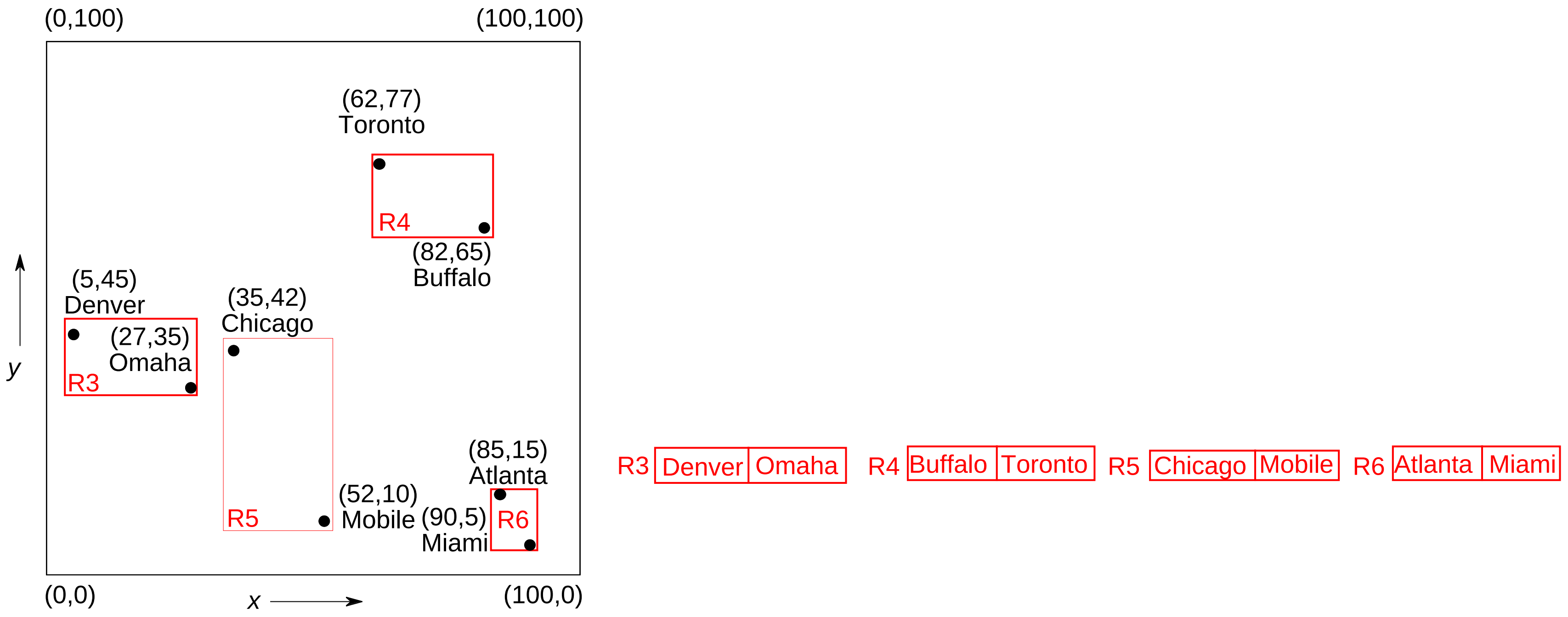
Minimum Bounding Rectangles: R-tree (Guttman)

- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode



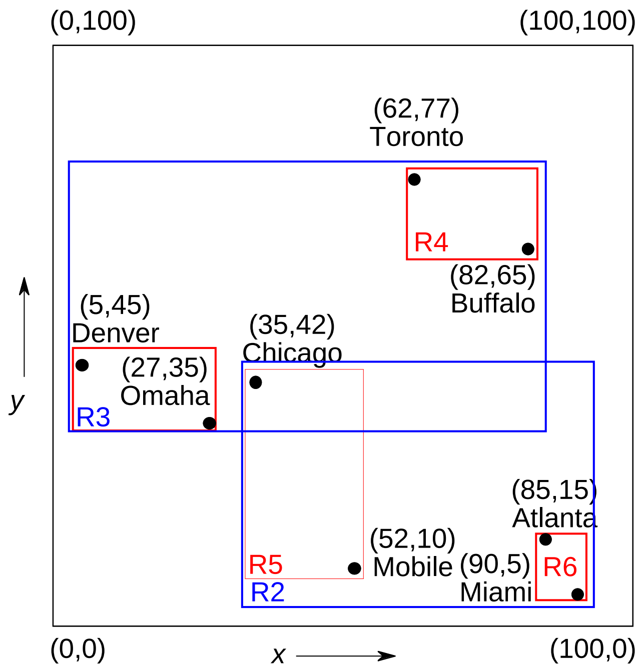
Minimum Bounding Rectangles: R-tree (Guttman)

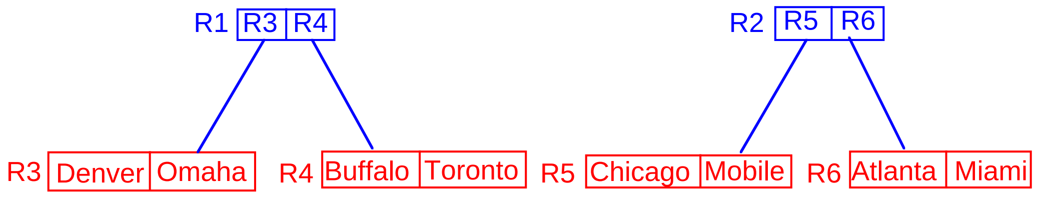
- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode



Minimum Bounding Rectangles: R-tree (Guttman)

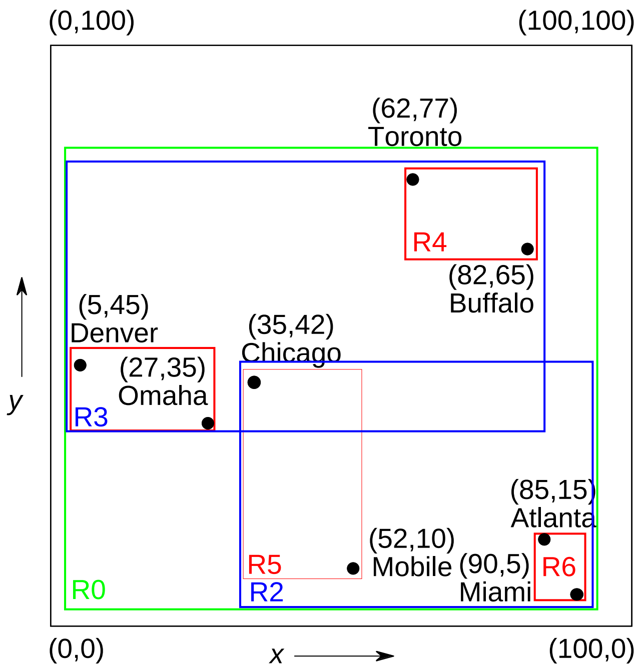
- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode

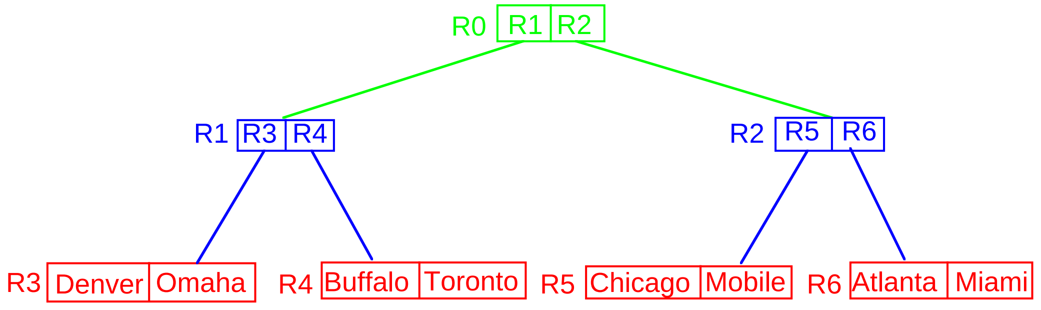




Minimum Bounding Rectangles: R-tree (Guttman)

- Objects grouped into hierarchies, stored in a structure similar to a B-tree
- Object has single bounding rectangle, yet area that it spans may be included in several bounding rectangles
- Drawback: not a disjoint decomposition of space (e.g., Chicago in R1+R2)
- Order (m, M) R-tree
  1. between $m \leq M/2$ and M entries in each node except root
  2. at least 2 entries in root unless a leaf node
- X-tree (Berchtold/Keim/Kriegel): if split creates too much overlap, then instead of splitting, create a supernode



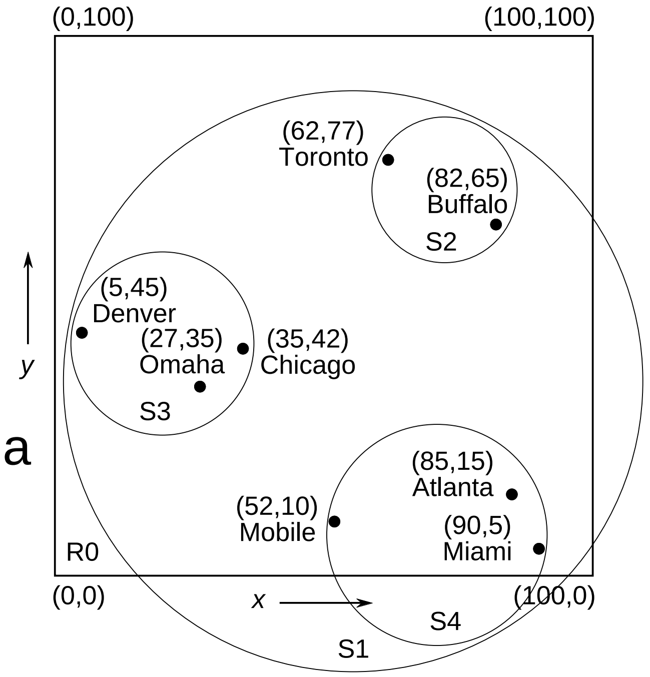


R*-tree (Beckmann et al.)

- Goal: minimize overlap for leaf nodes and area increase for nonleaf nodes
- Changes from R-tree:
  1. insert into leaf node p for which resulting bounding box has minimum increase in overlap with bounding boxes of p's brothers
     - compare with R-tree where insert into leaf node for which increase in area is a minimum (minimizes coverage)
  2. in case of overflow in p, instead of splitting p as in R-tree, reinsert a fraction of objects in p (e.g., farthest from centroid)
     - known as 'forced reinsertion' and similar to 'deferred splitting' or 'rotation' in B-trees
  3. in case of true overflow, use a two-stage process (goal: low coverage)
     - determine axis along which the split takes place
       a. sort bounding boxes for each axis on low/high edge to get $2d$ lists for d-dimensional data
       b. choose axis yielding lowest sum of perimeters for splits based on sorted orders
     - determine position of split
       a. position where overlap between two nodes is minimized
       b. resolve ties by minimizing total area of bounding boxes
- Works very well but takes time due to forced reinsertion

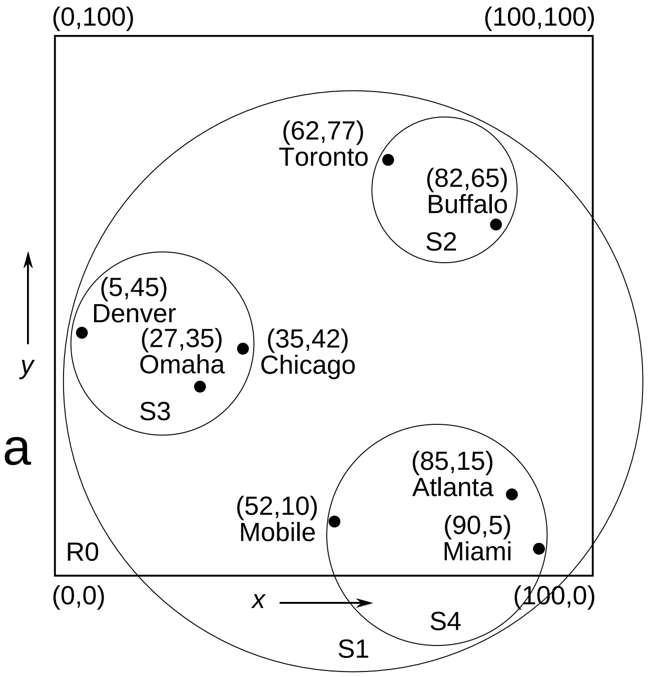
Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

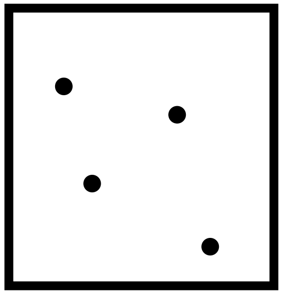


Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

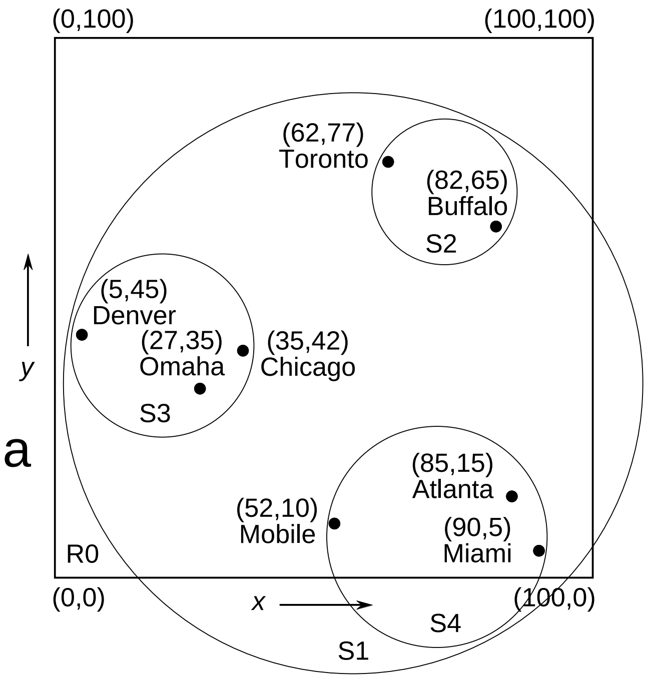


- SR-tree (Katayama/Satoh)
  1. bounding region is intersection of minimum bounding hyperrectangle and minimum bounding hypersphere
  2. motivated by desire to improve performance of SS-tree by reducing volume of minimum bounding boxes

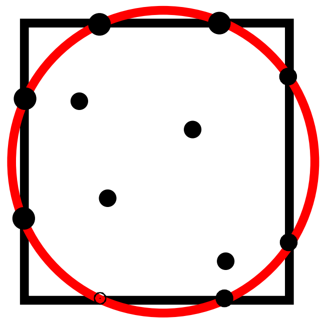


Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

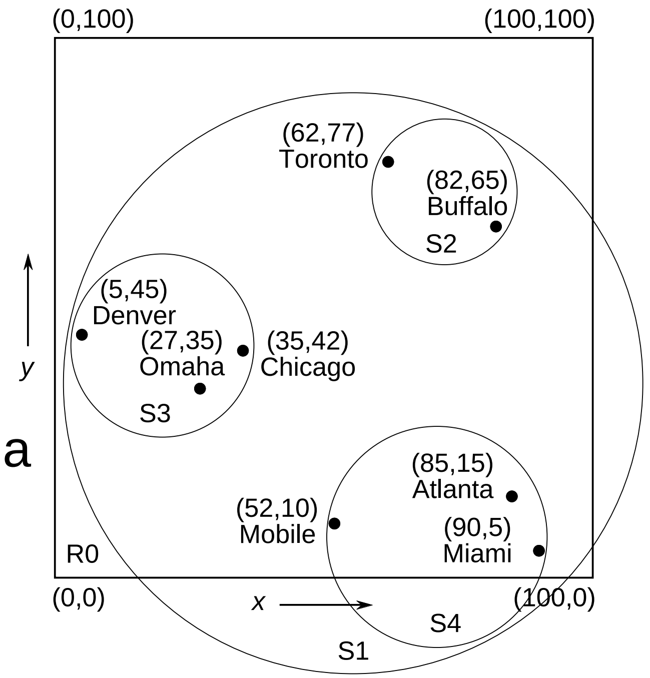


- SR-tree (Katayama/Satoh)
  1. bounding region is intersection of minimum bounding hyperrectangle and minimum bounding hypersphere
  2. motivated by desire to improve performance of SS-tree by reducing volume of minimum bounding boxes

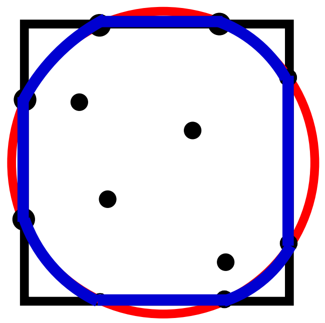


Minimum Bounding Hyperspheres

- SS-tree (White/Jain)
  1. make use of hierarchy of minimum bounding hyperspheres
  2. based on observation that hierarchy of minimum bounding hyperspheres is more suitable for hyperspherical query regions
  3. specifying a minimum bounding hypersphere requires slightly over one half the storage for a minimum bounding hyperrectangle
     - enables greater fanout at each node resulting in shallower trees
  4. drawback over minimum bounding hyperrectangles is that it is impossible cover space with minimum bounding hyperspheres without some overlap

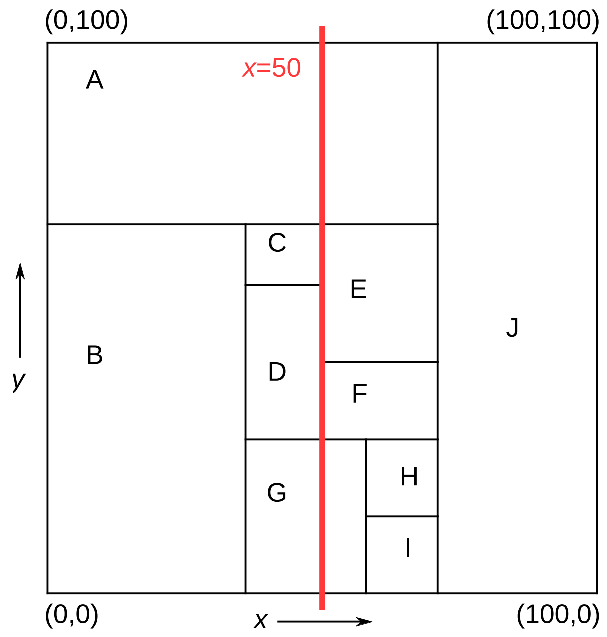


- SR-tree (Katayama/Satoh)
  1. bounding region is intersection of minimum bounding hyperrectangle and minimum bounding hypersphere
  2. motivated by desire to improve performance of SS-tree by reducing volume of minimum bounding boxes



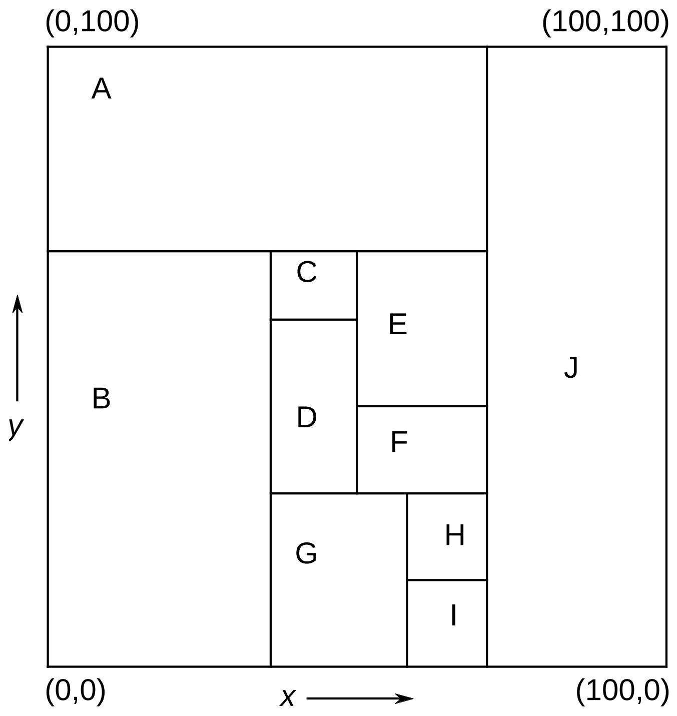
K-D-B-tree (Robinson)

- Rectangular embedding space is hierarchically decomposed into disjoint rectangular regions
- No dead space in the sense that at any level of the tree, entire embedding space is covered by one of the nodes
- Aggregate blocks of k-d tree partition of space into nodes of finite capacity
- When a node overflows, it is split along one of the axes
- Originally developed to store points but may be extended to non-point objects represented by their minimum bounding boxes
- Drawback: to get area covered by object, must retrieve all cells it occupies



Hybrid tree (Chakrabarti/Mehrotra)

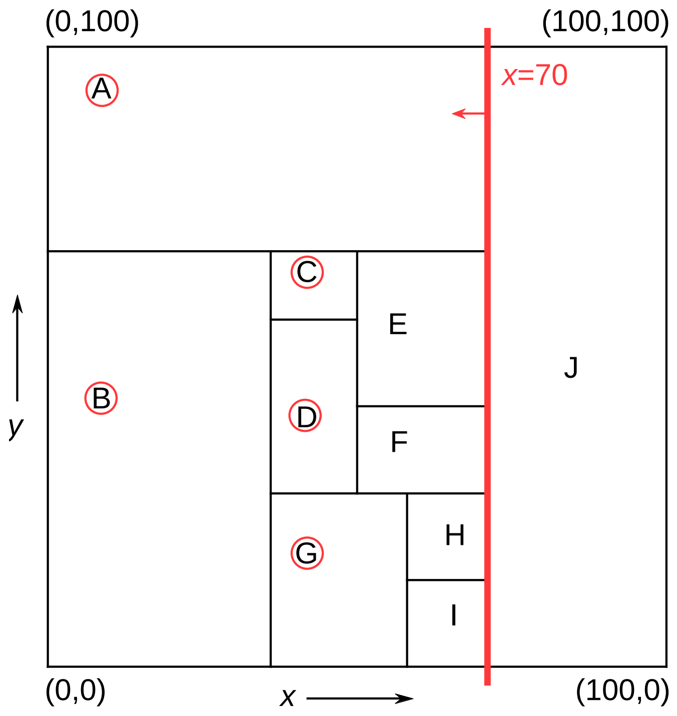
1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement



Hybrid tree (Chakrabarti/Mehrotra)

1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement

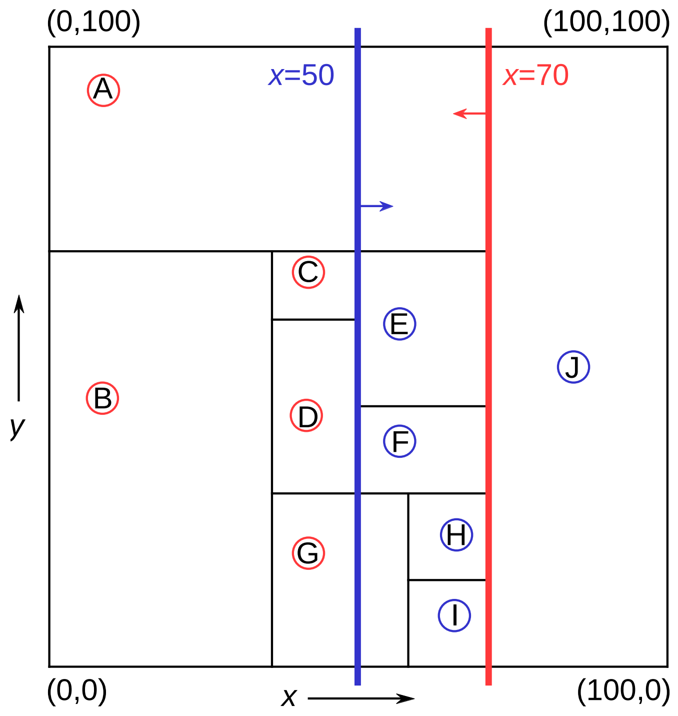
2. Add two partition lines at $x = 70$ for region low
   a. A, B, C, D, and G with region low



Hybrid tree (Chakrabarti/Mehrotra)

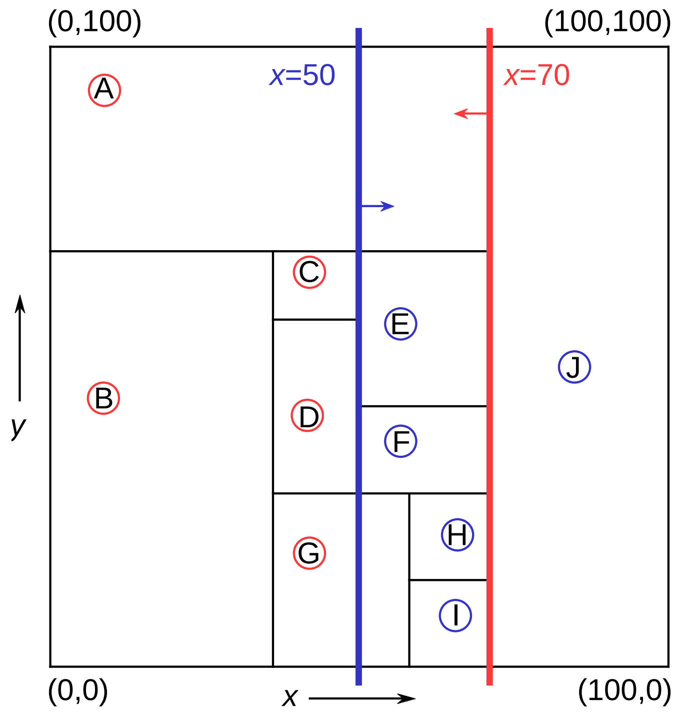
1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement

2. Add two partition lines at $x = 70$ for region low and $x = 50$ for region $high$
   a. A, B, C, D, and G with region low
   b. E, F, H, I, and J with region $high$



Hybrid tree (Chakrabarti/Mehrotra)

1. Variant of k-d-B-tree that avoids splitting the region and point pages that intersect a partition line l along partition axis a with value v by slightly relaxing the disjointness requirement



2. Add two partition lines at $x = 70$ for region low and $x = 50$ for region $high$
    a. A, B, C, D, and G with region low
    b. E, F, H, I, and J with region $high$

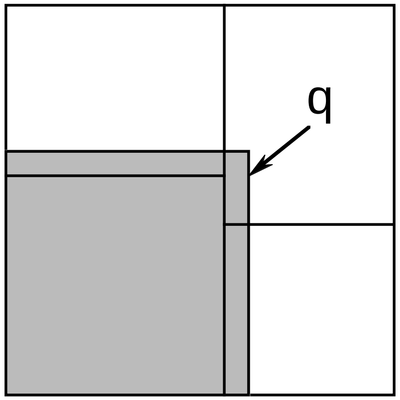
3. Associating two partition lines with each partition region is analogous to associating a bounding box with each region (also spatial k-d tree)
    - similar to bounding box in R-tree but not minimum bounding box
    - store approximation of bounding box by quantizing coordinate value along each dimension to b bits for a total of $2bd$ bits for each box thereby reducing fanout of each node (Henrich)

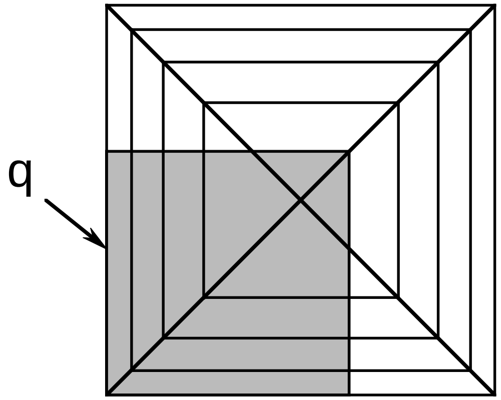
Avoiding Overlapping All of the Leaf Blocks

- Assume uniformly-distributed data
  1. most data points lie near the boundary of the space that is being split
     - Ex: for $d = 20$, 98.5% of the points lie within 10% of the surface
     - Ex: for $d = 100$, 98.3% of the points lie within 2% of the surface
  2. rarely will all of the dimensions be split even once
     - Ex: assuming at least $M/2$ points per leaf node blocks, and at least one split along each dimension, then total number of points N must be at least $2^d M/2$
     - if $d = 20$ and $M = 10$, then N must be at least 5 million to split along all dimensions once
  3. if each region is split at most once, and without loss of generality, split is in half, then query region usually intersects all the leaf node blocks
     - query selectivity of 0.01% for $d = 20$ leads to ‘side length of query region’=0.63 which means that it intersects all the leaf node blocks
     - implies a range query will visit each leaf node block
- One solution: use a 3-way split along each dimension into three parts of proportion r, $1 - 2r$, and r
- Sequential scan may be cheaper than using an index due to high dimensions
  - We assume our data is not of such high dimensionality!

Pyramid Technique (Berchtold/Böhm/Kriegel)

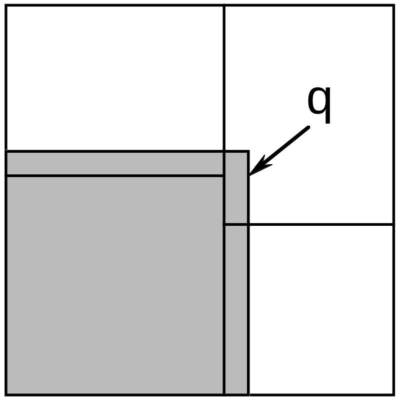
- Subdivide data space as if it is an onion by peeling off hypervolumes that are close to the boundary
- Subdivide hypercube into $2d$ pyramids having the center of the data space as the tip of their cones
- Each of the pyramids has one of the faces of the hypercube as its base
- Each pyramid is decomposed into slices parallel to its base
- Useful when query region side length is greater than half the width of the data space as won't have to visit all leaf node blocks

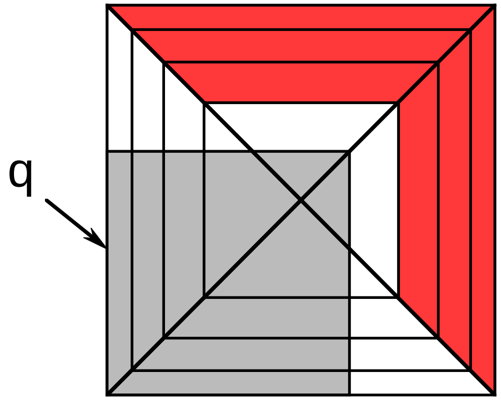




Pyramid Technique (Berchtold/Böhm/Kriegel)

- Subdivide data space as if it is an onion by peeling off hypervolumes that are close to the boundary
- Subdivide hypercube into $2d$ pyramids having the center of the data space as the tip of their cones
- Each of the pyramids has one of the faces of the hypercube as its base
- Each pyramid is decomposed into slices parallel to its base
- Useful when query region side length is greater than half the width of the data space as won't have to visit all leaf node blocks





- Pyramid containing q is the one corresponding to the coordinate i whose distance from the center point of the space is greater than all others
- Analogous to iMinMax method (Ooi/Tan/Yu/Bressan) with exception that iMinMax associates a point with its closest surface but the result is still a decomposition of the underlying space into $2d$ pyramids

Methods Based on a Sequential Scan

1. If neighbor finding in high dimensions must access every disk page at random, then a linear scan may be more efficient
   - advantage of sequential scan over hierarchical indexing methods is that actual I/O cost is reduced by being able to scan the data sequentially instead of at random as only need one disk seek
2. VA-file (Weber et al.)
   - use b_i bits per feature i to approximate feature
   - impose a d dimensional grid with $b = \sum_{i=1}^{d} b_i$ grid cells
   - sequentially scan all grid cells as a filter step to determine possible candidates which are then checked in their entirety via a disk access
   - VA-file is an additional representation in the form of a grid which is imposed on the original data
3. Other methods apply more intelligent quantization processes
   - VA+-file (Ferhatosmanoglu et al): decorrelate the data with KLT yielding new features and vary number of bits as well as use clustering to determine the region partitions
   - IQ-tree (Berchtold et al): hierarchical like an R-tree with unordered minimum bounding rectangles

Part 2: Distance-Based Indexing

1. Basic definitions
2. Properties for pruning the search
3. Ball partitioning methods
    a. vp-tree
    b. mvp-tree
4. General hyperplane partitioning methods
    a. gh-tree
    b. GNAT
    c. Bisector trees and mb-trees
5. M-tree
6. sa-tree
7. Distance matrix methods
    a. AESA
    b. LAESA

Basic Definitions

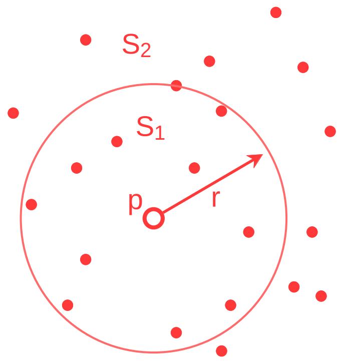
1. Often only information available is a distance function indicating degree of similarity (or dis-similarity) between all pairs of N data objects
2. Distance metric d: objects must reside in finite metric space (S, d) where for o_1, o_2, o_3 in S, d must satisfy
   - $d(o_1, o_2) = d(o_2, o_1)$ (symmetry)
   - $d(o_1, o_2) \geq 0$, $d(o_1, o_2) = 0$ iff $o_1 = o_2$ (non-negativity)
   - $d(o_1, o_3) \leq d(o_1, o_2) + d(o_2, o_3)$ (triangle inequality)
3. Triangle inequality is a key property for pruning search space
   - Computing distance is expensive
4. Non-negativity property enables ignoring negative values in derivations

Pivots

- Identify a distinguished object or subset of the objects termed *pivots* or *vantage points*
  1. sort remaining objects based on
     a. distances from the pivots, or
     b. which pivot is the closest
  2. and build index
  3. use index to achieve pruning of other objects during search
- Given pivot $p \in S$, for all objects $o \in S' \subseteq S$, we know:
  1. exact value of $d(p, o)$,
  2. $d(p, o)$ lies within range $[r_{\mathrm{lo}}, r_{\mathrm{hi}}]$ of values or
     - drawback is asymmetry of partition as outer shell is usually narrow
  3. o is closer to p than to some other object $p_2 \in S$
- Distances from pivots are useful in pruning the search

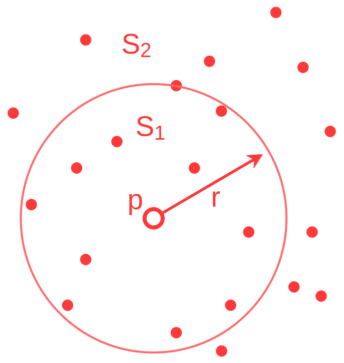
Pivots

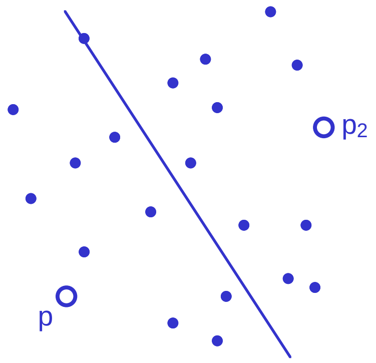
- Identify a distinguished object or subset of the objects termed *pivots* or *vantage points*
  1. sort remaining objects based on
     a. distances from the pivots, or
     b. which pivot is the closest
  2. and build index
  3. use index to achieve pruning of other objects during search
- Given pivot $p \in S$, for all objects $o \in S' \subseteq S$, we know:
  1. exact value of $d(p, o)$,
  2. $d(p, o)$ lies within range $[r_{\mathrm{lo}}, r_{\mathrm{hi}}]$ of values (ball partitioning) or
     - drawback is asymmetry of partition as outer shell is usually narrow
  3. o is closer to p than to some other object $p_2 \in S$
- Distances from pivots are useful in pruning the search



Pivots

- Identify a distinguished object or subset of the objects termed *pivots* or *vantage points*
  1. sort remaining objects based on
     a. distances from the pivots, or
     b. which pivot is the closest
  2. and build index
  3. use index to achieve pruning of other objects during search
- Given pivot $p \in S$, for all objects $o \in S' \subseteq S$, we know:
  1. exact value of $d(p,o)$,
  2. $d(p,o)$ lies within range $[r_{\mathrm{lo}}, r_{\mathrm{hi}}]$ of values (ball partitioning) or
     - drawback is asymmetry of partition as outer shell is usually narrow
  3. o is closer to p than to some other object $p_2 \in S$ (generalized hyperplane partitioning)
- Distances from pivots are useful in pruning the search

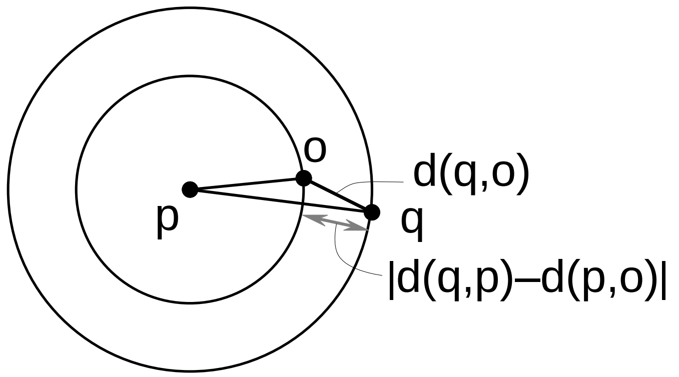


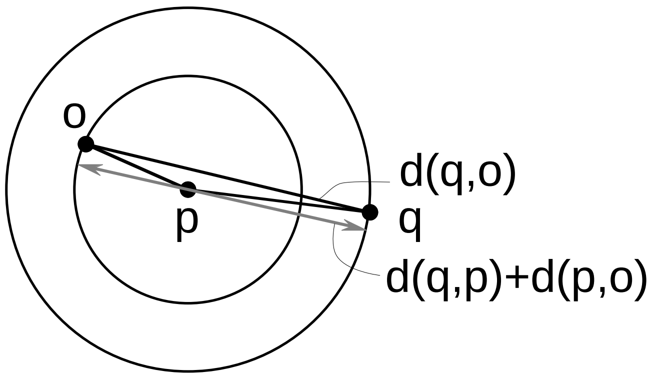


Pruning: Two Distances

Lemma 1: Knowing distance $d(p,q)$ from p to q and distance $d(p,o)$ from p to o enables bounding the distance $d(q,o)$ from q to o:

$$|d(q,p) - d(p,o)| \leq d(q,o) \leq d(q,p) + d(p,o)$$

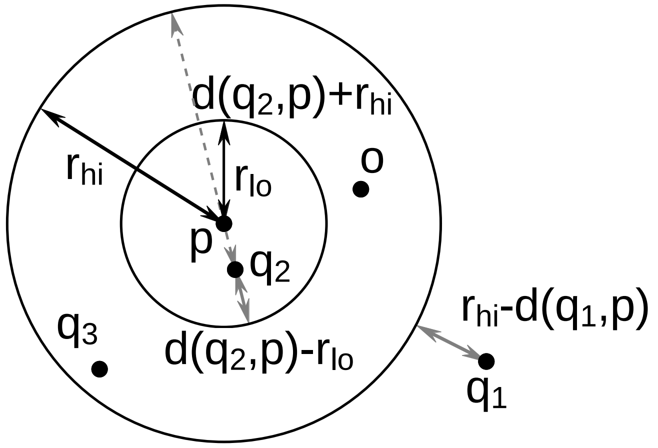




Pruning: One Distance and One Range

Lemma 2: Knowing distance $d(p,q)$ from p to q and that distance $d(p,o)$ from p to o is in the range $[r_{\mathrm{lo}}, r_{\mathrm{hi}}]$ enables bounding the distance $d(q,o)$ from q to o:

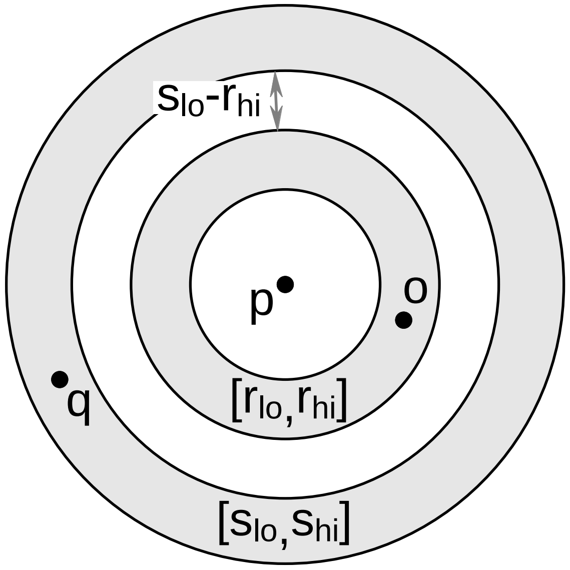
$$\max\{d(q,p) - r_{\mathrm{hi}}, r_{\mathrm{lo}} - d(q,p), 0\} \leq d(q,o) \leq d(q,p) + r_{\mathrm{hi}}$$

