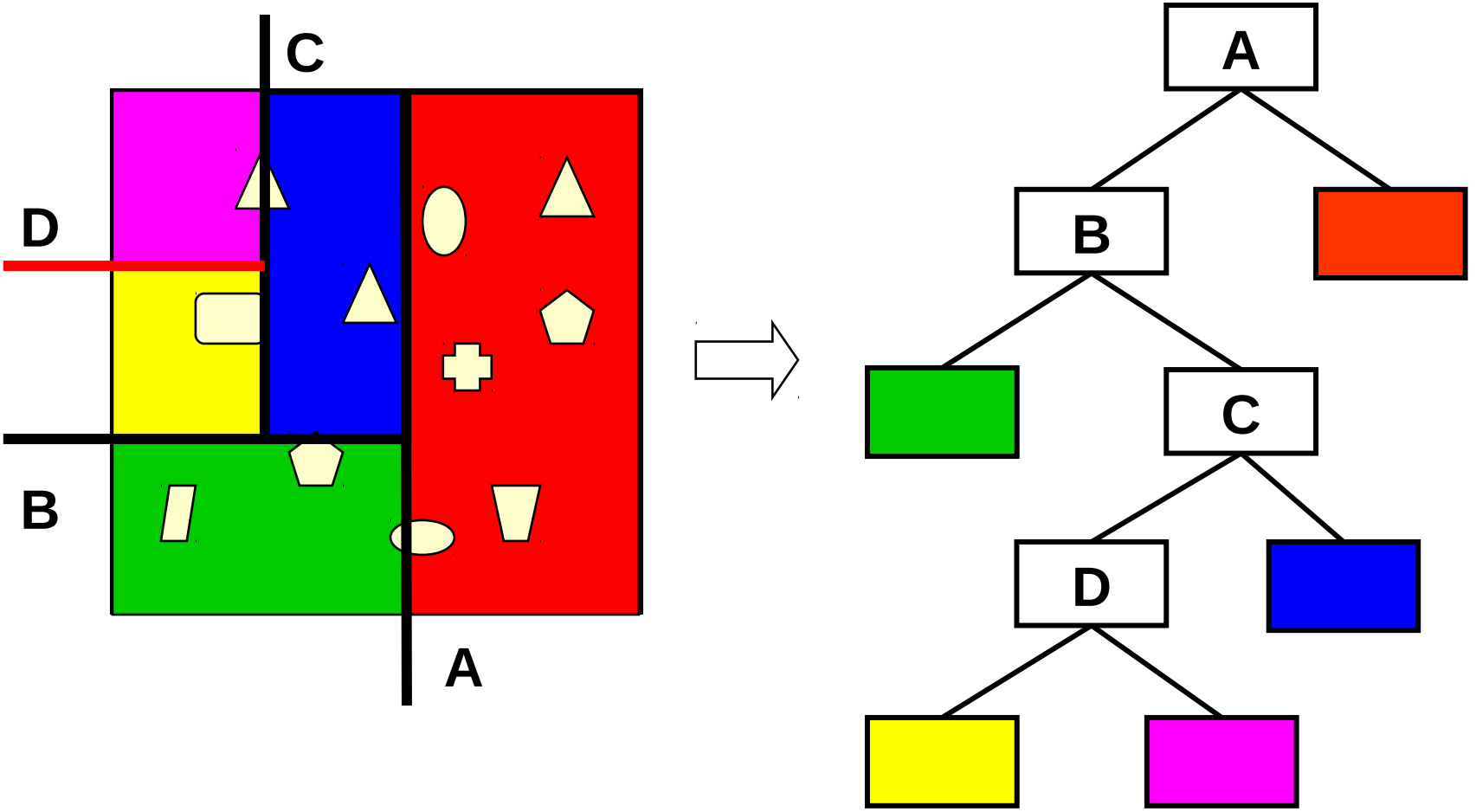
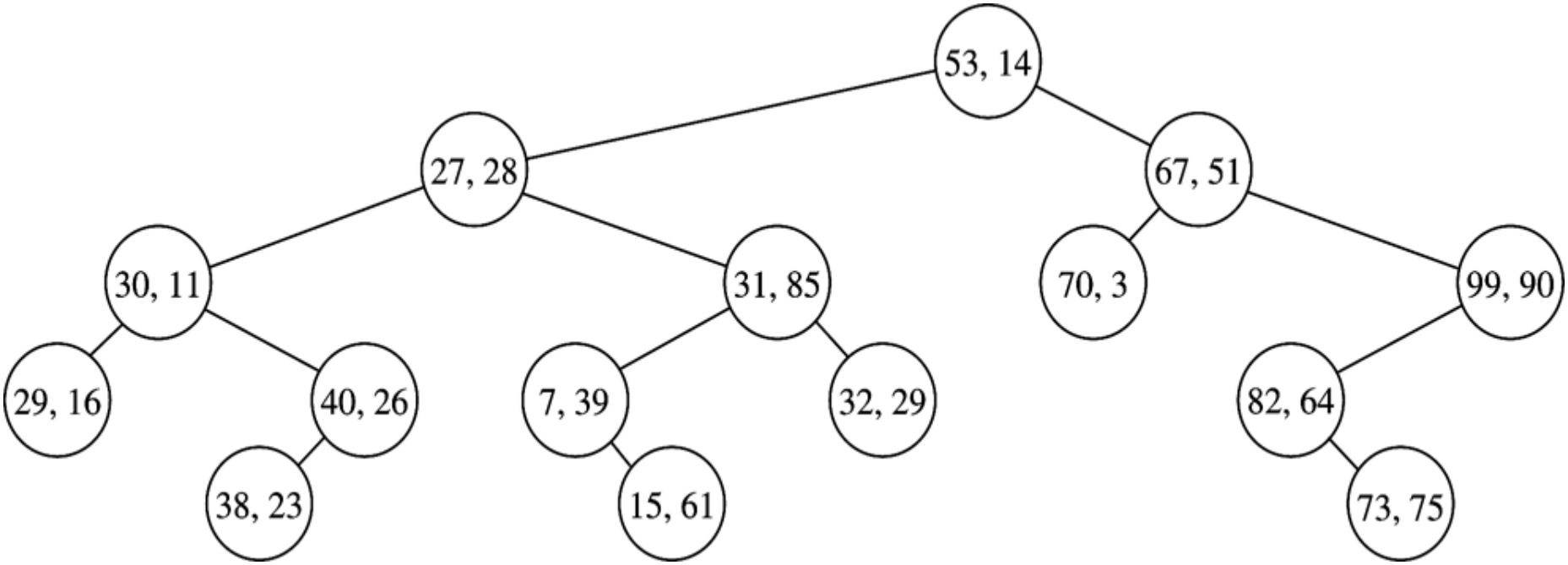
Distance-based models

k-d Trees

k-d tree



k-d tree


53, 14
27, 28
67, 51
30, 11
31, 85
70, 3
99, 90
29, 16
40, 26
7, 39
32, 29
82, 64
38, 23
15, 61
73, 75

k-d tree

- Invented in 1970s by Jon Bently
- Name originally meant “3d-trees, 4-d trees, etc., where k was the number of dimesnsions.
- Idea: each level of the tree compares against 1 dimension.
- The split can be specified by giving the splitting axes (x, y or whatever), also called *cutting dimesion*.
- Cycle through the dimensions as you walk down the tree.
- Each node contains a point P=(x,y).
- To find (x',y') you only compare coordinate from the cutting dimension. Ex: if cutting dimension is x, then: is x' < x ?

