CSE 521
Algorithms
Spring 2003

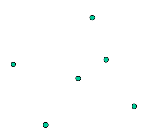
Randomized Incremental Algorithms
K-d Trees

Randomized Incremental Algorithms

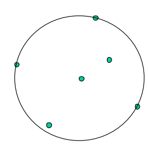
- Incremental Algorithm
  – Process the objects one at a time to solve problem
  – Objects might not be in an order causing bad worst case time complexity
- Randomized Incremental Algorithm
  – Permute the objects randomly
  – Objects not likely to be in a bad order
  – Good average time complexity

Smallest Enclosing Disc

- Given a set of points find the smallest enclosing disc.



Smallest Enclosing Disc

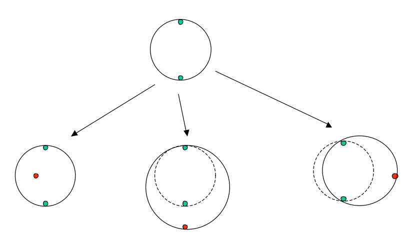


Smallest disc is unique

Nonincremental Algorithm

- For every two and three points construct the disc through the points and check that all the other points are inside.
- Pick the smallest of these discs.
- $O(n^4)$ time.

Incremental Algorithm



If the new point is outside the current disk then it is on the boundary of the new disc.

Incremental Algorithm 0

MinDisc0($\{p_1,p_2,\ldots,p_n\}$)
Let D_2 be the smallest disc containing p_1 and p_2
For i = 3 to n do
  if p_i in D_{i-1} then
    $D_i := D_{i-1}$
  else
    $D_i :=$ MinDisc1($\{p_1,p_2,\ldots,p_{i-1}\}$, p_i)
Return D_n

MinDisc1($\{p_1,p_2,\ldots,p_{i-1}\}$, p_i) returns the smallest disc that contains $\{p_1,p_2,\ldots,p_{i-1}\}$ with p_i on the boundary.

Incremental Algorithm 1

MinDisc1($\{p_1,p_2,\ldots,p_n\}$, p)
Let D_1 be the smallest disc containing p_1 and p
For i = 2 to n do
  if p_i in D_{i-1} then
    $D_i := D_{i-1}$
  else
    $D_i :=$ MinDisc2($\{p_1,p_2,\ldots,p_{i-1}\}$, p_i, p)
Return D_n

MinDisc1($\{p_1,p_2,\ldots,p_{i-1}\}$, p_i, p) returns the smallest disc that contains $\{p_1,p_2,\ldots,p_{i-1}\}$ with p_i and p on the boundary.

Incremental Algorithm 2

MinDisc2($\{p_1,p_2,\ldots,p_n\}$, p, q)
Let D_0 be the smallest disc containing p and q
For i = 1 to n do
  if p_i in D_{i-1} then
    $D_i := D_{i-1}$
  else
    $D_i :=$ the disc with p_i, p, q on the boundary
Return D_n

Crude Worst Case Time Analysis

- $T_i(n)$ = the running time of MinDisc(i) on n points
- $T_0(n) \le nT_1(n) + cn$
  $T_1(n) \le nT_2(n) + cn$
  $T_2(n) \le cn$
- By substitution
  $T_1(n) = O(n^2)$
  $T_0(n) = O(n^3)$

Better Analysis

- How often is MinDisc1 actually called in MinDisc0($\{p_1,p_2,\ldots,p_n\}$)?
- $T_0(n) \le T_1(i_1) + \ldots + T_1(i_k) + cn$
  - Where MinDisc1 called on just these values.
- Let's try to limit the number of calls to MinDisc1.

Randomized Incremental Algorithm

MinDisc0($\{p_1,p_2,\ldots,p_n\}$)
Randomly permute $\{p_1,p_2,\ldots,p_n\}$
Let D_2 be the smallest disc containing p_1 and p_2
For i = 3 to n do
  if p_i in D_{i-1} then
    $D_i := D_{i-1}$
  else
    $D_i :=$ MinDisc1($\{p_1,p_2,\ldots,p_{i-1}\}$, p_i)
Return D_n

Backwards Analysis

- What is the probability that MinDisc1 is called.
- Fix D_i to be the smallest disc containing $\{p_1,p_2,\ldots,p_i\}$
- Choose an element with equal probability to remove. What is the probability that smallest disc containing the remaining set is smaller?