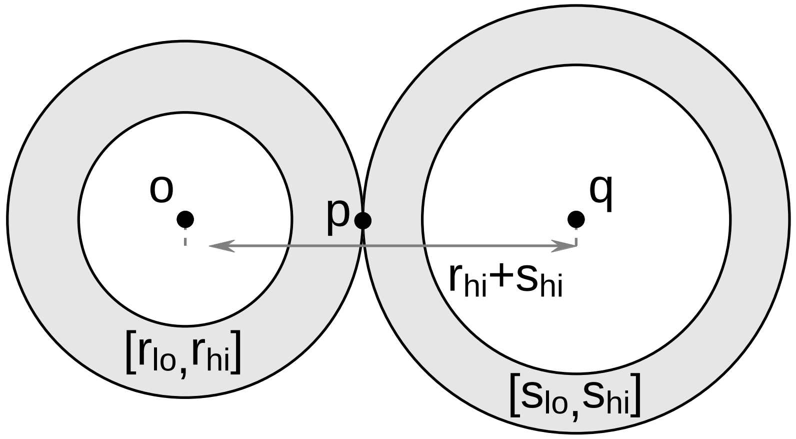


Pruning: Two Ranges

Lemma 3: Knowing that the distance $d(p,q)$ from p to q is in the range $[s_{\rm lo}, s_{\rm hi}]$ and and that distance $d(p,o)$ from p to o is in the range $[r_{\rm lo}, r_{\rm hi}]$ enables bounding the distance $d(q,o)$ from q to o:

$$\max\{s_{\rm lo} - r_{\rm hi}, r_{\rm lo} - s_{\rm hi}, 0\} \leq d(q,o) \leq r_{\rm hi} + s_{\rm hi}$$

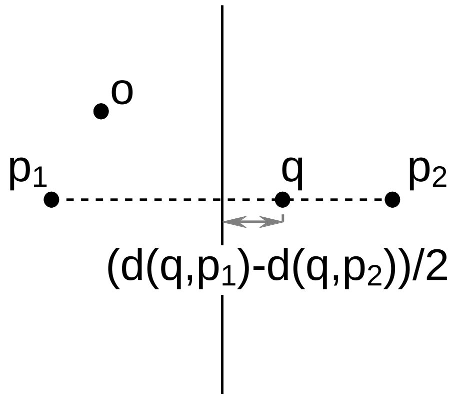




Pruning: Two Objects and Identity of Closest

Lemma 4: Knowing the distance $d(q, p_1)$ and $d(q, p_2)$ from q to pivot objects p_1 and p_2 and that o is closer to p_1 than to p_2 (or equidistant from both — i.e., $d(p_1, o) \leq d(p_2, o)$) enables a lower bound on the distance $d(q, o)$ from q to o:

$$\max\left\{\frac{d(q,p_1) - d(q,p_2)}{2}, 0\right\} \leq d(q,o)$$



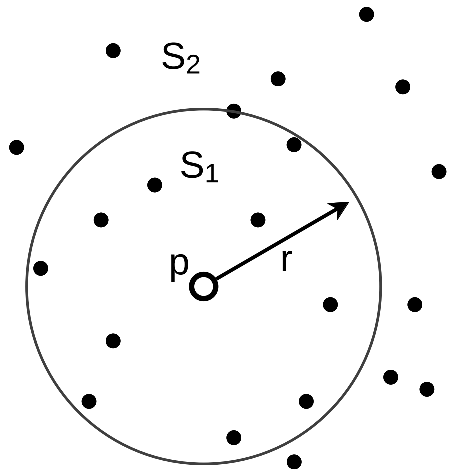
- Lower bound is attained when q is anywhere on the line from p_1 to p_2
- Lower bound decreases as q is moved off the line
- No upper bound as objects can be arbitrarily far from p_1 and p_2

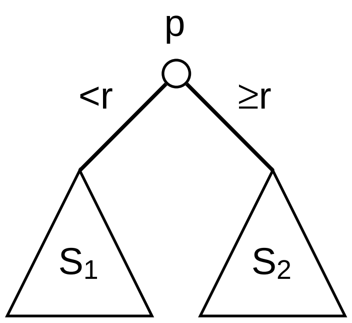
vp-tree (Metric tree; Uhlmann|Yianilos)

- Ball partitioning method
- Pick p from S and let r be median of distances of other objects from p
- Partition S into two sets S_1 and S_2 where:

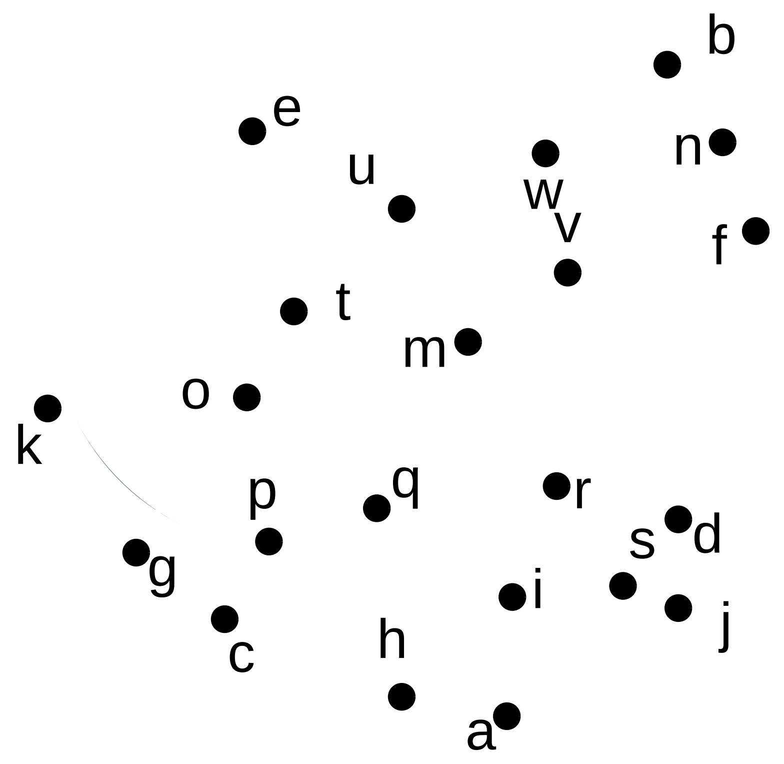
$$\begin{aligned} S_1 &= \{o \in S \setminus \{p\} \mid d(p,o) < r\} \\ S_2 &= \{o \in S \setminus \{p\} \mid d(p,o) \geq r\} \end{aligned}$$

- Apply recursively, yielding a binary tree with pivot and radius values at internal nodes
- Choosing pivots
  1. simplest is to pick at random
  2. choose a random sample and then select median

