CSE 490 GZ
Introduction to Data Compression
Winter 2002

Nearest Neighbor Search
for
Vector Quantization

VQ Encoding is Nearest Neighbor Search

- Given an input vector, find the closest codeword in the codebook and output its index.
- Closest is measured in squared Euclidian distance.
- For two vectors (w_1,x_1,y_1,z_1) and (w_2,x_2,y_2,z_2).

$$\text{Squared Distance} = (w_1 - w_2)^2 + (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2$$

---

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.

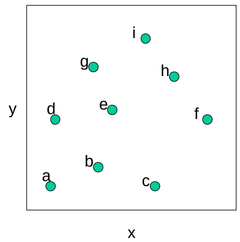
---

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.

---

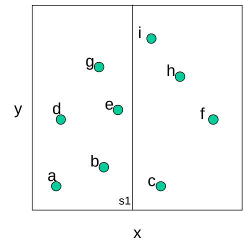
k-d Tree Construction (1)

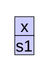


divide perpendicular to the widest spread.

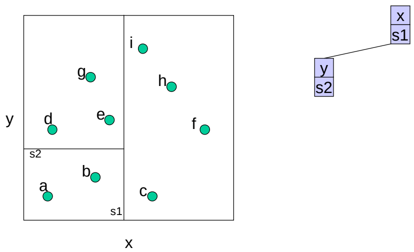
---

k-d Tree Construction (2)



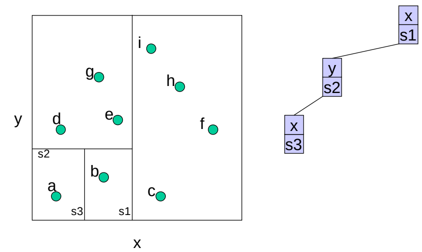


k-d Tree Construction (3)



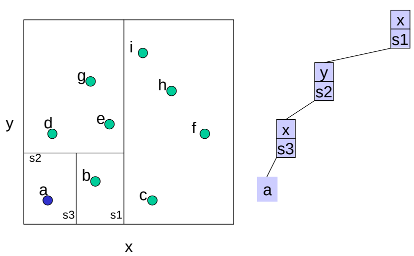
CSE 490gz - Lecture 13 - Winter 2002 7

k-d Tree Construction (4)



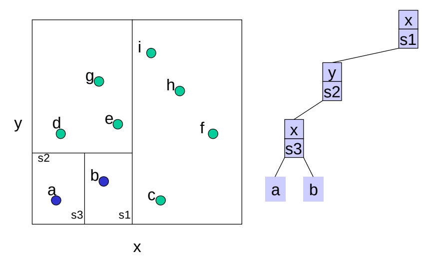
CSE 490gz - Lecture 13 - Winter 2002 8

k-d Tree Construction (5)



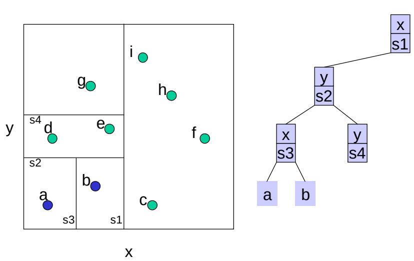
CSE 490gz - Lecture 13 - Winter 2002 9

k-d Tree Construction (6)



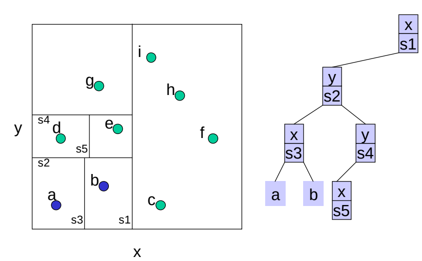
CSE 490gz - Lecture 13 - Winter 2002 10

k-d Tree Construction (7)