  $\leq 3/i$

  because 3 boundary points determine D_i.
- The probability that MinDisc1($\{p_1,p_2,\ldots,p_{i-1}\}$, p_i) is called is $3/i$.

Backward Analysis

- By a similar argument, the probability that MinDisc2($\{p_1,p_2,\ldots,p_{i-1}\}$, p_i, p) is called is $2/i$.
- Expected time analysis of MinDisc1

  $E_1(n) \leq (2/2)T_2(2) + \ldots + (2/n)T_2(n) + cn$

  $\leq (2/2)\, c2 + \ldots + (2/n)\, cn + cn$

  $= 3cn$
- Similar analysis of MinDisc0

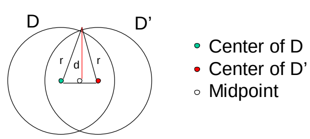
  $E_0(n) \leq (3/3)E_1(3) + \ldots + (3/n)E_1(n) + cn$

  $\leq (3/3)\, 3c3 + \ldots + (3/n)\, 3cn + cn$

  $= 10cn$

Disc Uniqueness

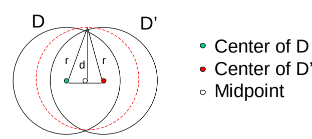
- All points are inside the circle centered at the midpoint between the centers of D and D'



d < r

Disc Uniqueness

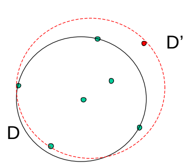
- Circle of radius d centered at midpoint contains all the points but is smaller.



d < r

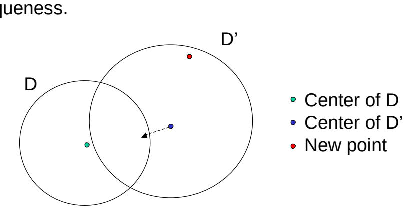
New Point Must Be on Boundary

- If p not in D then p on boundary of new smallest disc



New Point Must be on Boundary

- Move D' toward center of D. The new disc is the same size and contains all the points. Contradicting uniqueness.



Notes on Randomized Incremental Algorithms

- Randomized Incremental Algorithms first used for computing the intersection of half-planes. (Seidel 1991)
- Application to smallest disc problem. (Welzl 1991)

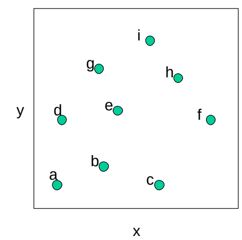
k-d Tree

- Jon Bentley, 1975
- Tree used to store spatial data.
  - Nearest neighbor search.
  - Range queries.
  - Fast look-up
- k-d tree are guaranteed $\log_2 n$ depth where n is the number of points in the set.
  - Traditionally, k-d trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

k-d Tree Construction

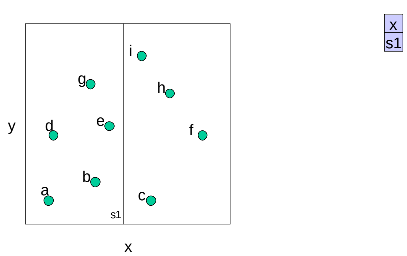
- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.
- Division strategies
  - divide points perpendicular to the axis with widest spread.
  - divide in a round-robin fashion.

k-d Tree Construction (1)

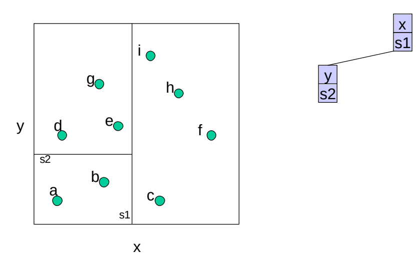


divide perpendicular to the widest spread.