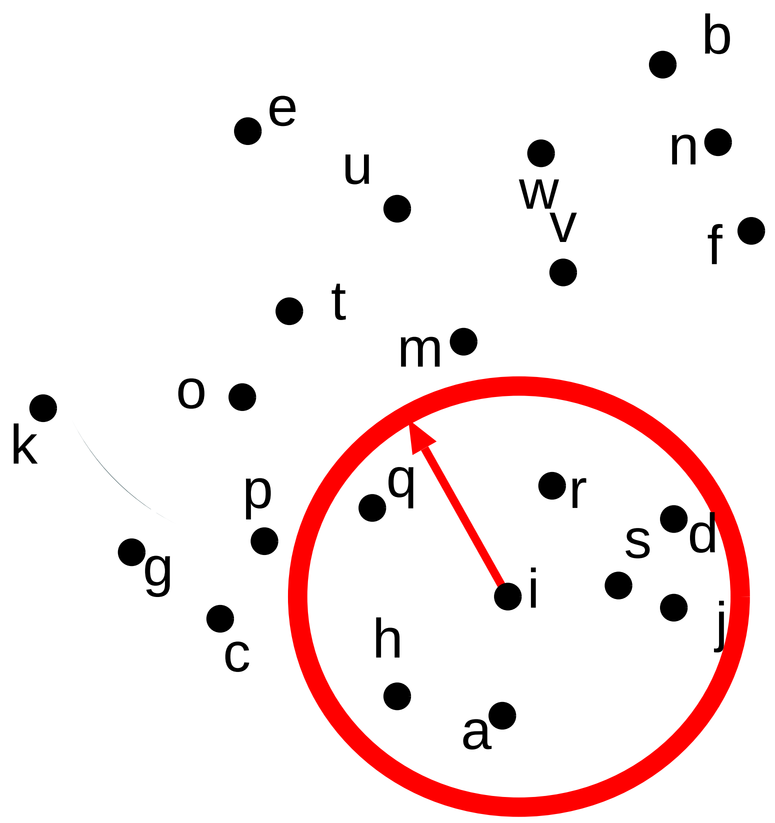


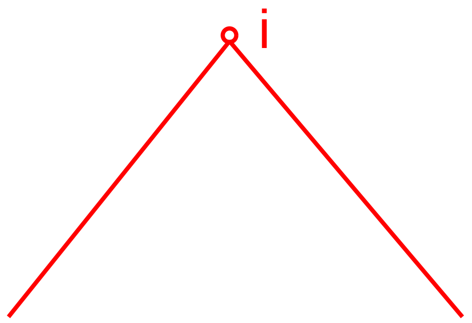


vp-tree Example

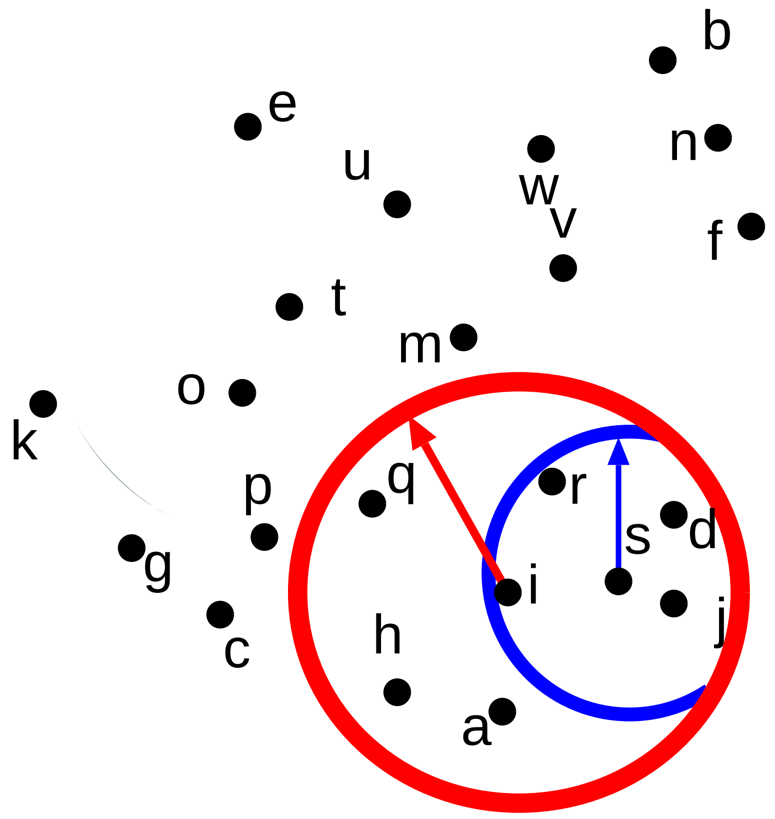


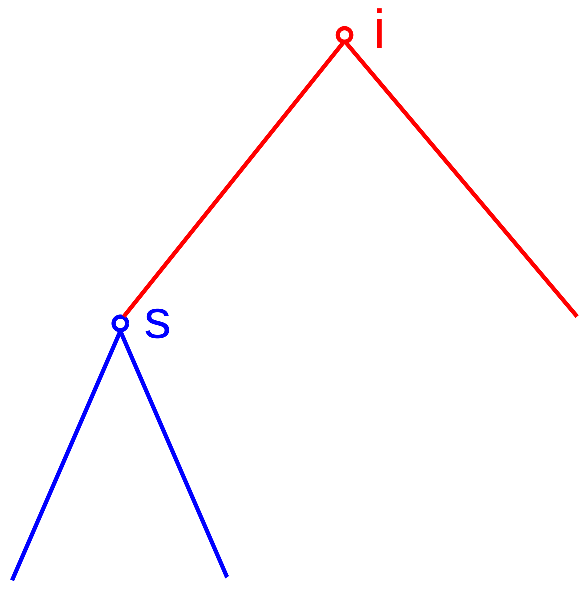
vp-tree Example



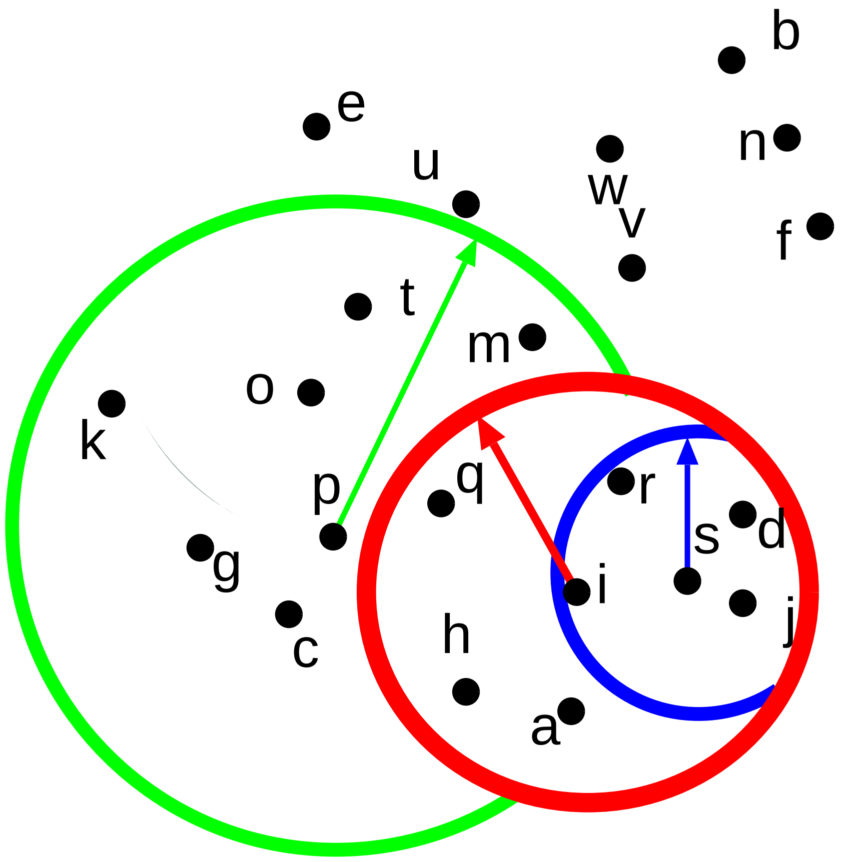


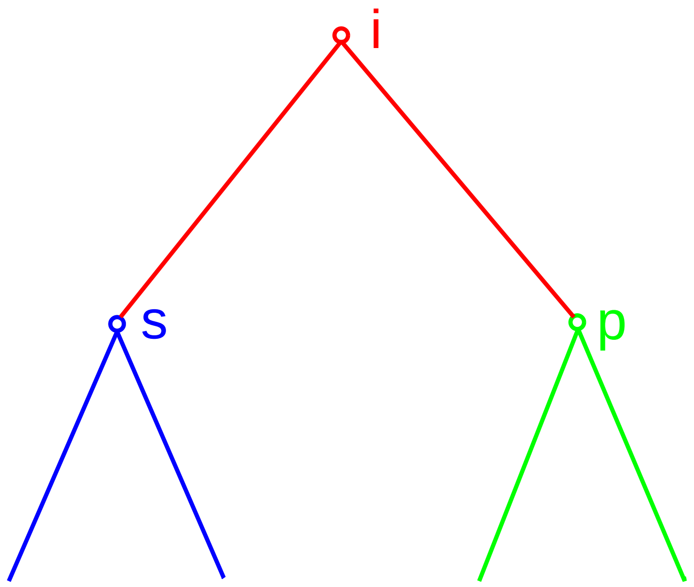
vp-tree Example



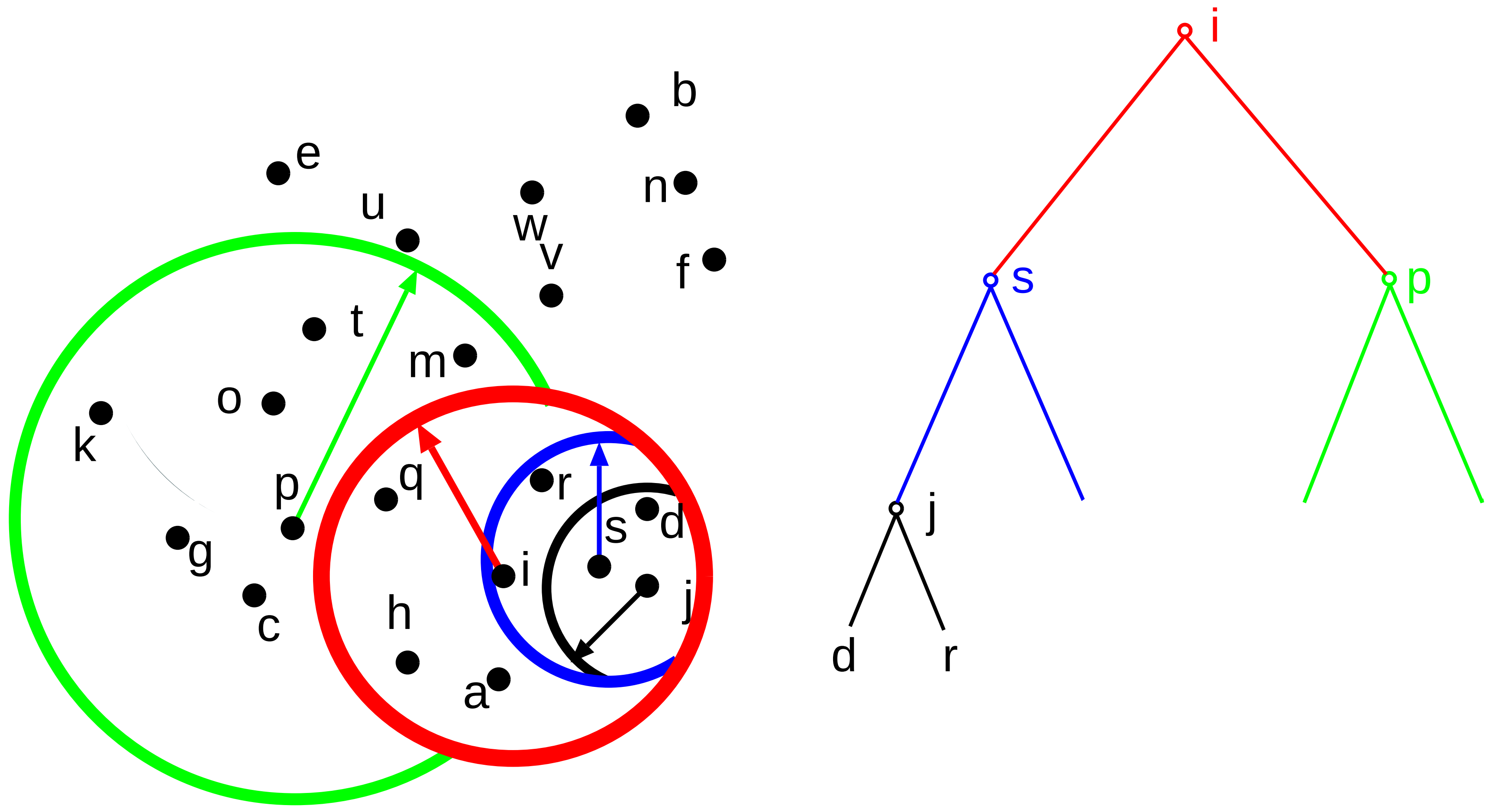


vp-tree Example

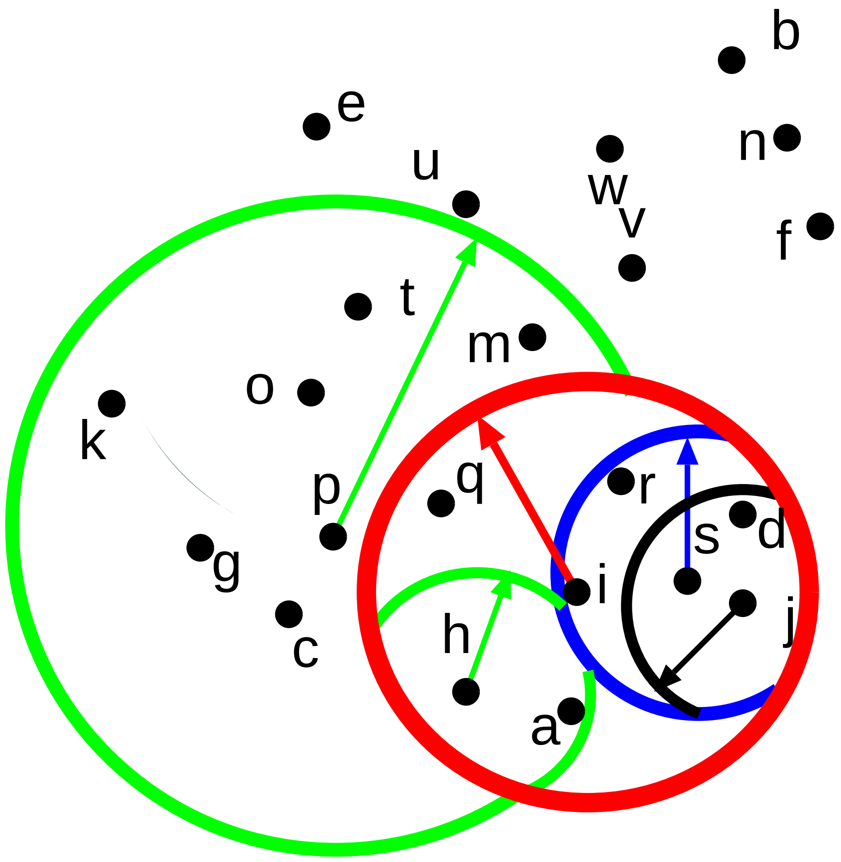


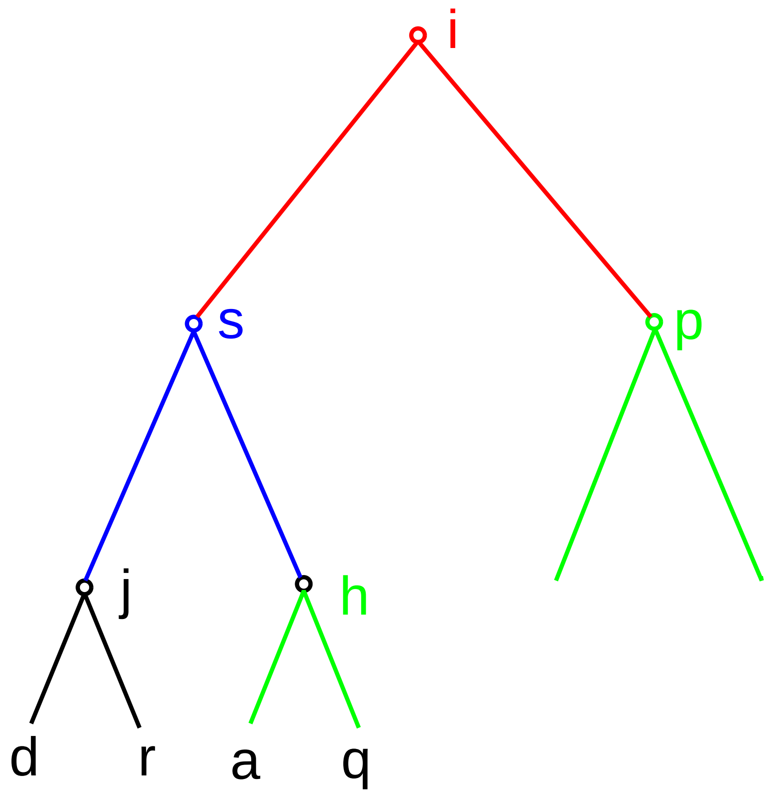


vp-tree Example

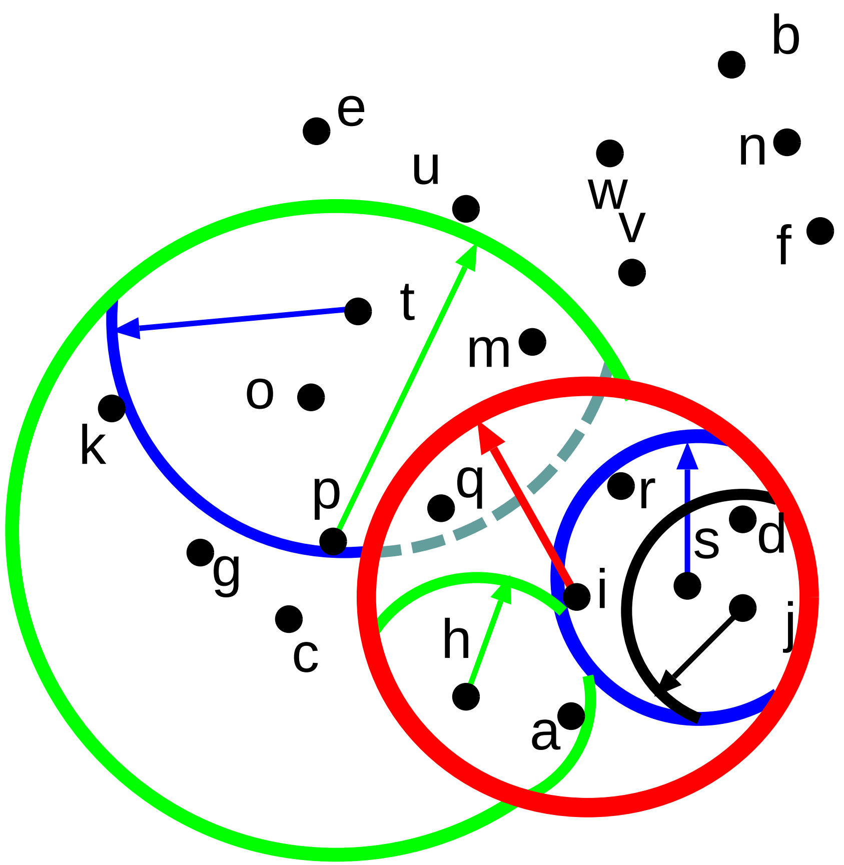


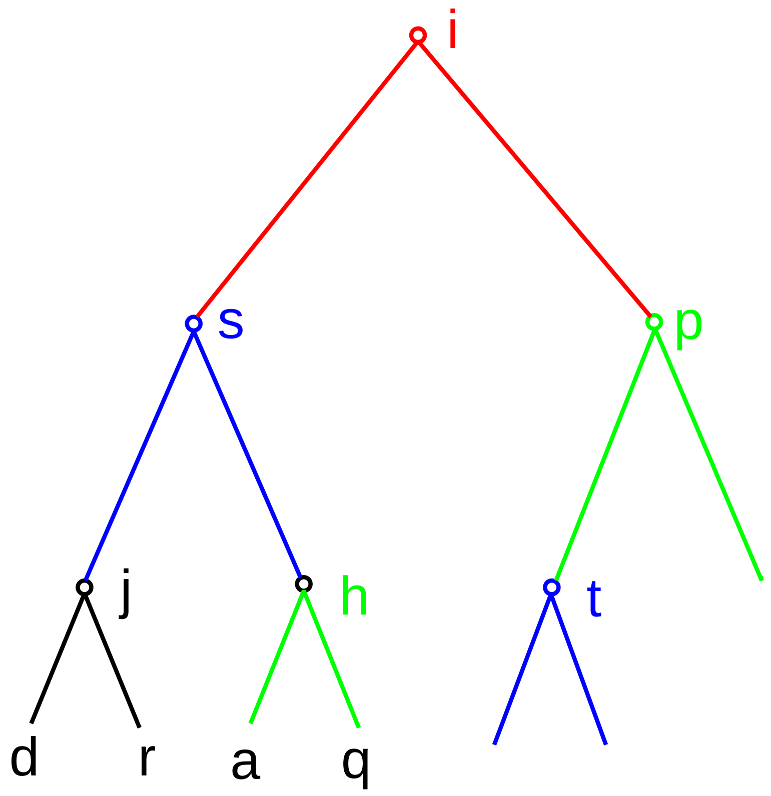
vp-tree Example



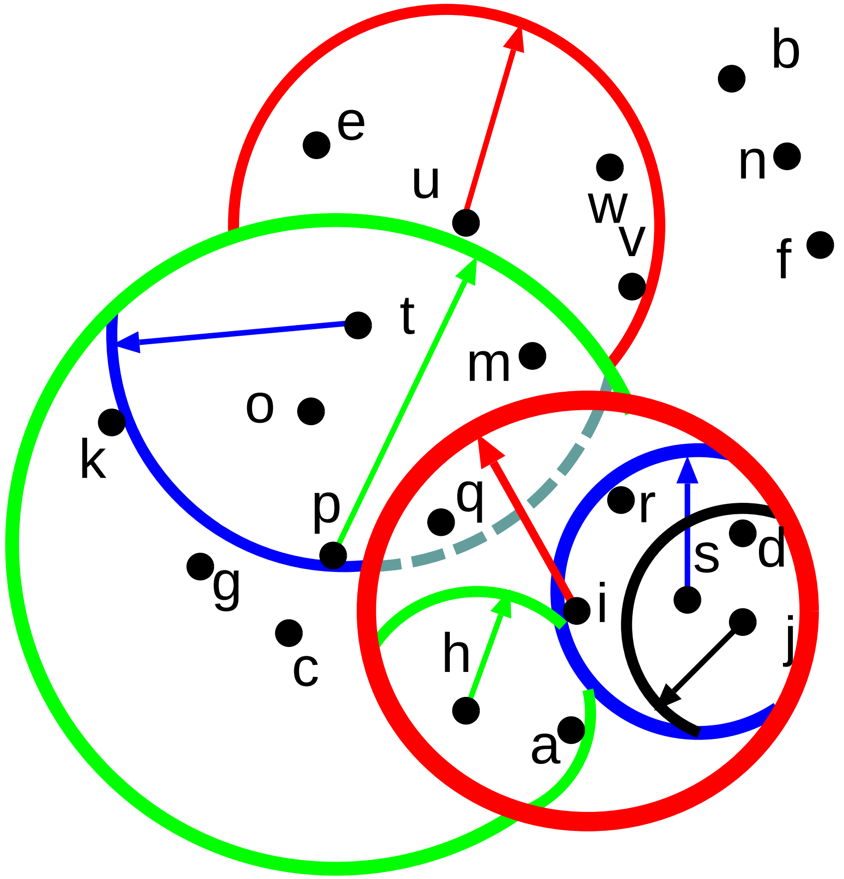


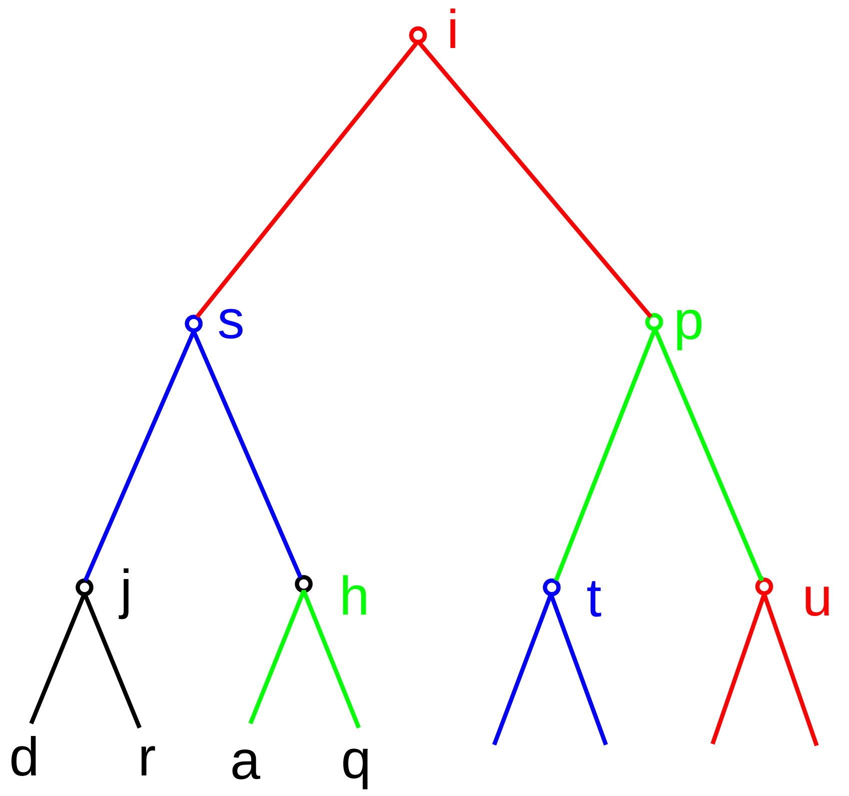
vp-tree Example



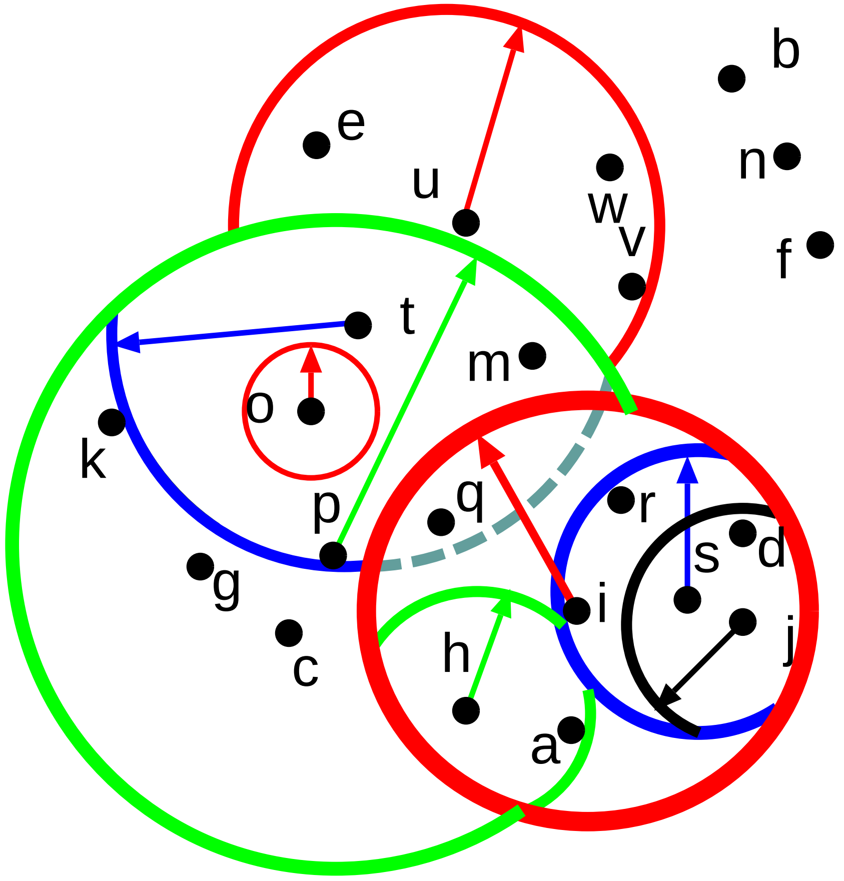


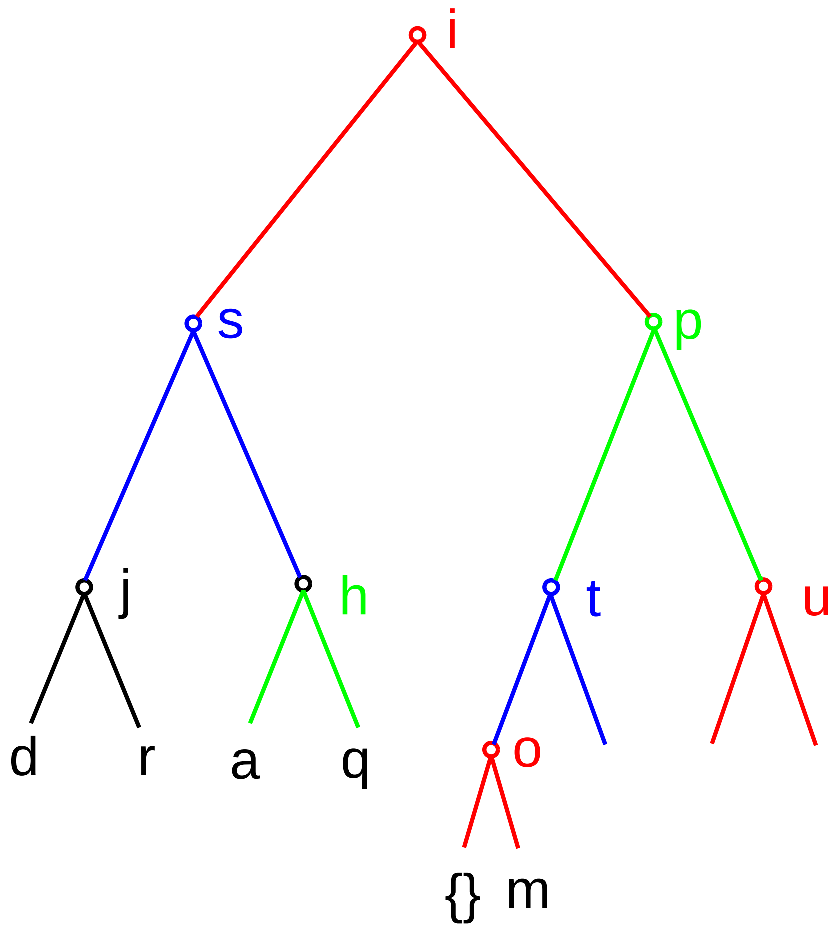
vp-tree Example



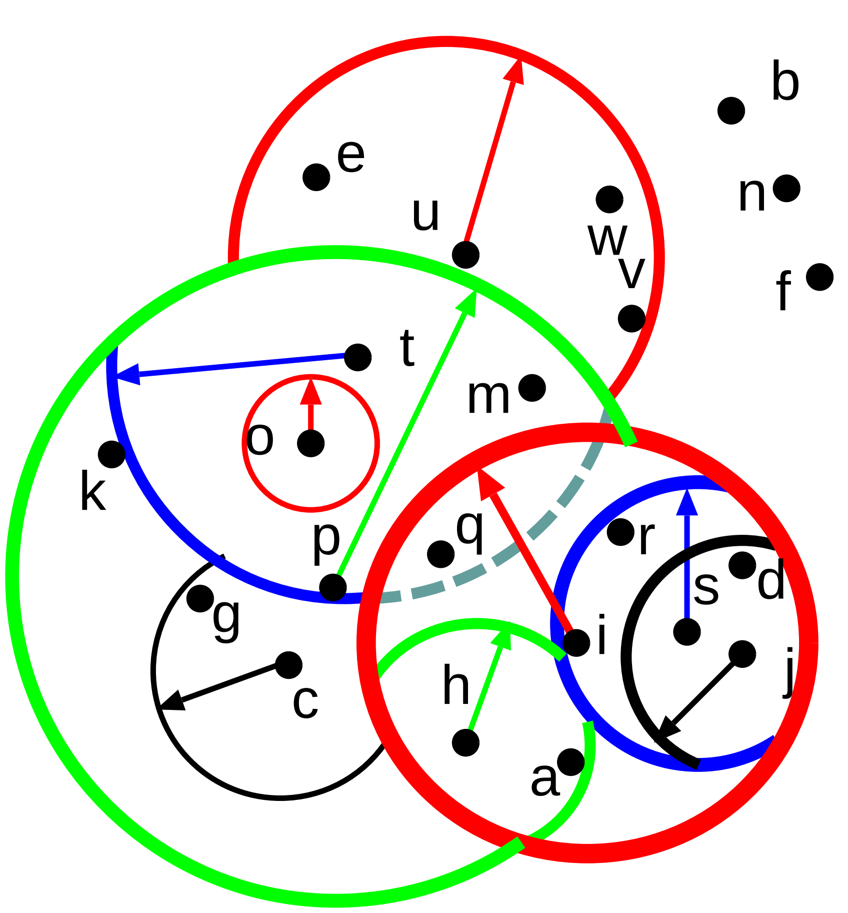


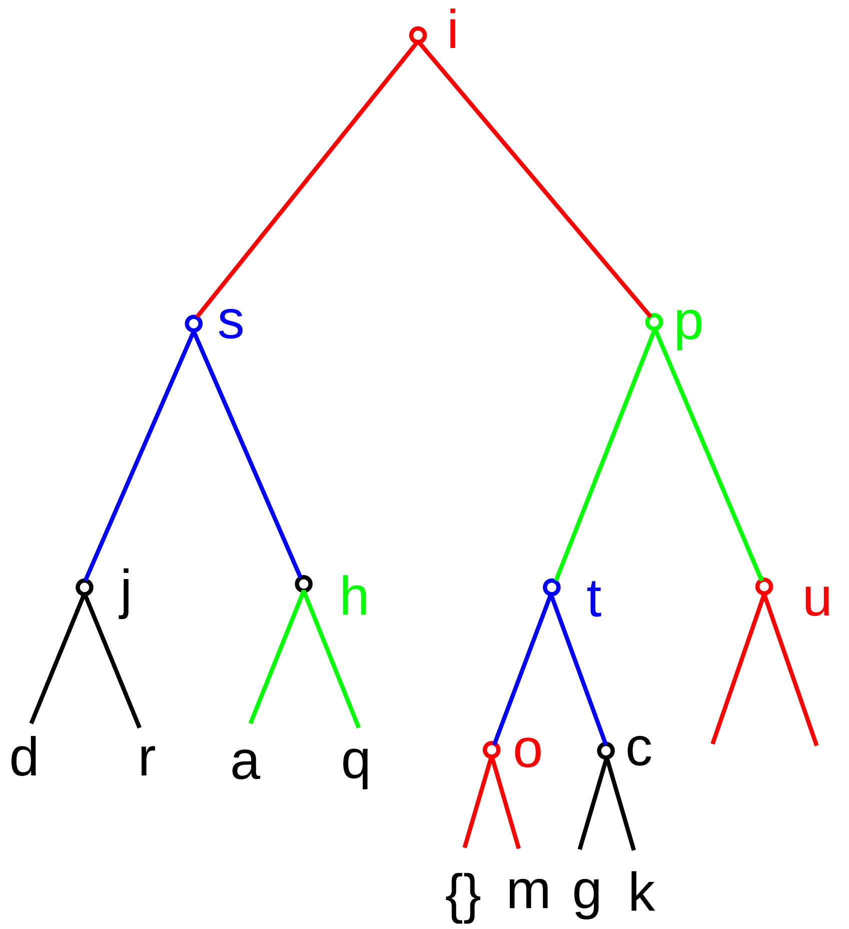
vp-tree Example



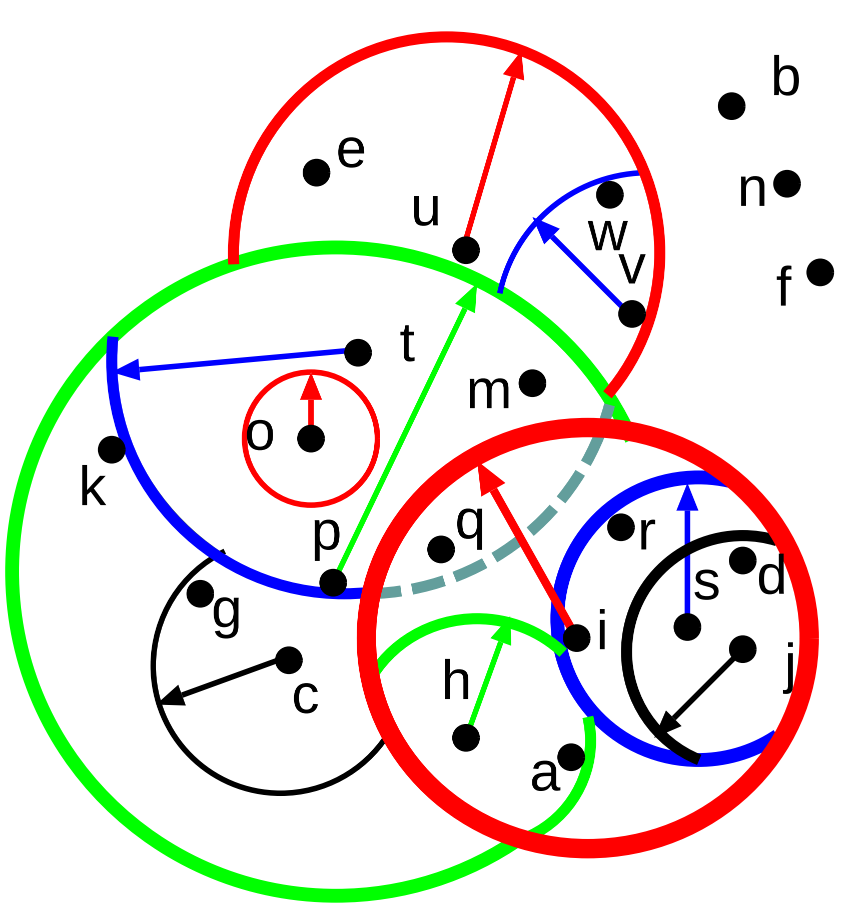


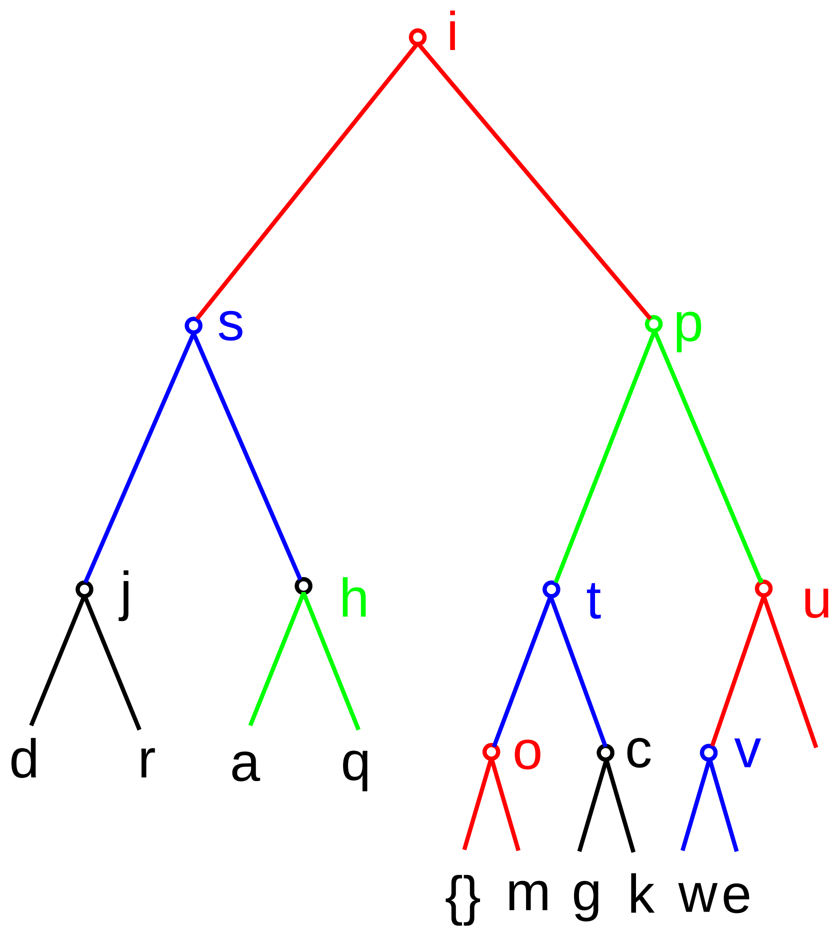
vp-tree Example



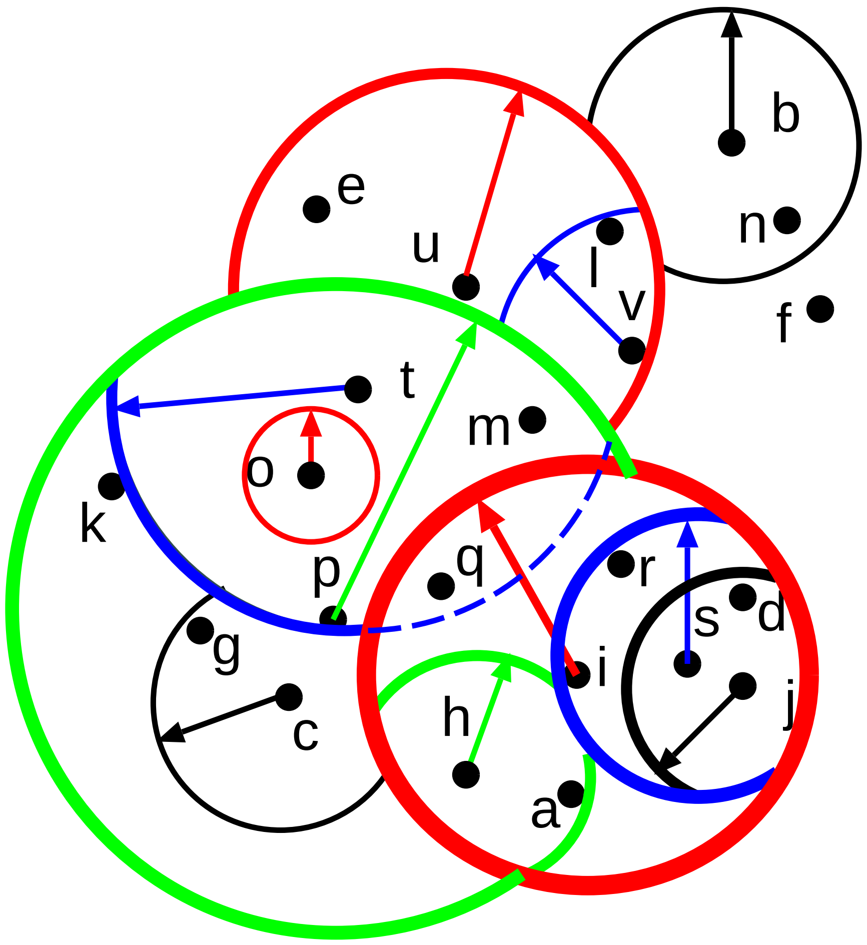


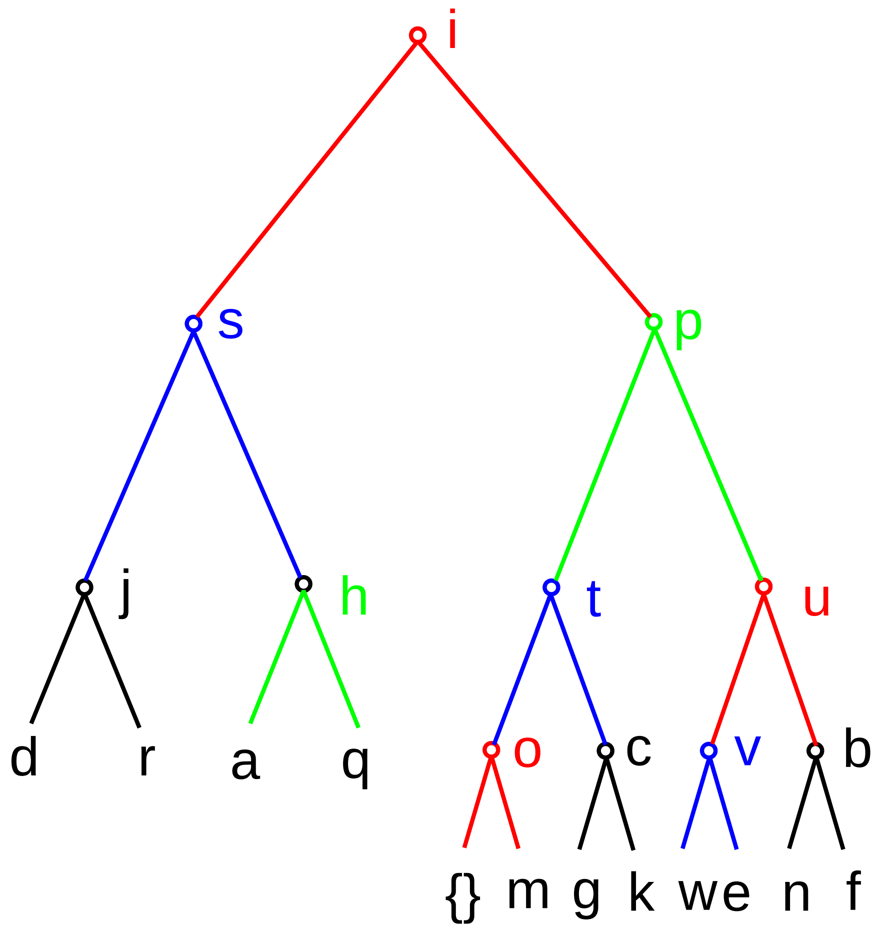
vp-tree Example





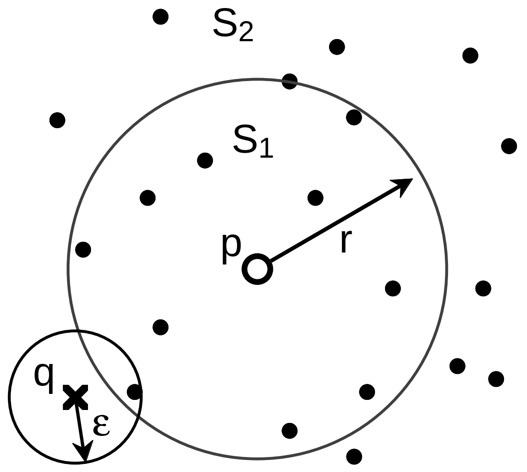
vp-tree Example





Range Searching with vp-tree

- Find all objects o such that $d(q,o) \leq \epsilon$

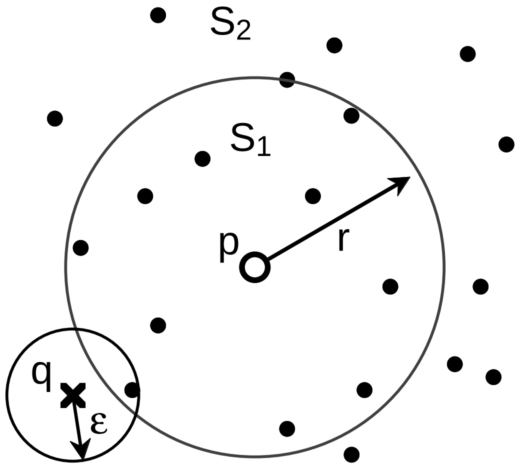


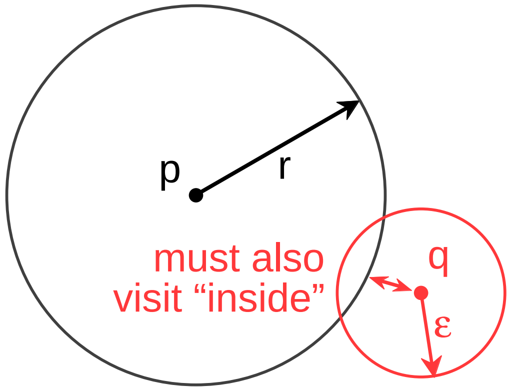
- Use Lemma 2 as know distance from pivot and bounds on the ranges in the two subtrees

$$\max\{d(q,p) - r_{\text{hi}}, r_{\text{lo}} - d(q,p), 0\} \leq d(q,o) \leq d(q,p) + r_{\text{hi}}$$

Range Searching with vp-tree

- Find all objects o such that $d(q,o) \leq \epsilon$





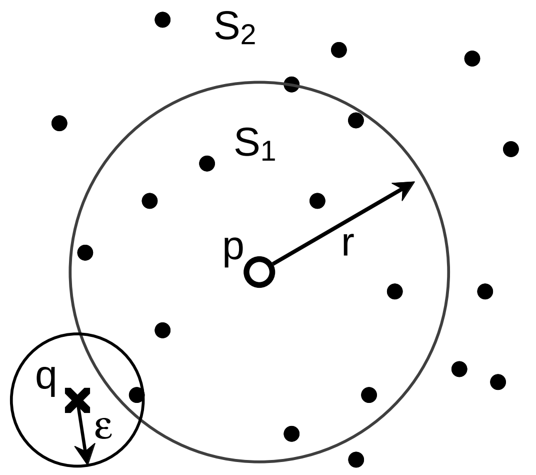
- Use Lemma 2 as know distance from pivot and bounds on the ranges in the two subtrees

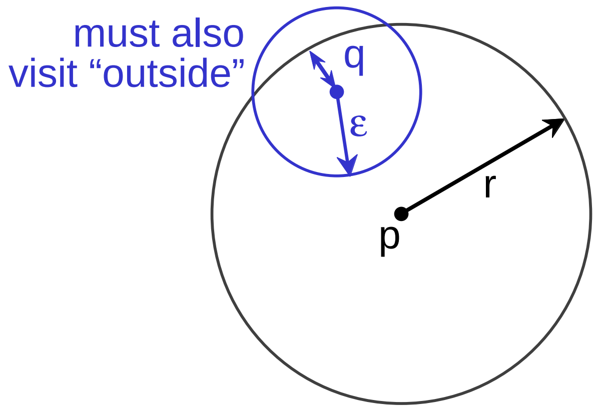
$$\max\{d(q,p) - r_{\text{hi}}, r_{\text{lo}} - d(q,p), 0\} \leq d(q,o) \leq d(q,p) + r_{\text{hi}}$$

1. visit left subtree iff $d(q,p) - r \leq \epsilon \Rightarrow d(q,p) \leq r + \epsilon$
   - $r_{\text{lo}} = 0$ and $r_{\text{hi}} = r$

Range Searching with vp-tree

- Find all objects o such that $d(q,o) \leq \epsilon$





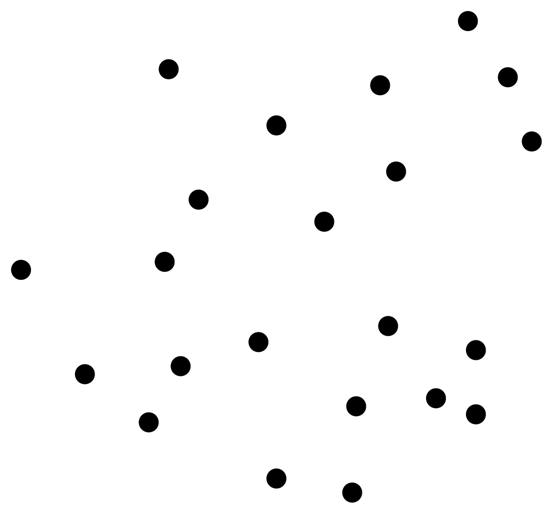
- Use Lemma 2 as know distance from pivot and bounds on the ranges in the two subtrees

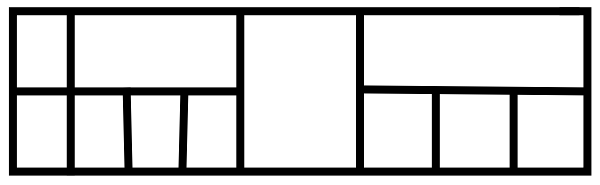
$$\max\{d(q,p) - r_{\mathrm{hi}}, r_{\mathrm{lo}} - d(q,p), 0\} \leq d(q,o) \leq d(q,p) + r_{\mathrm{hi}}$$

1. visit left subtree iff $d(q,p) - r \leq \epsilon \Rightarrow d(q,p) \leq r + \epsilon$
   - $r_{\mathrm{lo}} = 0$ and $r_{\mathrm{hi}} = r$
2. visit right subtree iff $r - d(q,p) \leq \epsilon \Rightarrow d(q,p) \geq r - \epsilon$
   - $r_{\mathrm{lo}} = r$ and $r_{\mathrm{hi}} = \infty$

Increasing Fanout in vp-tree

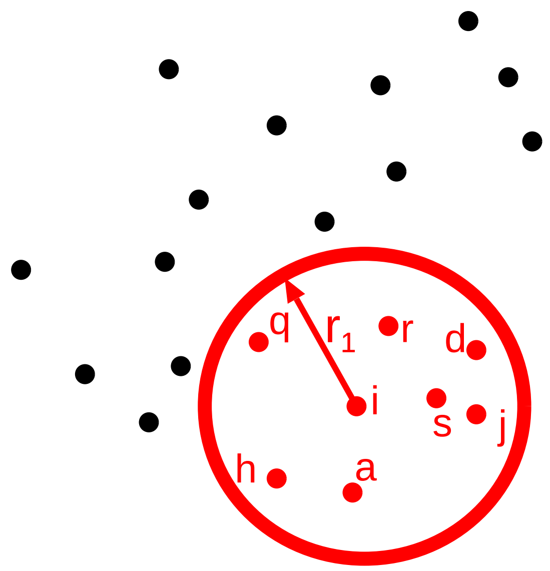
- Fanout of a node in vp-tree is low
- Options
  1. increase fanout by splitting S into m equal-sized subsets based on $m+1$ bounding values $r_0, \ldots, r_m$ or even let $r_0 = 0$ and $r_m = \infty$
  2. mvp-tree
     - each node is equivalent to collapsing nodes at several levels of vp-tree
     - use same pivot for each subtree at a level although the ball radius values differ
     - rationale: only need one distance computation per level to visit all nodes at the level (useful when search backtracks)

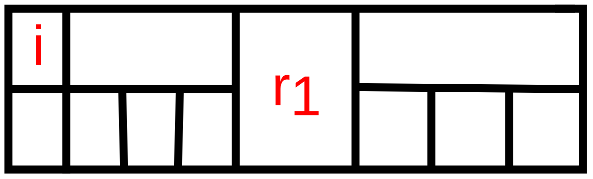




Increasing Fanout in vp-tree

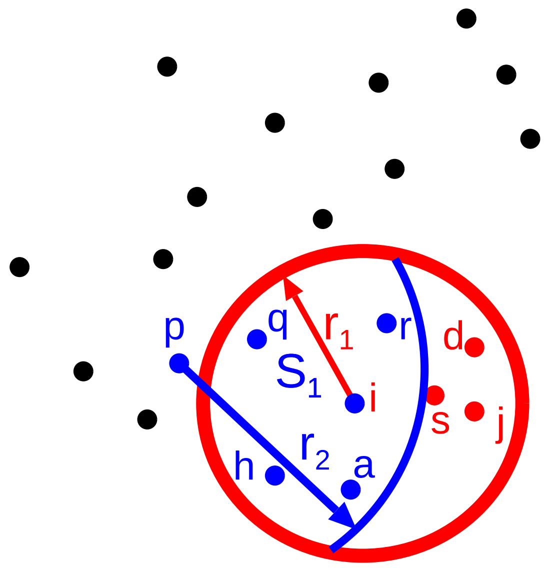
- Fanout of a node in vp-tree is low
- Options
  1. increase fanout by splitting S into m equal-sized subsets based on $m+1$ bounding values $r_0,\ldots,r_m$ or even let $r_0 = 0$ and $r_m = \infty$
  2. mvp-tree
     - each node is equivalent to collapsing nodes at several levels of vp-tree
     - use same pivot for each subtree at a level although the ball radius values differ
     - rationale: only need one distance computation per level to visit all nodes at the level (useful when search backtracks)
       a. first pivot i partitions into ball of radius r_1

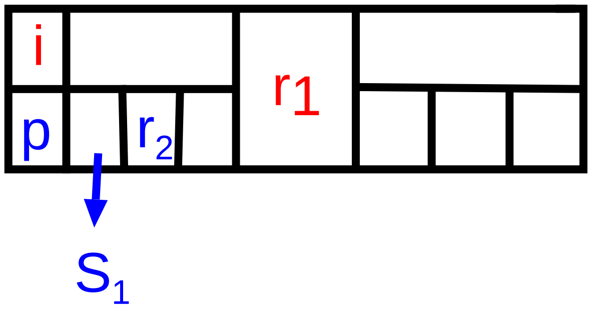




Increasing Fanout in vp-tree

- Fanout of a node in vp-tree is low
- Options
  1. increase fanout by splitting S into m equal-sized subsets based on $m+1$ bounding values $r_0,\ldots,r_m$ or even let $r_0=0$ and $r_m=\infty$
  2. mvp-tree
     - each node is equivalent to collapsing nodes at several levels of vp-tree
     - use same pivot for each subtree at a level although the ball radius values differ
     - rationale: only need one distance computation per level to visit all nodes at the level (useful when search backtracks)
       a. first pivot i partitions into ball of radius r_1
       b. second pivot p partitions inside of the ball for i into subsets S_1



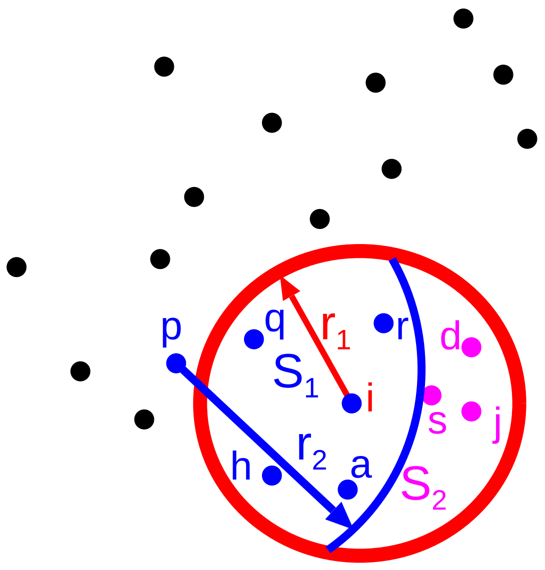


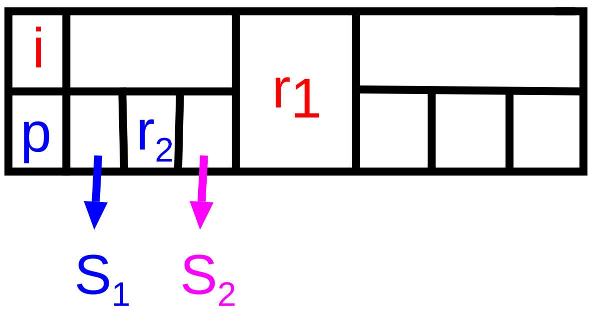
S_1 ={a,h,q,t}

Increasing Fanout in vp-tree

- Fanout of a node in vp-tree is low
- Options
  1. increase fanout by splitting S into m equal-sized subsets based on $m+1$ bounding values $r_0, \ldots, r_m$ or even let $r_0 = 0$ and $r_m = \infty$
  2. mvp-tree

     - each node is equivalent to collapsing nodes at several levels of vp-tree
     - use same pivot for each subtree at a level although the ball radius values differ
     - rationale: only need one distance computation per level to visit all nodes at the level (useful when search backtracks)
       a. first pivot i partitions into ball of radius r_1
       b. second pivot p partitions inside of the ball for i into subsets S_1 and S_2





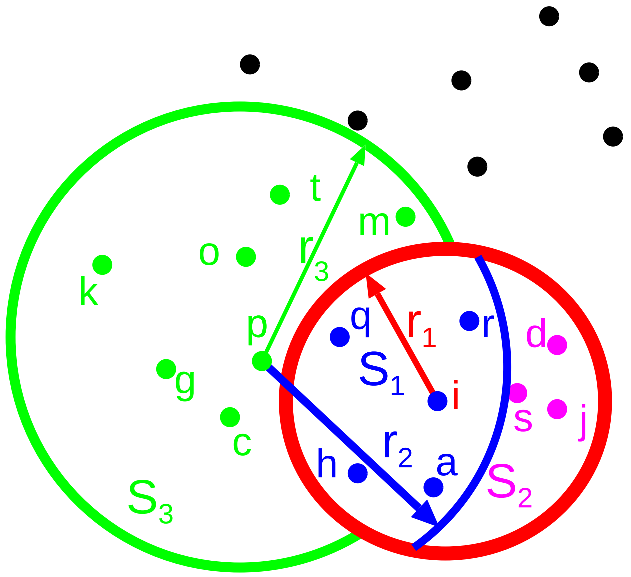
S_1 ={a,h,q,t}

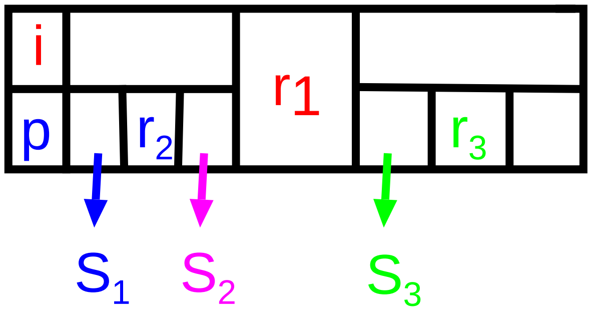
S_2 ={d,j,s}

Increasing Fanout in vp-tree

- Fanout of a node in vp-tree is low
- Options
  1. increase fanout by splitting S into m equal-sized subsets based on $m+1$ bounding values $r_0, \ldots, r_m$ or even let $r_0 = 0$ and $r_m = \infty$
  2. mvp-tree

     - each node is equivalent to collapsing nodes at several levels of vp-tree
     - use same pivot for each subtree at a level although the ball radius values differ
     - rationale: only need one distance computation per level to visit all nodes at the level (useful when search backtracks)
       a. first pivot i partitions into ball of radius r_1
       b. second pivot p partitions inside of the ball for i into subsets S_1 and S_2 , and outside of the ball for i into subsets S_3



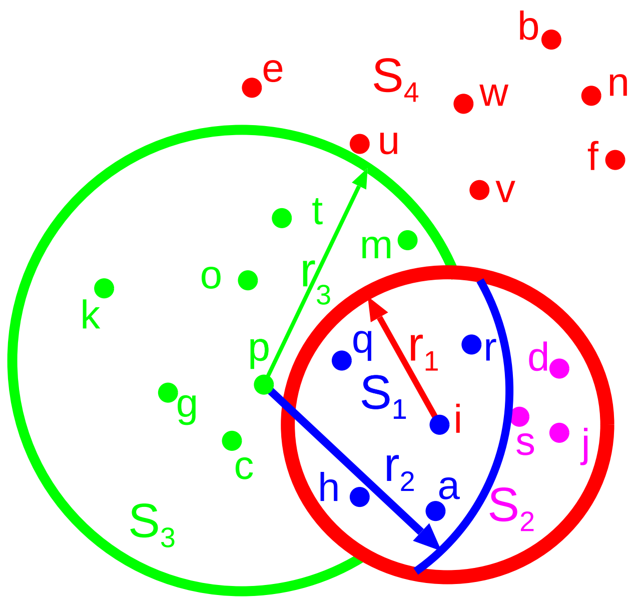


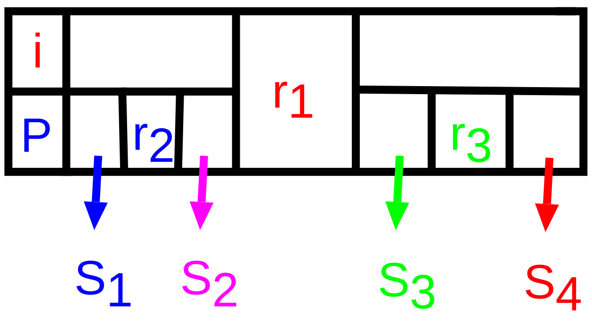
S_1 ={a,h,q,t}
S_2 ={d,j,s}
S_3 ={c,g,k,m,o,t}

Increasing Fanout in vp-tree

- Fanout of a node in vp-tree is low
- Options
  1. increase fanout by splitting S into m equal-sized subsets based on $m+1$ bounding values $r_0, \ldots, r_m$ or even let $r_0 = 0$ and $r_m = \infty$
  2. mvp-tree

  - each node is equivalent to collapsing nodes at several levels of vp-tree
  - use same pivot for each subtree at a level although the ball radius values differ
  - rationale: only need one distance computation per level to visit all nodes at the level (useful when search backtracks)
    a. first pivot i partitions into ball of radius r_1
    b. second pivot p partitions inside of the ball for i into subsets S_1 and S_2 , and outside of the ball for i into subsets S_3 and S_4





S_1 ={a,h,q,t}

S_2 ={d,j,s}

S_3 ={c,g,k,m,o,t}

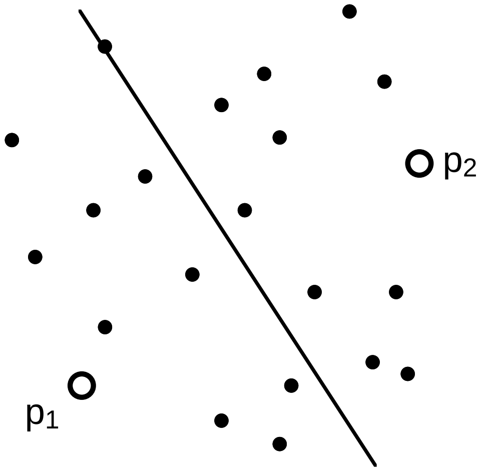
S_4 ={b,e,f,n,u,v,w}

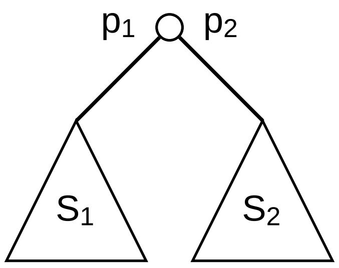
gh-tree (Metric tree; Uhlmann)

- Generalized hyperplane partitioning method
- Pick p_1 and p_2 from S and partition S into two sets S_1 and S_2 where:

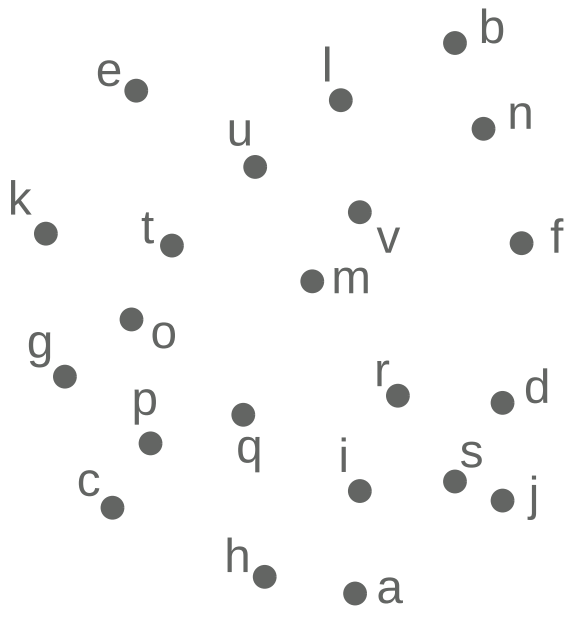
$$\begin{aligned} S_1 &= \{o \in S \setminus \{p_1, p_2\} \mid d(p_1, o) \leq d(p_2, o)\} \\ S_2 &= \{o \in S \setminus \{p_1, p_2\} \mid d(p_2, o) < d(p_1, o)\} \end{aligned}$$

- Objects in S_1 are closer to p_1 than to p_2 (or equidistant from both), and objects in S_2 are closer to p_2 than to p_1
  - hyperplane corresponds to all points o satisfying $d(p_1, o) = d(p_2, o)$
  - can also "move" hyperplane, by using $d(p_1, o) = d(p_2, o) + m$
- Apply recursively, yielding a binary tree with two pivots at internal nodes





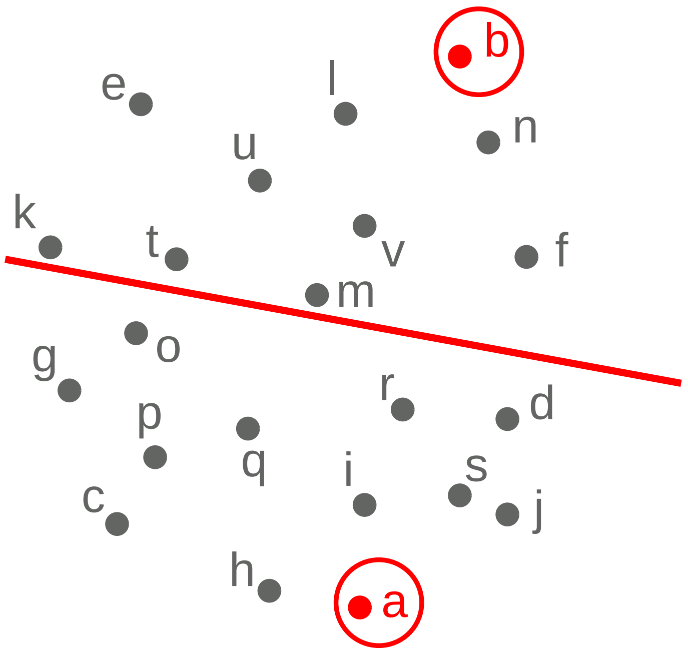
gh-tree Example



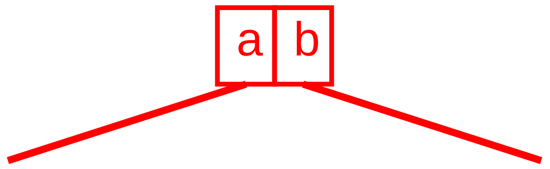
(a)

(b)

gh-tree Example

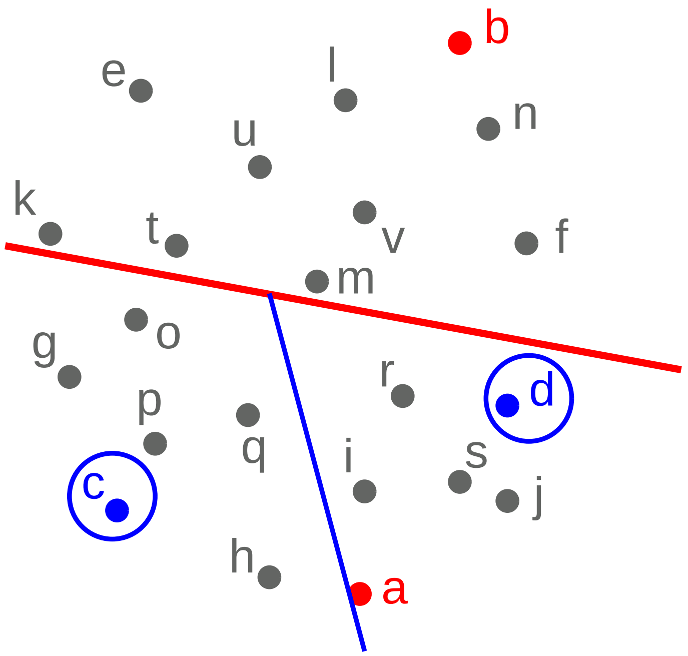


(a)

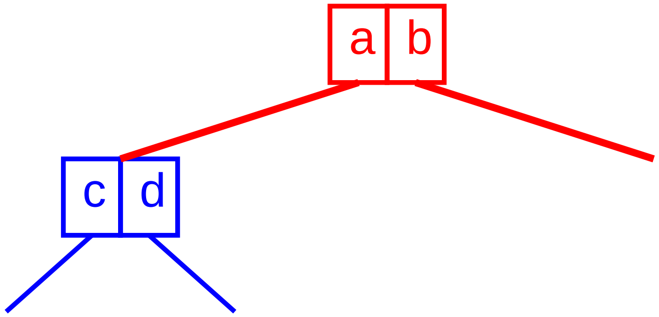


(b)

gh-tree Example

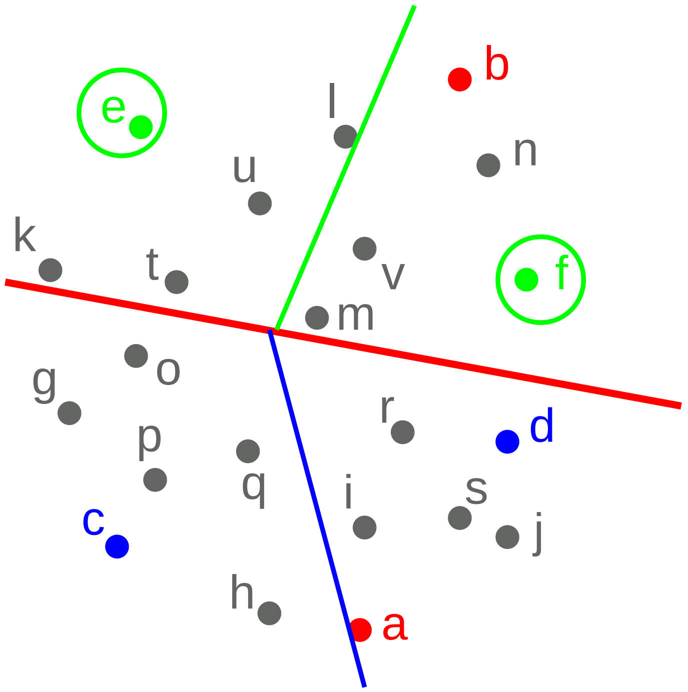


(a)