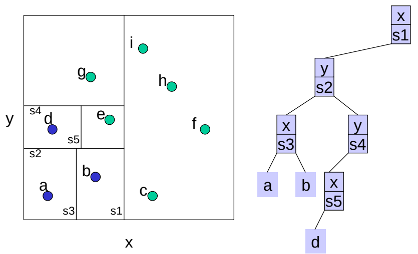
CSE 490gz - Lecture 13 - Winter 2002 11

k-d Tree Construction (8)

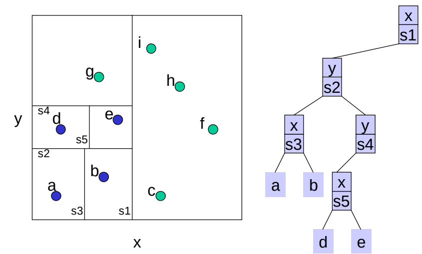


CSE 490gz - Lecture 13 - Winter 2002 12

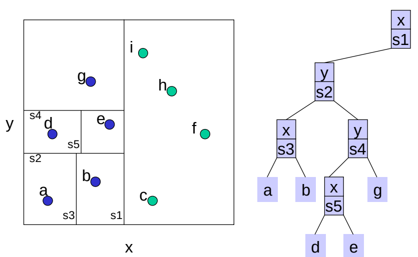
k-d Tree Construction (9)



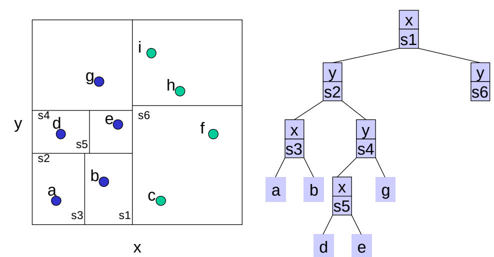
k-d Tree Construction (10)



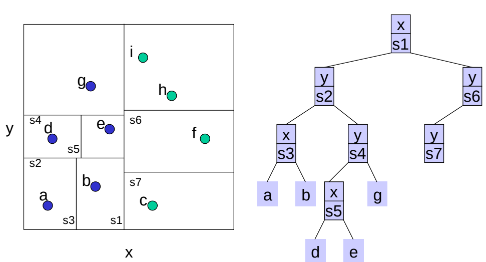
k-d Tree Construction (11)



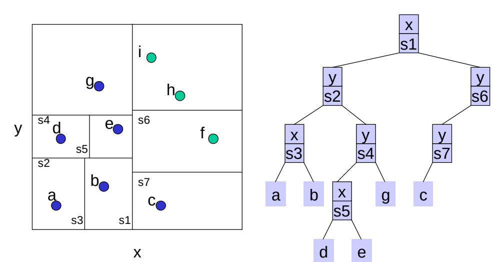
k-d Tree Construction (12)



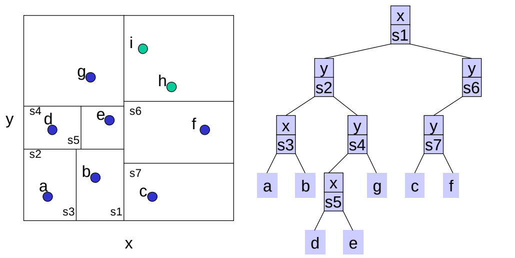
k-d Tree Construction (13)



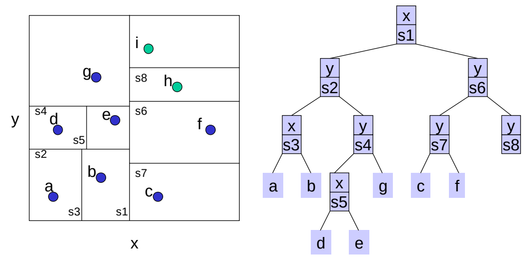
k-d Tree Construction (14)