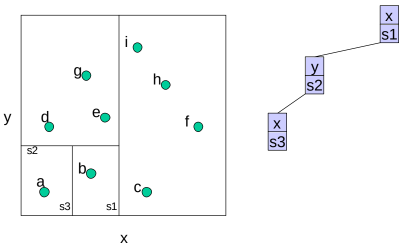
k-d Tree Construction (2)



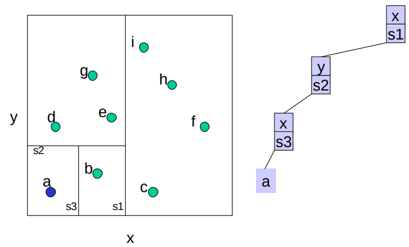
k-d Tree Construction (3)



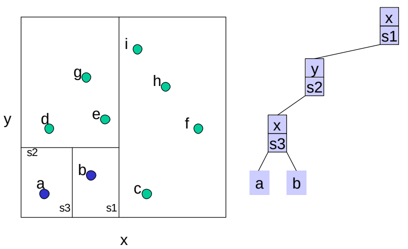
k-d Tree Construction (4)



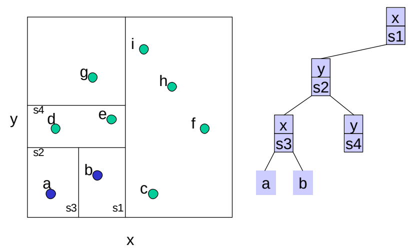
k-d Tree Construction (5)



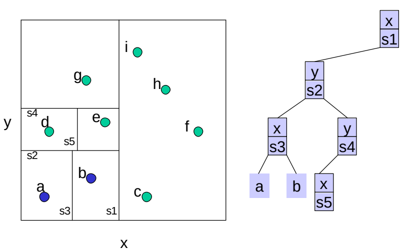
k-d Tree Construction (6)



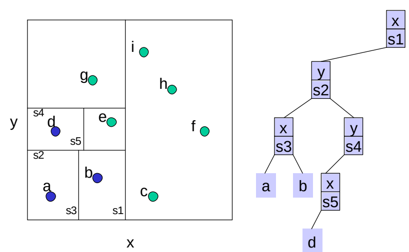
k-d Tree Construction (7)



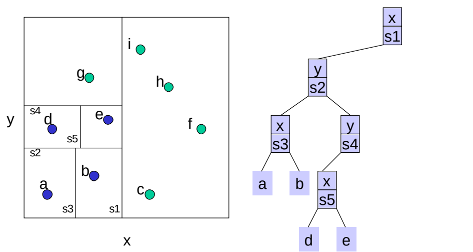
k-d Tree Construction (8)



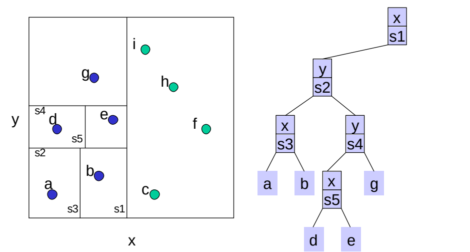
k-d Tree Construction (9)



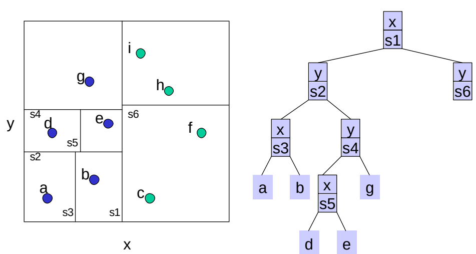
k-d Tree Construction (10)



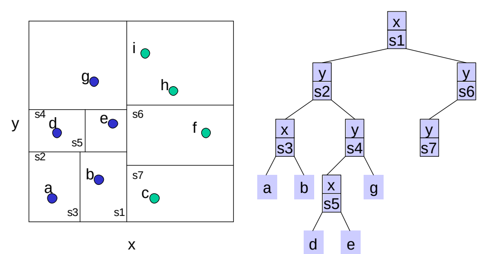
k-d Tree Construction (11)