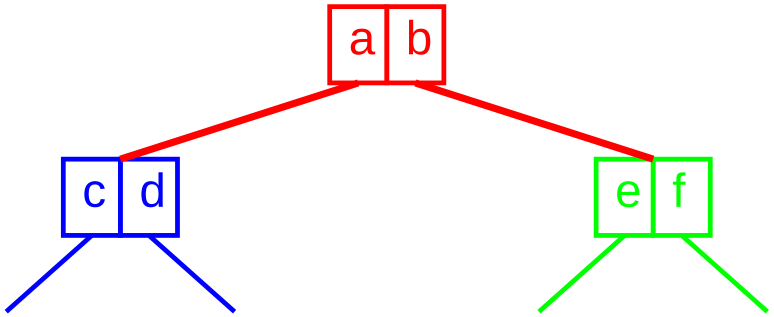


(b)

gh-tree Example

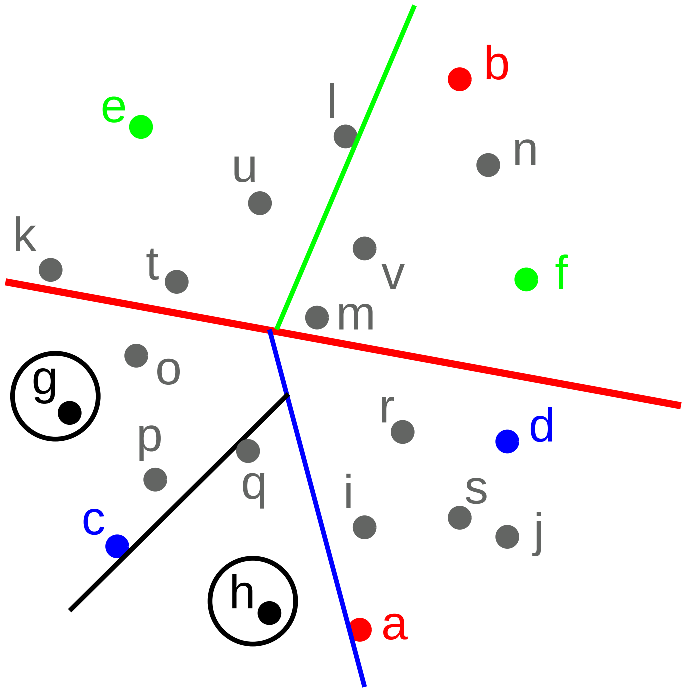


(a)

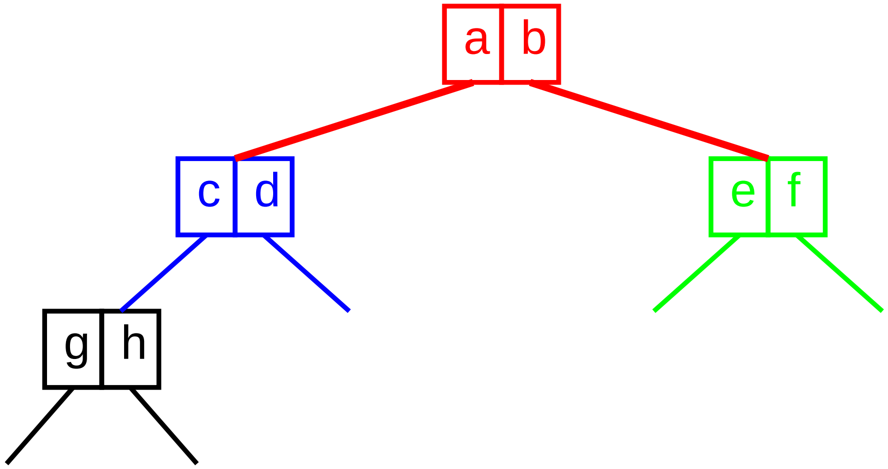


(b)

gh-tree Example

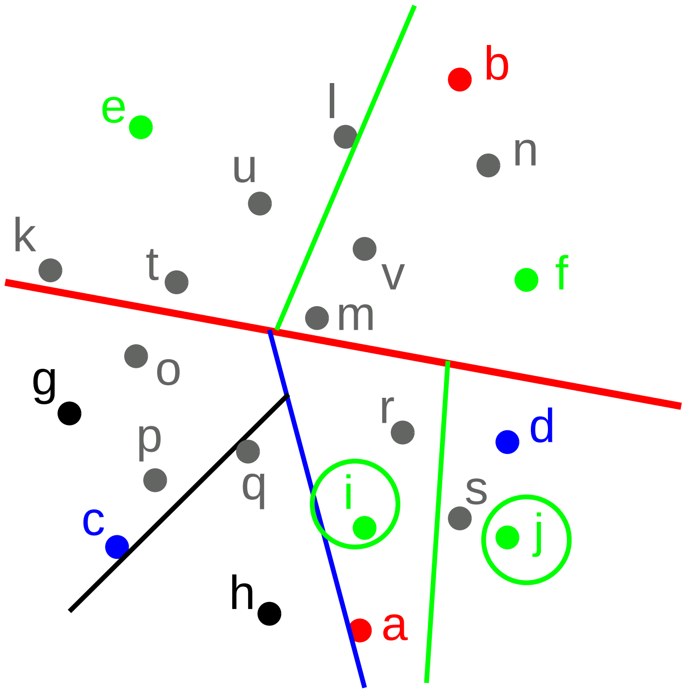


(a)

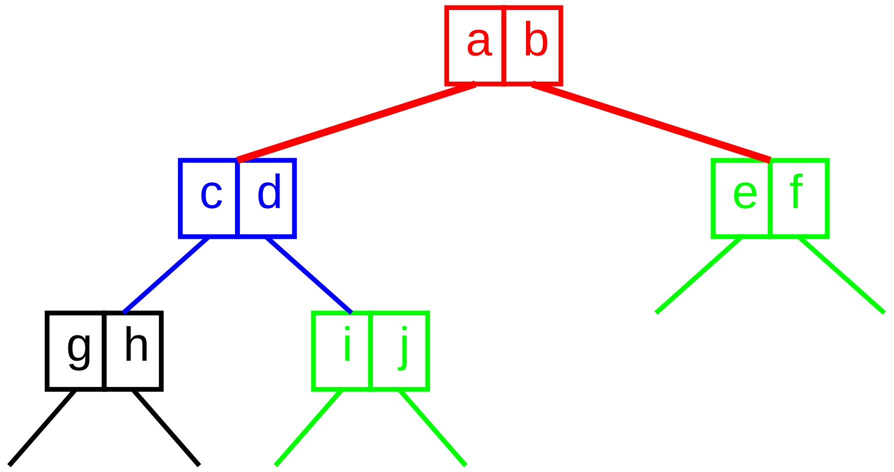


(b)

gh-tree Example

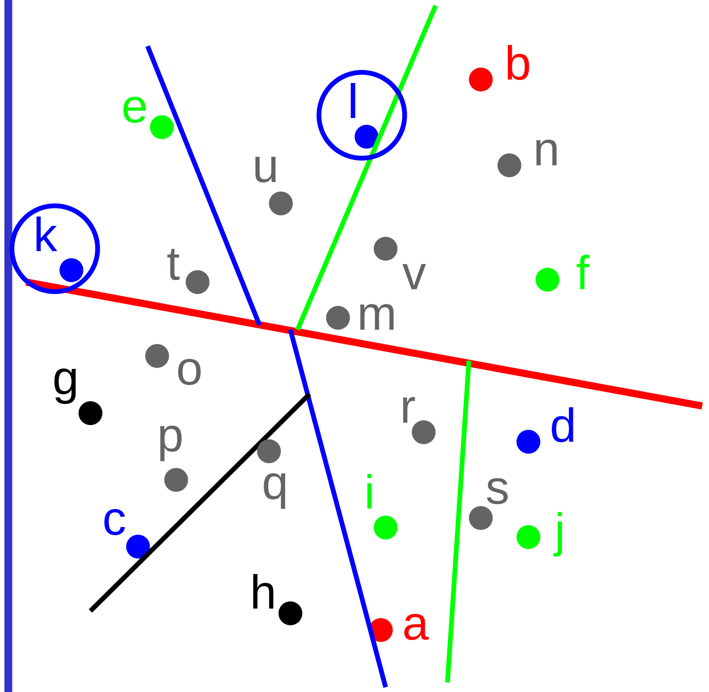


(a)

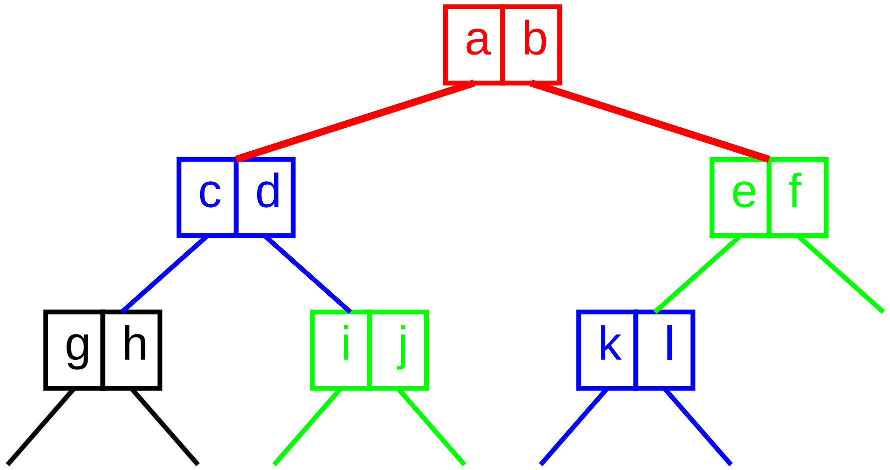


(b)

gh-tree Example

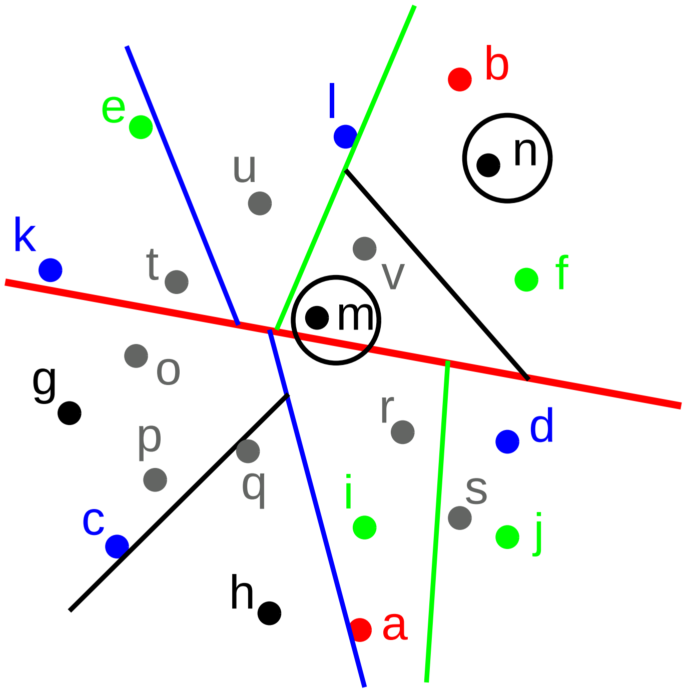


(a)

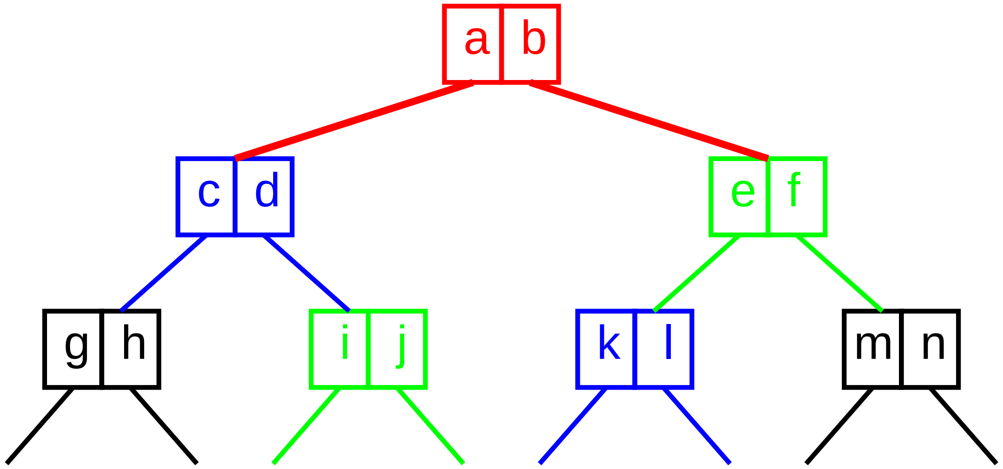


(b)

gh-tree Example

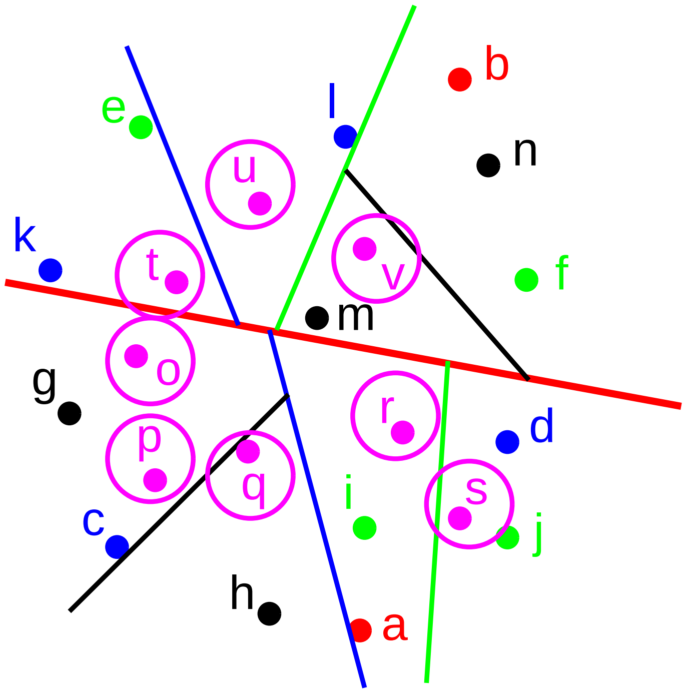


(a)

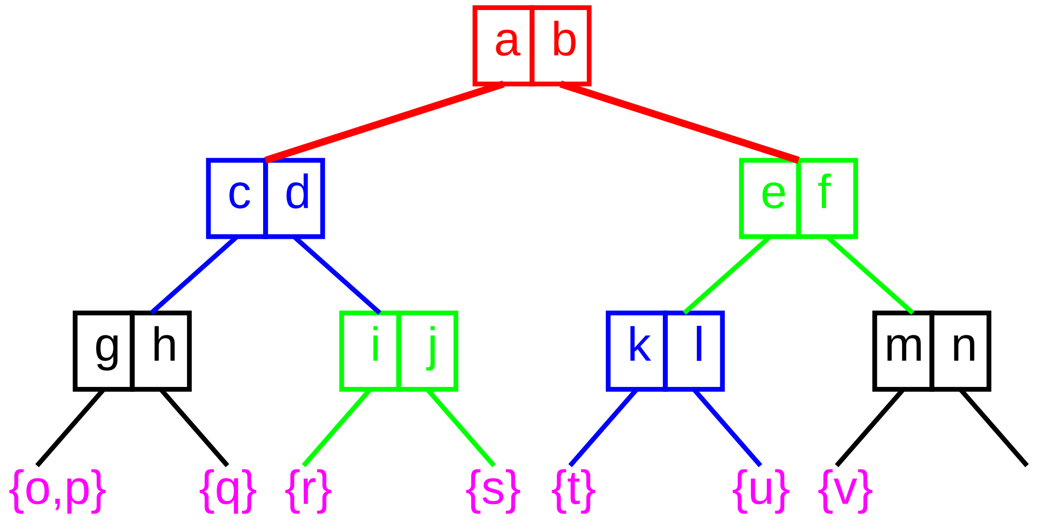


(b)

gh-tree Example

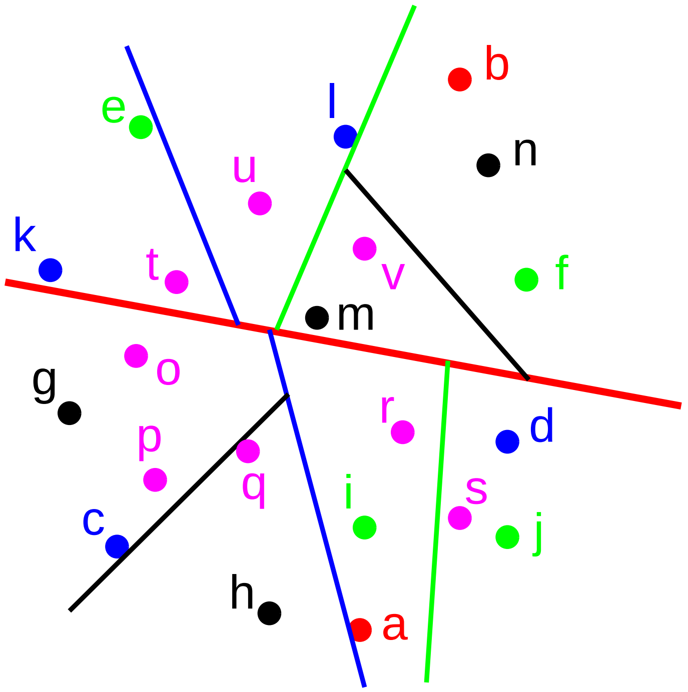


(a)

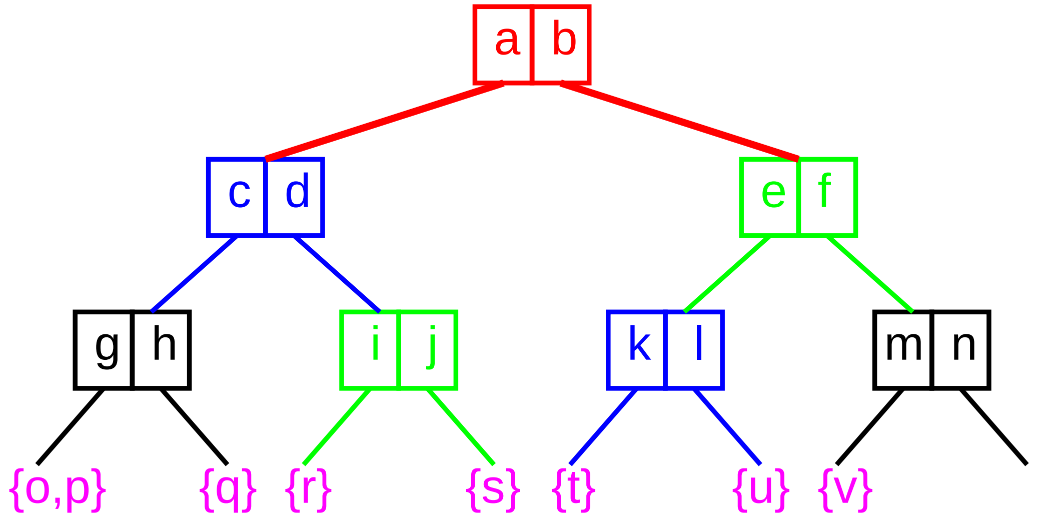


(b)

gh-tree Example



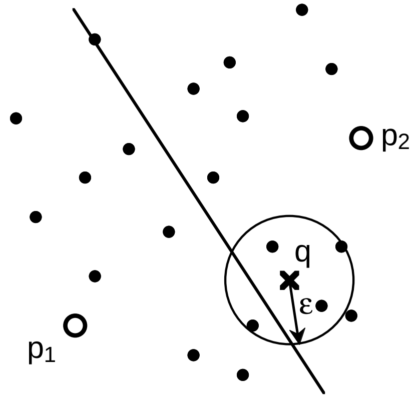
(a)



(b)

Range Searching with gh-tree

- Find all objects o such that $d(q, o) \leq \epsilon$

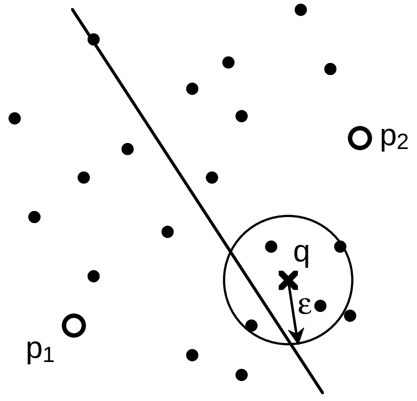


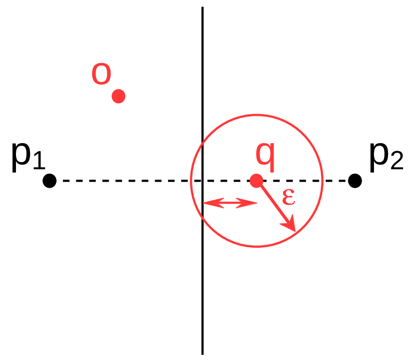
- Lower bound on $d(q, o)$ is distance to hyperplane (or zero)
  - can only use directly in Euclidean spaces
  - otherwise, no direct representation of the “generalized hyperplane”
- But, can use Lemma 4 with distance from pivots

$$\max\left\{\frac{d(q,p_1) - d(q,p_2)}{2}, 0\right\} \leq d(q,o)$$

Range Searching with gh-tree

- Find all objects o such that $d(q,o) \leq \epsilon$





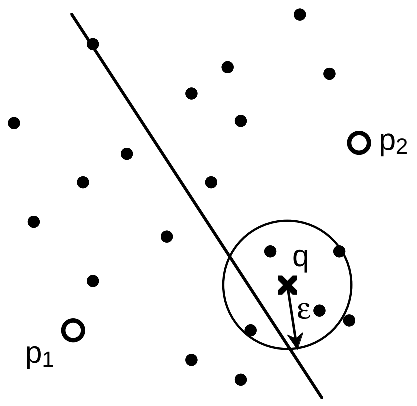
- Lower bound on $d(q,o)$ is distance to hyperplane (or zero)
  - can only use directly in Euclidean spaces
  - otherwise, no direct representation of the “generalized hyperplane”
- But, can use Lemma 4 with distance from pivots

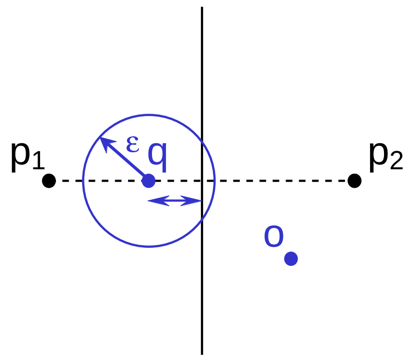
$$\max\left\{\frac{d(q,p_1)-d(q,p_2)}{2},0\right\} \leq d(q,o)$$

1. visit left subtree iff $\frac{d(q,p_1)-d(q,p_2)}{2} \leq \epsilon \Rightarrow d(q,p_1) \leq d(q,p_2)+2\epsilon$

Range Searching with gh-tree

- Find all objects o such that $d(q,o) \leq \epsilon$





- Lower bound on $d(q,o)$ is distance to hyperplane (or zero)
  - can only use directly in Euclidean spaces
  - otherwise, no direct representation of the “generalized hyperplane”
- But, can use Lemma 4 with distance from pivots

$$\max\left\{\frac{d(q,p_1)-d(q,p_2)}{2},0\right\} \leq d(q,o)$$

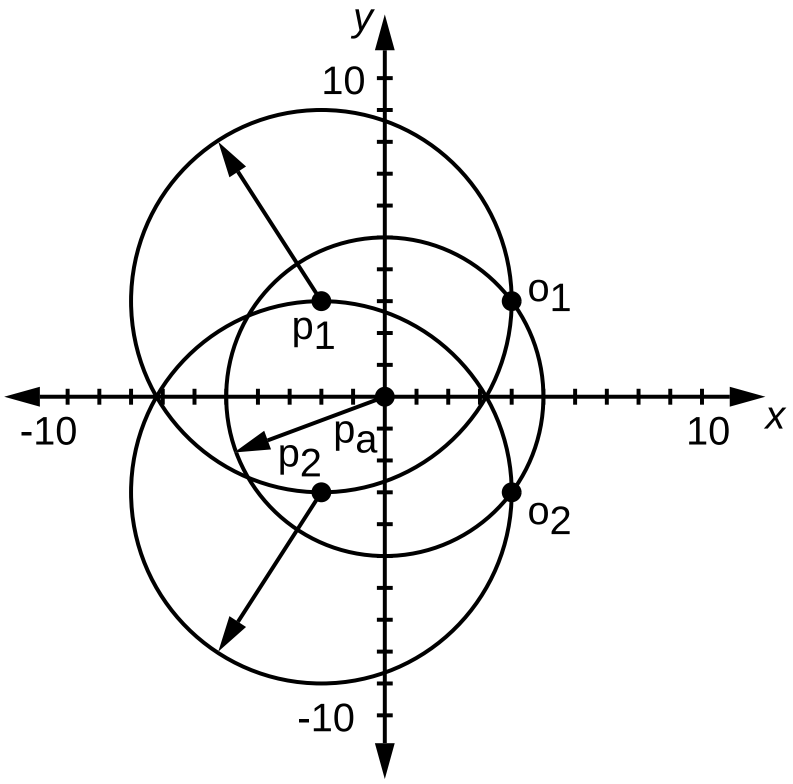
1. visit left subtree iff $\frac{d(q,p_1)-d(q,p_2)}{2} \leq \epsilon \Rightarrow d(q,p_1) \leq d(q,p_2)+2\epsilon$
2. visit right subtree iff $\frac{d(q,p_2)-d(q,p_1)}{2} \leq \epsilon \Rightarrow d(q,p_2) \leq d(q,p_1)+2\epsilon$

Increasing Fanout in gh-tree

- Fanout of a node in gh-tree is low
- Geometric Near-neighbor Access tree (GNAT; Brin)
  1. increase fanout by adding m pivots $P = \{p_1, \ldots, p_m\}$ to split S into $S_1, \ldots, S_m$ based on which of the objects in P is the closest
  2. for any object $o \in S \setminus P$, o is a member of S_i if $d(p_i, o) \leq d(p_j, o)$ for all $j = 1, \ldots, m$
  3. store information about ranges of distances between pivots and objects in the subtrees to facilitate pruning search

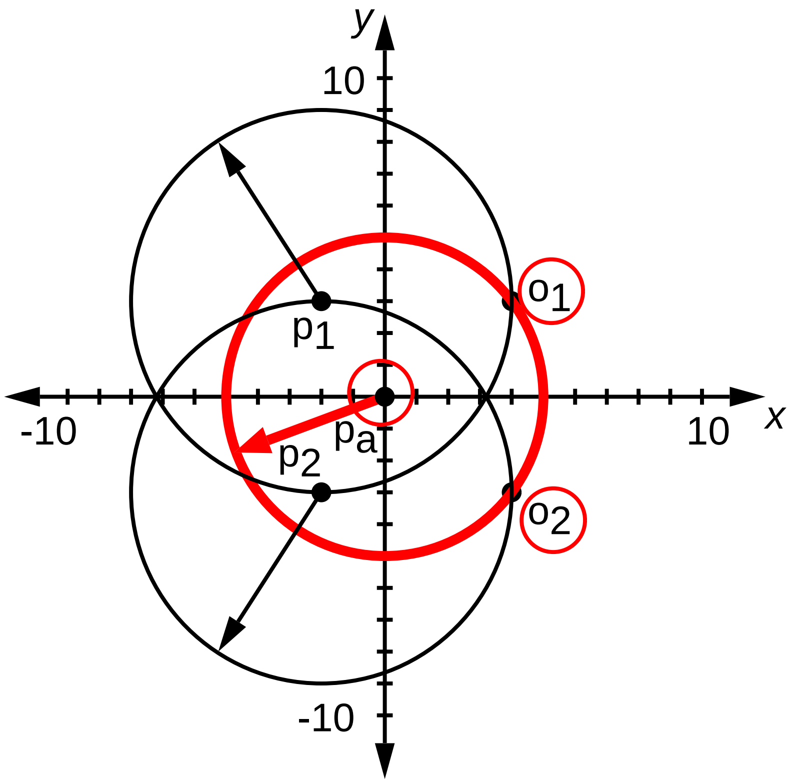
Bisector tree (bs-tree) (Kalantari/McDonald)

1. gh-trees with covering balls
2. Drawback: radius of covering ball of a node is sometimes smaller than the radii of the covering balls of its descendants (termed eccentric)
3. Bad for pruning as ideally we want radii of covering balls to decrease as search descends



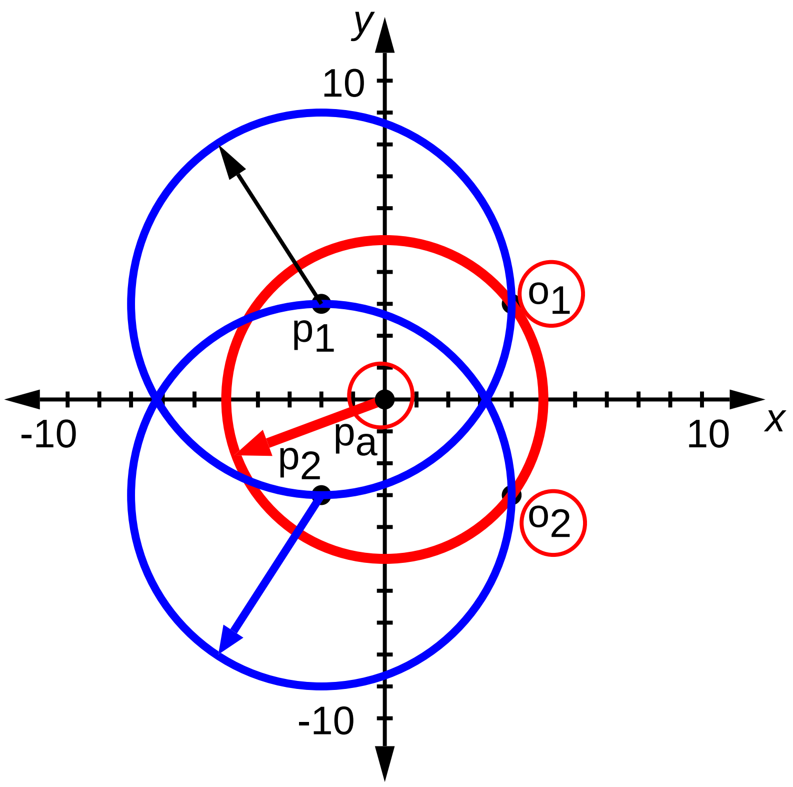
Bisector tree (bs-tree) (Kalantari/McDonald)

1. gh-trees with covering balls
2. Drawback: radius of covering ball of a node is sometimes smaller than the radii of the covering balls of its descendants (termed eccentric)
3. Bad for pruning as ideally we want radii of covering balls to decrease as search descends



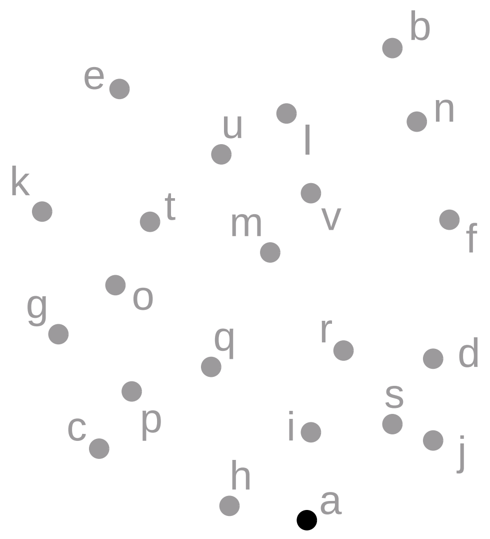
Bisector tree (bs-tree) (Kalantari/McDonald)

1. gh-trees with covering balls
2. Drawback: radius of covering ball of a node is sometimes smaller than the radii of the covering balls of its descendants (termed eccentric)
3. Bad for pruning as ideally we want radii of covering balls to decrease as search descends



mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit

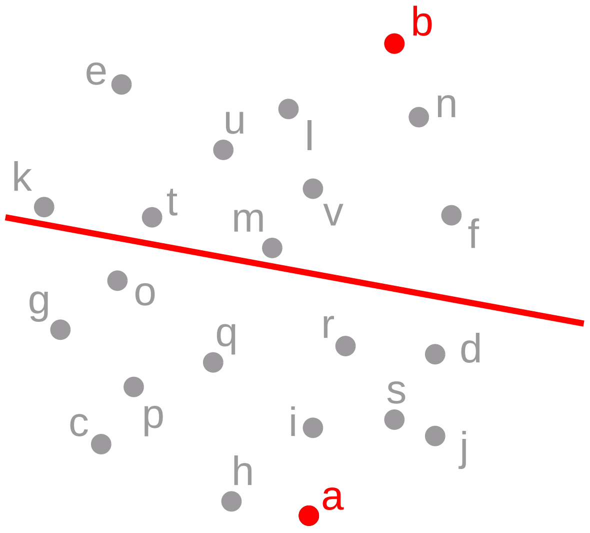


(a)

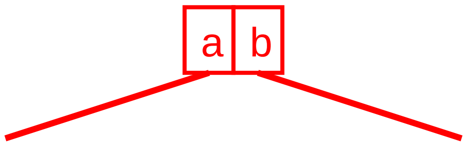
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



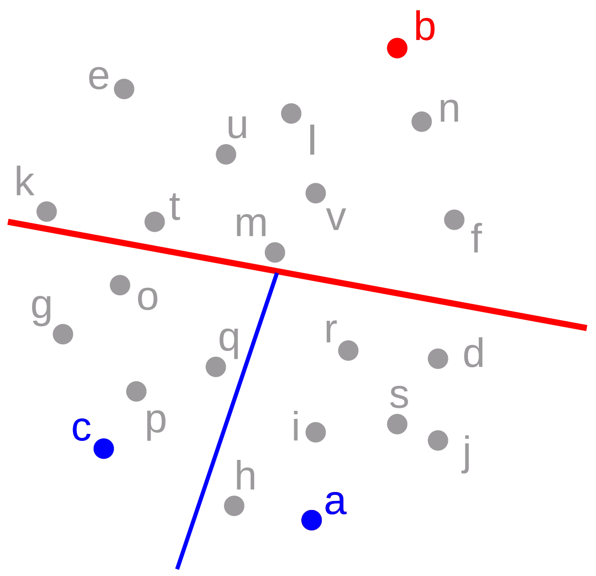
(a)



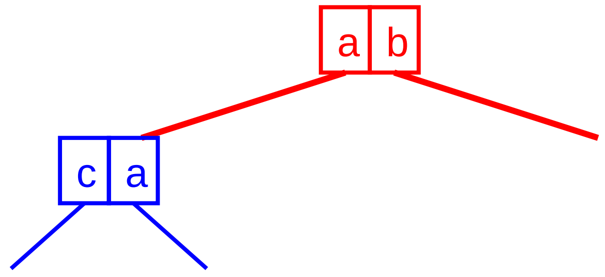
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



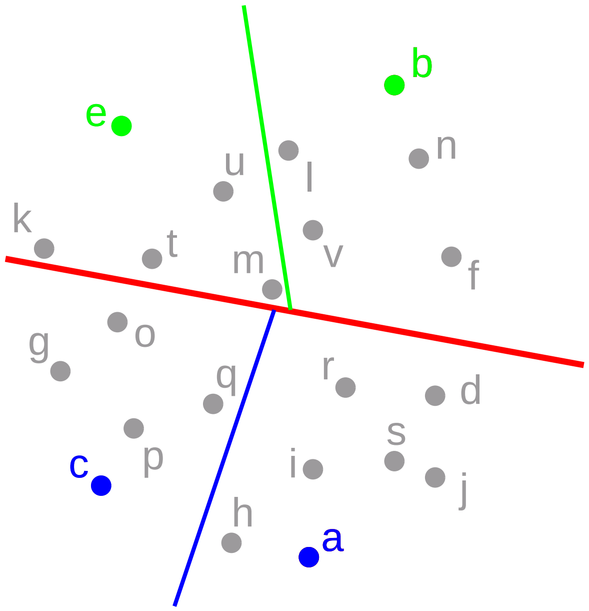
(a)



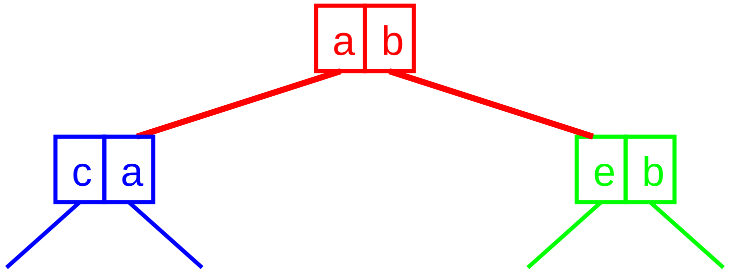
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



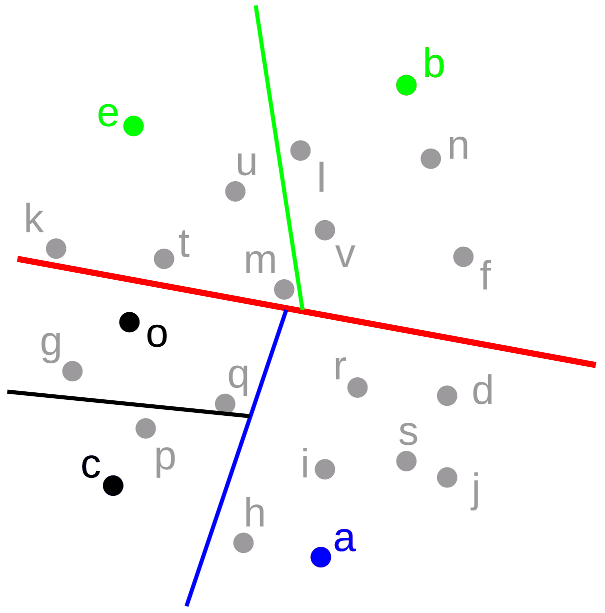
(a)



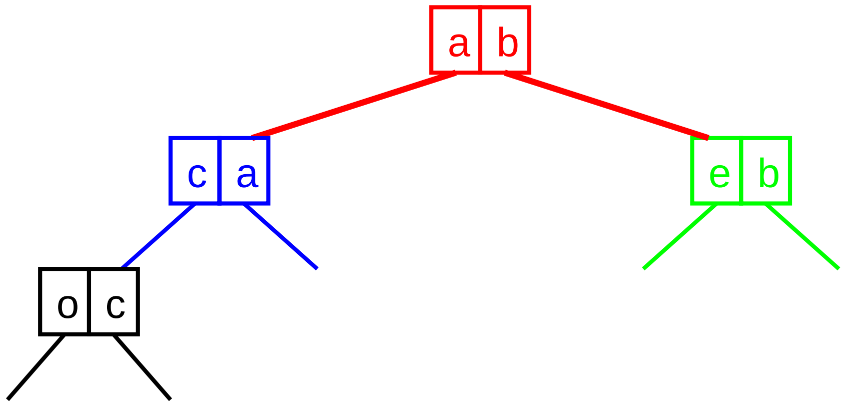
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



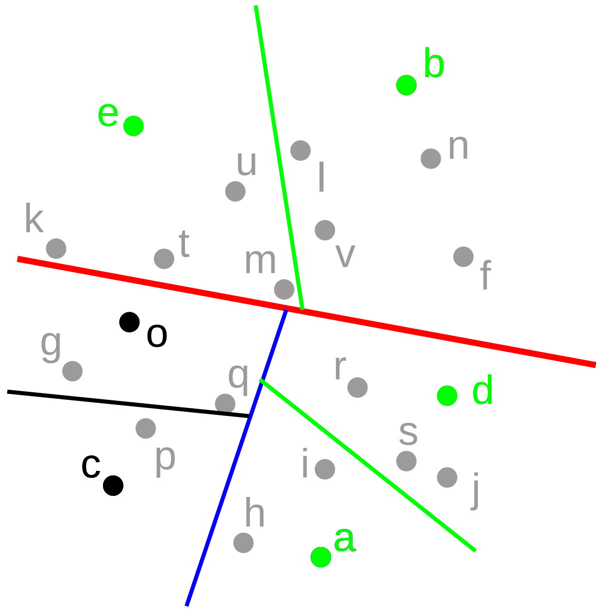
(a)