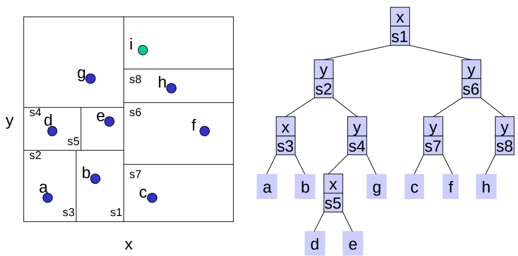
k-d Tree Construction (15)



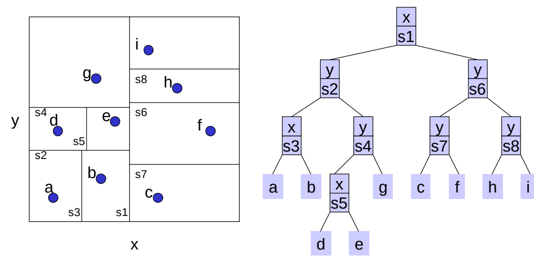
k-d Tree Construction (16)



k-d Tree Construction (17)



k-d Tree Construction (18)

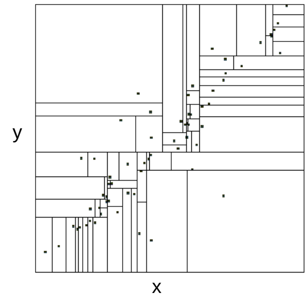


k-d Tree Construction Complexity

- 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 Codebook Organization

2-d vectors
(x,y)



Node Structure for k-d Trees

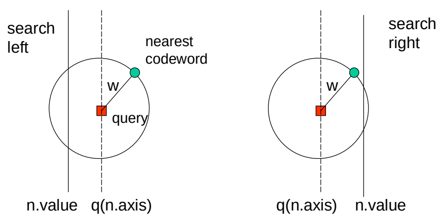
- A node has 5 fields
  - axis (splitting axis)
  - value (splitting value)
  - left (left subtree)
  - right (right subtree)
  - point (holds a point if left and right children are null)

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 $\boxed{\text{NNS(q, root, p, infinity)}}$

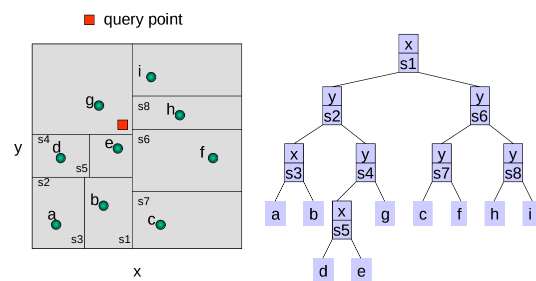
Explanation



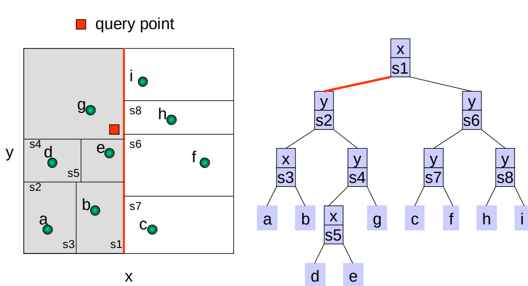
q(n.axis) – w ≤ n.value means the circle overlap the left subtree.

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

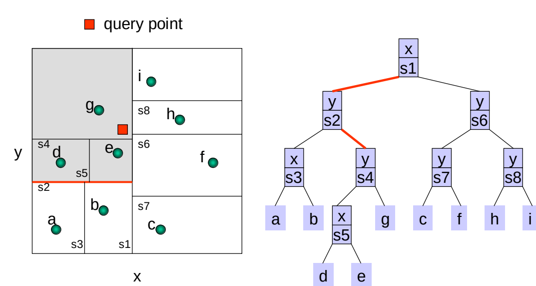
k-d Tree NNS (1)



k-d Tree NNS (2)