k-d tree

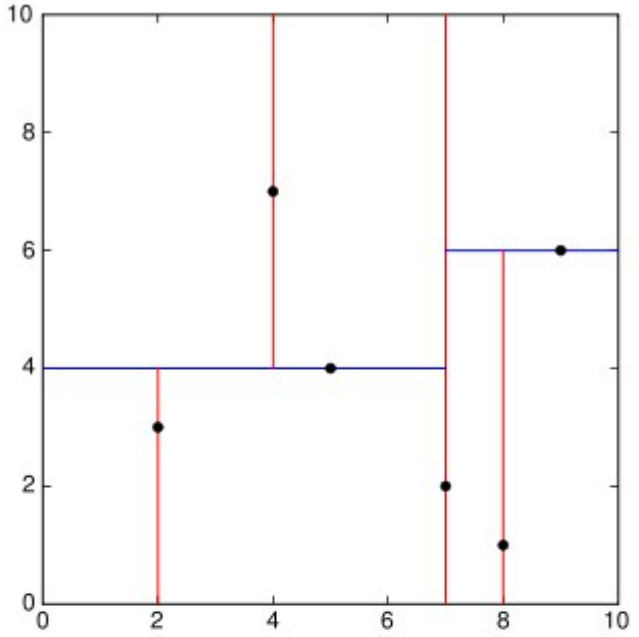
- Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces.
- Points to the left of this hyperplane are represented by the left subtree of that node and points right of the hyperplane are represented by the right subtree.
- Every node in the tree is associated with one of the k-dimensions, with the hyperplane perpendicular to that dimension's axis.

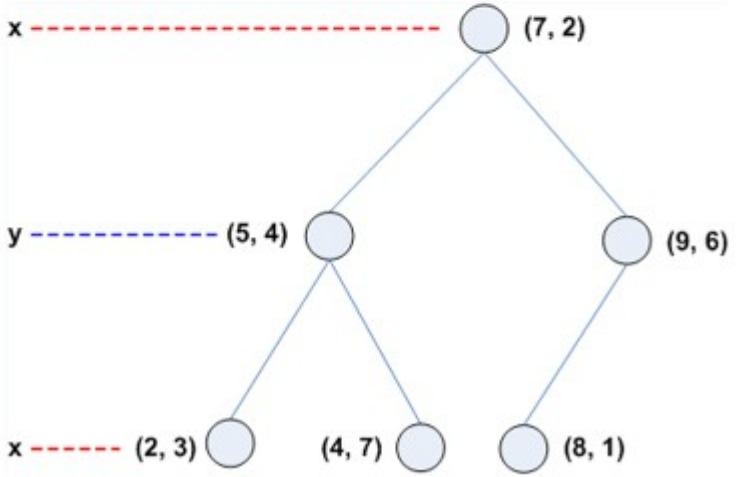
k-d tree

- The data points are:

  (7,2) (5,4) (4,7)

  (9,6) (2,3) (8,1)
- The top is space splitting
- The bottom is k-d tree





Nearest Neighbor in k-d tree

- Given a k-d tree and a point in space (called the *test point*), which point in the kd-tree is closest to the test point?
- The point in the data set closest to the test point i s called its nearest neighbor.
- If there is a point in this data set that is closer to the *test point*, it must lie in the circle centred at the *test point* that passes through the guess point.
- This region is a circle, in three dimension it is called a hypersphere.
- The nearest neighbor to the test point must lie inside this hypersphere.

Nearest Neighbor in k-d tree

- Determine if this hypershpere intersects / crosses the splitting hyperplane.
- If this hypersphere is fully to one side of a splitting hyperplane, then all the points on the other side of the splitting hyperplane can not be the nearest neighbor.

NN in k-d tree – Algorithm

- 1. Starting with the root node, the algorithm moves down the tree recursively (i.e. it goes left or right depending on whether the point is less than or greater than the current node in the split dimension).
- 2. Once the algorithm reaches a leaf node, it saves that node point as the "current best".
- 3. The algorithm unwinds the recursion of the tree, performing the following steps at each node:
  - 3.1. If the current node is closer than the current best, then it becomes the current best.

NN in k-d tree – Algorithm

- 3.2. The algorithm checks whether there could be any points on the other side of the splitting plane that are closer to the search point than the current best.
  - In concept, this is done by intersecting the splitting hyperplane with a hypersphere around the search point.
  - Check whether the difference between the splitting coordinate of the search point and current node is less than the distance (overall coordinates) from the search point to the current best.

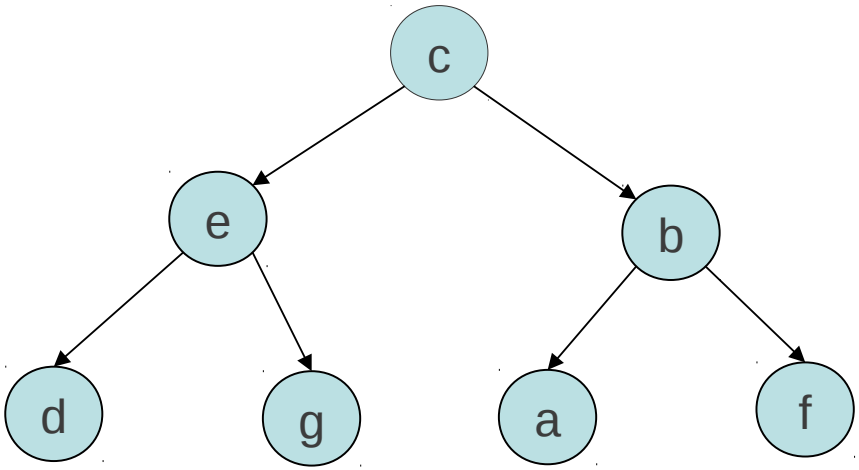
NN in k-d tree – Algorithm

- 3.2.1. If the hypersphere crosses the plane, there could be nearer points on the other side of the plane, so the algorithm must move down the other branch of the tree from the current node looking for closer points, following the same recursive process as the entire search.
- 3.2.2. If the hypersphere doesn't intersect the splitting plane, then the algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.
- 4. When the algorithm finishes this process for the root node, then the search is complete.