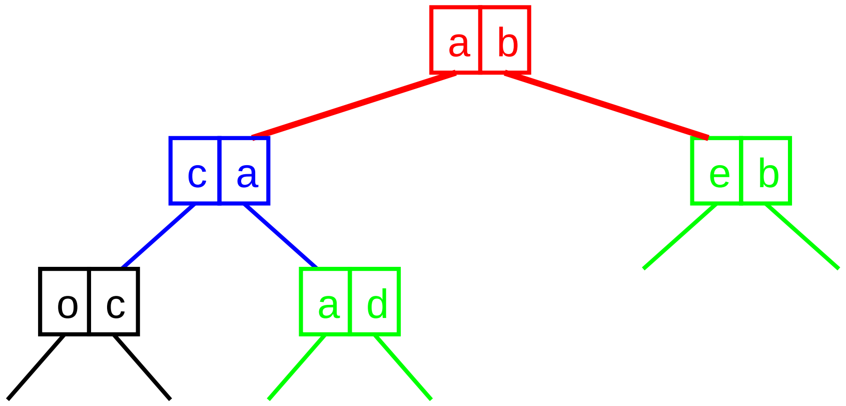
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



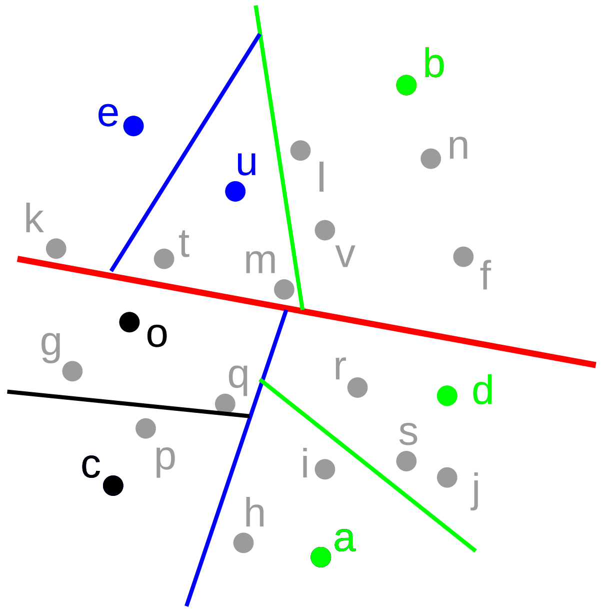
(a)



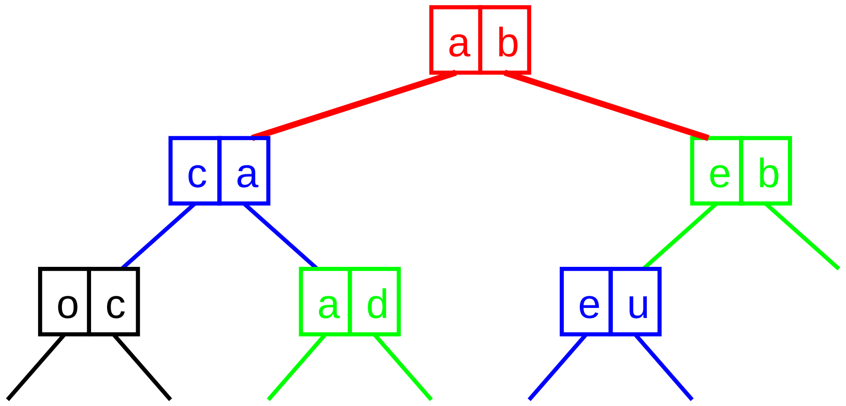
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



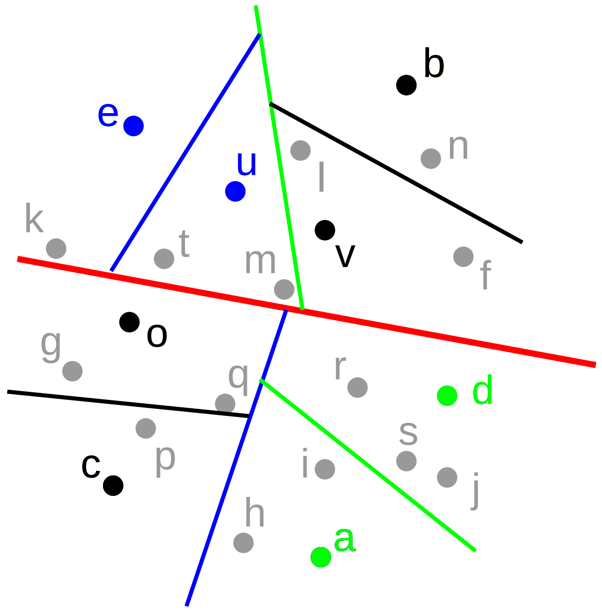
(a)



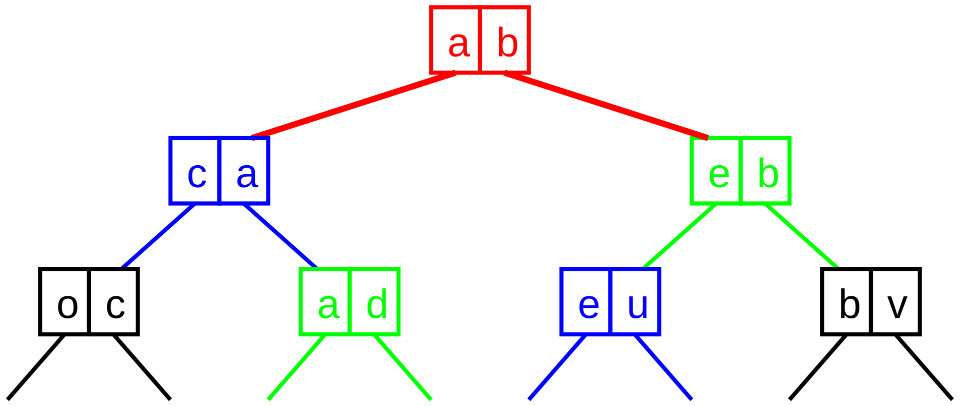
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



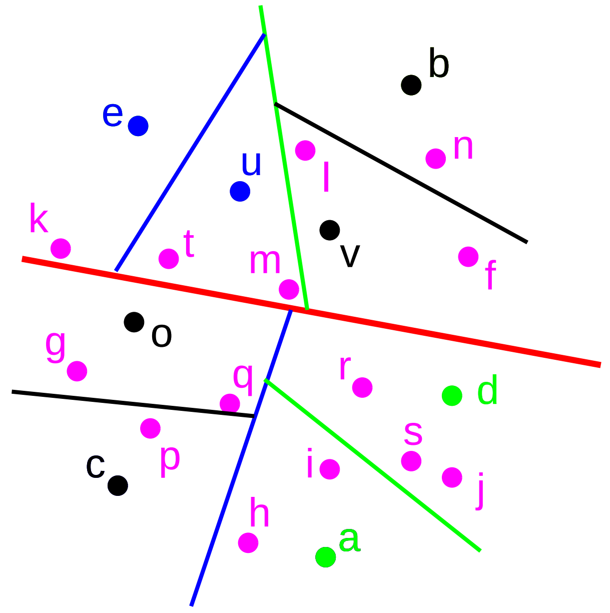
(a)



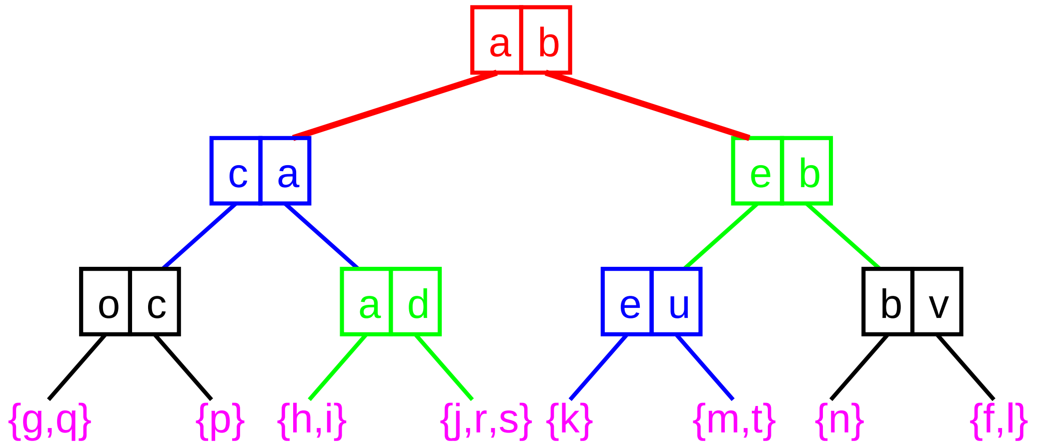
(b)

mb-tree (Dehne/Noltemeier)

1. Inherit one pivot from ancestor node
2. Fewer pivots and fewer distance computations but perhaps deeper tree
3. Like bucket (k) PR k-d tree as split whenever region has $k > 1$ objects but region partitions are implicit (defined by pivot objects) instead of explicit



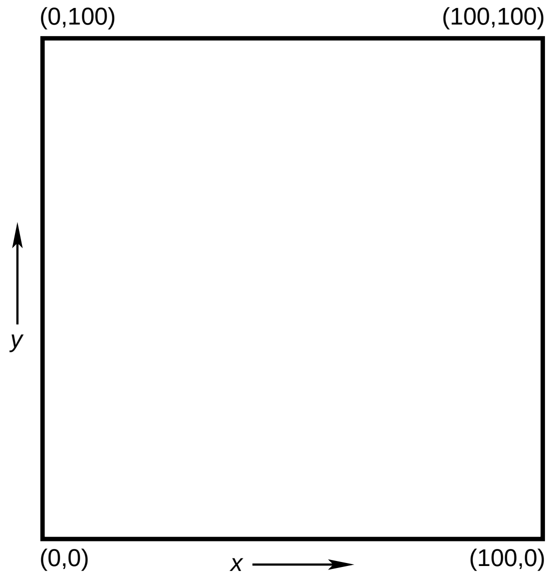
(a)



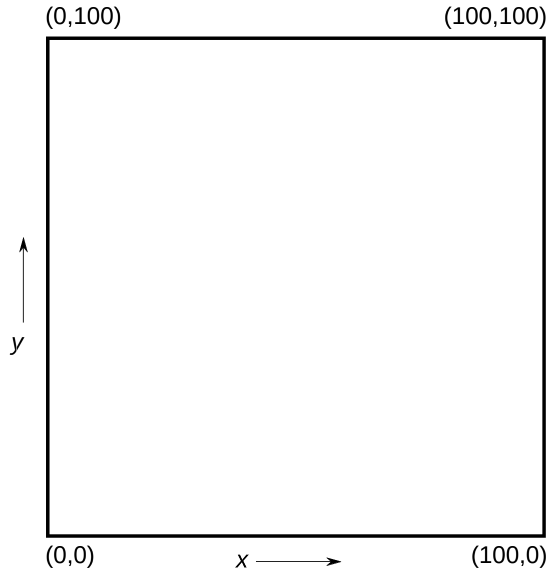
(b)

Comparison of mb-tree (BSP tree) and PR k-d tree

PR k-d tree

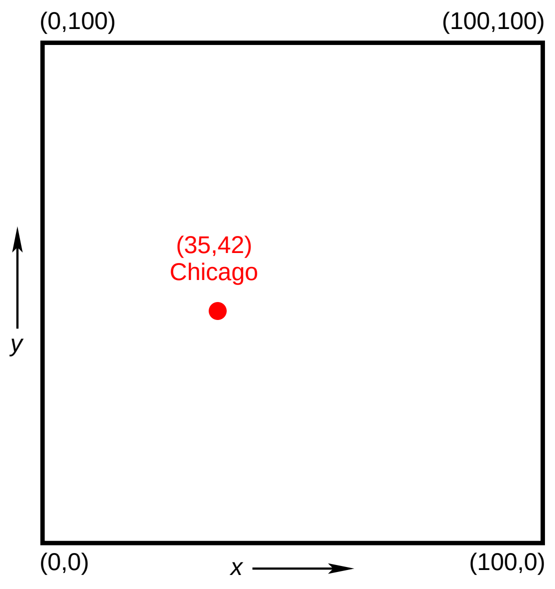


BSP tree

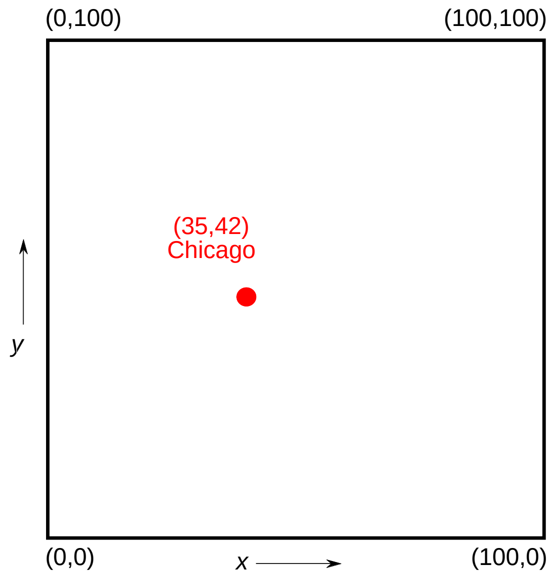


Comparison of mb-tree (BSP tree) and PR k-d tree

PR k-d tree

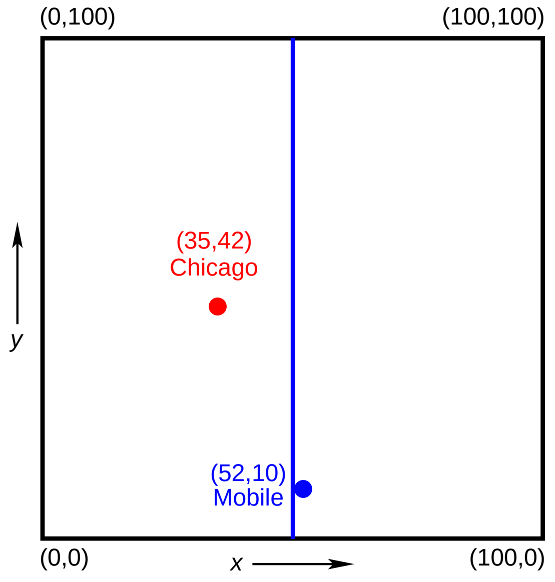


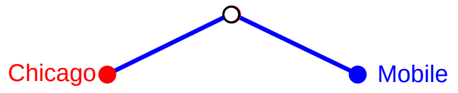
BSP tree



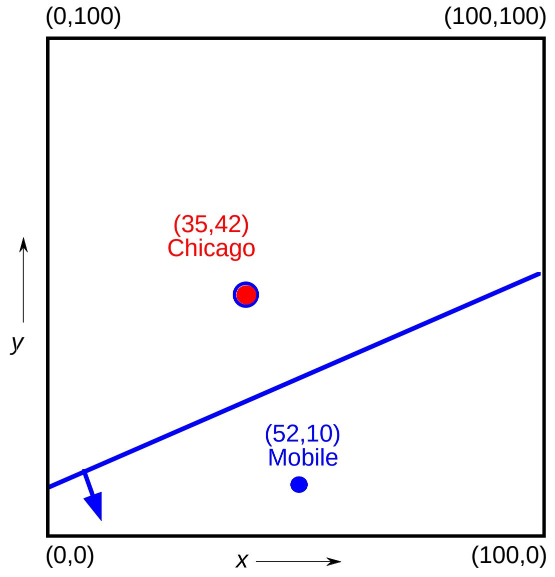
Comparison of mb-tree (BSP tree) and PR k-d tree

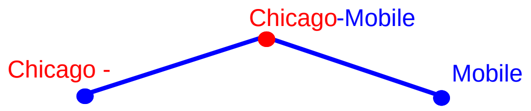
PR k-d tree





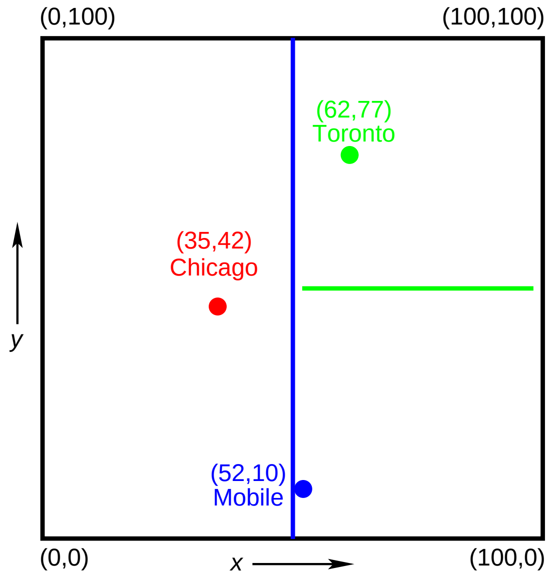
BSP tree

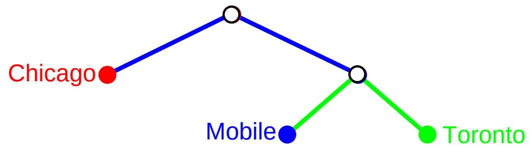




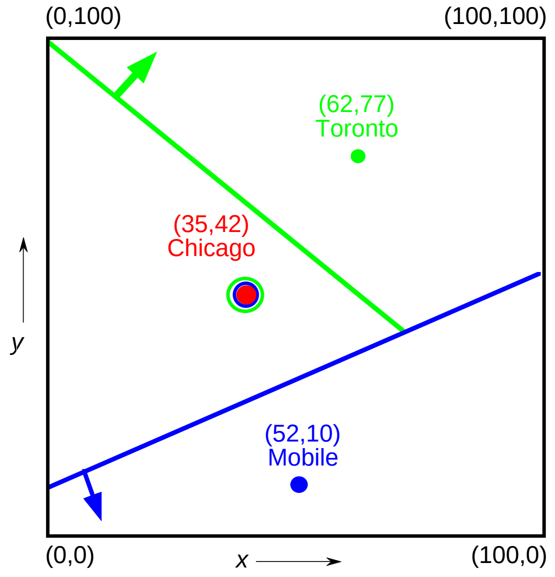
Comparison of mb-tree (BSP tree) and PR k-d tree

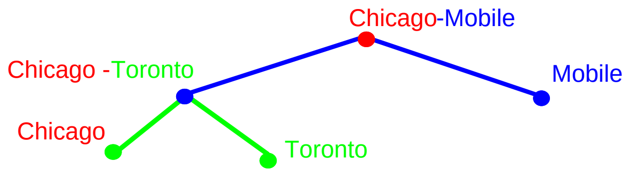
PR k-d tree





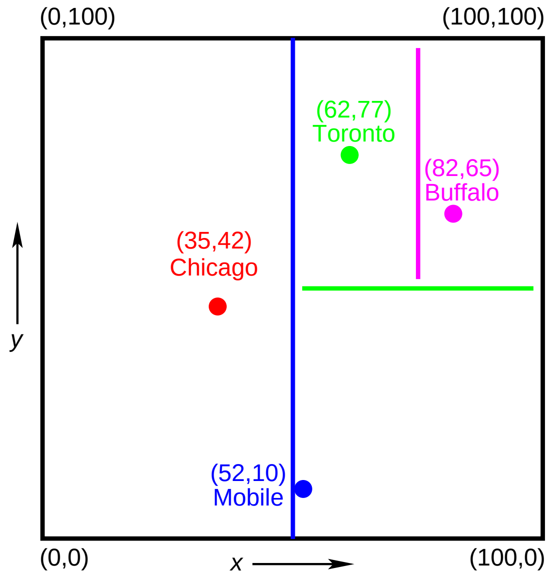
BSP tree

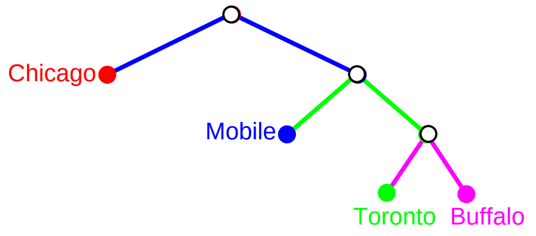




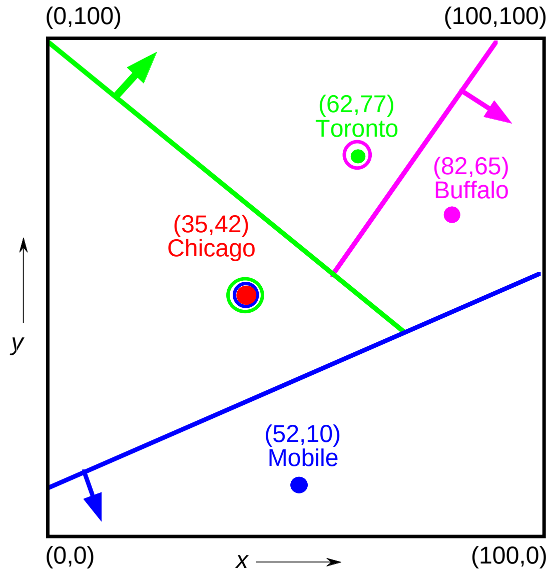
Comparison of mb-tree (BSP tree) and PR k-d tree

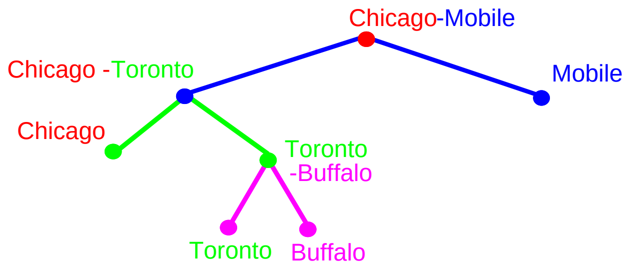
PR k-d tree





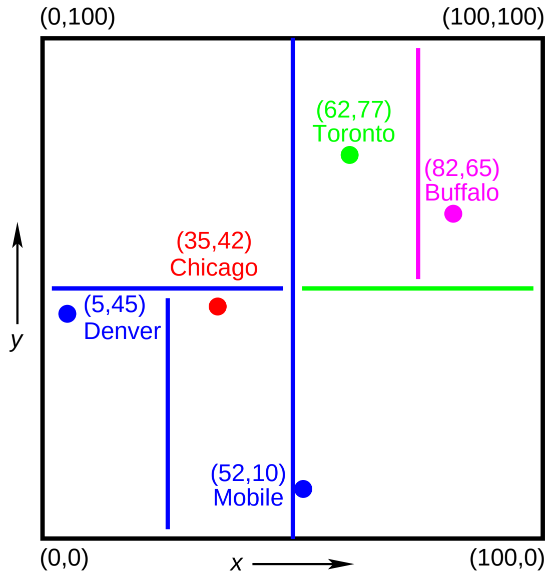
BSP tree

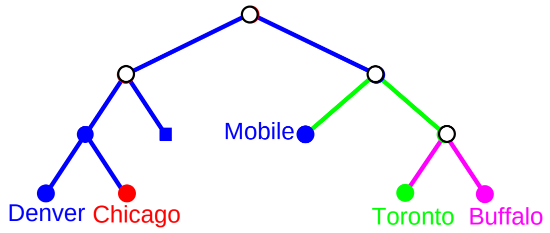




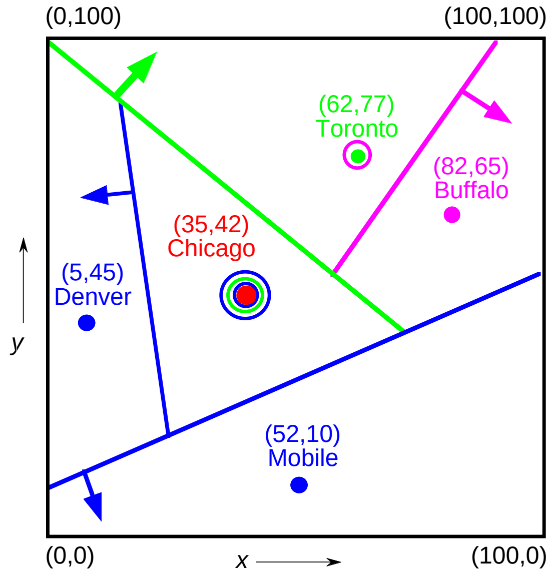
Comparison of mb-tree (BSP tree) and PR k-d tree

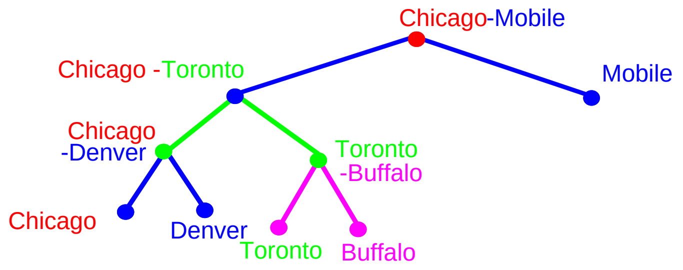
PR k-d tree





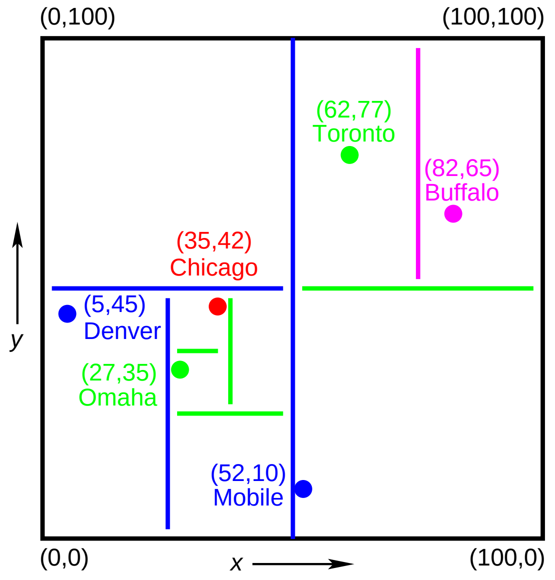
BSP tree

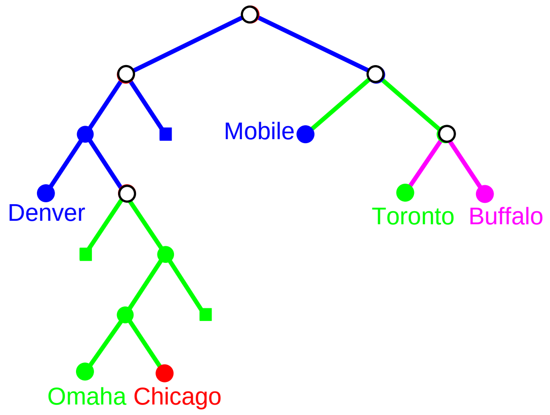




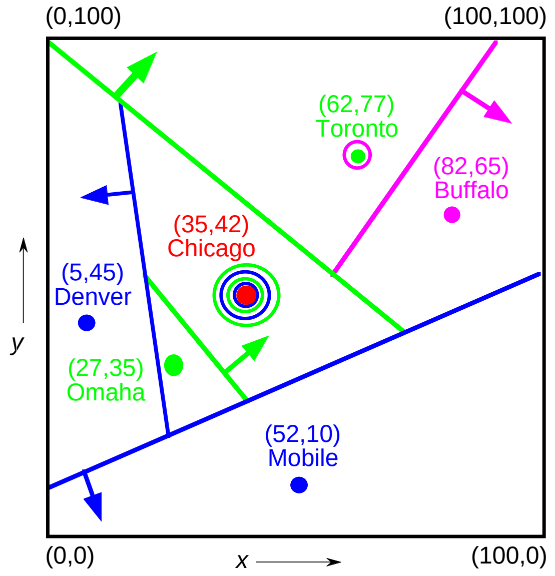
Comparison of mb-tree (BSP tree) and PR k-d tree

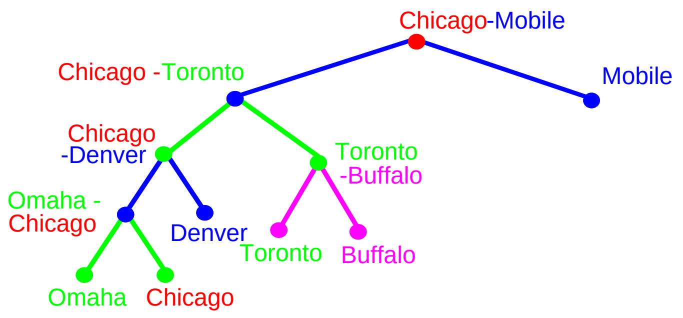
PR k-d tree





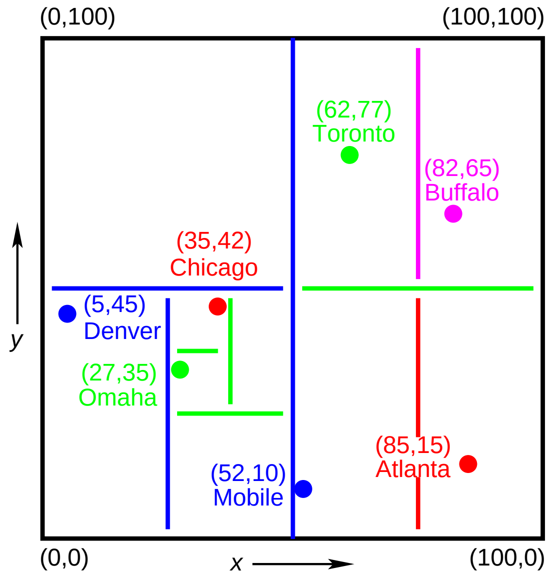
BSP tree

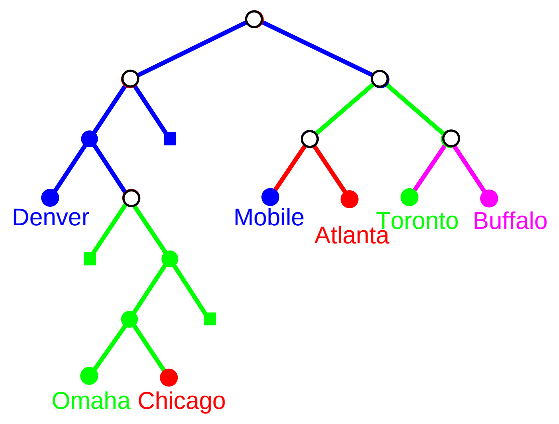




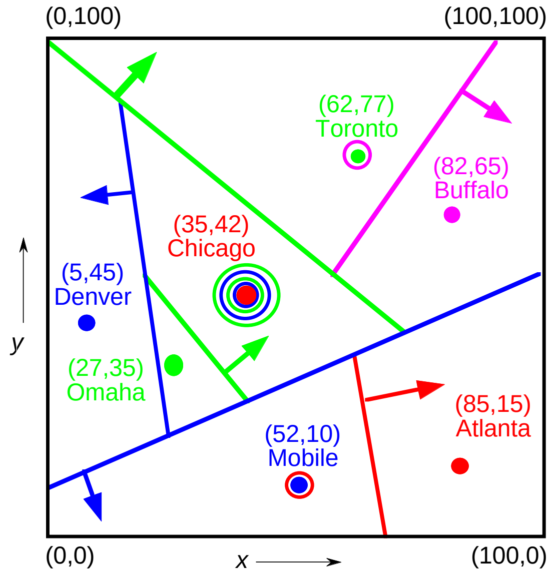
Comparison of mb-tree (BSP tree) and PR k-d tree

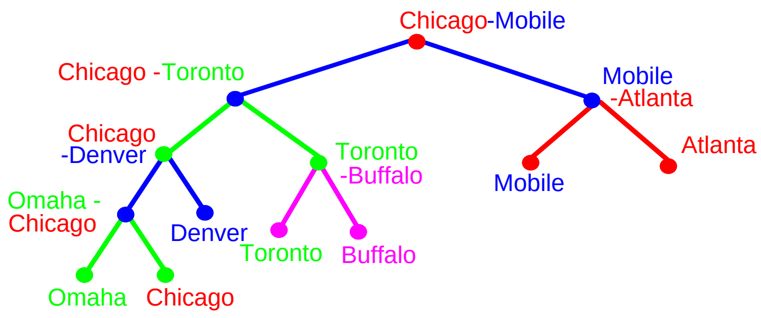
PR k-d tree





BSP tree

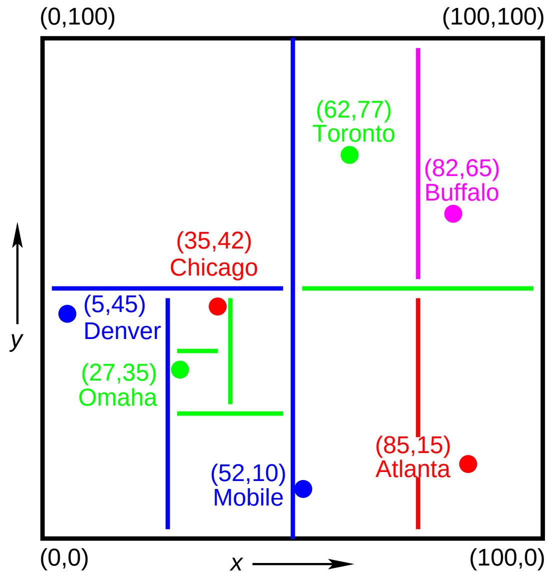


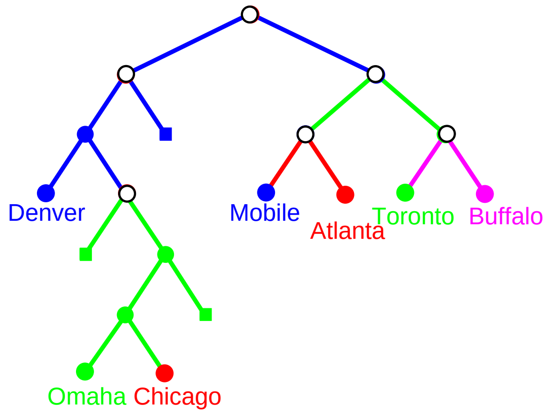


Comparison of mb-tree (BSP tree) and PR k-d tree

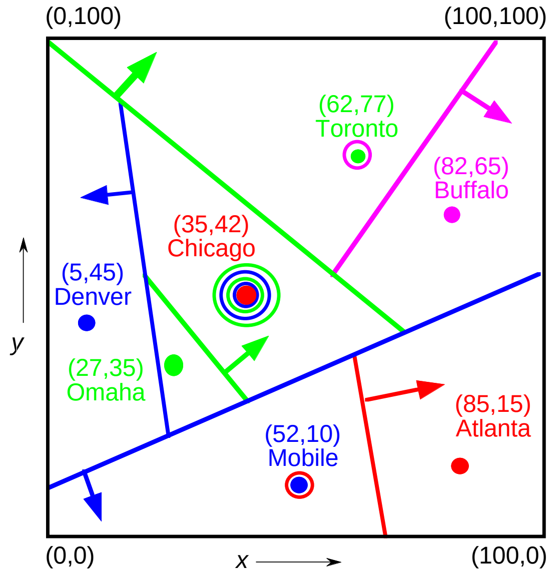
- Partition of underlying space analogous to that of BSP tree for points

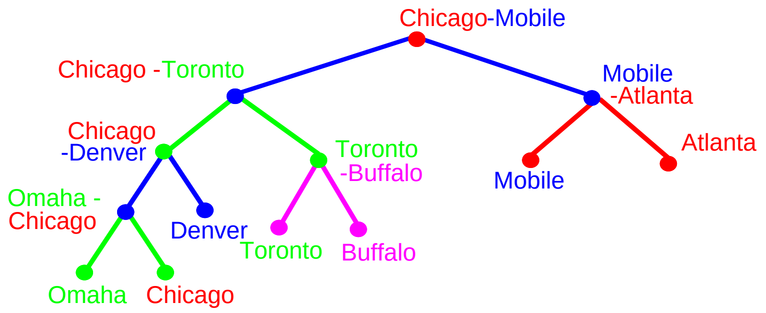
PR k-d tree





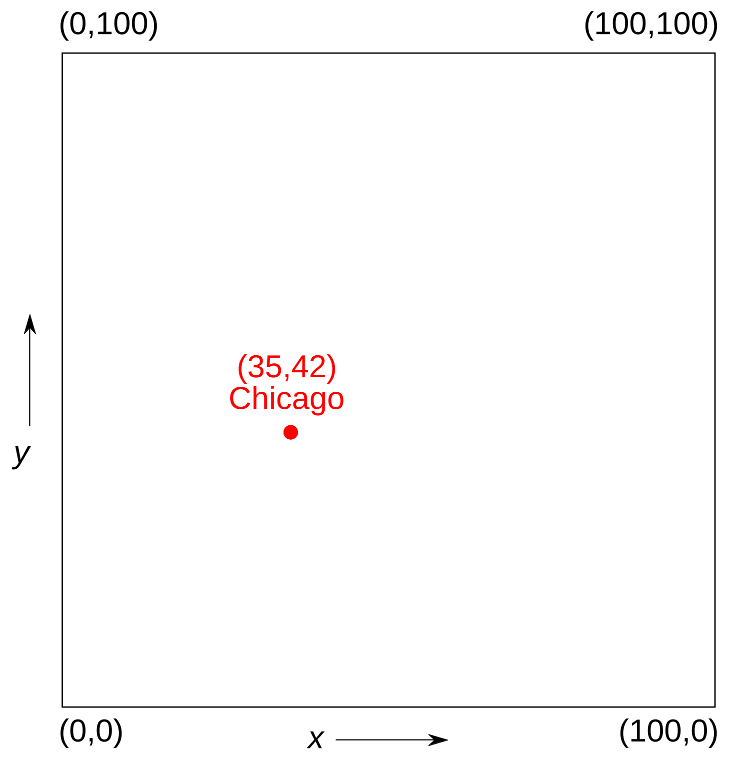
BSP tree





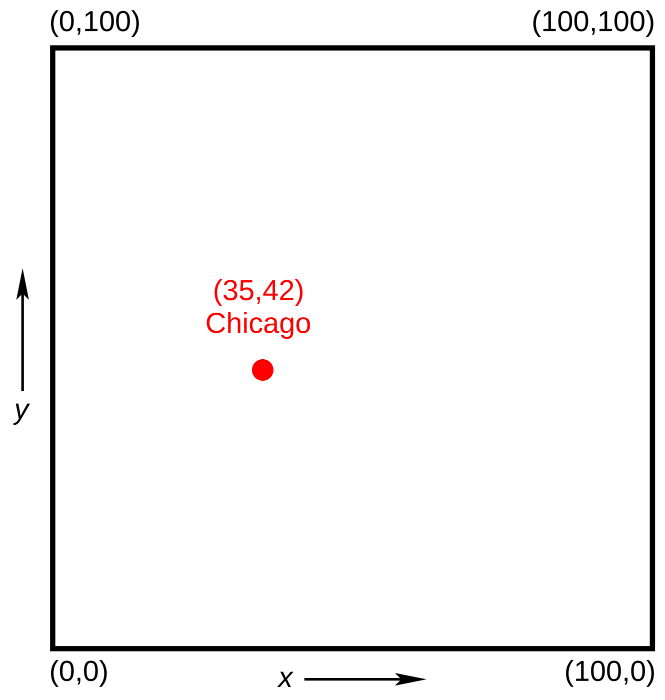
PR k-d tree

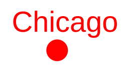
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points



PR k-d tree

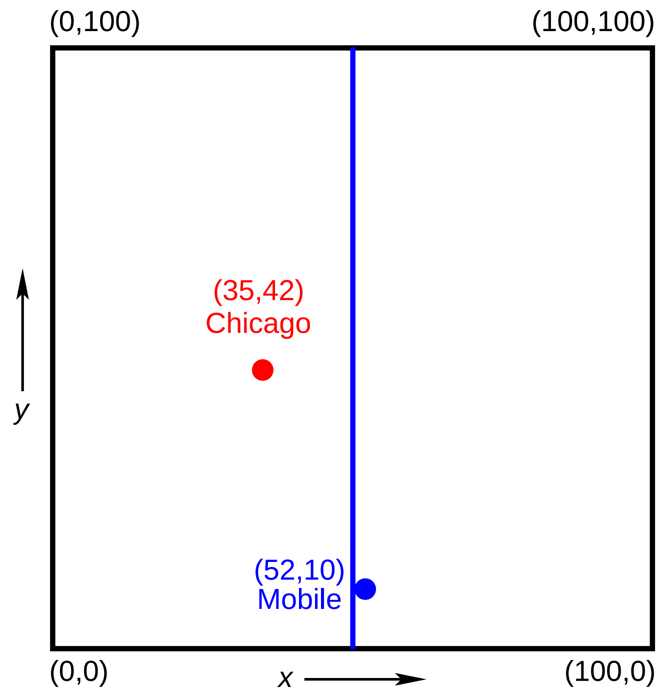
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