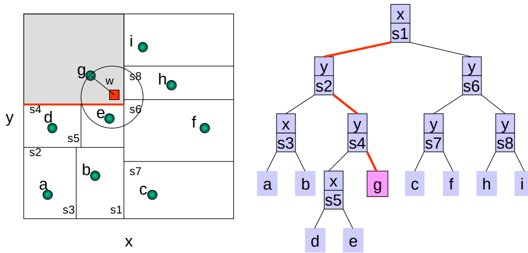


k-d Tree NNS (3)



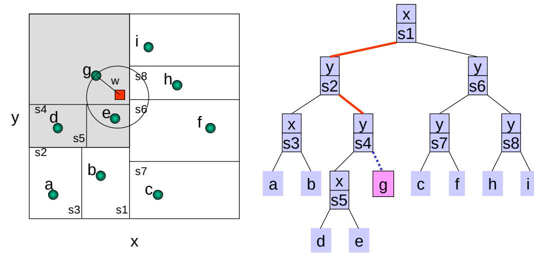
k-d Tree NNS (4)

query point



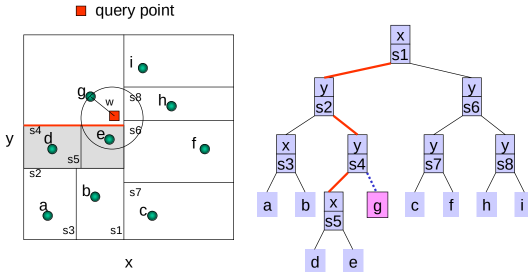
k-d Tree NNS (5)

query point



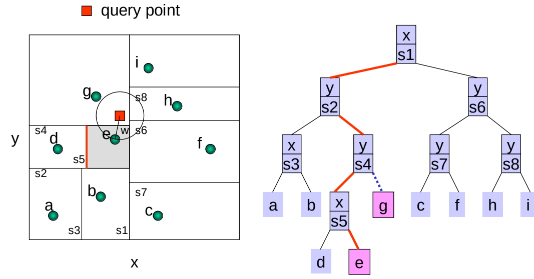
k-d Tree NNS (6)

query point



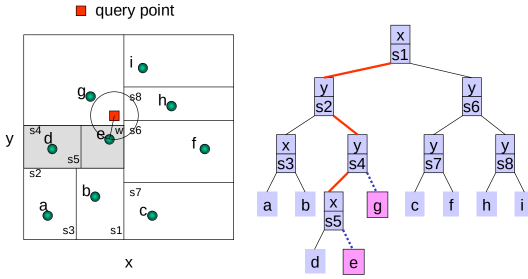
k-d Tree NNS (7)

query point



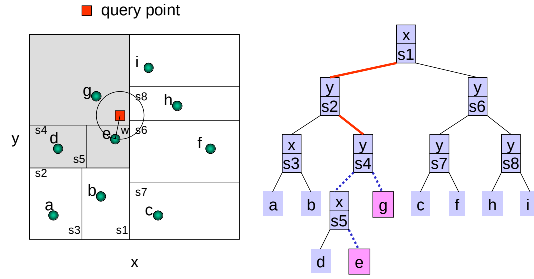
k-d Tree NNS (8)

query point



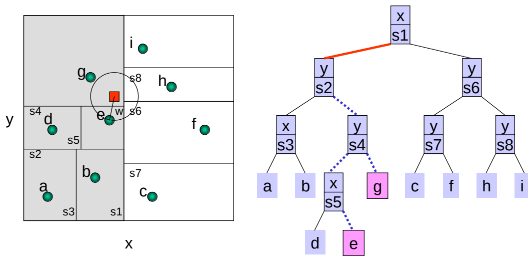
k-d Tree NNS (9)

query point



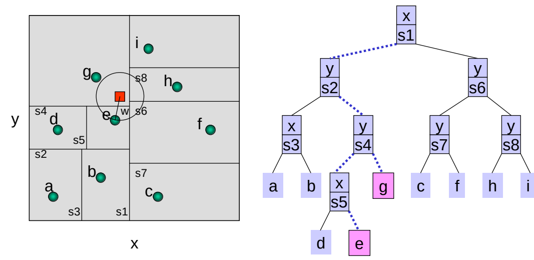
k-d Tree NNS (10)