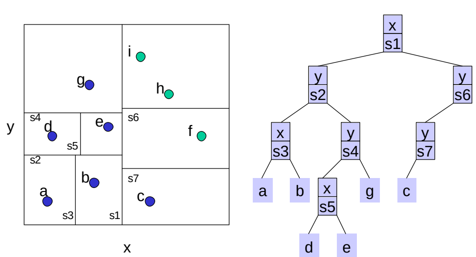
k-d Tree Construction (12)



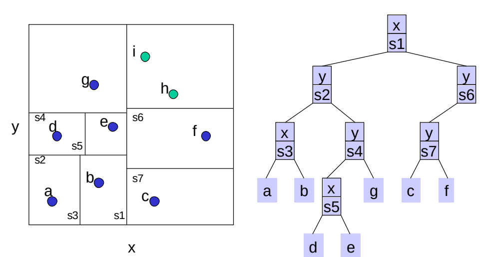
k-d Tree Construction (13)



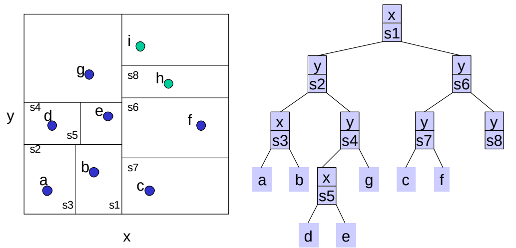
k-d Tree Construction (14)



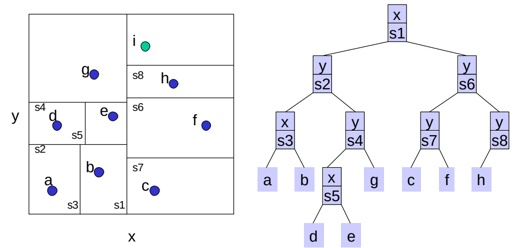
k-d Tree Construction (15)



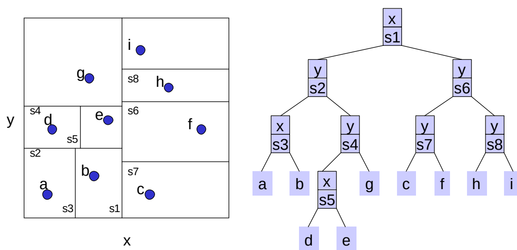
k-d Tree Construction (16)



k-d Tree Construction (17)



k-d Tree Construction (18)

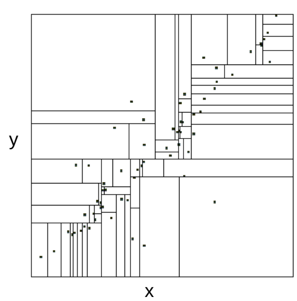


k-d Tree Construction Complexity

- First sort the points in each dimension.
  - O(dn log n) time and dn storage.
  - These are stored in A[1..d,1..n]
- Finding the widest spread and equally divide into two subsets can be done in O(dn) time.
- Constructing the k-d tree can be done in O(dn log n) and dn storage

k-d Tree Geometrically

2-d vectors
(x,y)



Node Structure for k-d Trees

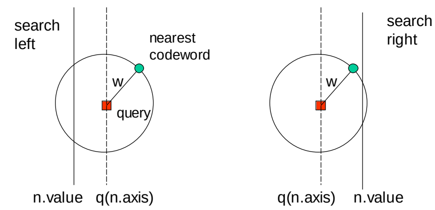
- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)

k-d Tree Nearest Neighbor Search

```
NNS(q: point, n: node, p: ref point w: ref distance)
if n.left = n.right = null then {leaf case}
   w' := ||q - n.point||;
   if w' < w then w := w'; p := n.point;
else
   if w = infinity then
       if q(n.axis) ≤ n.value then
           NNS(q, n.left, p, w);
           if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
       else
           NNS(q, n.right, p, w);
           if q(n.axis) - w ≤ n.value then NNS(q, n.left, p, w)
   else {w is finite}
       if q(n.axis) - w ≤ n.value then NNS(q, n.left, p, w)
       if q(n.axis) + w > n.value then NNS(q, n.right, p, w);
```

initial call NNS(q, root, p, infinity)