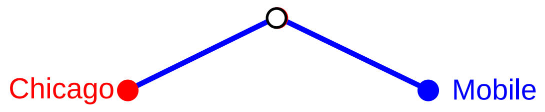




PR k-d tree

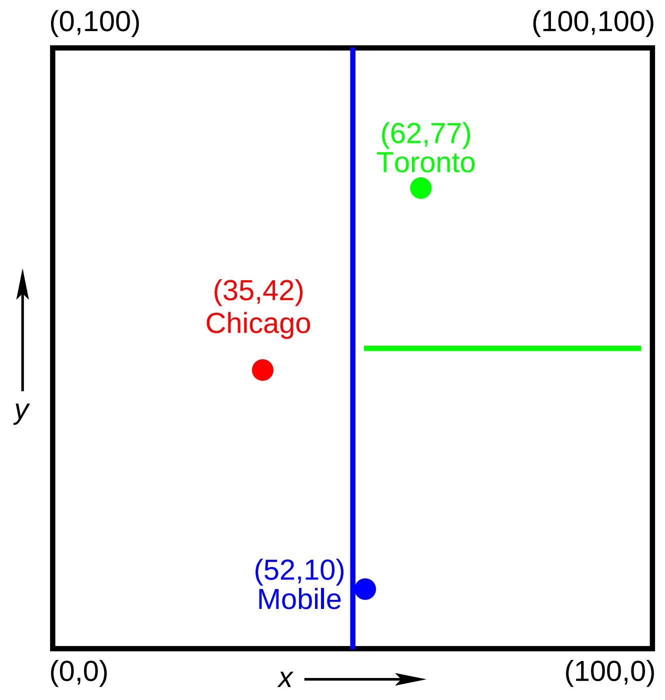
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

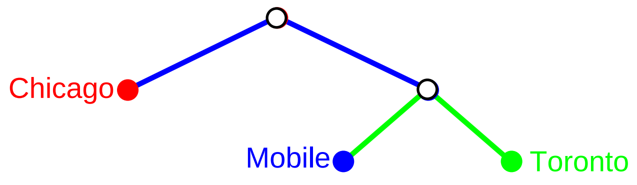




PR k-d tree

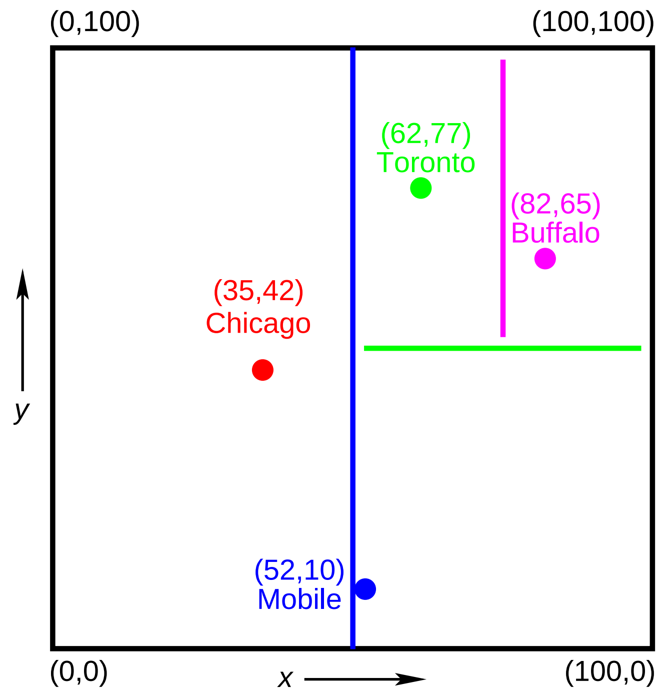
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

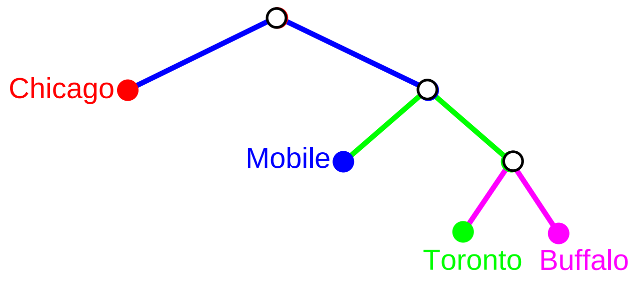




PR k-d tree

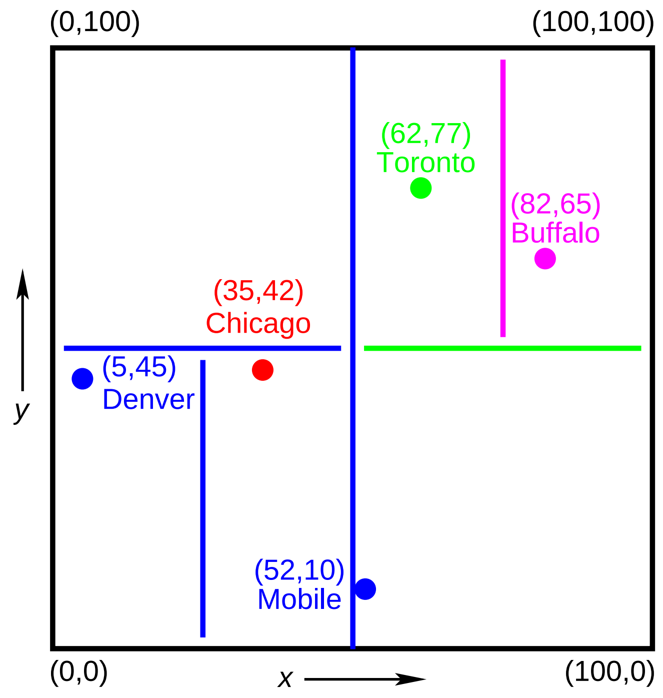
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

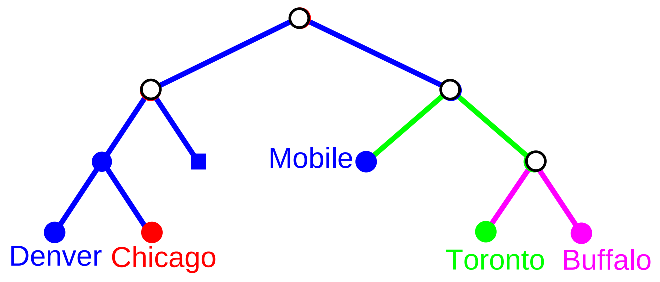




PR k-d tree

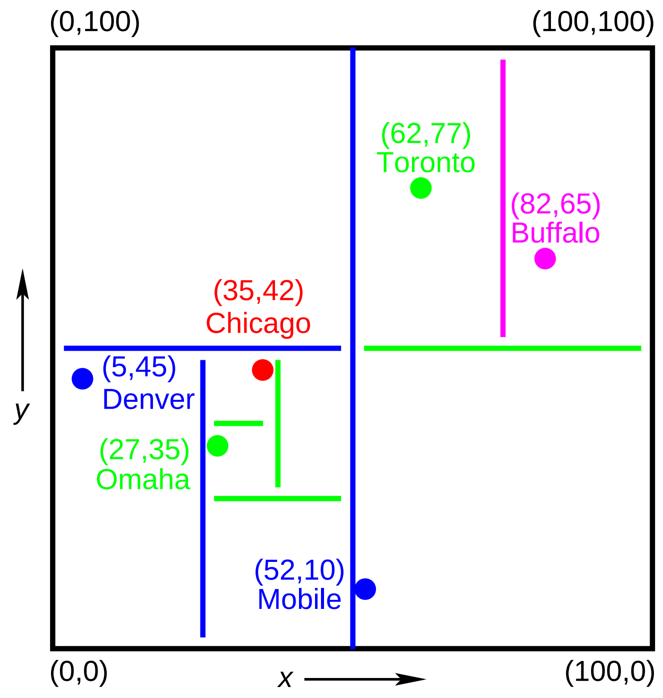
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

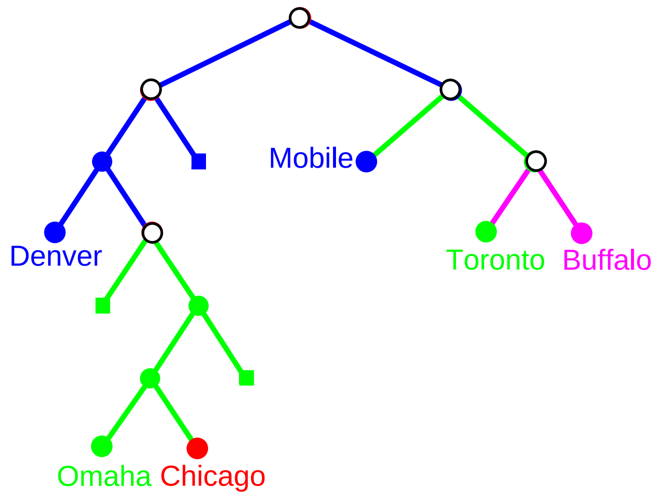




PR k-d tree

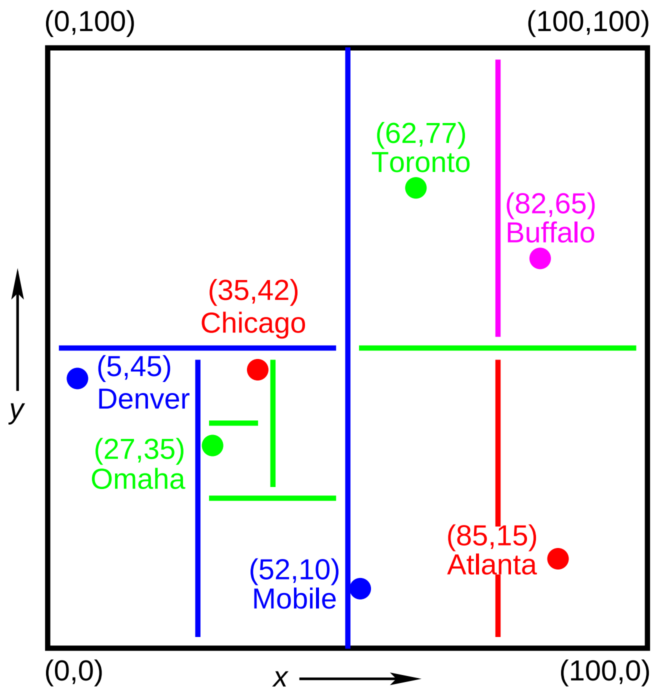
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points

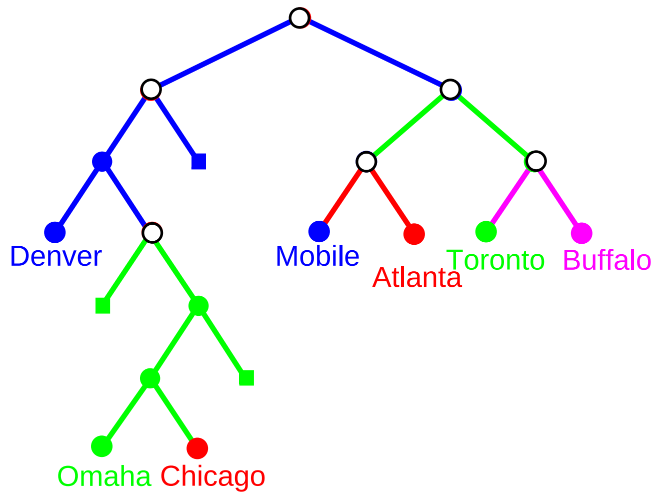




PR k-d tree

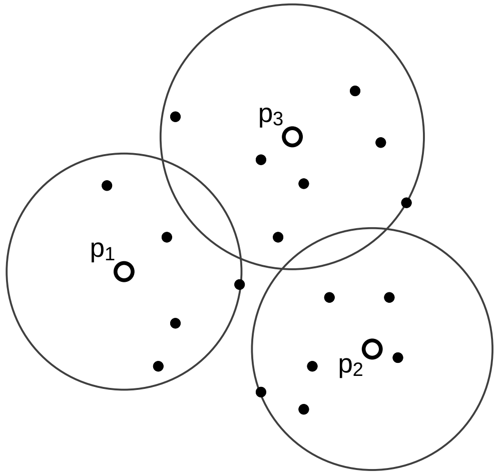
1. Regular decomposition point representation
2. Decompose whenever a block contains more than one point, while cycling through attributes
3. Maximum level of decomposition depends on minimum point separation
   - if two points are very close, then decomposition can be very deep
   - can be overcome by viewing blocks as buckets with capacity c and only decomposing a block when it contains more than c points





M-tree (Ciaccia et al.)

- Dynamic structure based on R-tree (actually SS-tree)
- All objects in leaf nodes
- Balls around “routing” objects (like pivots) play same role as minimum bounding boxes

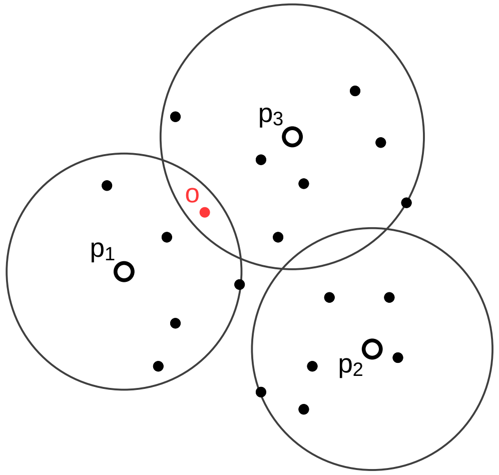


- Pivots play similar role as in GNAT, but:
  1. all objects are stored in the leaf nodes and an object may be referenced several times in the M-tree as it could be a routing object in more than one nonleaf node
  2. for an object o in a subtree of node n, the subtree’s pivot p is not always the one closest to o among all pivots in n

- Each nonleaf node n contains up to c entries of format (p, r, D, T)
  1. p is the pivot (i.e., routing object)
  2. r is the covering radius
  3. D is distance from p to its parent pivot p'
  4. T points to the subtree

M-tree (Ciaccia et al.)

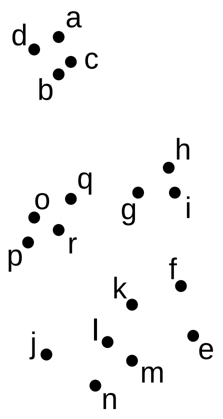
- Dynamic structure based on R-tree (actually SS-tree)
- All objects in leaf nodes
- Balls around "routing" objects (like pivots) play same role as minimum bounding boxes



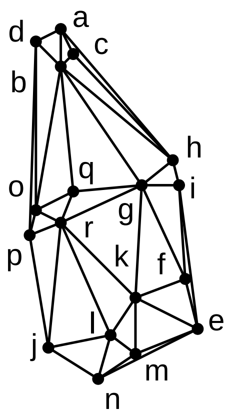
- Pivots play similar role as in GNAT, but:
  1. all objects are stored in the leaf nodes and an object may be referenced several times in the M-tree as it could be a routing object in more than one nonleaf node
  2. for an object o in a subtree of node n, the subtree's pivot p is not always the one closest to o among all pivots in n
  3. object o can be inserted into subtrees of several pivots: a choice
- Each nonleaf node n contains up to c entries of format (p, r, D, T)
  1. p is the pivot (i.e., routing object)
  2. r is the covering radius
  3. D is distance from p to its parent pivot p'
  4. T points to the subtree

Delaunay Graph

- Definition
  1. each object is a node and two nodes have an edge between them if their Voronoi cells have a common boundary
  2. explicit representation of neighbor relations that are implicitly represented in a Voronoi diagram
     - equivalent to an index or access structure for the Voronoi diagram
  3. search for a nearest neighbor of q starts with an arbitrary object and then proceeds to a neighboring object closer to q as long as this is possible
- Unfortunately we cannot construct Voronoi cells explicitly if only have interobject distances
- Spatial Approximation tree (sa-tree): approximation of the Delaunay graph



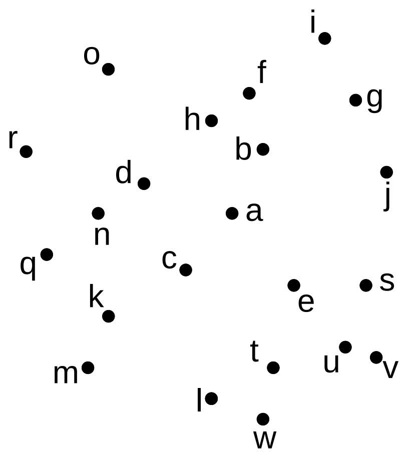
Point Set



Delaunay graph

sa-tree (Navarro)

- Definition:
  1. choose arbitrary object a as root of tree
  2. find N(a), smallest possible set of neighbors of a, so that any neighbor is closer to a than to any other object in N(a)
     - i.e., x is in N(a) iff for all $y \in N(a) - \{x\}$, $d(x,a) < d(x,y)$
     - all objects in $S \setminus N(a)$ are closer to some object in N(a) than to a
  3. objects in N(a) become children of a
  4. associate remaining objects in S with closest child of a, and recursively define subtrees for each child of a

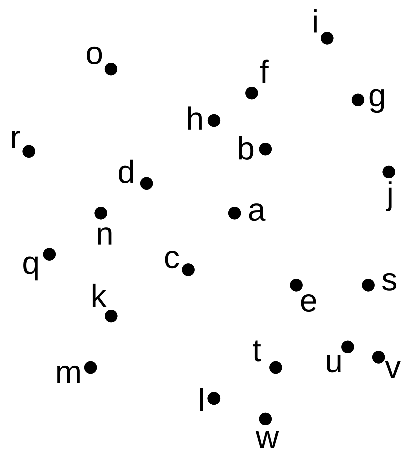


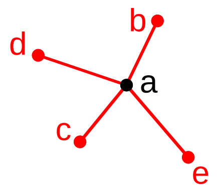
1. a is root

- Use heuristics to construct sa-tree as N(a) is used in the definition which makes it circular, and thus resulting tree is not necessarily minimal and not unique

sa-tree (Navarro)

- Definition:
  1. choose arbitrary object a as root of tree
  2. find N(a), smallest possible set of neighbors of a, so that any neighbor is closer to a than to any other object in N(a)
     - i.e., x is in N(a) iff for all $y \in N(a) - \{x\}$, $d(x,a) < d(x,y)$
     - all objects in $S \setminus N(a)$ are closer to some object in N(a) than to a
  3. objects in N(a) become children of a
  4. associate remaining objects in S with closest child of a, and recursively define subtrees for each child of a



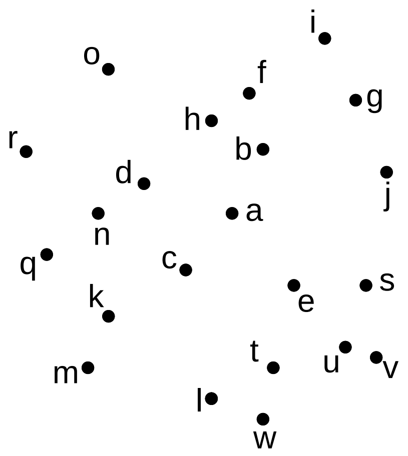


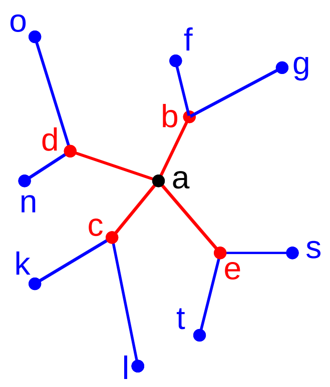
1. a is root
2. N(a)={b,c,d,e}

- Use heuristics to construct sa-tree as N(a) is used in the definition which makes it circular, and thus resulting tree is not necessarily minimal and not unique

sa-tree (Navarro)

- Definition:
  1. choose arbitrary object a as root of tree
  2. find N(a), smallest possible set of neighbors of a, so that any neighbor is closer to a than to any other object in N(a)
     - i.e., x is in N(a) iff for all $y \in N(a) - \{x\}$, $d(x,a) < d(x,y)$
     - all objects in $S \setminus N(a)$ are closer to some object in N(a) than to a
  3. objects in N(a) become children of a
  4. associate remaining objects in S with closest child of a, and recursively define subtrees for each child of a



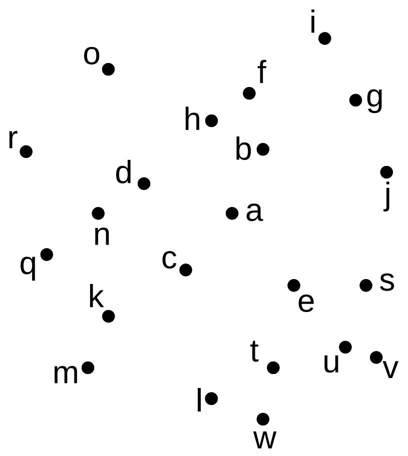


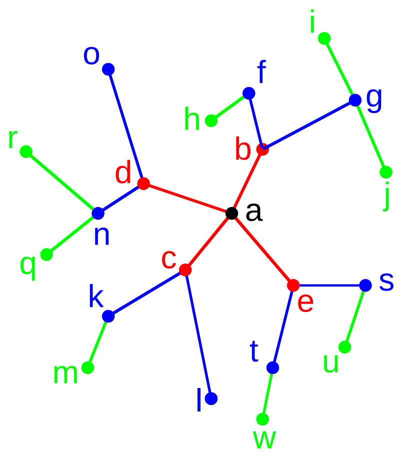
1. a is root
2. N(a)={b,c,d,e}
3. second level

- Use heuristics to construct sa-tree as N(a) is used in the definition which makes it circular, and thus resulting tree is not necessarily minimal and not unique

sa-tree (Navarro)

- Definition:
  1. choose arbitrary object a as root of tree
  2. find N(a), smallest possible set of neighbors of a, so that any neighbor is closer to a than to any other object in N(a)
     - i.e., x is in N(a) iff for all $y \in N(a) - \{x\}$, $d(x,a) < d(x,y)$
     - all objects in $S \setminus N(a)$ are closer to some object in N(a) than to a
  3. objects in N(a) become children of a
  4. associate remaining objects in S with closest child of a, and recursively define subtrees for each child of a



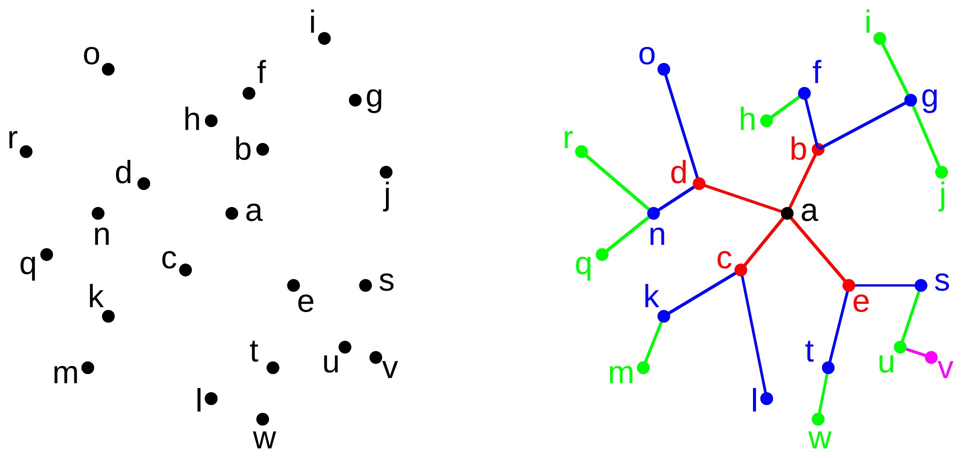


1. a is root
2. N(a)={b,c,d,e}
3. second level
4. h $\notin$ N(a) and N(b) as h closer to F than to b or a

- Use heuristics to construct sa-tree as N(a) is used in the definition which makes it circular, and thus resulting tree is not necessarily minimal and not unique

sa-tree (Navarro)

- Definition:
  1. choose arbitrary object a as root of tree
  2. find N(a), smallest possible set of neighbors of a, so that any neighbor is closer to a than to any other object in N(a)
     - i.e., x is in N(a) iff for all $y \in N(a) - \{x\}$, $d(x,a) < d(x,y)$
     - all objects in $S \setminus N(a)$ are closer to some object in N(a) than to a
  3. objects in N(a) become children of a
  4. associate remaining objects in S with closest child of a, and recursively define subtrees for each child of a



1. a is root
2. N(a)={b,c,d,e}
3. second level
4. h $\notin$ N(a) and N(b) as h closer to F than to b or a
5. fourth level

- Use heuristics to construct sa-tree as N(a) is used in the definition which makes it circular, and thus resulting tree is not necessarily minimal and not unique

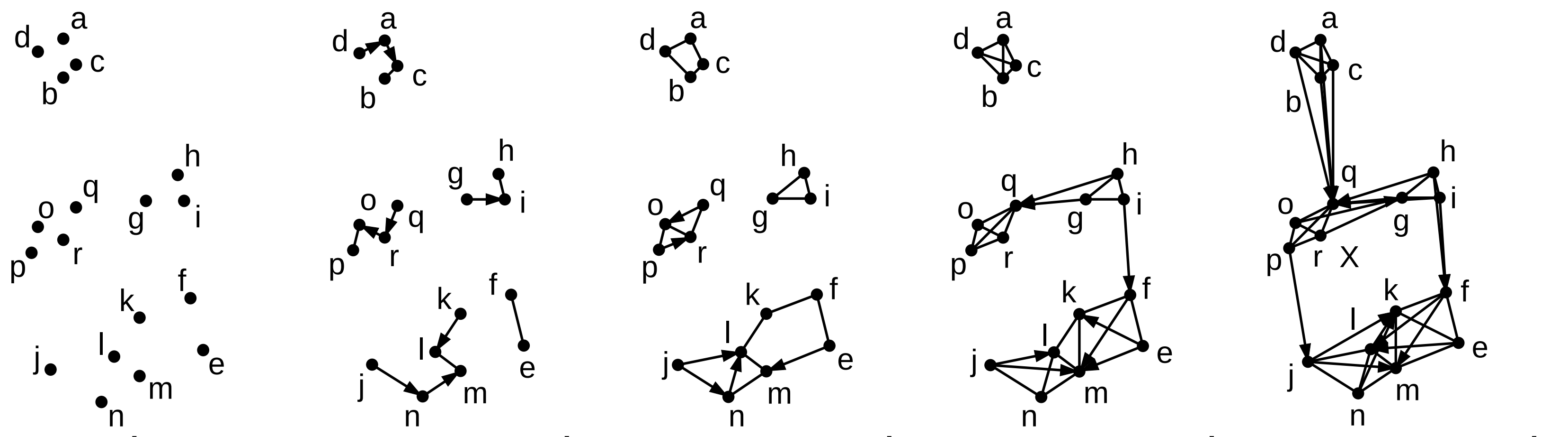
Range Searching with sa-tree

- Search algorithms make use of Lemma 4 which provides a lower bound on distances
  1. know that for c in $\{a\} \cup N(a)$, b in $N(a)$, and o in tree rooted at b, then o is closer to b than to c
     - therefore,$(d(q,b) - d(q,c))/2 \leq d(q,o)$ from Lemma 4
  2. want to avoid visiting as many children of a as possible
     - must visit any object o for which $d(q,o) \leq \epsilon$
     - must visit any object o in b if lower bound $(d(q,b) - d(q,c))/2 \leq \epsilon$
     - no need to visit any objects o in b for which there exist c in $\{a\} \cup N(a)$ so that $(d(q,b) - d(q,c))/2 > \epsilon$
     - higher lower bound implies less likely to visit
     - $d(q,o)$ is maximized when $d(q,c)$ is minimized
     - c is object in $\{a\} \cup N(a)$ which is closest to q
  3. choose c so that lower bound $(d(q,b) - d(q,c))/2$ on $d(q,o)$ is maximized
     - c is object in $\{a\} \cup N(a)$ closest to q
- Once find c, traverse each child $b \in N(a)$ except those for which

$$(d(q,b) - d(q,c))/2 > \epsilon$$

kNN Graphs (Sebastian/Kimia)

1. Each vertex has an edge to each of its k nearest neighbors

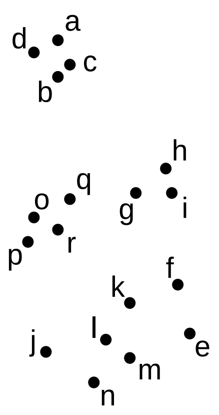


Point Set 1NN graph 2NN graph 3NN graph 4NN graph

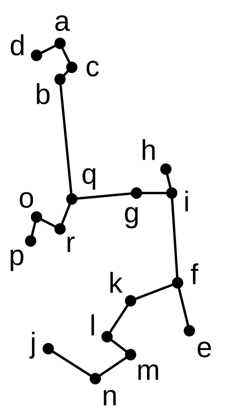
2. Problems
   - graph is not necessarily connected
   - even if increase k so graph is connected, search may halt at object p which is closer to q than any of the k nearest neighbors of p but not closer than all of the objects in p's neighbor set (e.g., the $k + 1^{st}$ nearest neighbor)
     - Ex: search for nearest neighbor of X in 4NN graph starting at any one of {e,f,j,k,l,m,n} will return k instead of r
   - overcome by extending size of search neighborhood as in approximate nearest neighbor search
   - use several starting points for search (i.e., seeds)
3. Does not require triangle inequality and thus works for arbitrary distances

Alternative Approximations of the Delaunay Graph

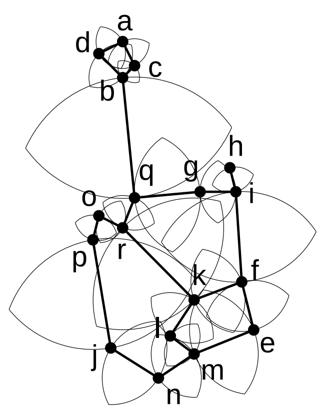
1. Other approximation graphs of the Delaunay graph are connected by virtue of being supersets of the minimal spanning tree (MST) of the vertices
2. Relative neighborhood graph (RNG): an edge between vertices u and v if for all vertices p, u is closer to v than is p or v is closer to u than is p — that is, $d(u,v) \leq Max\{d(p,u), d(p,v)\}$
3. Gabriel graph (GG): an edge between vertices u and v if for all other vertices p we have that $d(u,p)^2 + d(v,p)^2 \geq d(u,v)^2$
4. RNG and GG are not restricted to Euclidean plane or Minkowski metrics
5. MST(E) $\subset$RNG(E) $\subset$GG(E) $\subset$DT(E) in Euclidean plane with edges E
6. MST(E) $\subset$RNG(E) $\subset$GG(E) in any metric space as DT is only defined for the two-dimensional Euclidean plane



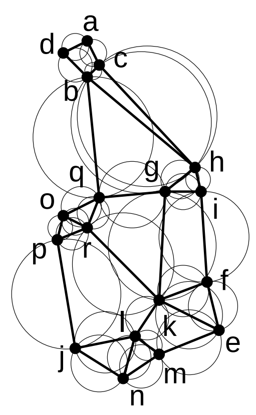
Point Set



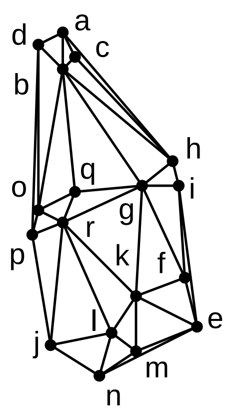
MST



RNG



GG



Delaunay Graph (DT)

Use of Delaunay Graph Approximations

1. Unless approximation graph is a superset of Delaunay graph (which it is not), to be useful in nearest neighbor searching, we need to be able to force the algorithm to move to other neighbors of current object p even if they are farther from q than p
2. Examples:
   - kNN graph: use extended neighborhood
   - sa-tree: prune search when can show (with aid of triangle inequality) that it is impossible to reach the nearest neighbor via a transition to nearest neighbor or set of neighbors
   - RNG and GG have advantage that are always connected and don't need seeds
   - advantage of kNN graph is that k nearest neighbors are precomputed

Spatial Approximation Sample Hierarchy (SASH)(Houle)

- Hierarchy of random samples of set of objects S of size $S/2, S/4, S/8, \ldots, 1$
- Makes use of approximate nearest neighbors
- Has similar properties as the kNN graph
  1. both do not require that the triangle inequality be satisfied
  2. both are indexes
     - $O(N^2)$ time to build kNN graph as no existing index
     - SASH is built incrementally level by level starting at root with samples of increasing size making use of index already built for existing levels thereby taking $O(N \log N)$ time
     - each level of SASH is a kNN tree with maximum $k = c$
- Key to approximation is to treat the “nearest neighbor relation” as an “equivalence relation” even though this is not generally true
  1. assumption of “equivalence” relation is the analog of ϵ
  2. no symmetry: x being approximate nearest neighbor of x' does not mean that x' must be an approximate nearest neighbor of x
  3. no transitivity: x being approximate nearest neighbor of q and x' being approximate nearest neighbor of x does not mean that x' must be an approximate nearest neighbor of q
  4. construction of SASH is analog of UNION operation
  5. finding approximate nearest neighbor is analog of FIND operation

SASH vis-a-vis Triangle Inequality

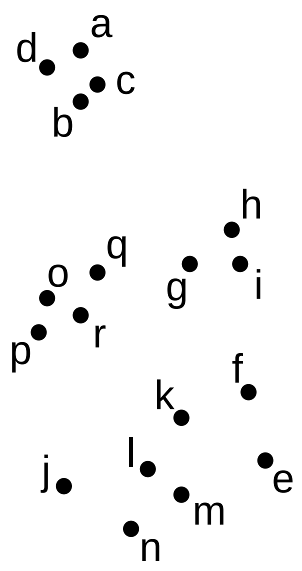
- Triangle inequality is analogous to transitivity with $\leq$ corresponding to "approximate nearest neighbor" relation
- Appeal to triangle inequality, $d(x', q) \leq d(q, x) + d(x', x)$, regardless of whether or not it holds
  1. to establish links to objects likely to be neighbors of query object q,
     - when $d(q, x)$ and $d(x', x)$ are both very small, then $d(q, x')$ is also very small (analogous to "nearest")
     - implies if $x \in S \setminus S'$ is a highly ranked neighbor of both q and $x' \in S'$ among objects in $S \setminus S'$, then x' is also likely to be a highly ranked neighbor of q among objects in S'
       - x' is a highly ranked neighbor of x (symmetry)
       - AND x is a highly ranked neighbor of q
       - RESULT: x' is a highly ranked neighbor of q (transitivity)
  2. INSTEAD of to eliminate objects that are guaranteed not to be neighbors

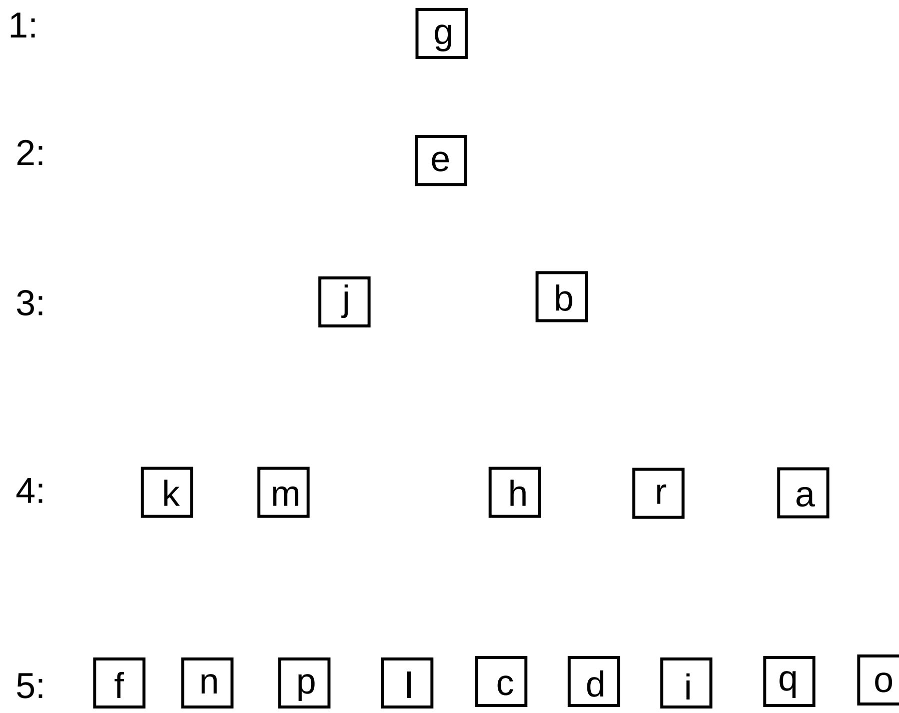
Mechanics of SASH

- SASH construction (UNION of UNION-FIND)
  1. form hierarchy of samples
  2. assume SASH$_i$ has been built and process sample S'
     - know that x in SASH$_i \backslash$SASH$_{i-1}$ is one of p approximate nearest neighbors of $x' \in S'$ and use SASH$_i$ to determine x
     - infer that x' is one of $c > p$ approximate nearest neighbors in S' of x (symmetry)
  3. special handling to ensure that every object at level $i+1$ is an approximate nearest neighbor of at least one object at level i (i.e., no orphan objects)
- Finding k approximate nearest neighbors of q (FIND of UNION-FIND)
  1. follow links from level $i-1$ of SASH to level i retaining in U_i the k_i approximate nearest neighbors of q at level i of the SASH
  2. determine k approximate nearest neighbors of q from the union of U_i over all levels of the SASH
  3. know that x in U_i is an approximate nearest neighbor of q
  4. know that x' in U_{i+1} is an approximate nearest neighbor of x in U_i
  5. infer that x' in U_{i+1} is an approximate nearest neighbor of q (transitivity)