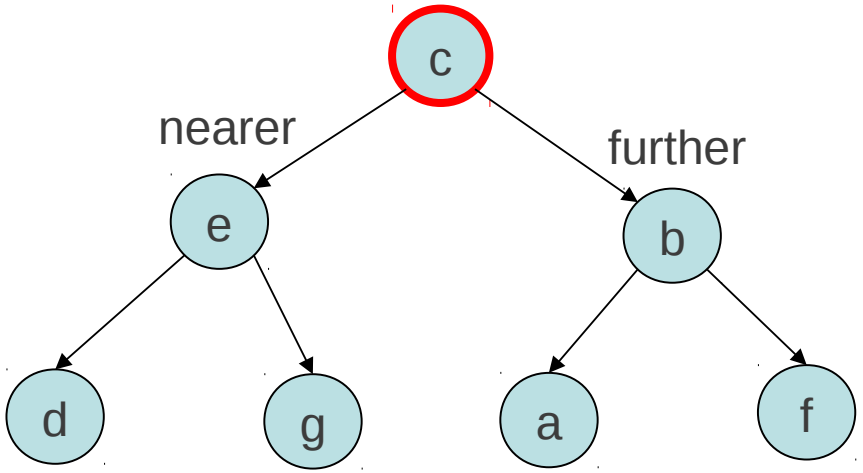
Example

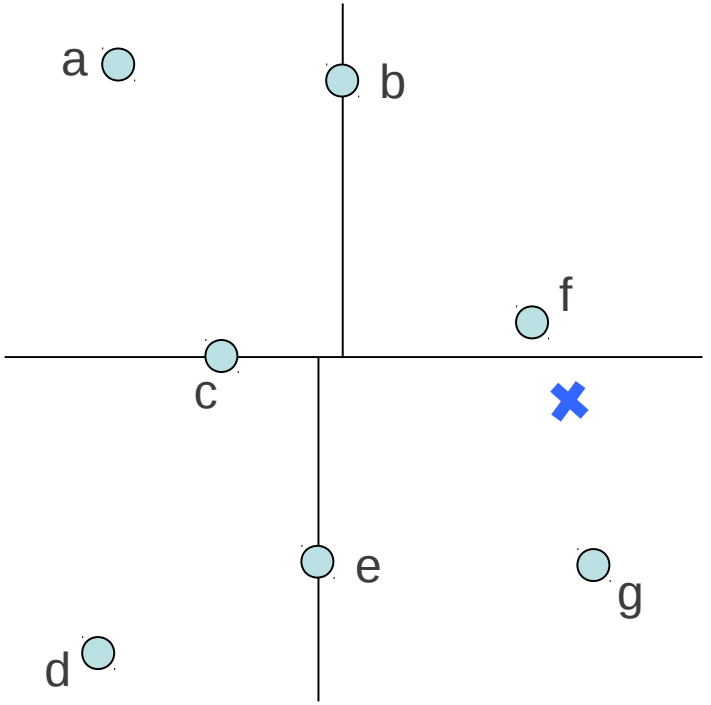
- The points are:

  c(10,20), e(15,10), g(30,8), b(17,35), a(5,37), f(28,22), d(3,5)



Example





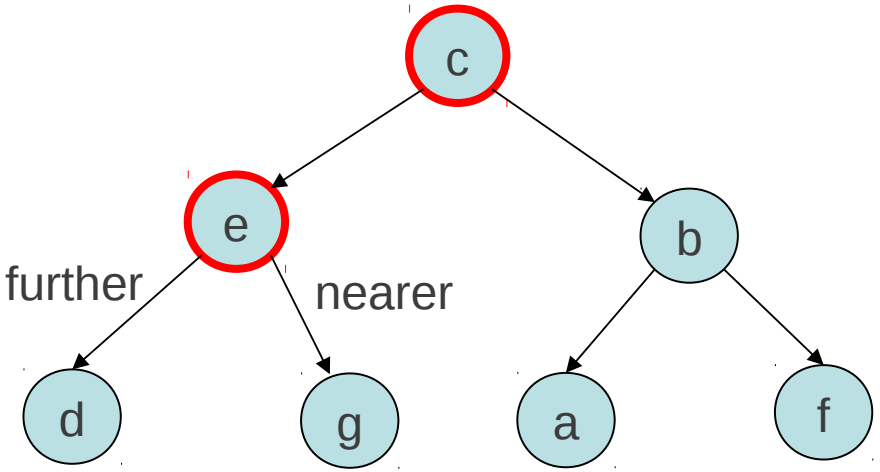
Nearest = ?
dist-sqd = ∞

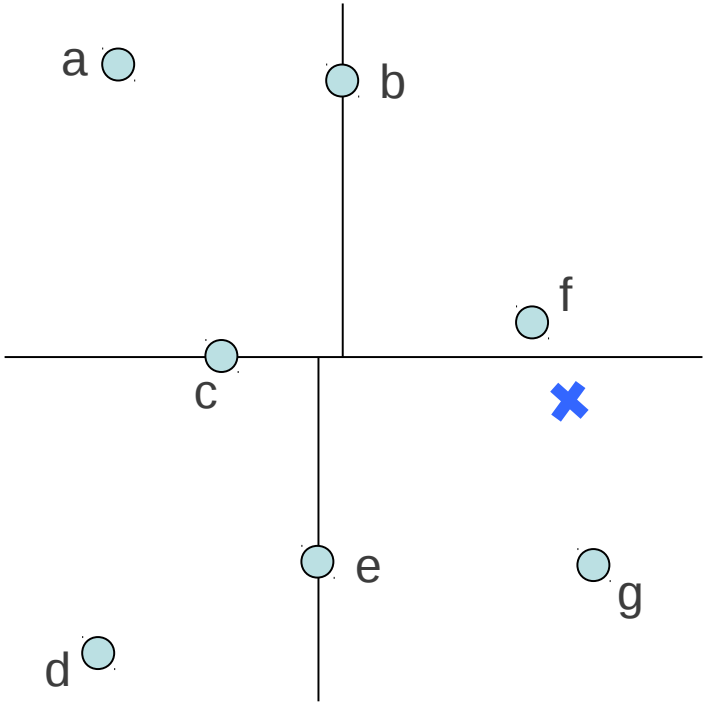
Nearer = e
Further = b

NN(c, x)

NN (e, x)