Explanation



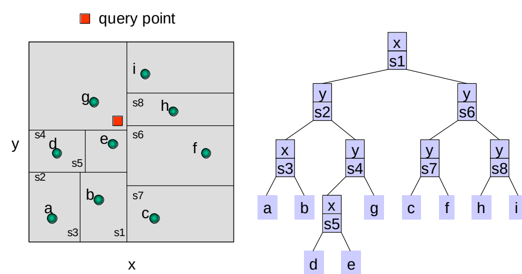
q(n.axis) – w ≤ n.value means the circle overlap the left subtree.

q(n.axis) + w > n.value means the circle overlap the right subtree.

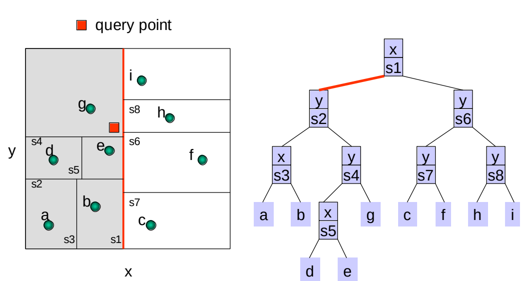
Nearest Neighbor Search

- Input q
- Find the leaf containing q and let w be the distance from the point in the leaf to q.
- Search each subtree recursively that may have a point of distance < w to q.
- Update w when a closer point is found.

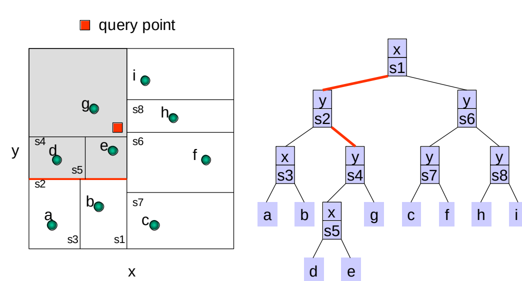
k-d Tree NNS (1)



k-d Tree NNS (2)

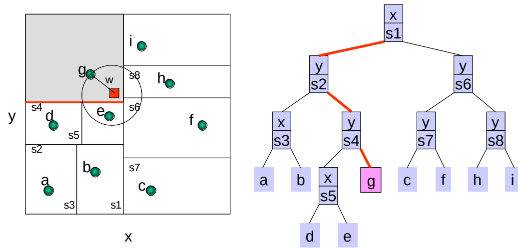


k-d Tree NNS (3)



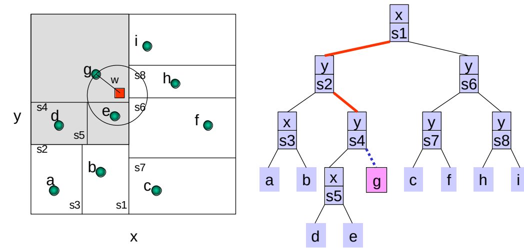
k-d Tree NNS (4)

query point



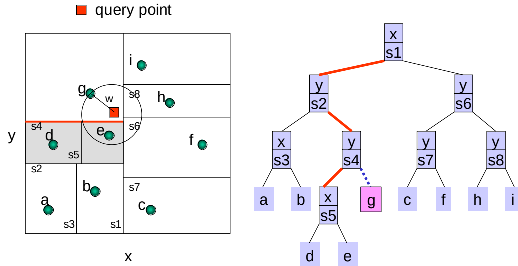
k-d Tree NNS (5)

query point



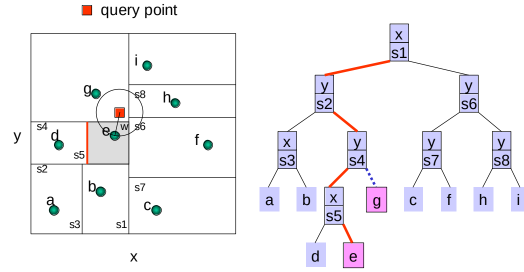
k-d Tree NNS (6)