Example of SASH construction

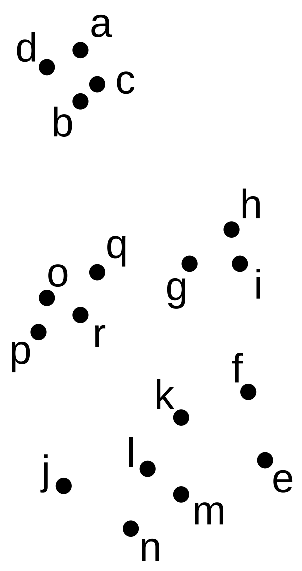
- Ex: P=2 C=5

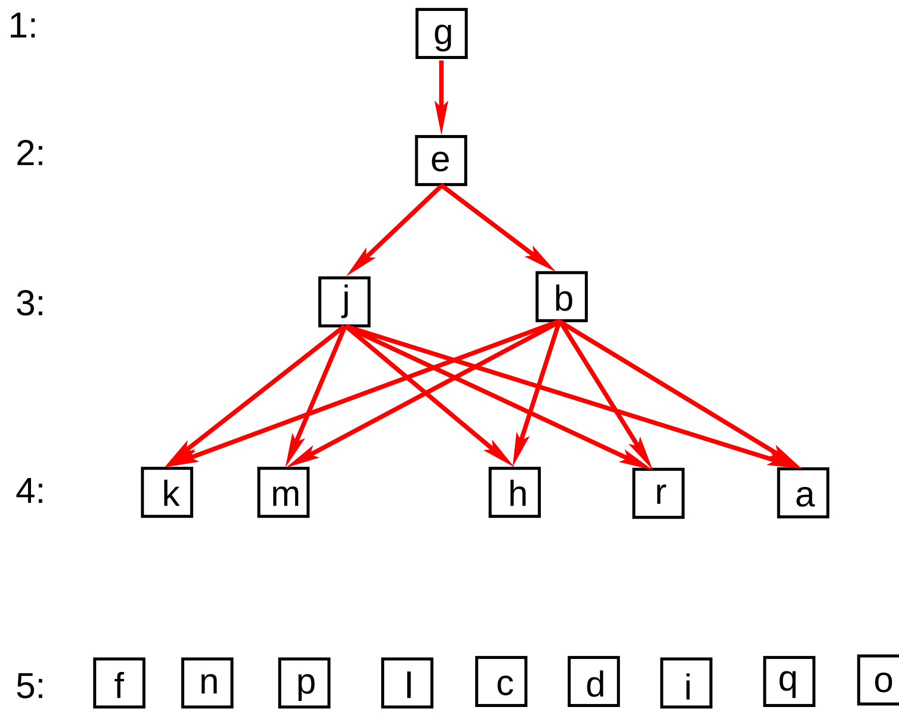




Example of SASH construction

- Ex: P=2 C=5
- Initially, no choice in the first 3 levels

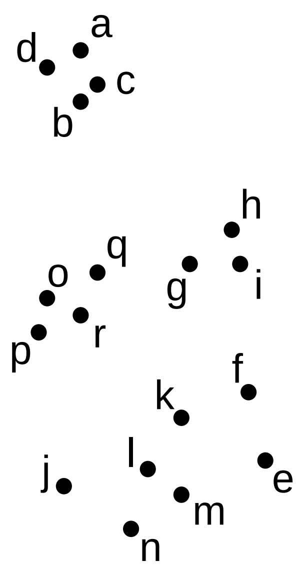


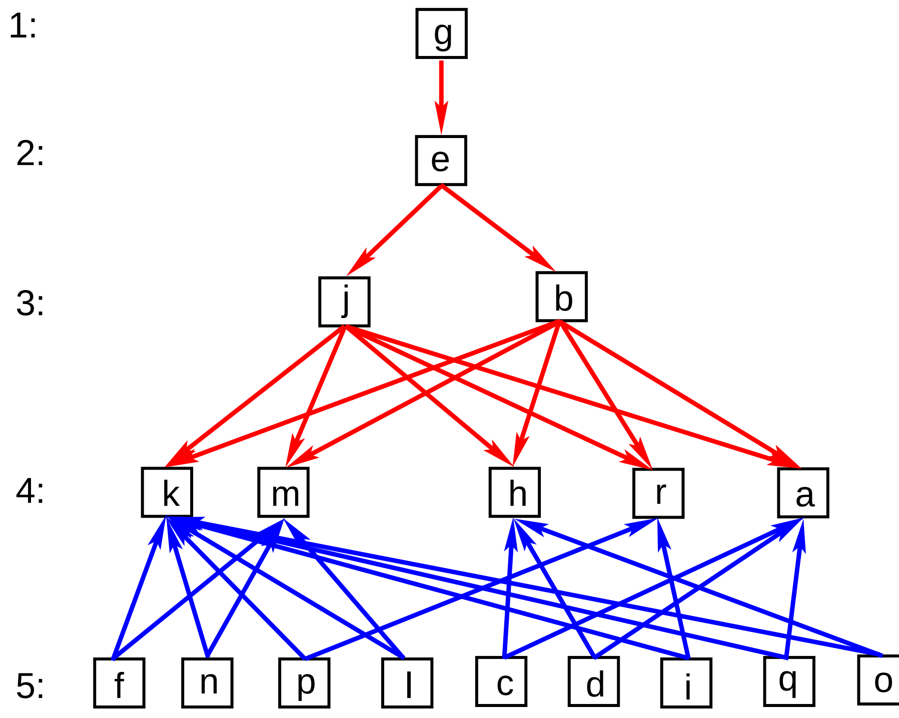


Example of SASH construction

- Ex: P=2 C=5
- Initially, no choice in the first 3 levels
- Find two closest objects at level 4 for each object at level 5

f:k,m	n:k,m
p:k,r	l:k,m
c:a,h	d:a,h
i:h,k	d:h,r
o:k,r	



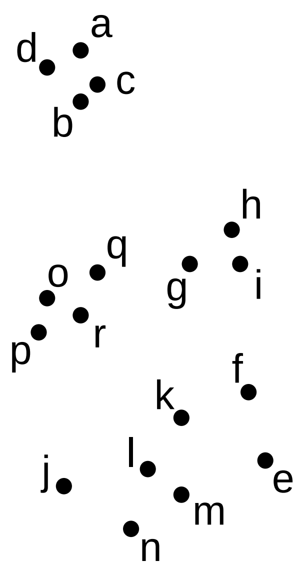


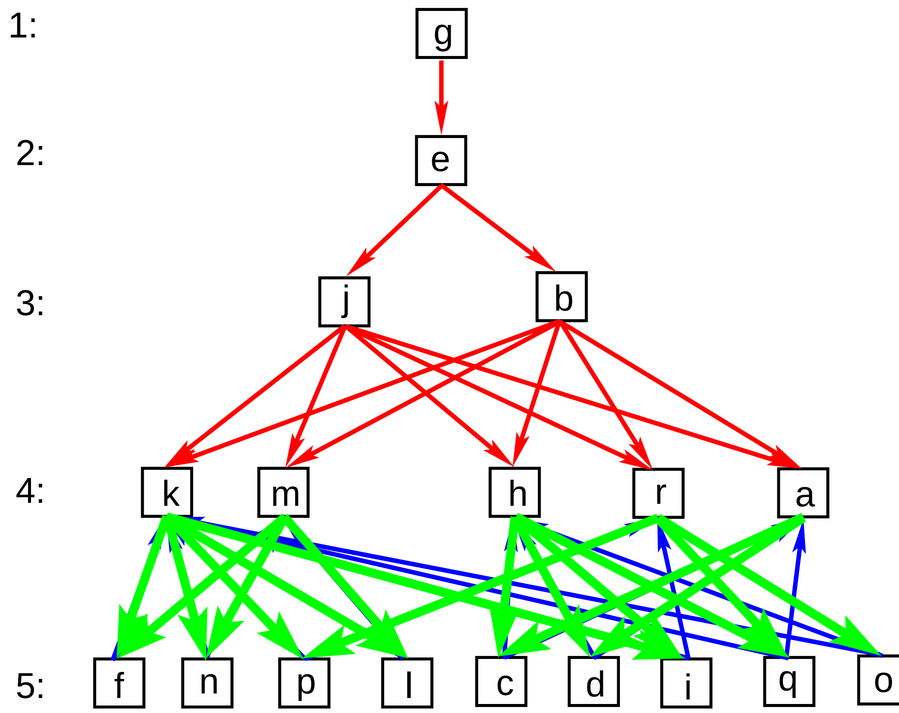
Example of SASH construction

- Ex: P=2 C=5
- Initially, no choice in the first 3 levels
- Find two closest objects at level 4 for each object at level 5

f:k,m	n:k,m
p:k,r	l:k,m
c:a,h	d:a,h
i:h,k	d:h,r
o:k,r	

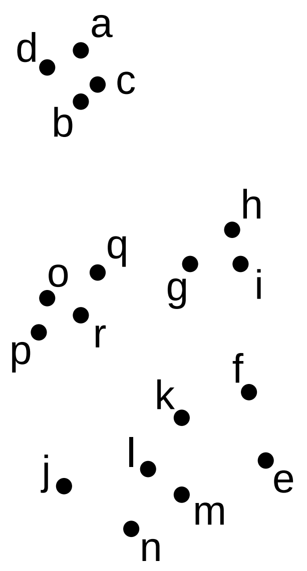
- Retain 5 nearest neighbors at level 5 to each object at level 4
  - k:{f,n,p,l,i,o }
  - m:f,n,l
  - n:c,d,i,q
  - a:c,d

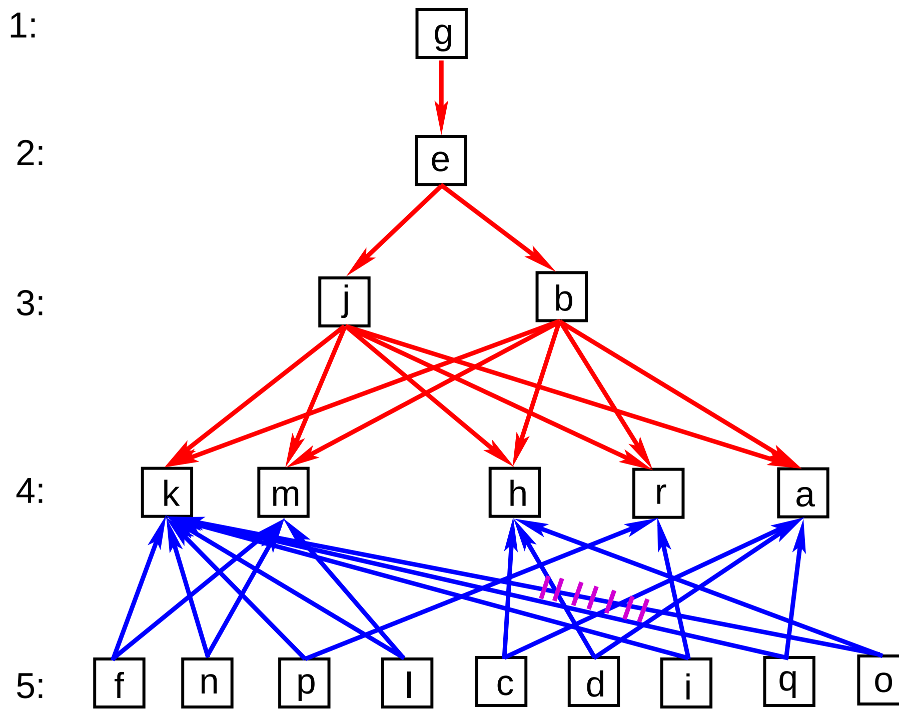




Example of SASH construction

- Ex: P=2 C=5
- Initially, no choice in the first 3 levels
- Find two closest objects at level 4 for each object at level 5
  - f:k,m n:k,m
  - p:k,r l:k,m
  - c:a,h d:a,h
  - i:h,k d:h,r
  - o:k,r
- Retain 5 nearest neighbors at level 5 to each object at level 4
  - k:{f,n,p,l,i,~~o~~}
  - m:f,n,l
  - n:c,d,i,q
  - a:c,d
- Ignore o as k has 5 closer neighbors



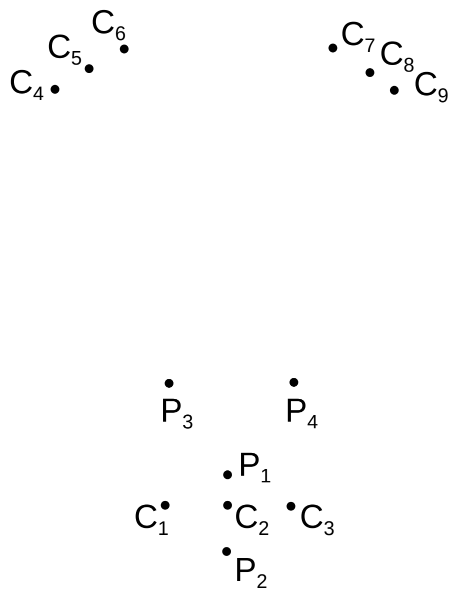


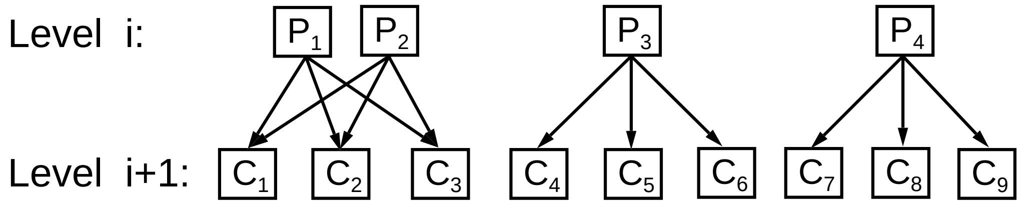
Example SASH Approximate k Nearest Neighbor Finding

- Ex: $k = 3$ and query object c
- Let $f(k,i) = k_i = k^{1-(h-i)/\log_2 N}$ yielding $k_i = (1,1,2,2,3)$
- $U_1 =$ root g of SASH
- $U_2 =$ objects reachable from U_1 which is e
- $U_3 =$ objects reachable form U_2 which is b and j which are retained as $k_3 = 2$
- U_4 objects reachable from U_3 which is {a,h,k,m,r} and we retain just a and h in U_4 as $k_4 = 2$
- $U_5 =$ objects reachable form U_4 which is {c,d,i,q}, and we retain just c, d, and q in U_5 as $k_5 = 3$
- Take union of U_1, U_2, U_3, U_4, U_5 which is the set {a,b,c,d,e,g,h,i,j,k,m,q,r}, and the closest three neighbors to query object c are a, b, and d

Drawback of SASH

- Assumes that if a at level i is an approximate nearest neighbor of o at level $i+1$, then by symmetry o is likely to be an approximate nearest neighbor of a, which is not generally true
- Ex: objects at level i are not necessarily linked to their nearest neighbors at level $i+1$





- P_3 and P_4 at level i are linked to the sets of three objects $\{C_4, C_5, C_6\}$ and $\{C_7, C_8, C_9\}$, respectively, at level i+1, instead of to their nearest neighbors C_1, C_2, and C_3 at level i+1.

AESA (Vidal Ruiz)

- Precomputes $O(N^2)$ interobject distances between all N objects in S and stores them in a distance matrix
- Distance matrix is used to provide lower bounds on distances from query object q to objects whose distances have not yet been computed
- Only useful if static set of objects and number of queries $\gg N$ as otherwise can use brute force to find nearest neighbor with N distance computations
- Algorithm for range search:

  S_u: objects whose distance from q has not been computed and that have not been pruned, initially S

  $d_{\mathrm{lo}}(q,o)$: lower bound on $d(q,o)$ for $o \in S_u$, initially zero

  1. remove from S_u the object p with lowest value $d_{\mathrm{lo}}(q,p)$
     - terminate if S_u is empty or if $d_{\mathrm{lo}}(q,p) > \epsilon$
  2. compute $d(q,p)$, adding p to result if $d(q,p) \leq \epsilon$
  3. for all $o \in S_u$, update $d_{\mathrm{lo}}(q,o)$ if possible
     - $d_{\mathrm{lo}}(q,o) \leftarrow \max\{d_{\mathrm{lo}}(q,o), |d(q,p) - d(p,o)|\}$
     - lower bound property by Lemma 1: $|d(q,p) - d(p,o)| \leq d(q,o)$
  4. go to step 1
- Other heuristic possible for choosing next object: random, highest d_{lo}, etc.

LAESA (Micó et al.)

- AESA is costly as treats all N objects as pivots
- Choose a fixed number M of pivots
- Similar approach to searching as in AESA but
  1. non-pivot objects in S_c do not help in tightening lower bound distances of the objects in S_u
  2. eliminating pivot objects in S_u may hurt later in tightening the distance bounds
- Differences:
  1. selecting a pivot object in S_u over any non-pivot object, and
  2. eliminating pivot objects from S_u only after a certain fraction f of the pivot objects have been selected into S_c (f can range from 0 to 100%
     - if $f = 100\%$ then pivots are never eliminated from S_u

Classifying Distance-Based Indexing Methods

1. Pivot-based methods:
   - pivots, assuming k of them, can be viewed as coordinates in a k-dimensional space and the result of the distance computation for object x is equivalent to a mapping of x to a point $(x_0, x_1, \ldots, x_{k-1})$ where coordinate value x_i is the distance $d(x, p_i)$ of x from pivot p_i
   - result is similar to embedding methods
   - also includes distance matrix methods which contain precomputed distances between some (e.g., LAESA) or all (e.g., AESA) objects
     - difference from ball partitioning as no hierarchical partitioning of data set
2. Clustering-based methods:
   - partition data into spatial-like zones based on proximity to distinguished object called the *cluster center*
   - each object associated with closest cluster center
   - also includes sa-tree which records subset of Delaunay graph of the data set which is a graph whose vertices are the Voronoi cells
   - different from pivot-based methods where an object o is associated with a pivot p on the basis of o's distance from p rather than because p is the closest pivot to o

Pivot-Based vs: Clustering-Based Indexing Methods

1. Both achieve a partitioning of the underlying data set into spatial-like zones
2. Difference:
   - pivot-based: boundaries of zones are more well-defined as they can be expressed explicitly using a small number of objects and a known distance value
   - clustering-based methods: boundaries of zones are usually expressed implicitly in terms of the cluster centers, instead of explicitly, which may require quite a bit of computation to determine
     - in fact, very often, the boundaries cannot be expressed explicitly as, for example, in the case of an arbitrary metric space (in contrast to a Euclidean space) where we do not have a direct representation of the ‘generalized hyperplane’ that separates the two partitions

Distance-Based vs: Multidimension Indexing

1. Distance computations are used to build index in distance-based indexing, but once index has been built, similarity queries can often be performed with significantly fewer distance computations than a sequential scan of entire dataset
2. Drawback is that if we want to use a different distance metric, then need to build a separate index for each metric in distance-based indexing
   - not the case for multidimensional indexing methods which can support arbitrary distance metrics when performing a query, once the index has been built
   - however, multidimensional indexing is not very useful if don’t have a feature value and only know relative interobject distances (e.g., DNA sequences)

Part 3: Dimension Reduction

1. Motivation
   - overcoming curse of dimensionality
   - want to use traditional indexing methods (e.g., R-tree and quadtree variants) which lose effectiveness in higher dimensions
2. Searching in a dimensionally-reduced space
3. Using only one dimension
4. Representative point methods
5. Singular value decomposition (SVD, PCA, KLT)
6. Discrete Fourier transform (DFT)

Searching in a Dimensionally-Reduced Space

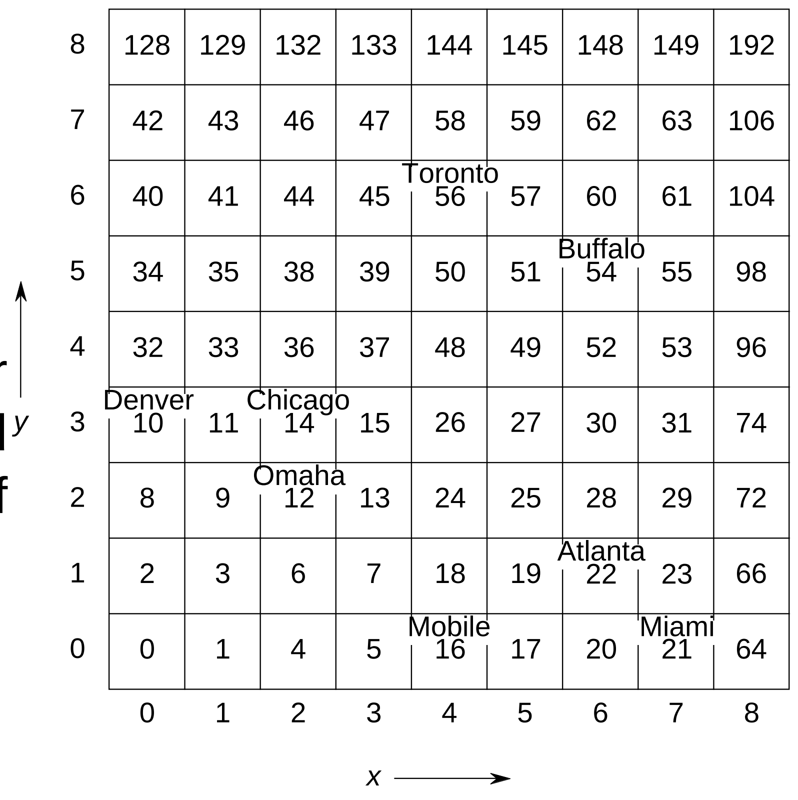
- Want a mapping f so that $d(v,u) \approx d'(f(v),f(u))$ where d' is the distance in the transformed space
- Range searching
  1. reduce query radius
     - implies more precision as reduce false hits
  2. increase query radius
     - implies more recall as reduce false dismissals
  3. $d'(f(a),f(b)) \leq d(a,b)$ for any pair of objects a and b
     - mapping f is contractive and 100% recall

Nearest Neighbors in a Dimensionally-Reduced Space

1. Ideally $d(a,b) \leq d(a,c)$ implies $d'(f(a),f(b)) \leq d'(f(a),f(c))$, for any objects a, b, and c
   - proximity preserving property
   - implies that nearest neighbor queries can be performed directly in the transformed space
   - rarely holds
     a. holds for translation and scaling with any Minkowski metric
     b. holds for rotation when using Euclidean metric in both original and transformed space
2. Use “filter-and-refine” algorithm with no false dismissals (i.e., 100% recall) as long as f is contractive
   - if o is nearest neighbor of q, contractiveness ensures that ‘filter’ step finds all candidate objects o' such that $d'(f(q),f(o')) \leq d(q,o)$
   - ‘refine’ step calculates actual distance to determine actual nearest neighbor

Using only One Dimension

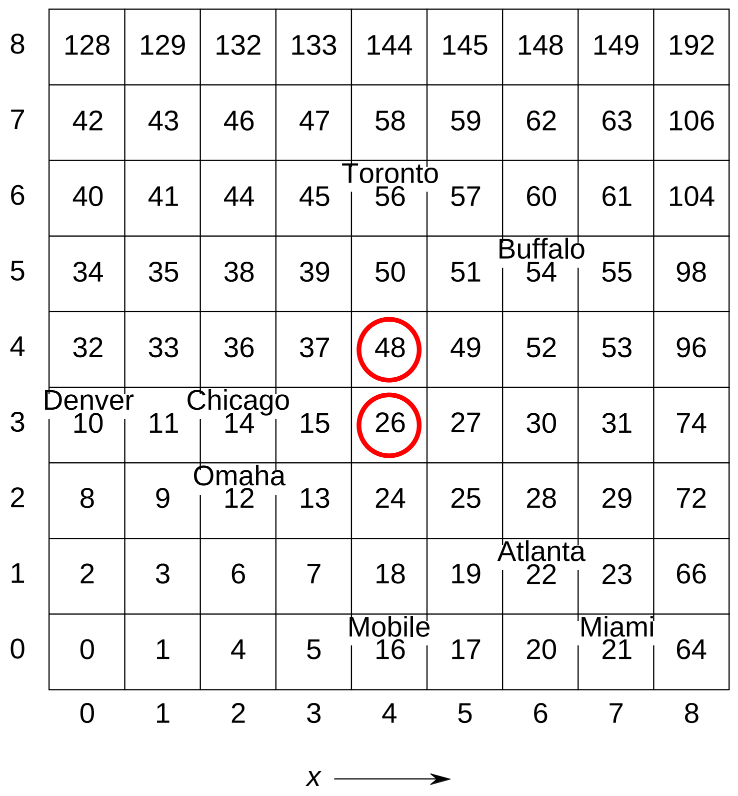
1. Can keep just one of the features
   - global: feature f with largest range
   - local: feature f with largest range of expected values about the value of feature f for query object q
   - always contractive if distance metric for the single feature is suitably derived from the distance metric used on all of the features
2. Combine all features into one feature
   - concatenate a few bits from each feature
   - use bit interleaving or Peano-Hilbert code



y									
8	128	129	132	133	144	145	148	149	192
7	42	43	46	47	58	59	62	63	106
6	40	41	44	45	56	57	60	61	104
5	34	35	38	39	50	51	54	55	98
4	32	33	36	37	48	49	52	53	96
3	10	11	14	15	26	27	30	31	74
2	8	9	12	13	24	25	28	29	72
1	2	3	6	7	18	19	22	23	66
0	0	1	4	5	16	17	20	21	64
x	0	1	2	3	4	5	6	7	8

Using only One Dimension

1. Can keep just one of the features
   - global: feature f with largest range
   - local: feature f with largest range of expected values about the value of feature f for query object q
   - always contractive if distance metric for the single feature is suitably derived from the distance metric used on all of the features
2. Combine all features into one feature
   - concatenate a few bits from each feature
   - use bit interleaving or Peano-Hilbert code
   - not contractive: points (4,3) and (4,4) are adjacent, but codes 26 and 48 are not!

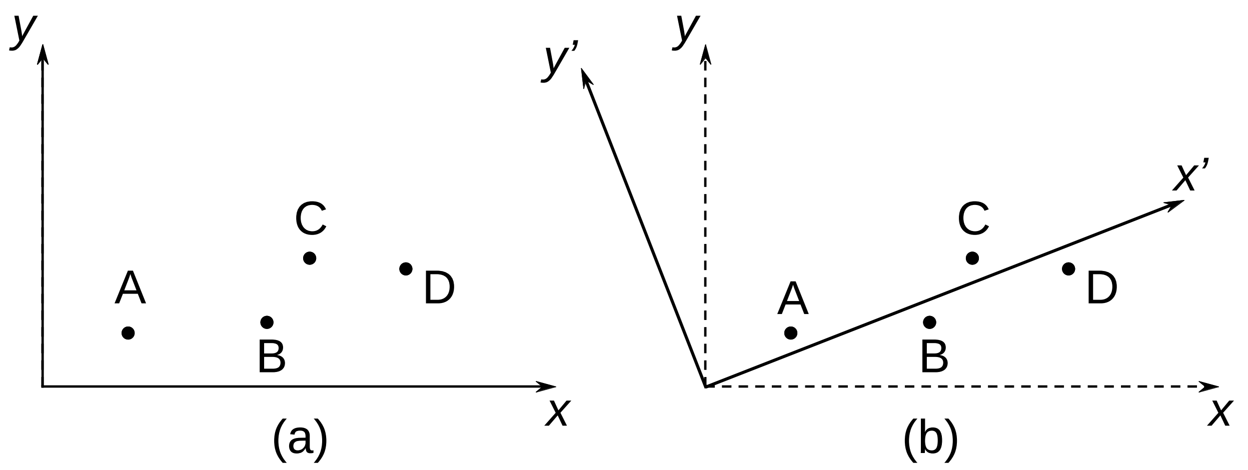


y \ x	0	1	2	3	4	5	6	7	8
8	128	129	132	133	144	145	148	149	192
7	42	43	46	47	58	59	62	63	106
6	40	41	44	45	Toronto 56	57	60	61	104
5	34	35	38	39	50	51	Buffalo 54	55	98
4	32	33	36	37	48	49	52	53	96
3	Denver 10	11	Chicago 14	15	26	27	30	31	74
2	8	9	Omaha 12	13	24	25	28	29	72
1	2	3	6	7	18	19	Atlanta 22	23	66
0	0	1	4	5	Mobile 16	17	20	Miami 21	64

Representative Points

- Often, objects with spatial extent are represented by a representative point such as a sphere by its radius and the coordinate values of its center
- Really a transformation into a point in a higher dimensional space and thus not a dimensional reduction
- Transformation is usually not contractive as distance between the transformed objects is greater than the distance between the original objects

Transformation into a Different and Smaller Feature Set



- Rotate x, y axes to obtain x', y' axes
- x' is dominant axis and can even drop axis y'

SVD (KLT, PCA)

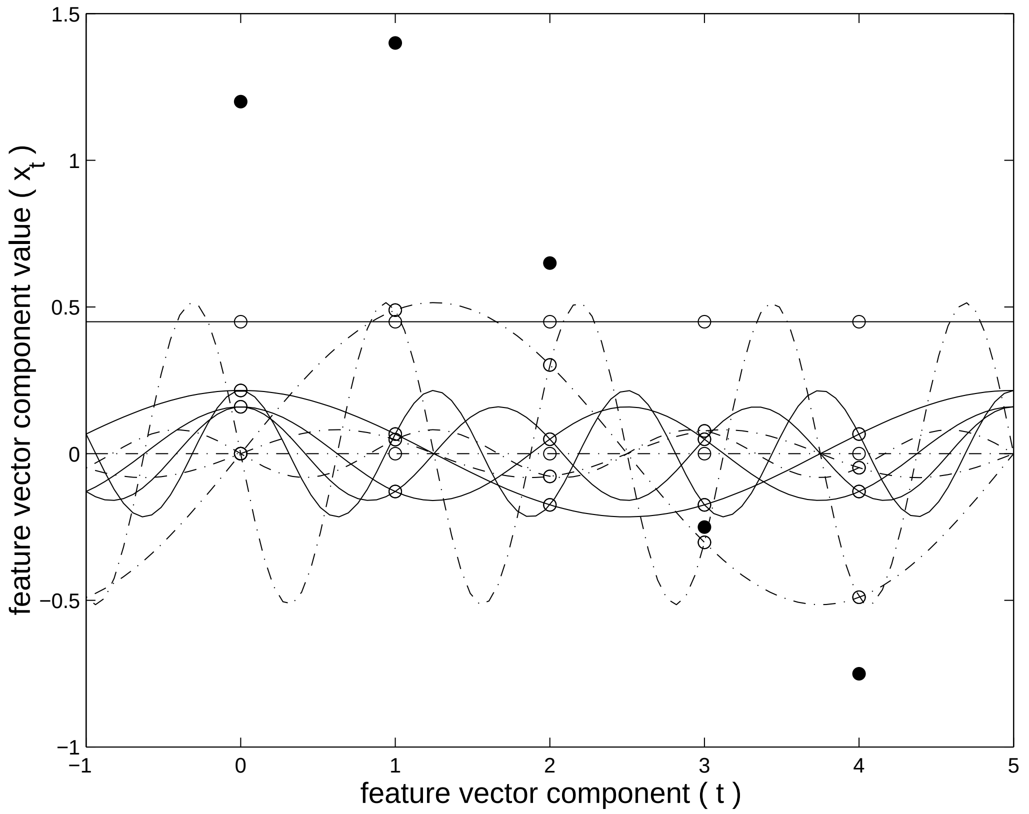
- Method of finding a linear transformation of n-dimensional feature vectors that yields good dimensionality reduction
  1. after transformation, project feature vectors on “first” k axes, yielding k-dimensional vectors ($k \leq n$)
  2. projection minimizes the sum of the squares of the Euclidean distances between the set of n-dimensional feature vectors and their corresponding k-dimensional feature vectors
- Letting F denote the original feature vectors, calculate V, the SVD transform matrix, and obtain transformed feature vectors T so that $FV = T$
- $F = U\Sigma V^T$ and retain the k most discriminating values in Σ (i.e., the largest ones and zeroing the remaining ones)
- Start with m n-dimensional points
- Drawback is the need to know all of the data in advance which means that need to recompute if any of the data values change
- Transformation preserves Euclidean distance and thus projection is contractive

Discrete Fourier Transform (DFT)

- Drawback of SVD: need to recompute when one feature vector is modified
- DFT is a transformation from time domain to frequency domain or vice versa
- DFT of a feature vector has same number of components (termed *coefficients*) as original feature vector
- DFT results in the replacement of a sequence of values at different instances of time by a sequence of an equal number of coefficients in the frequency domain
- Analogous to a mapping from a high-dimensional space to another space of equal dimension
- Provides insight into time-varying data by looking into the dependence of the variation on time as well as its repeatability, rather than just looking at the strength of the signal (i.e., the amplitude) as can be seen from the conventional representation of the signal in the time domain

Invertibility of DFT

- Ex: decomposition of real-valued five-dimensional feature vector $\vec{x} =$ (1.2,1.4,0.65,-0.25,-0.75)



- Cosine basis functions are solid
- Sine basis functions are broken
- Solid circle shows original feature vector

Use of DFT for Similarity Searching

- Euclidean distance norm of feature vector and its DFT are equal
- Can apply a form of dimension reduction by eliminating some of the Fourier coefficients
- Zeroth coefficient is average of components of feature vector
- Hard to decide which coefficients to retain
  1. choose just the first k coefficients
  2. find dominant coefficients (i.e., highest magnitude, mean, variance, etc.)
     - requires knowing all of the data and not so dynamic

Part 4: Embedding Methods

1. Problem statement
2. Lipschitz embeddings
3. SparseMap
4. FastMap
5. MetricMap

Overview of Embedding Methods

1. Given a finite set of N objects and a distance metric d indicating distance between them
2. Find function F that maps N objects into a vector space of dimension k using a distance function d' in this space
   - ideally, k is low: $k \ll N$
   - computing F should be fast — $O(N)$ or $O(N \log N)$
   - avoid examining all $O(N^2)$ inter-object distance pairs
   - fast way of obtaining $F(o)$ given o
3. Problem setting also includes situation where the N original objects are described by an n-dimensional feature vector
4. Ideally, the distances between the objects are preserved exactly by the mapping F
   - exact preservation means that (S, d) and $(F(S), d')$ are isometric
   - possible when d and d' are both Euclidean, in which case it is always true when $k = N - 1$
   - difficult in general for arbitrary combinations of d and d' regardless of value of k

Exact Distance Preservation May Be Impossible

Ex: 4 objects a, b, c, e

1. $d(a,b) = d(b,c) = d(a,c) = 2$ and $d(e,a) = d(e,b) = d(e,c) = 1.1$
   - d satisfies triangle inequality
   - Cannot embed objects into a 3-d Euclidean space — that is, with d' as the Euclidean distance while preserving d
2. Can embed if distance between e and a, b, and c is at least $2/\sqrt{3}$
   - place a, b, and c in plane p and place e on line perpendicular to p that passes through the centroid of the triangle in p formed by a, b, and c
3. Also possible if use City Block distance metric d_A (L_1)
   - place a, b, and c at (0,0,0), (2,0,0), and (1,1,0), respectively, and e at (1,0,0.1)

Exact Distance Preservation Always Possible with Chessboard Distance

- One dimension for each object
- Map object o into vector $\{d(o,o_1),d(o,o_2),\ldots,d(o,o_N)\}$
- For any pair of objects o_i and o_j,

$$d'(F(o_i),F(o_j)) = d_M(F(o_i),F(o_j)) = \max_l\{|d(o_i,o_l)-d(o_j,o_l)|\}$$

- For any l, $|d(o_i,o_l)-d(o_j,o_l)| \leq d(o_i,o_j)$ by the triangle inequality
- $|d(o_i,o_l)-d(o_j,o_l)| = d(o_i,o_j)$ for $l=i$ and $l=j$ in which case $d'(F(o_i),F(o_j)) = d(o_i,o_j)$
- Therefore, distances are preserved by F when using the Chessboard metric d_M (L_∞)
- Number of dimensions here is high: $k=N$
- At times, define F in terms of a subset of the objects

Properties of Embeddings

1. Contractiveness:
   - $d'(F(a), F(b)) \leq d(a, b)$
   - alternative to exact distance preservation
   - ensures 100% recall when use the same search radius in both the original and embedding space as no correct responses are missed
   - but precision may be less than 100% due to false candidates
2. Distortion: measures how much larger or smaller the distances in the embedding space $d'(F(o_1), F(o_2))$ are than the corresponding distances $d(o_1, o_2)$ in the original space
   - defined as $c_1 c_2$ where $\frac{1}{c_1} \cdot d(o_1, o_2) \leq d'(F(o_1), F(o_2)) \leq c_2 \cdot d(o_1, o_2)$ for all object pairs o_1 and o_2 where $c_1, c_2 \geq 1$
   - similar effect to contractiveness
3. SVD is optimal way of linearly transforming n-dimensional points to k-dimensional points ($k \leq n$)
   - ranks features by importance
   - drawbacks:
     a. can't be applied if only know distance between objects
     b. slow: $O(N \cdot m^2)$ where m is dimension of original space
     c. only works if d and d' are the Euclidean distance

Lipschitz Embeddings (Linial et al.)