Example





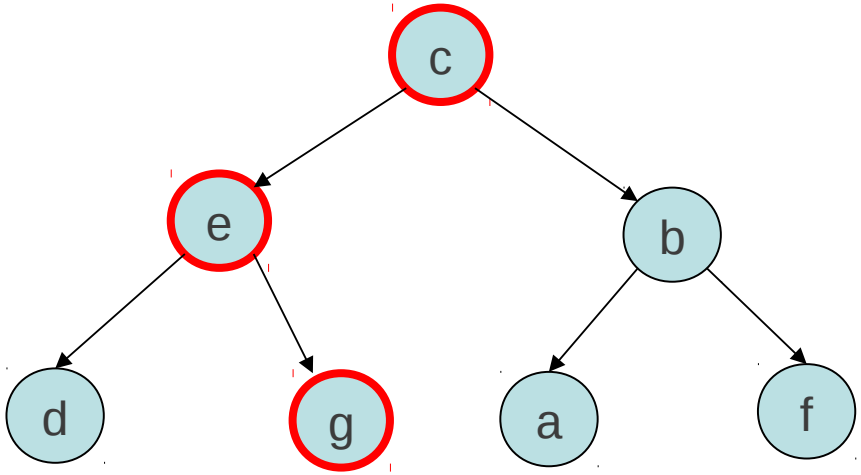
Nearest = ?
dist-sqd = ∞

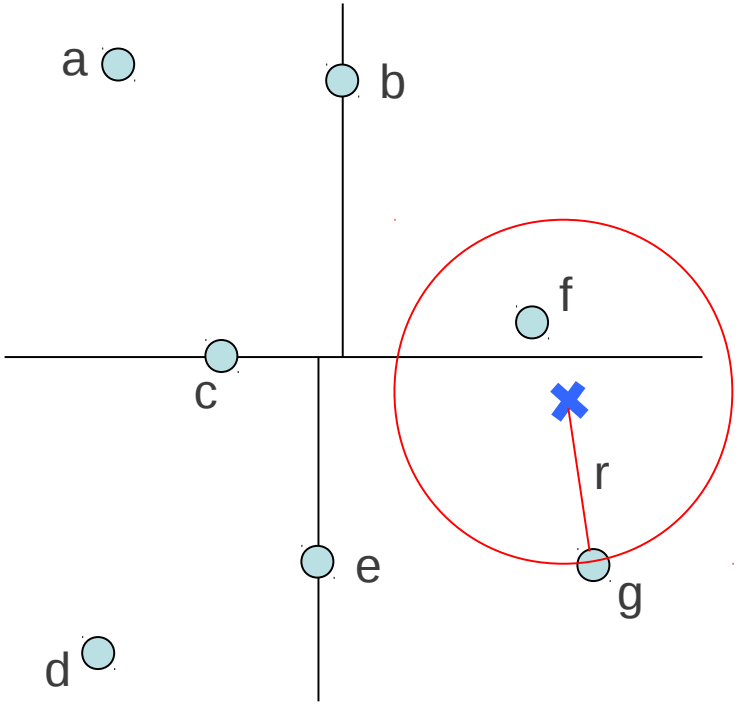
Nearer = g
Further = d

NN(e, x)

NN (g, x)

Example

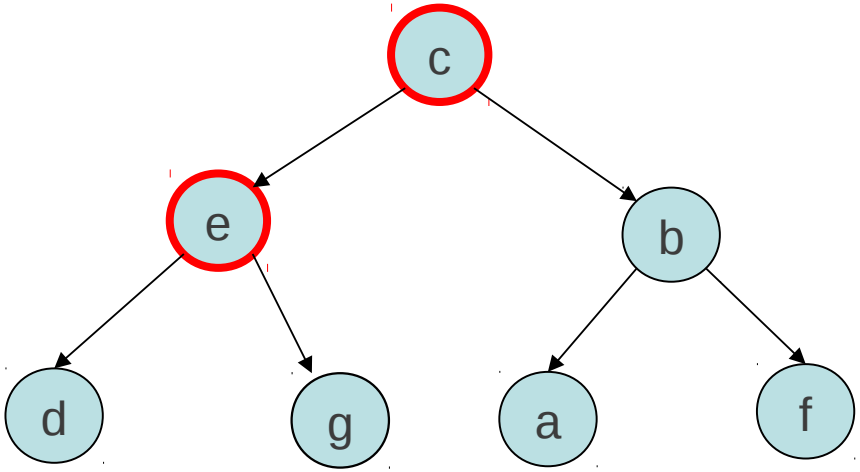


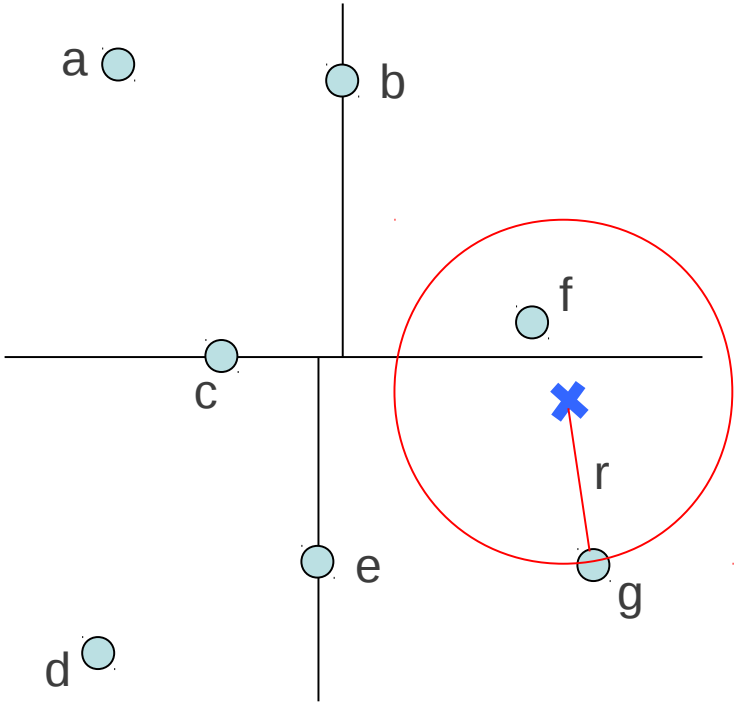


Nearest = g
dist-sqd = r

NN(g, x)

Example





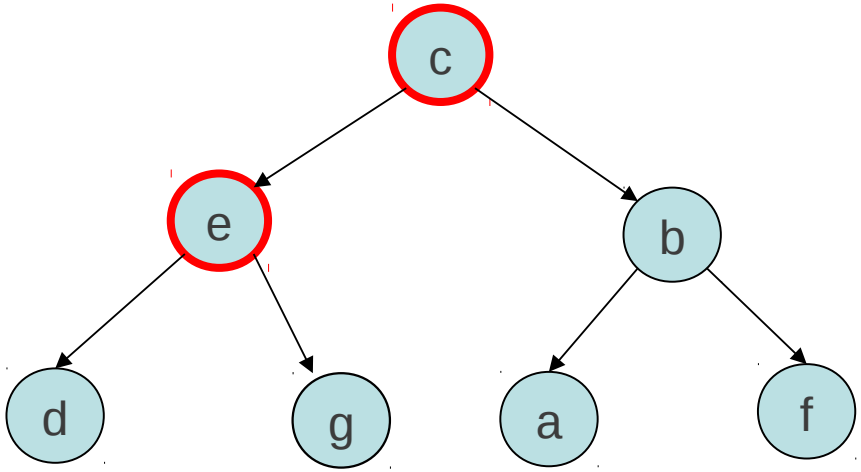
Nearest = g
dist-sqd = r

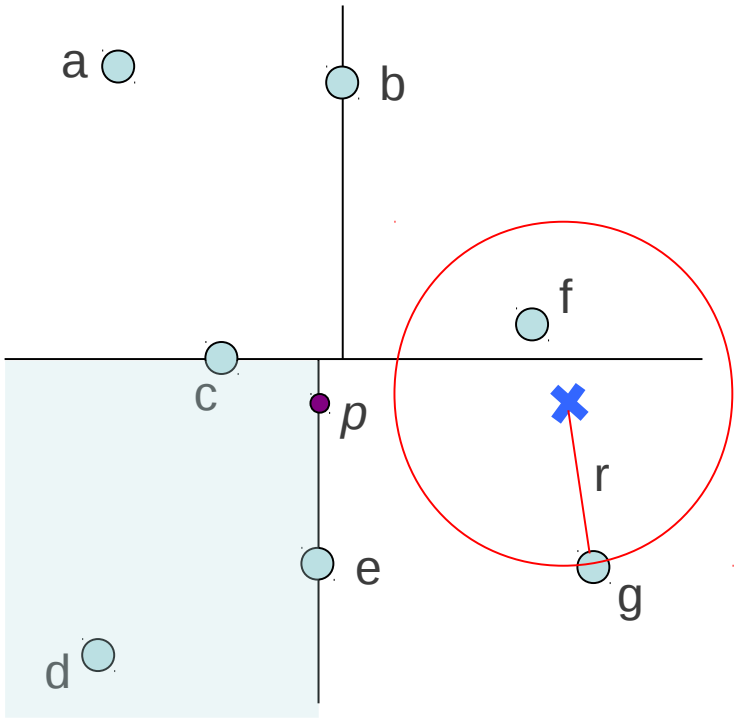
NN(e, x)