query point



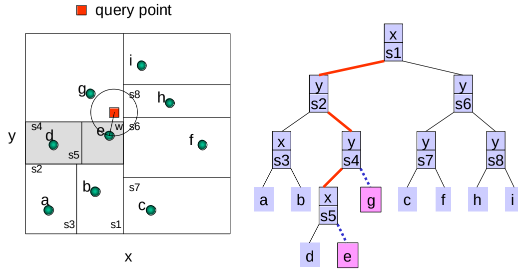
k-d Tree NNS (7)

query point



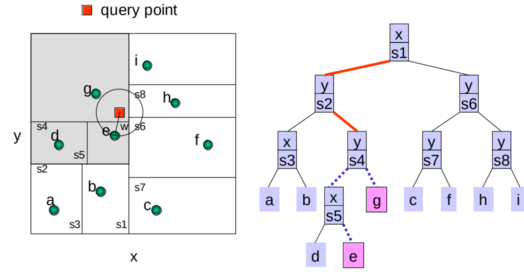
k-d Tree NNS (8)

query point



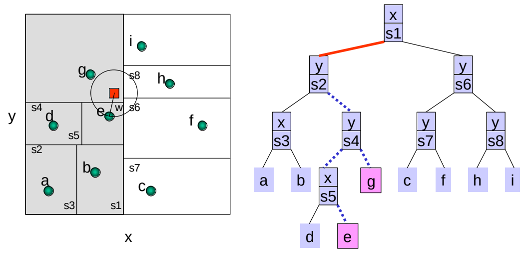
k-d Tree NNS (9)

query point



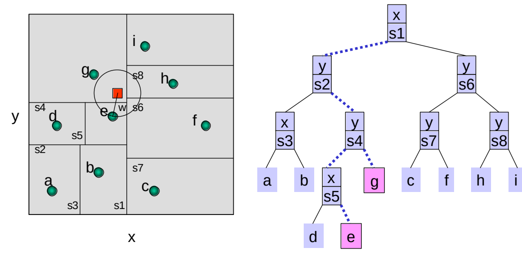
k-d Tree NNS (10)

query point



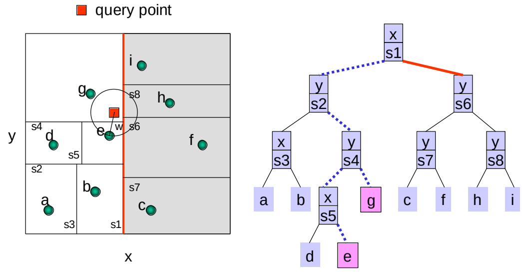
k-d Tree NNS (11)

query point



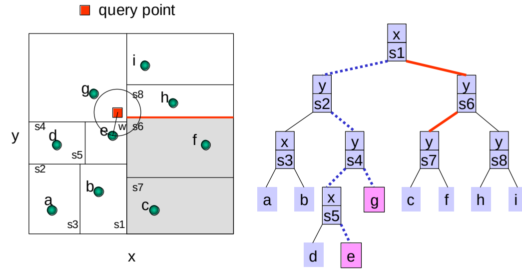
k-d Tree NNS (12)

query point



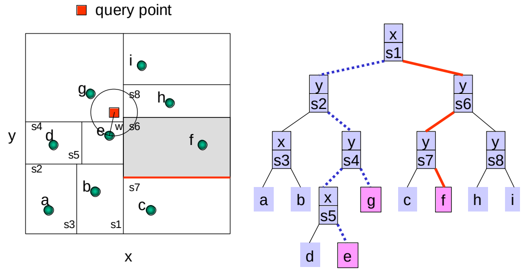
k-d Tree NNS (13)