- Based on defining a coordinate space where each axis corresponds to a reference set which is a subset of the objects
- Definition
  1. set R of subsets of S, $R = \{A_1, A_2, \ldots, A_k\}$
  2. $d(o, A) = \min_{x \in A}\{d(o, x)\}$ for $A \subset S$
  3. $F(o) = (d(o, A_1), d(o, A_2), \ldots, d(o, A_k))$
     - coordinate values of o are distances from o to the closest element in each of A_i
     - saw one such embedding earlier using L_∞ where R is all singleton subsets of S — that is, $R = \{\{o_1\}, \{o_2\}, \ldots, \{o_N\}\}$

Motivation for Lipschitz Embeddings

- If x is an arbitrary object, can obtain some information about $d(o_1, o_2)$ for arbitrary objects o_1 and o_2 by comparing $d(o_1, x)$ and $d(o_2, x)$ — that is, $|d(o_1, x) - d(o_2, x)|$
- $|d(o_1, x) - d(o_2, x)| \leq d(o_1, o_2)$ by Lemma 1
- Extend to subset A so that $|d(o_1, A) - d(o_2, A)| \leq d(o_1, o_2)$
- Proof:
  1. let $x_1, x_2 \in A$ be such that $d(o_1, A) = d(o_1, x_1)$ and $d(o_2, A) = d(o_2, x_2)$
  2. $d(o_1, x_1) \leq d(o_1, x_2)$ and $d(o_2, x_2) \leq d(o_2, x_1)$ implies $|d(o_1, A) - d(o_2, A)| = |d(o_1, x_1) - d(o_2, x_2)|$
  3. $d(o_1, x_1) - d(o_2, x_2)$ can be positive, while a negative value implies $|d(o_1, x_1) - d(o_2, x_2)| \leq \max\{|d(o_1, x_1) - d(o_2, x_1)|, |d(o_1, x_2) - d(o_2, x_2)|\}$
  4. from triangle inequality, $\max\{|d(o_1, x_2) - d(o_2, x_2)|, |d(o_1, x_1) - d(o_2, x_1)|\} \leq d(o_1, o_2)$
  5. therefore, $|d(o_1, A) - d(o_2, A)| \leq d(o_1, o_2)$ as $|d(o_1, A) - d(o_2, A)| = |d(o_1, x_1) - d(o_2, x_2)|$
- By using R of the subsets, we increase likelihood that $d(o_1, o_2)$ is captured by $d'(F(o_1), F(o_2))$

Mechanics of Lipschitz Embeddings

- Linial et al. let R be $O(\log^2 N)$ randomly selected subsets of S
- For $d' = L_p$, define F so that

  $F(o) = (d(o,A_1)/q, d(o,A_2)/q, \ldots, d(o,A_k)/q)$, where $q = k^{1/p}$
- F satisfies $\frac{c}{\lfloor \log_2 N \rfloor} \cdot d(o_1,o_2) \leq d'(F(o_1),F(o_2)) \leq d(o_1,o_2)$
- Distortion of $O(\log N)$ is large and may make F ineffective at preserving relative distances as want to use distance value in original space
- Since sets A_i are chosen at random, proof is probabilistic and c is a constant with high probability
- Embedding is impractical
  1. large number and sizes of subsets in R mean that there is a high probability that all N objects appear in a subset of R
     - implies need to compute distance between query object q and all objects in S
  2. number of coordinates is $\lfloor \log_2 N \rfloor^2$, which is relatively large
     - $N = 100$ yields $k = 36$ which is too high

SparseMap (Hristescu/Farach-Colton)

- Attempts to overcome high cost of computing Lipschitz embedding of Linial in terms of number of distance computations and dimensions
- Uses regular Lipschitz embedding instead of Linial et al. embedding
  1. does not divide the distances $d(o, A_i)$ by $k^{1/p}$
  2. uses Euclidean distance metric
- Two heuristics
  1. reduce number of distance computations by calculating an upper bound $\hat{d}(o, A_i)$ instead of the exact value $d(o, A_i)$
     - only calculate a fixed number of distance values for each object as opposed to $|A_i|$ distance values
  2. reduce number of dimensions by using a “high quality” subset of R instead of the entire set
     - use greedy resampling to reduce number of dimensions by eliminating poor reference sets
- Heuristics do not lead to a contractive embedding but can be made contractive (Hjaltason and Samet)
  1. modify first heuristic to compute actual value $d(o, A_i)$, not upper bound
  2. use d_M (L_∞) as d' instead of d_E (L_2)

FastMap (Faloutsos/Lin)

- Inspired by dimension reduction methods for Euclidean space based on linear transformations such as SVD, KLT, PCA
- Claimed to be general but assumes that d is Euclidean as is d' and only for these cases is it contractive
- Objects are assumed to be points

Mechanics of FastMap

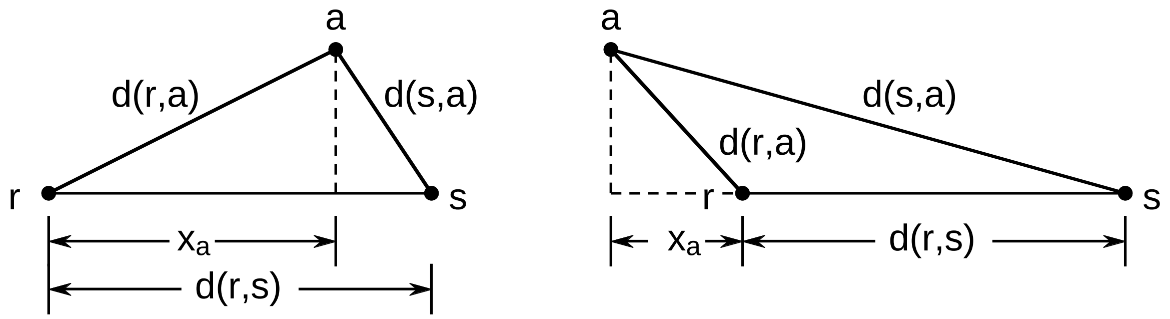
- Obtain coordinate values for points by projecting them on k mutually orthogonal coordinate axes
- Compute projections using the given distance function d
- Construct coordinate axes one-by-one
  1. choose two objects (pivots) at each iteration
  2. draw a line between them that serves as the coordinate axis
  3. determine coordinate value along this axis for each object o by mapping (i.e., projecting) o onto this line
- Prepare for next iteration
  1. determine the $(m-1)$-dimensional hyperplane H perpendicular to the line that forms the previous coordinate axis
  2. project all of the objects onto H
     - perform projection by defining a new distance function d_H measuring distance between projections of objects on H
     - d_H is derived from original distance function d and coordinate axes determined so far
  3. recur on original problem with m and k reduced by one, and a new distance function d_H
     - continue process until have enough coordinate axes

Choosing Pivot Objects

- Pivot objects serve to anchor the line that forms the newly-formed coordinate axis
- Ideally want a large spread of the projected values on the line between the pivot objects
  1. greater spread generally means that more distance information can be extracted from the projected values
     - for objects a and b, more likely that $|x_a - x_b|$ is large, thereby providing more information
  2. similar to principle in KLT but different as spread is weaker notion than variance which is used in KLT
     - large spread can be caused by a few outliers while large variance means values are really scattered over a wide range
- Use an $O(N)$ heuristic instead of $O(N^2)$ process for finding approximation of farthest pair
  1. arbitrarily choose one of the objects a
  2. find object r which is farthest from a
  3. find object s which is farthest from r
     - could iterate more times to obtain a better estimate of farthest pair

Deriving First Coordinate Value

1. Two possible positions for projection of object for first coordinate



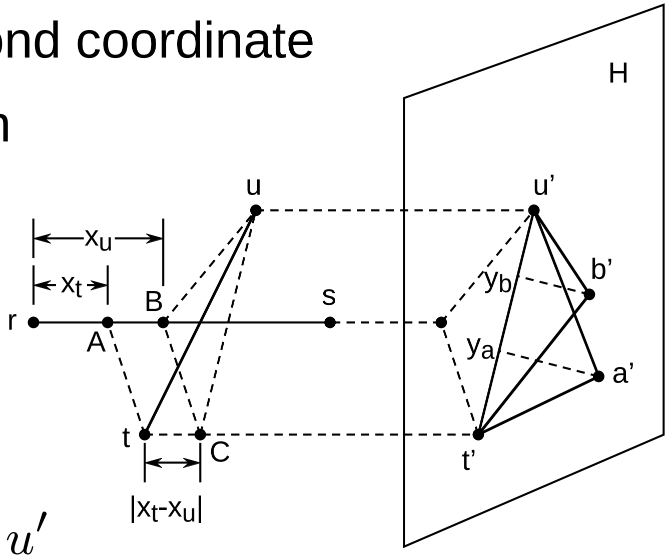
   - x_a obtained by solving $d(r,a)^2 - x_a^2 = d(s,a)^2 - (d(r,s) - x_a)^2$
   - Expanding and rearranging yields $x_a = \frac{d(r,a)^2 + d(r,s)^2 - d(s,a)^2}{2d(r,s)}$

2. Used Pythagorean Theorem which is only applicable to Euclidean space
   - implicit assumption that d is Euclidean distance
   - equation is only a heuristic when used for general metric spaces
   - implies embedding may not be contractive

3. Observations about x_a
   - can show $|x_a| \leq d(r,s)$
   - maximum spread between arbitrary a and b is $2d(r,s)$
   - bounds may not hold if d is not Euclidean as then the distance function used in subsequent iterations may possibly not satisfy triangle inequality

Projected Distance

- Ex: 3-d space and just before determining second coordinate
- d_H: distance function for the distances between objects when projected onto the hyperplane H perpendicular to the first coordinate axis (through pivots r and s)



- Determining $d_H(t,u)$ for some objects t and u:
  1. let t' and u' be their projections on H
     - $d_H(t,u)$ equals distance between t' and u'
     - also know: $d(t',u') = d(C,u)$
  2. angle at C in triangle tuC is 90°, so can apply Pythagorean theorem:

$$d(t,u)^2 = d(t,C)^2 + d(C,u)^2 = (x_t - x_u)^2 + d(t',u')^2$$

  3. rearranging and $d_H(t,u) = d(t',u')$ yields

$$d_H(t,u)^2 = d(t,u)^2 - (x_t - x_u)^2$$

- Implicit assumption that d is Euclidean distance

Side-Effects of Non-Euclidean Distance d

1. d_H can fail to satisfy triangle inequality
   - produce coordinate values that lead to non-contractiveness
2. Non-contractiveness may cause negative values of $d_H(a, b)^2$
   - complicates search for pivot objects
   - problem: square root of negative number is a complex number which means that a and b (really their projections) cannot serve as pivot objects

Subsequent Iterations

- Distance function at iteration i is the distance function d_H from previous iteration
- Notation:
  1. x_o^i: i^{th} coordinate value obtained for object o
  2. $F_i(o) = \{x_o^1, x_o^2, \ldots, x_o^i\}$: first i coordinate values of $F(o)$
  3. d_i: distance function used in iteration i
  4. p_1^i and p_2^i: two pivot objects chosen in iteration i
     - p_2^i is the farthest object from p_1^i
- $x_o^i = \frac{d_i(p_1^i,o)^2 + d_i(p_1^i,p_2^i)^2 - d_i(p_2^i,o)^2}{2d_i(p_1^i,p_2^i)}$
- Recursive distance function:

$$\begin{aligned} d_1(a,b) &= d(a,b) \\ d_i(a,b)^2 &= d_{i-1}(a,b)^2 - (x_a^{i-1} - x_b^{i-1})^2 \\ &= d(a,b)^2 - d_E(F_{i-1}(a), F_{i-1}(b))^2 \end{aligned}$$

Computational Complexity

1. $O(k \cdot N)$ distance computations to map N objects to k-dimensional space
   - $O(N)$ distance computations at each iteration
2. $O(k \cdot N)$ space to record the k coordinate values of each of the points corresponding to the N objects
3. $2 \times k$ array to record identities of k pairs of pivot objects, as this information is needed to process queries
4. Query objects are transformed to k-dimensional points by applying same algorithm used to construct points corresponding to original objects, except that we use existing pivot objects
   - $O(k)$ process as $o(k)$ distance computations
5. Can also record distance between pivot objects so no need to recompute

Properties of FastMap

1. Contractiveness
   - yes as long as d and d' are both Euclidean

   a. no if d is Euclidean and d' is not
      - Ex: use city block distance d_A (L_1) for d' as $d_A((0,0),(3,4)) = 7$ while $d_E((0,0),(3,4)) = 5$

   b. no if d is not Euclidean regardless of d'
      - Ex: four objects, a through e, with distances $d(a,b) = 10$, $d(a,c) = 4$, $d(a,e) = 5$, $d(b,c) = 8$, $d(b,e) = 7$, and $d(c,e) = 1$
      - letting a and b be pivots in the first iterations, results in $x_e - x_c = 6/5 = 1.2 < 1 = d(c,e)$
   - if d non-Euclidean, then eventually non-contractive if enough iterations
2. With Euclidean distances, distance can preserved given enough iterations
   - $\min\{m, N-1\}$ for m-dimensional space and N points
3. Distance expansion can be very large if non-contractive
4. If d is not Euclidean, then d_H could violate triangle inequality
   - Ex: four objects, a through e, with distances $d(a,b) = d(c,e) = 6$, $d(a,c) = 5$, $d(a,e) = d(b,e) = 4$, and $d(b,c) = 3$
   - letting a and b be pivots, yields $d_H(a,c) + d_H(a,e) \approx 5.141 < 5.850 \approx d_H(c,e)$, violating triangle inequality

Implications of Non-Contractiveness of FastMap

1. Not guaranteed to to be able to determine k coordinate axes
   - limits extent of distance preservation
   - failure to determine more coordinate axes does not necessarily imply that relative distances among the objects are effectively preserved
2. Distance distortion can be very large
3. Presence of many non-positive, or very small positive, distance values (which can cause large distortion) in the intermediate distance functions (i.e., those used to determine the second and subsequent coordinate axes) may cause FastMap to no longer satisfy the claimed $O(N)$ bound on the number of distance computations in each iteration
   - finding a legal pivot pair may, in the worst case, require examining the distances between a significant fraction of all possible pairs of objects, or $\Omega(N^2)$ distance computations

MetricMap (Wang et al.)

1. Similar to SVD, FastMap, and a special class of Lipschitz embeddings
   - in Euclidean spaces, equivalent to applying SVD for dimension reduction
   - based on an analogy to rotation and projection in Euclidean spaces
2. Differs from FastMap as embedding space is *pseudo-Euclidean*
   - some coordinate axes make a negative contribution to “distances” between the points
3. Makes use of $2k$ reference objects which form a coordinate space in a $(2k-1)$-dimensional space
   - one reference object is mapped to origin and rest are mapped to unit vectors in the $(2k-1)$-dimensional space
   - forms a matrix that preserves distance between reference objects
4. Mapping each object is less expensive than in FastMap
   - only need $k+1$ distance computations
5. Employs different strategy to handle non-Euclidean metrics
   - maps into a pseudo-Euclidean space, which may result in less distortion in the distances
   - may possibly not be contractive

Part 5: Nearest Neighbor Finding

1. Classical methods such as branch and bound
2. K nearest neighbors
3. Incremental nearest neighbor finding
    a. general method
    b. permitting duplicate instances of objects
4. Approximate nearest neighbor finding
5. Probably approximately correct (PAC) nearest neighbor finding

Branch and Bound Algorithm (Fukunaga/Narendra)

1. Visit elements in hierarchy using a depth-first traversal
   - maintain a list L of current candidate k nearest neighbors
2. D_k: distance between q and the farthest object in L
   - $D_k = \max_{o \in L}\{d(q,o)\}$), or ∞ if L contains fewer than k objects
   - D_k is monotonically non-increasing over the course of the search traversal, and eventually reaches the distance of the k^{th} nearest neighbor of q
3. If element e_t being visited represents an object o (i.e., $t = 0$), then insert o into L, removing farthest if $|L| > k$
4. Otherwise, e_t ($t \geq 1$) is not an object
   - construct an *active list* $A(e_t)$ of child elements of e_t, ordered by "distance" from q
   - recursively visit the elements in $A(e_t)$ in order, backtracking when
     a. all elements have been visited, or
     b. reaching an element $e_{t'} \in A(e_t)$ with $d_{t'}(q, e_{t'}) > D_k$
        - condition ensures that all objects at distance of k^{th} nearest neighbor are reported
        - if sufficient to report k objects, then use $d_{t'}(q, e_{t'}) \geq D_k$

Branch and Bound Enhancements

- Process elements of active list in an order more closely correlated with finding the k nearest neighbors
  1. process elements that are more likely to contain the k nearest neighbors before those that are less likely to do so
  2. possibly prune elements from further consideration by virtue of being farther away from the query object than any of the members of list L of the current candidate k nearest neighbors
     - in case of distance-based indexes for metric space searching, prune with aid of triangle inequality
- Can use cost estimate functions
  1. $\textsc{MinDistObject}(q, n)$ is least possible distance from query object q to an object in tree rooted at n
  2. $\textsc{MaxDistObject}(q, n)$ is greatest possible distance between q and an object in tree rooted at n
- When use a spatial index with bounding box hierarchies, then order on basis of minimum distance to the bounding box associated with each element

Incremental Nearest Neighbors (Hjaltason/Samet)

- Motivation
  1. often don’t know in advance how many neighbors will need
  2. e.g., want nearest city to Chicago with population > 1 million
- Several approaches
  1. guess some area range around Chicago and check populations of cities in range
     - if find a city with population > 1 million, must make sure that there are no other cities that are closer with population > 1 million
     - inefficient as have to guess size of area to search
     - problem with guessing is we may choose too small a region or too large a region
       a. if size too small, area may not contain any cities with right population and need to expand the search region
       b. if size too large, may be examining many cities needlessly
  2. sort all the cities by distance from Chicago
     - impractical as we need to re-sort them each time pose a similar query with respect to another city
     - also sorting is overkill when only need first few neighbors
  3. find k closest neighbors and check population condition

Mechanics of Incremental Nearest Neighbor Algorithm

- Make use of a search hierarchy (e.g., tree) where
  1. objects at lowest level
  2. object approximations are at next level (e.g., bounding boxes in an R-tree)
  3. nonleaf nodes in a tree-based index
- Traverse search hierarchy in a “best-first” manner similar to A*-algorithm instead of more traditional depth-first or breadth-first manners
  1. at each step, visit element with smallest distance from query object among all unvisited elements in the search hierarchy
     - i.e., all unvisited elements whose parents have been visited
  2. use a global list of elements, organized by their distance from query object
     - use a priority queue as it supports necessary insert and delete minimum operations
     - ties in distance: priority to lower type numbers
     - if still tied, priority to elements deeper in search hierarchy

Incremental Nearest Neighbor Algorithm

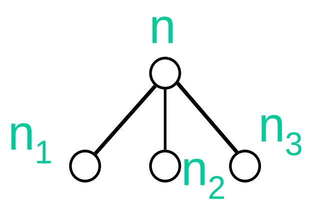
Algorithm:

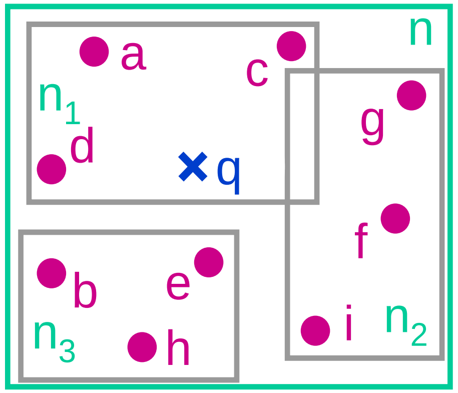
```
INCNEAREST($q$, $S$, $T$)
1  $Q \leftarrow$ NEWPRIORITYQUEUE()
2  $e_t \leftarrow$ root of the search hierarchy induced by $q$, $S$, and $T$
3  ENQUEUE($Q$, $e_t$, 0)
4  while not ISEMPTY($Q$) do
5     $e_t \leftarrow$ DEQUEUE($Q$)
6     if $t = 0$ then /* $e_t$ is an object */
7        Report $e_t$ as the next nearest object
8     else
9        for each child element $e_{t'}$ of $e_t$ do
10          ENQUEUE($Q$, $e_{t'}$, $d_{t'}(q,\ e_{t'})$)
```

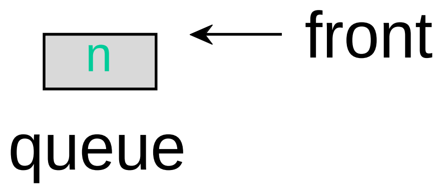
1. Lines 1-3 initialize priority queue with root
2. In main loop take element e_t closest to q off the queue
   - report e_t as next nearest object if e_t is an object
   - otherwise, insert child elements of e_t into priority queue

Example of IncNearest

- Initially, algorithm descends tree to leaf node containing q

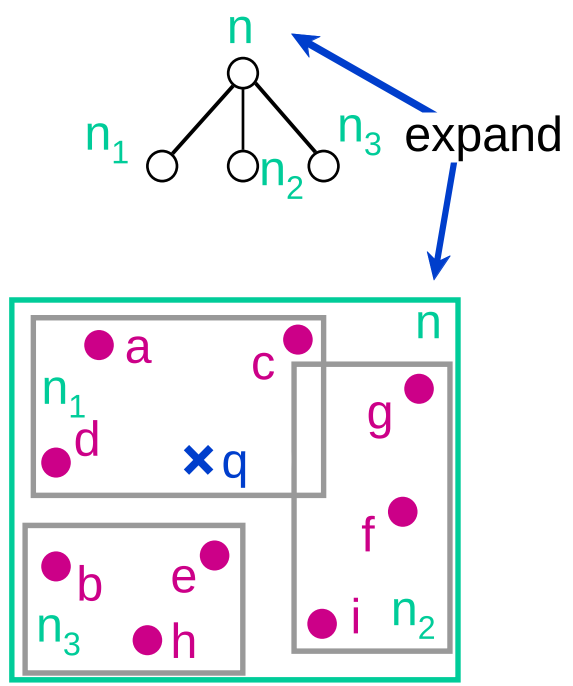


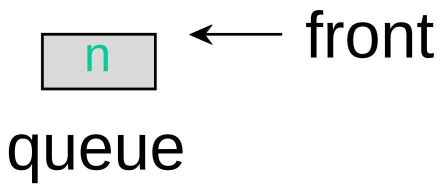




Example of INCNEAREST

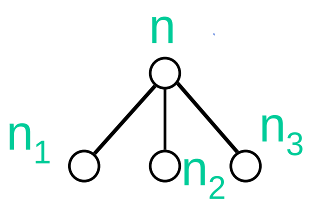
- Initially, algorithm descends tree to leaf node containing q
  - expand n

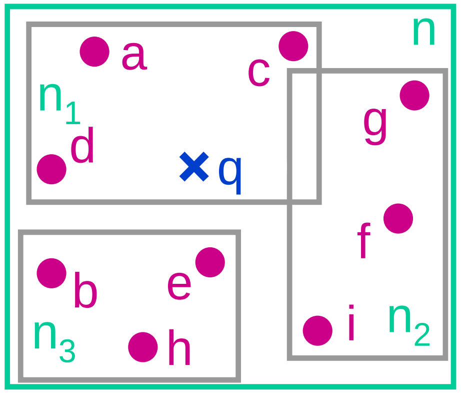


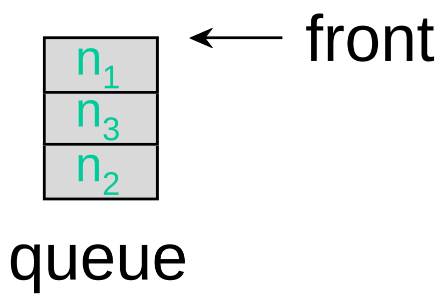


Example of INCNEAREST

- Initially, algorithm descends tree to leaf node containing q
  - expand n

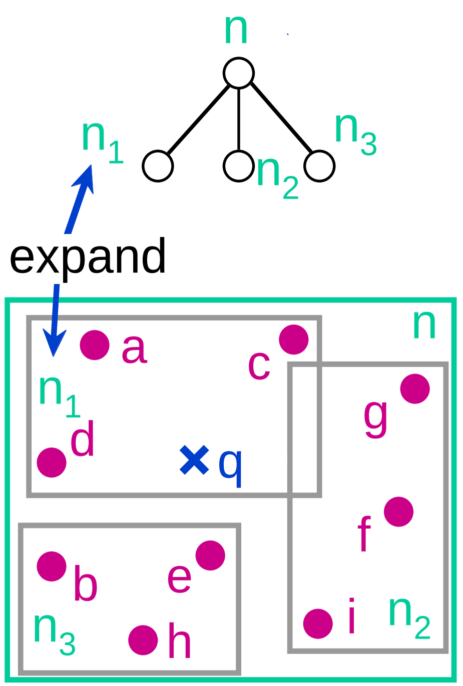


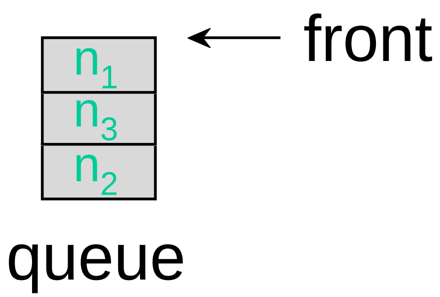




Example of IncNearest

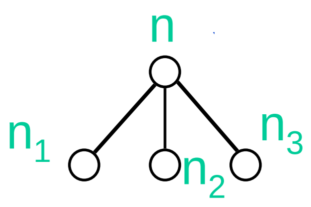
- Initially, algorithm descends tree to leaf node containing q
  - expand n
  - expand n_1

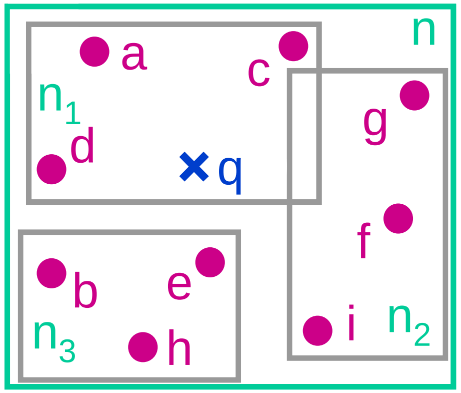


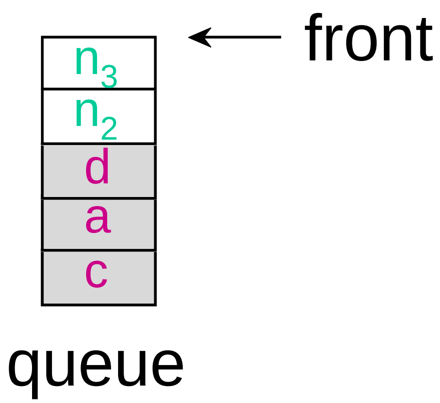


Example of INCNEAREST

- Initially, algorithm descends tree to leaf node containing q
  - expand n
  - expand n_1

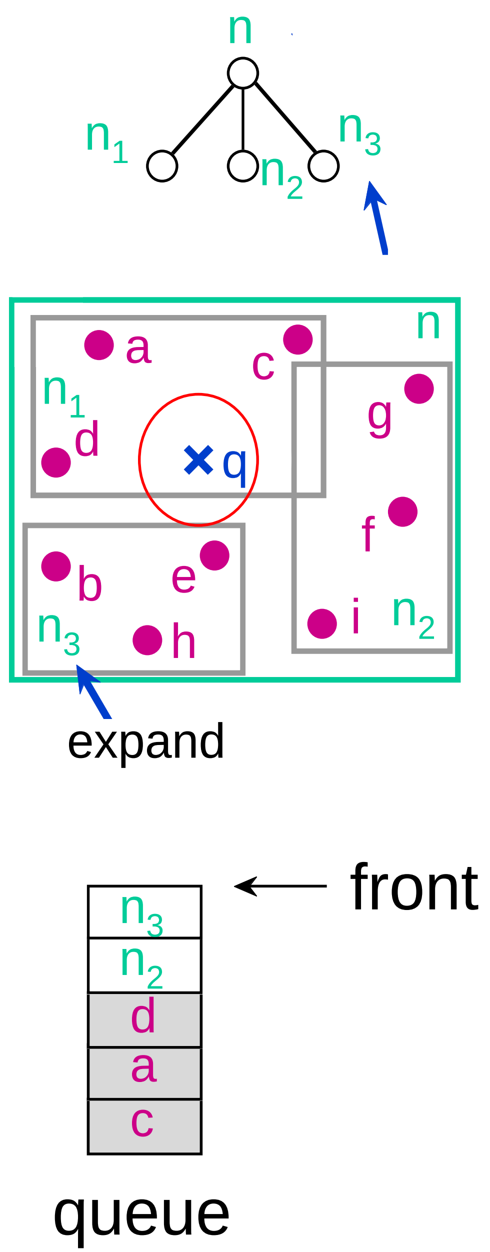






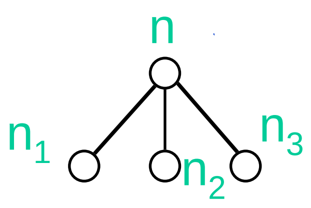
Example of IncNearest

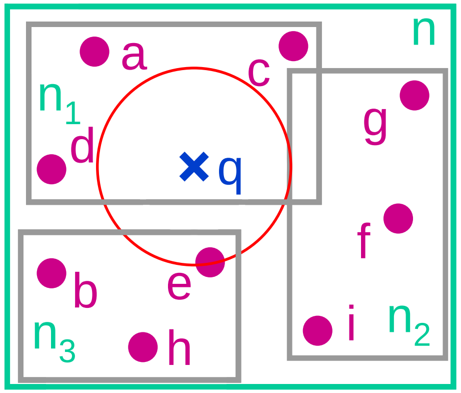
- Initially, algorithm descends tree to leaf node containing q
  - expand n
  - expand n_1
- Start growing search region
  - expand n_3

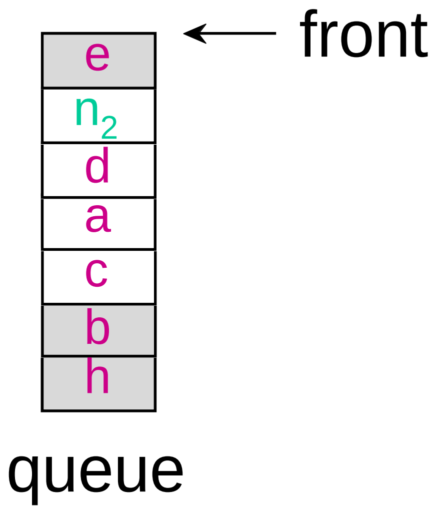


Example of IncNearest

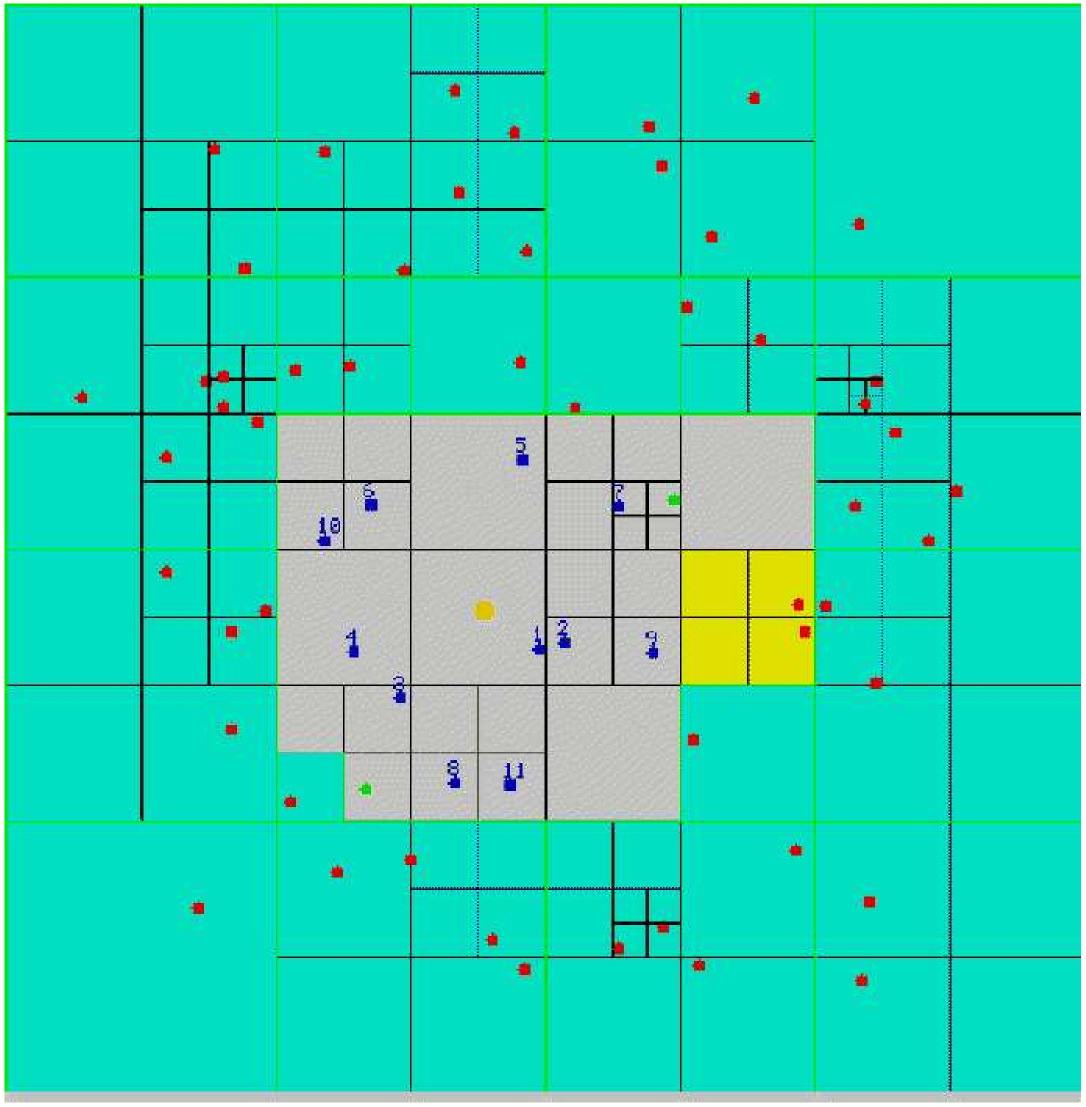
- Initially, algorithm descends tree to leaf node containing q
  - expand n
  - expand n_1
- Start growing search region
  - expand n_3
  - report e as nearest neighbor







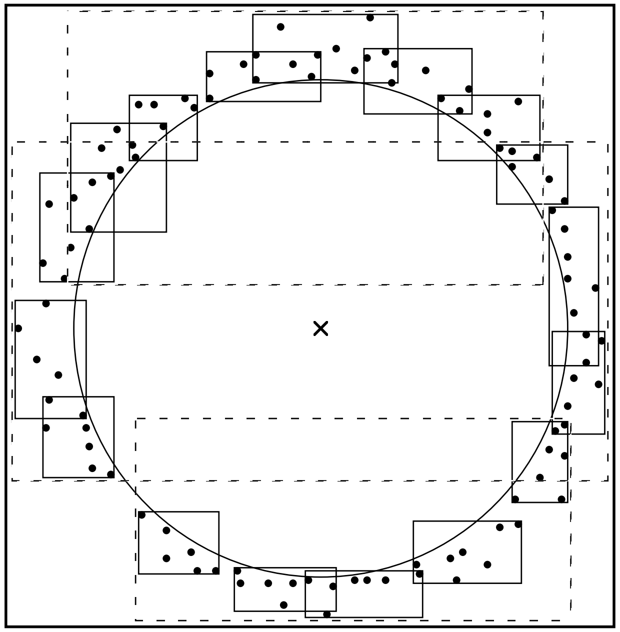
VASCO Spatial Applet



http://www.cs.umd.edu/ hjs/quadtree/index.html

Complexity Analysis

- Algorithm is I/O optimal
  - no nodes outside search region are accessed
  - better pruning than branch and bound algorithm
- Observations for finding k nearest neighbors for uniformly-distributed two-dimensional points
  - expected # of points on priority queue: $c \cdot \sqrt{k}$
  - expected # of leaf nodes intersecting search region: $c \cdot (k + \sqrt{k})$
- In worst case, priority queue will be as large as entire data set
  - e.g., when data objects are all nearly equidistant from query object
  - probability of worst case very low, as it depends on a particular configuration of both the data objects and the query object (but: curse of dimensionality!)



Duplicate Instances of Objects

- Objects with extent such as lines, rectangles, regions, etc. are indexed by methods that associate the objects with the different blocks that they occupy
- Indexes employ a disjoint decomposition of space in contrast to non-disjoint as is the case for bounding box hierarchies (e.g., R-tree)
- Search hierarchies will contain multiple references to some objects
- Adapting incremental nearest neighbor algorithm:
    1. make sure to detect all duplicate instances that are currently in priority queue
    2. avoid inserting duplicate instances of an object that has already been reported

Duplicate Instances Algorithm

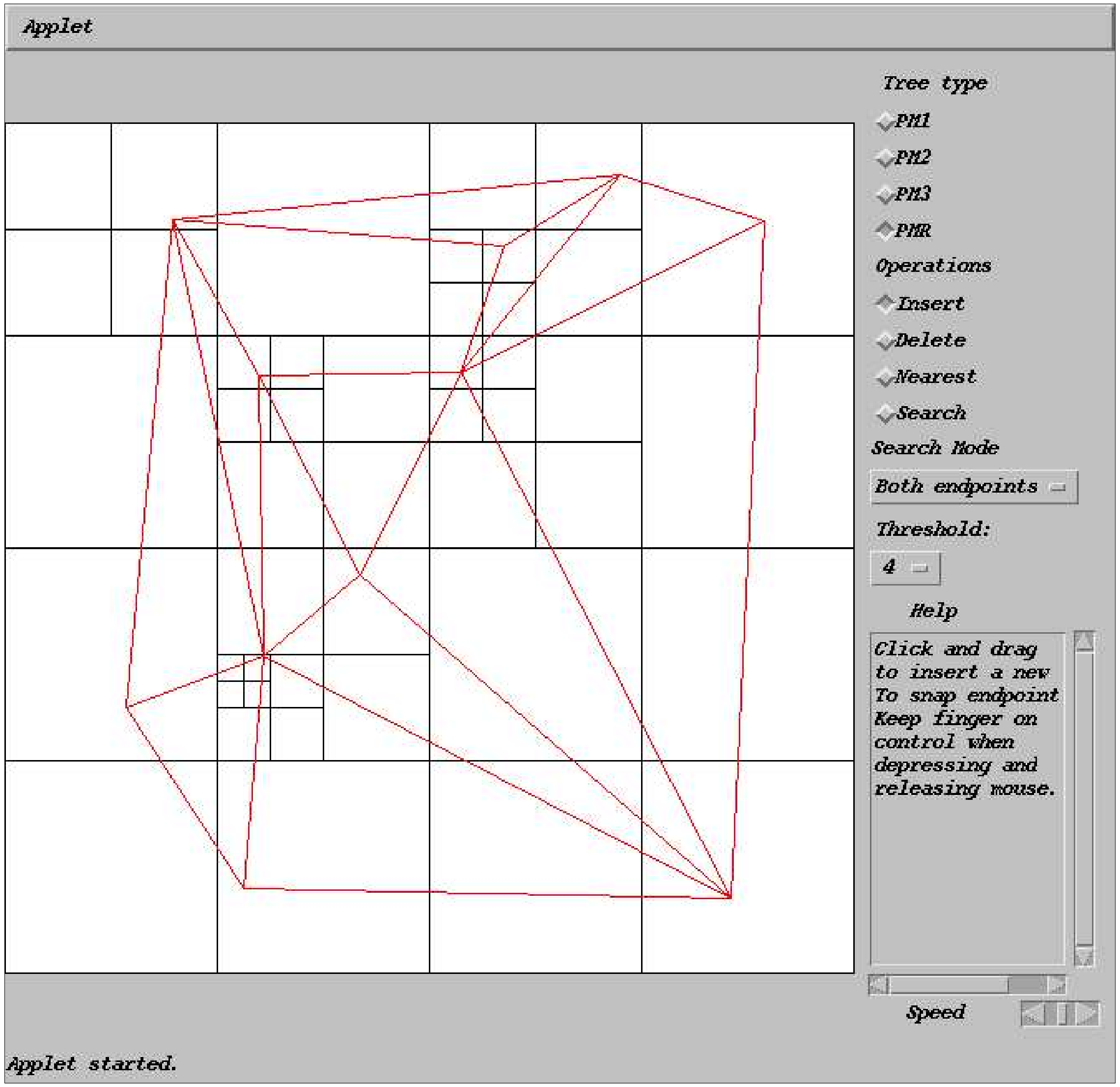
```
INCNEARESTDUP($q$, $S$, $T$)

 1  $Q \leftarrow$ NEWPRIORITYQUEUE()
 2  $e_t \leftarrow$ root of the search hierarchy induced by $q$, $S$, and $T$
 3  ENQUEUE($Q$, $e_t$, 0)
 4  while not ISEMPTY($Q$) do
 5    $e_t \leftarrow$ DEQUEUE($Q$)
 6    if $t = 0$ then /* $e_t$ is an object */
 7      while $e_t$ = FIRST($Q$) do
 8        DELETEFIRST($Q$)
 9      Report $e_t$ as the next nearest object
10    else /* $e_t$ is not an object */
11      for each child element $e_{t'}$ of $e_t$ do
12        if $t' > 0$ or $d_{t'}(q, e_{t'}) \geq d_t(q, e_t)$ then
13          ENQUEUE($Q$, $e_{t'}$, $d_{t'}(q, e_{t'})$)
```

Differences from INCNEAREST

1. Object o ($e_{t'}$) is enqueued only if o has not yet been reported
   - check if o's distance from q is less than distance from e_t to q (line 12)
   - if yes, then o must have been encountered in an element $e_{t''}$ which was closer to q and hence already been reported
2. Check for multiple instances of object o and report only once (lines 7–9)
3. Order objects in queue by identity when at same distance
4. Retrieve all nodes in the queue before objects at same distance
   - important because an object can have several ancestor nodes of the same type
   - interesting as unlike INCNEAREST where want to report neighbors as soon as possible so break ties by giving priority to elements with lower type numbers

VASCO Spatial Applet



Tree type: [✓ PMR]

Operations: [✓ Insert]

Search Mode: Both endpoints

Threshold: 4

Click and drag
to insert a new
To snap endpoint
Keep finger on
control when
depressing and
releasing mouse.

Applet started.

http://www.cs.umd.edu/ hjs/quadtree/index.html

INCNEAREST Extensions

1. Incremental range query
2. Incremental retrieval of k nearest neighbors
   - need an extra queue to keep track of k neighbors found so far and can use distance d_k from q of the k^{th} candidate nearest neighbor o_k to reduce number of priority queue operations
3. Farthest neighbor
4. Pairs of objects
   - distance join
   - distance semi-join

Approximate Nearest Neighbors

1. Often, obtaining exact results is not critical and willing to trade off accuracy for improved performance
2. Let ϵ denote the approximation error tolerance
   - common criterion is that the distance between q and the resulting candidate nearest neighbor o' is within a factor of $1+\epsilon$ of the distance to the actual nearest neighbor o
   - i.e., $d(q,o') \leq (1+\epsilon)d(q,o)$

Approximate Nearest Neighbors with INCNEAREST

1. Modify INCNEAREST by multiplying the key values for non-object elements on the priority queue by $1 + \epsilon$
   - in a practical sense, non-object element e_t is enqueued with a larger distance value — that is, by a factor of $(1 + \epsilon)$
   - implies that we delay its processing, thereby allowing objects to be reported ‘before their time’
   - e.g., once e_t is finally processed, all objects o satisfying $d(q, o) \leq (1 + \epsilon)d_t(q, e_t)$ (which is greater than $d_t(q, e_t)$ if $\epsilon > 0$) would have already been reported
   - thus an object c in e_t with a distance $d(q, c) \leq d(q, o)$ could exist, yet o is reported before c
   - algorithm does not necessarily report the resulting objects in strictly increasing order of their distance from q
2. Different from Arya/Mount algorithm which cannot be incremental as priority queue only contains non-object elements
   - shrinks distance r from q to the closest object o by a factor of $1 + \epsilon$ and only inserts a non-object element e into the priority queue if the distance $d(b, q)$ of e’s corresponding block b from q is less than the shrunken distance

Probably Approximately Correct (PAC) Nearest Neighbors (Ciaccia/Patella)

- Relax approximate nearest neighbor condition by stipulating a maximum probability δ for tolerating failure, thereby enabling the decision process to halt sooner at the risk δ of being wrong
- Object o' is considered a PAC-nearest neighbor of q if the probability that $d(q,o') \leq (1+\epsilon) \cdot d(q,o)$ is at least $1-\delta$, where o is actual nearest neighbor
- Alternatively, given ϵ and δ, $1-\delta$ is the minimum probability that o' is the $(1+\epsilon)$-approximate nearest neighbor of q
- Ciaccia and Patella use information about the distances between q and the data objects to derive an upper bound s on the distance between q and a PAC-nearest neighbor o'
- Distance bound s is used during the actual nearest neighbor search as a pre-established halting condition — that is, the search can be halted once locating an object o' with $d(q,o') \leq s$
- Method is analogous to executing a variant of a range query, where the range is defined by the distance bound s, which halts on the first object in the range
- Difficulty is determining a relationship between δ and the distance bound s

Concluding Remarks

1. Similarity search is a broad area of research
2. Much relation to geometry; geometric setting is usually missing
3. Progress is heavily influenced by applications
4. Need to look at old literature to be able to evaluate current research results
5. Much is left to do as difficult to say what is best solution

Selected Overview References

- H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.
- H. Samet. *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.
- V. Gaede and O. Günther. Multidimensional access methods. *ACM Computer Surveys*, 20(2):170–231, June 1998.
- C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, Sept. 2001.
- E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–322, Sept. 2001.
- G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, May 2003. Also University of Maryland Computer Science TR-4102.
- G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, Dec. 2003.
- H. Samet. *Foundations of Multidimensional and Metric Data Structures*, Morgan-Kaufmann, San Francisco, CA, 2006.