Check d2(e,x) > r
No need to update

d2: distance squared

Example



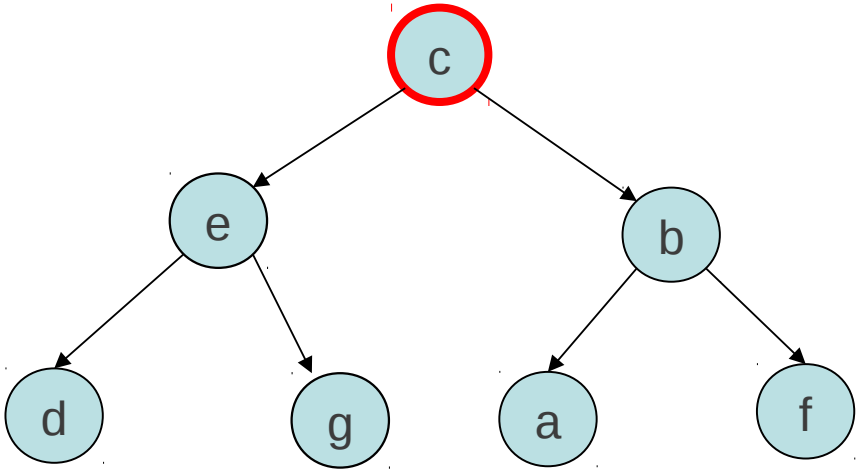


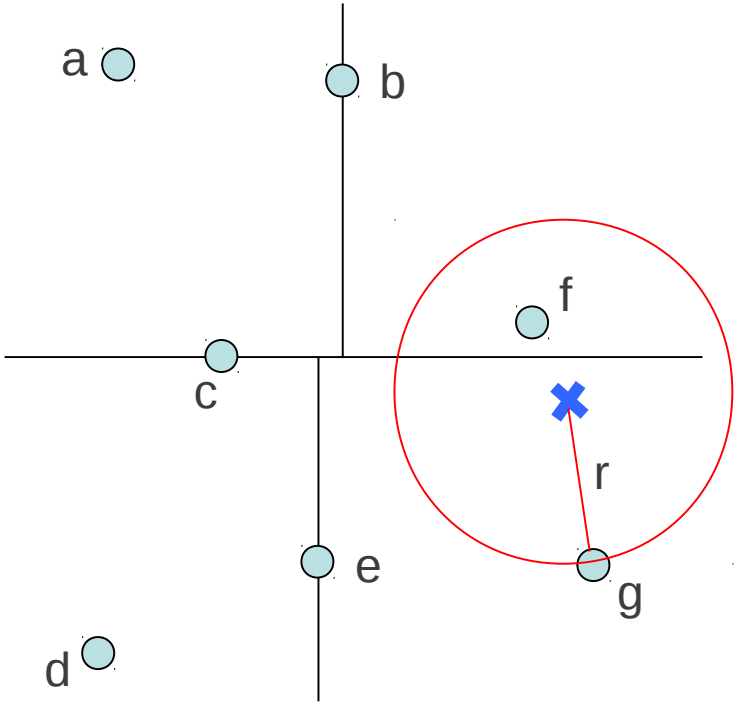
Nearest = g
dist-sqd = r

NN(e, x)

Check further of e: find *p*
d (p,x) > r
No need to update

Example



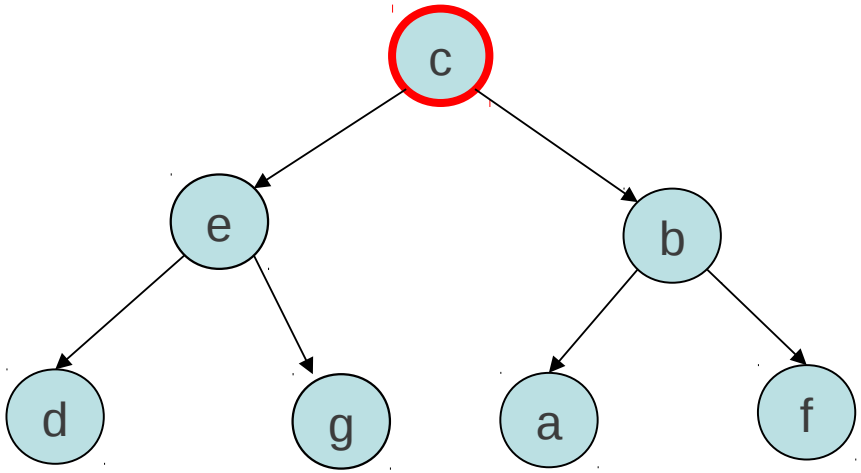


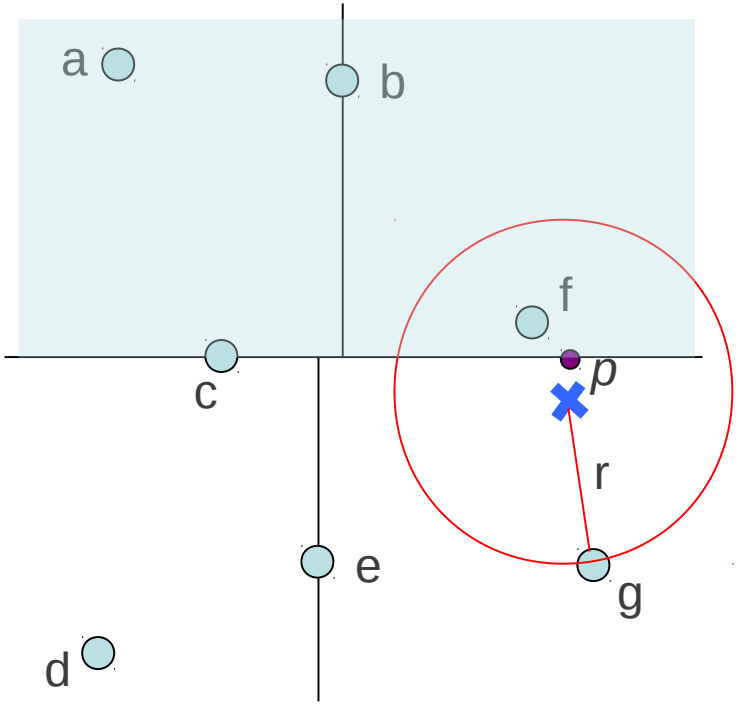
Nearest = g
dist-sqd = r

NN(c, x)