query point



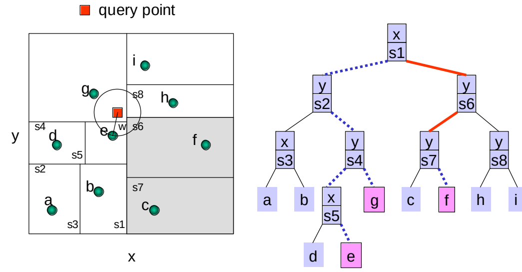
k-d Tree NNS (14)

query point

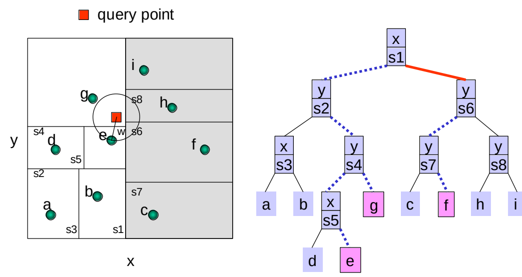


k-d Tree NNS (15)

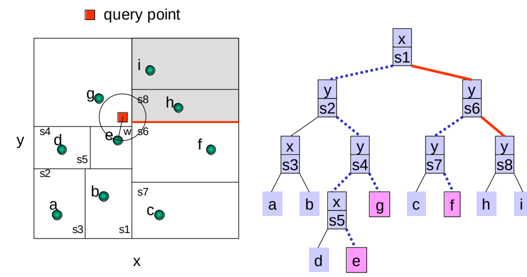
query point



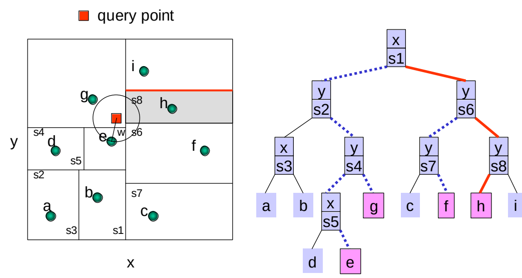
k-d Tree NNS (16)



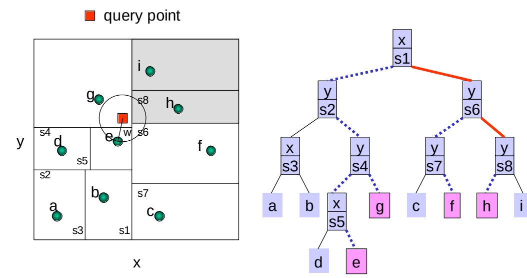
k-d Tree NNS (17)



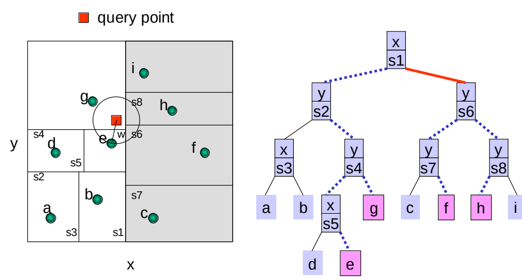
k-d Tree NNS (18)



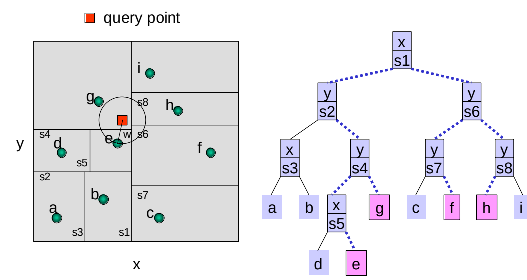
k-d Tree NNS (19)



k-d Tree NNS (20)



k-d Tree NNS (21)



Notes on k-d Tree NNS

- K-d tree NNS been shown to run in O(log n) average time per search in a reasonable model. (Assume d a constant)
  – Points come from the same distribution as the queries.
- Storage for the k-d tree is O(n).
- Preprocessing time is O(n log n) assuming d is a constant.

Computational Geometry Problems of Note

- Triangulation
- Binary Space Partition Trees
- Range queries
- Quad and Oct Trees
- Motion planning
- Paper Folding
- On-line Algorithms
- Kinetic Algorithms