query point



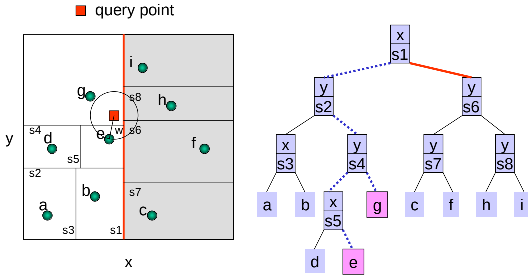
k-d Tree NNS (11)

query point



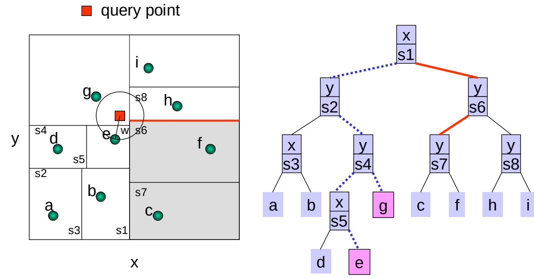
k-d Tree NNS (12)

query point



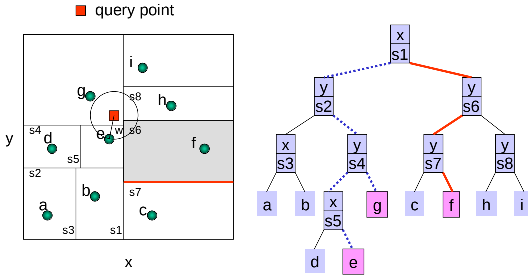
k-d Tree NNS (13)

query point



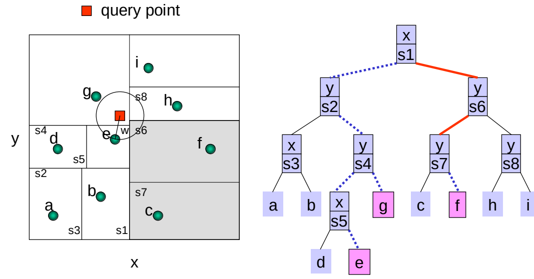
k-d Tree NNS (14)

query point

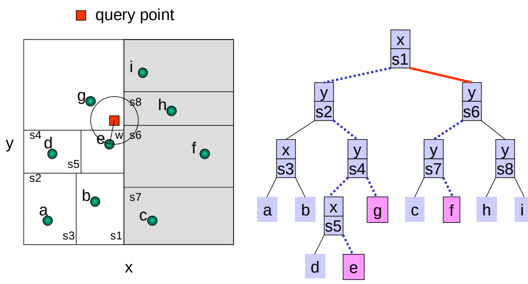


k-d Tree NNS (15)

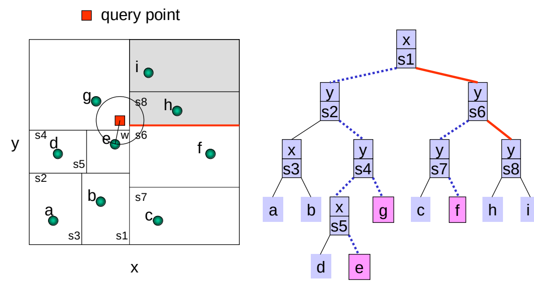
query point



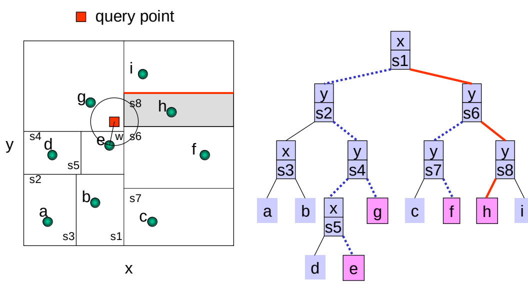
k-d Tree NNS (16)