Check d2(c,x) > r
No need to update

Example





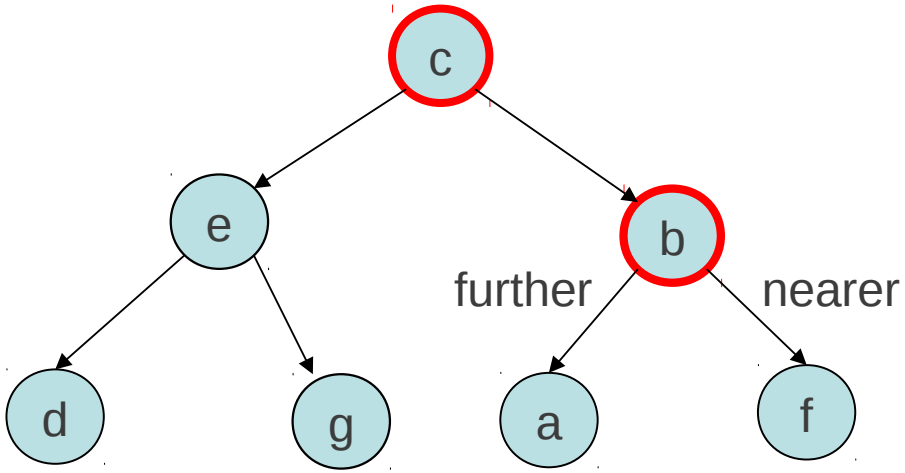
Nearest = g
dist-sqd = r

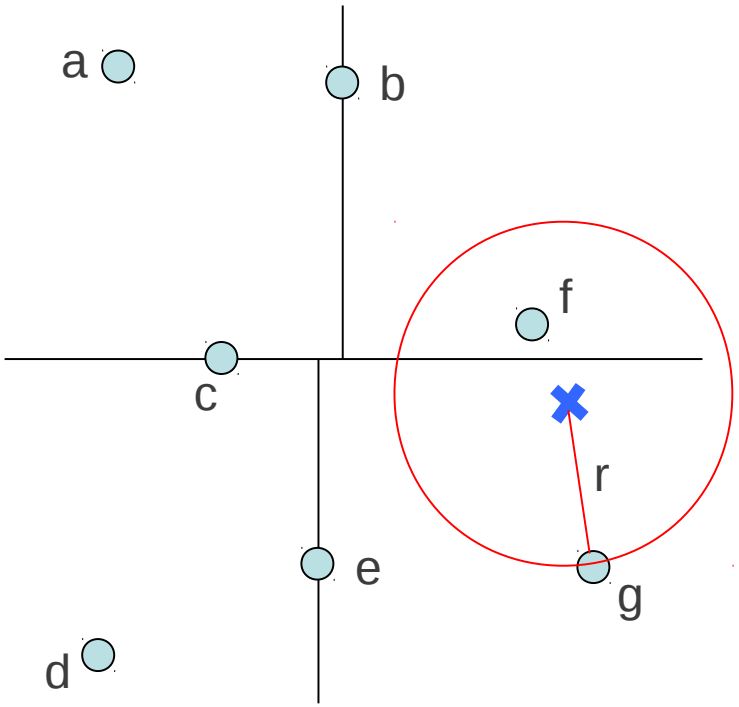
NN(c, x)

Check further of c: find p
d(p,x) < r !!

NN (b,x)

Example





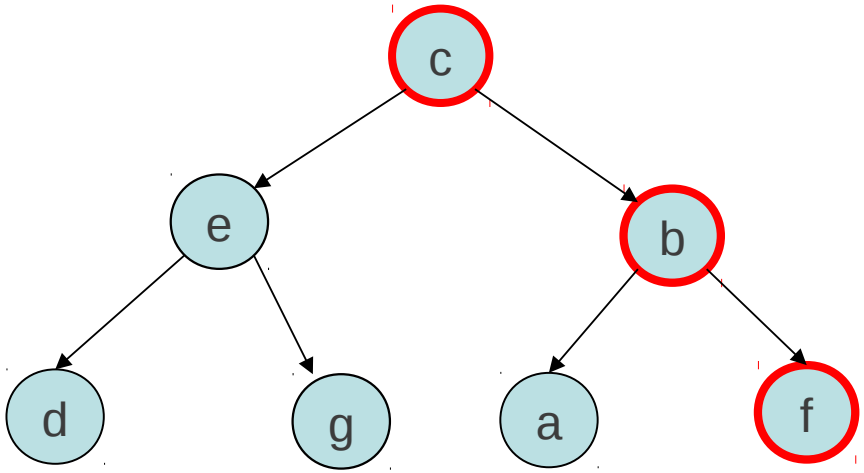
Nearest = g
dist-sqd = r

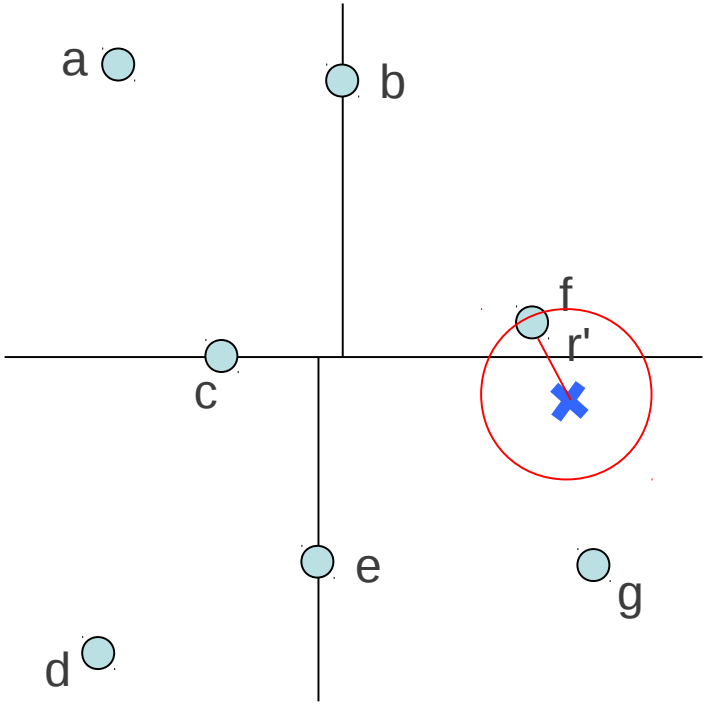
NN(b, x)

Nearer = f
Further = g

NN (f,x)

Example





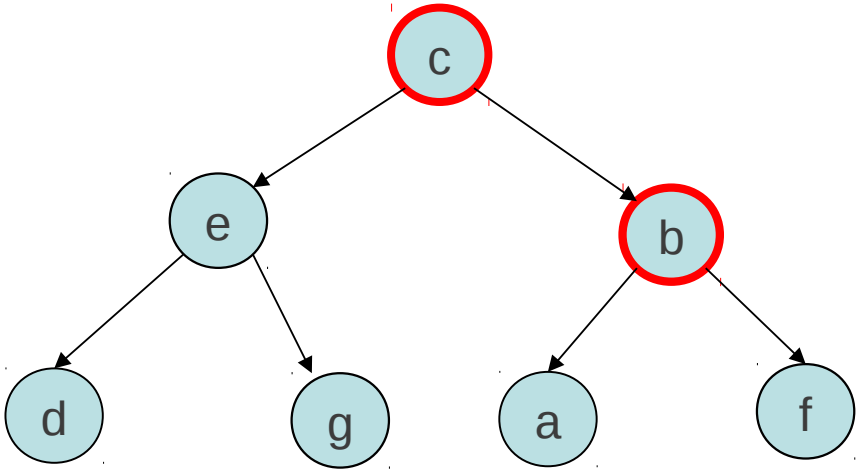
Nearest = g
dist-sqd = r

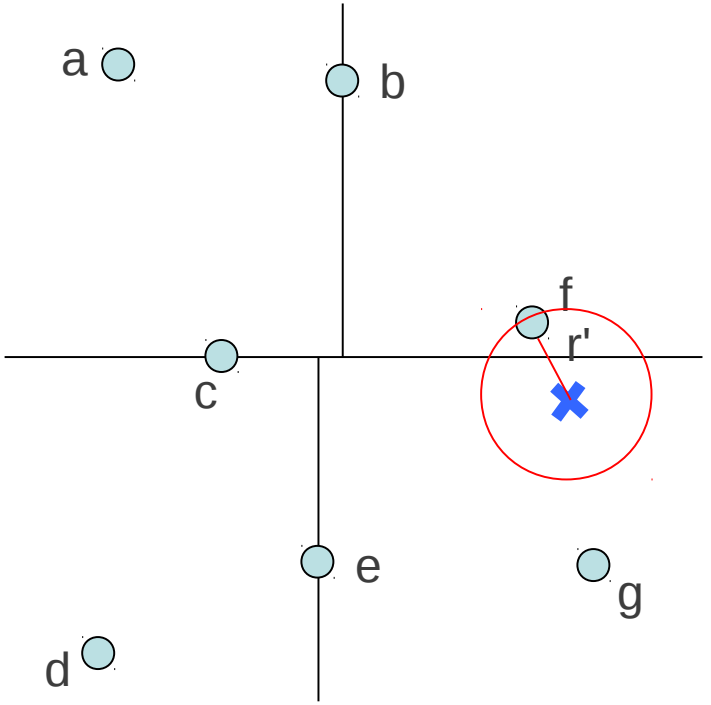
NN(f, x)

r’ = d2 (f,x) < r

dist-sqd ← r’
nearest ←f

Example





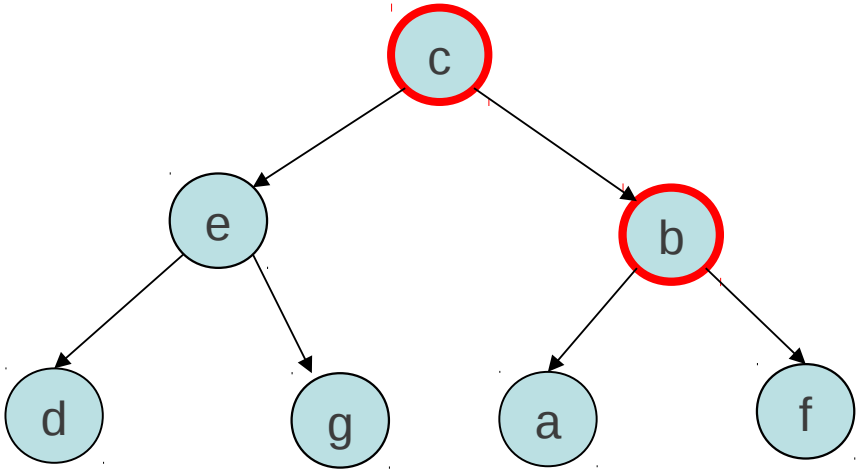
Nearest = f
dist-sqd = r'

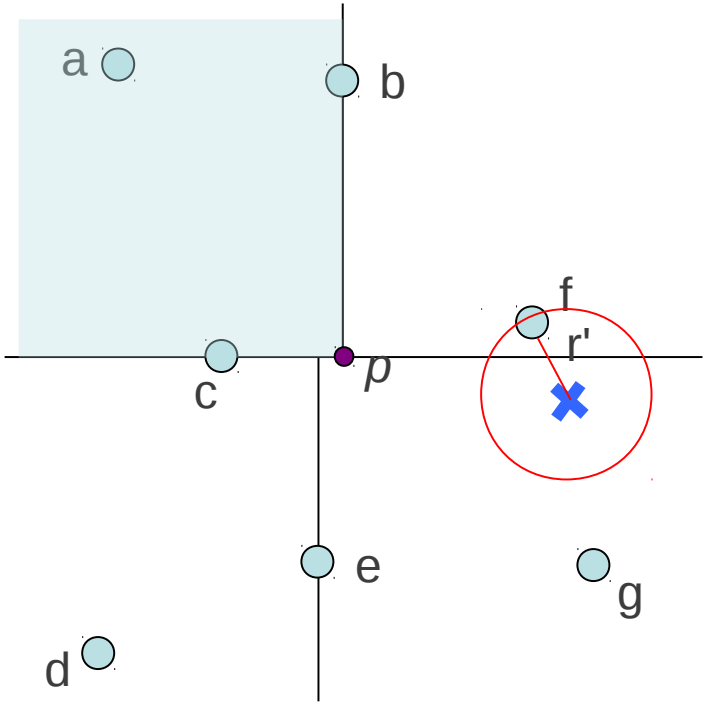
NN(b, x)

Check d(b,x) < r’

No need to update

Example





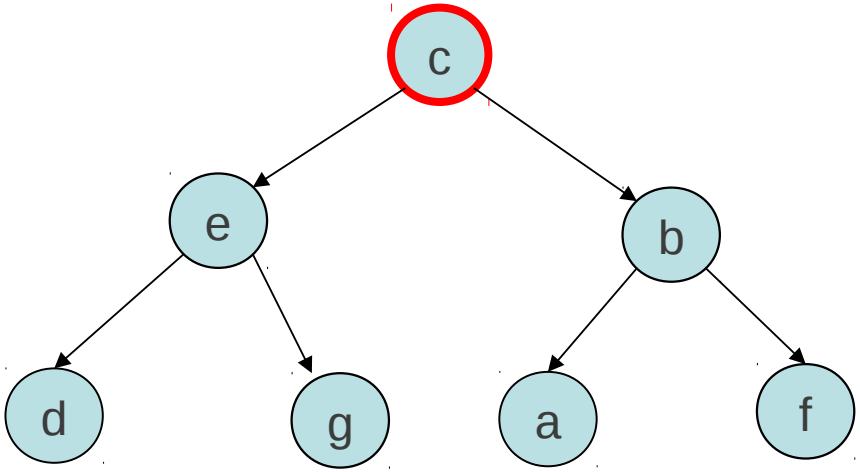
Nearest = f
dist-sqd = r'

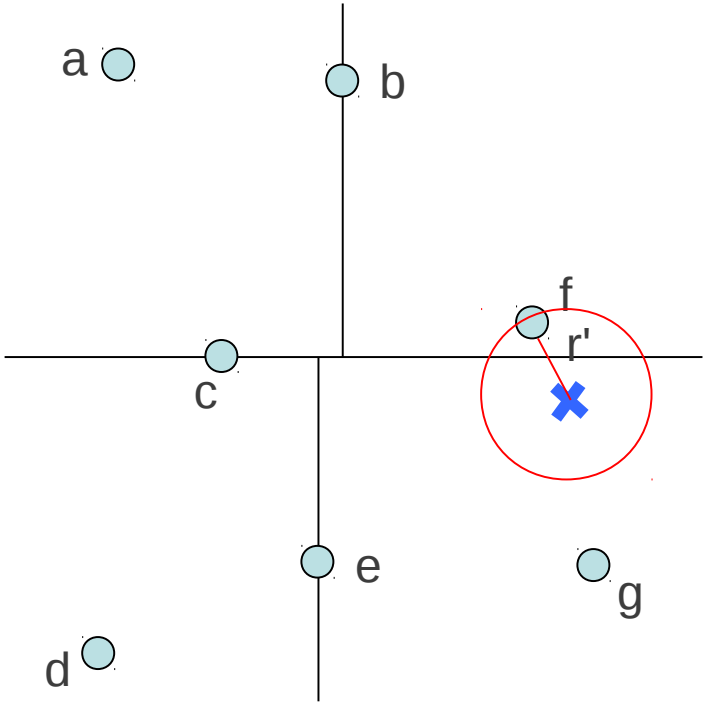
Check further of b; find p
d(p,x) > r'

NN(b, x)

No need to update

Example





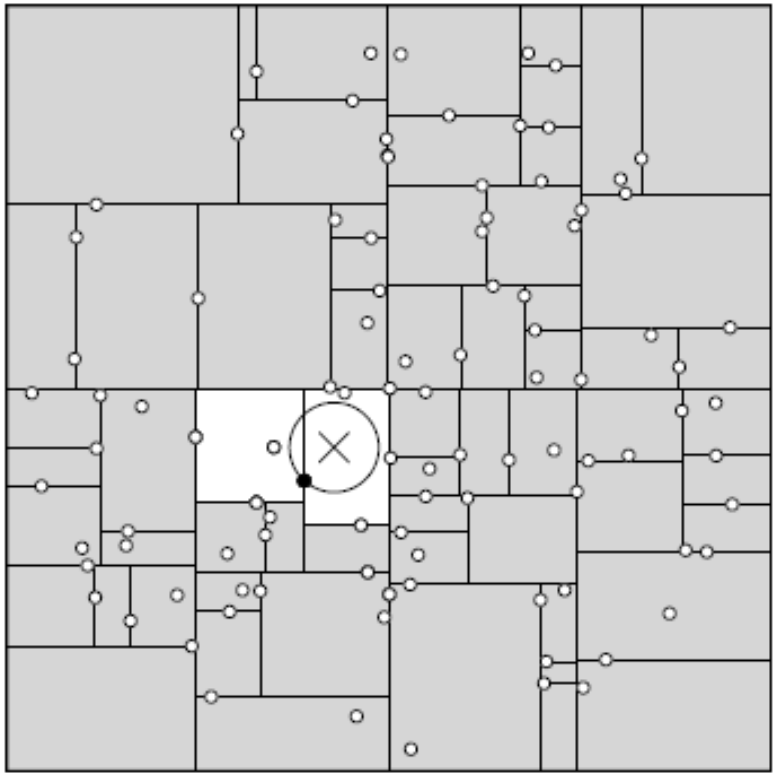
Nearest = f
dist-sqd = r'

NN(c, x)

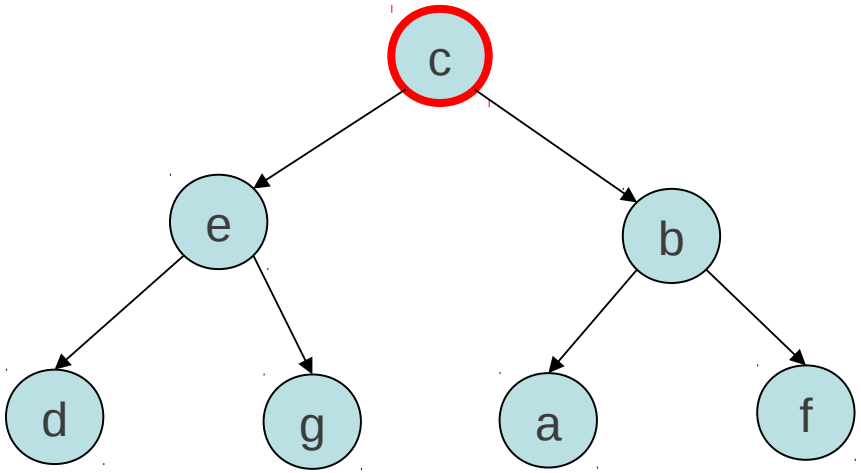
Time complexity

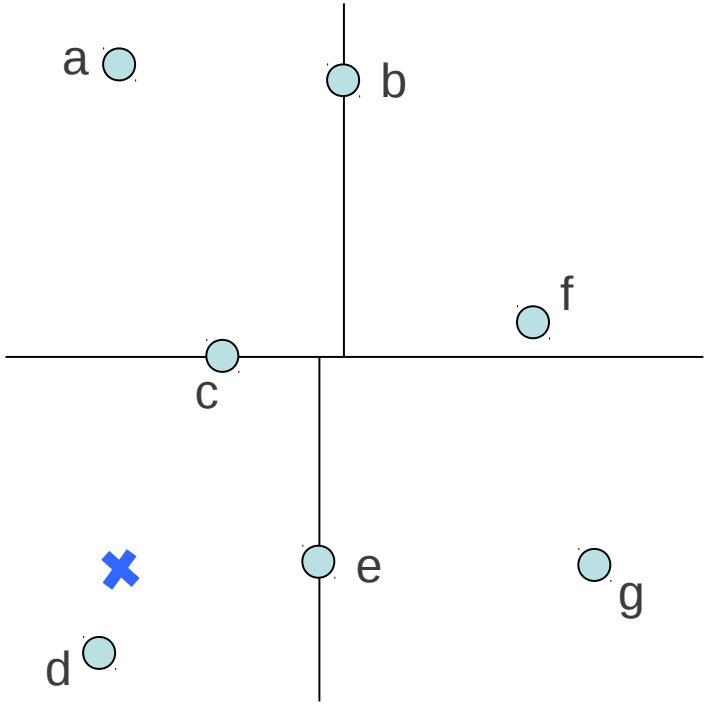
- At least O(log N) inspections are necessary.
- No more than N nodes are searched: the algorithm visits each node at most once.
- Depends on the point distribution.

Only a few leaf nodes need to be inspected



Exercise





nearest = ?
min-dist-sqd = ∞

NN(c, x)

Thank you

- Ensemble learning.....?