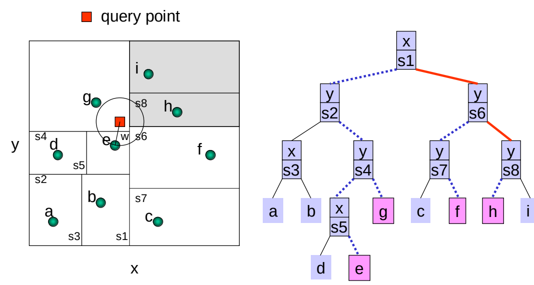
k-d Tree NNS (17)



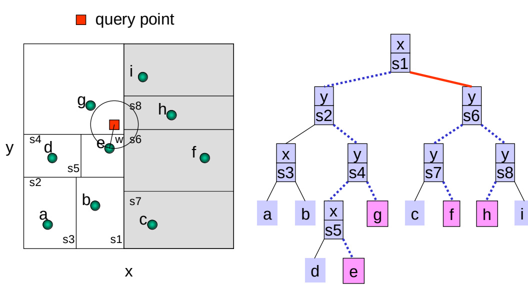
k-d Tree NNS (18)



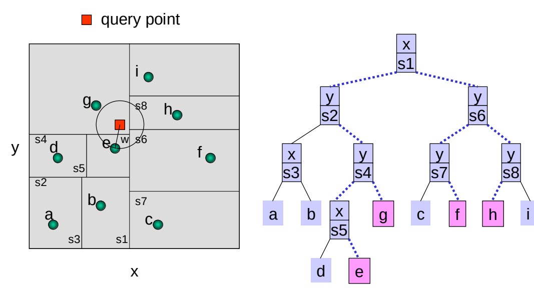
k-d Tree NNS (19)



k-d Tree NNS (20)



k-d Tree NNS (21)



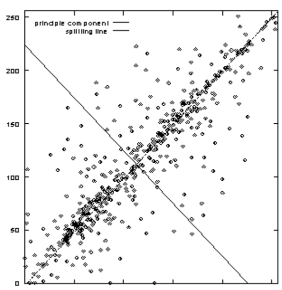
Notes on k-d Tree NNS

- Has been shown to run in O(log n) average time per search in a reasonable model. (Assume d a constant)
- For VQ it appears that O(log n) is correct.
- Storage for the k-d tree is O(n).
- Preprocessing time is O(n log n) assuming d is a constant.

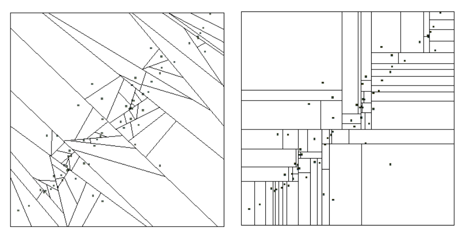
Alternatives

- Orchard's Algorithm (1991)
  - Uses O(n^2) storage but is very fast
- Annulus Algorithm
  - Similar to Orchard but uses O(n) storage. Does many more distance calculations.
- PCP Principal Component Paritioning
  - Zatloukal, Johnson, Ladner (1999)
  - Similar to k-d trees
  - Also very fast

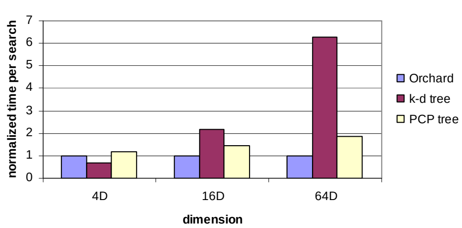
Principal Component Partition



PCP Tree vs. k-d tree



Comparison in Time per Search



4,096 codewords

Notes on VQ

- Works well in some applications.
  - Requires training
- Has some interesting algorithms.
  - Codebook design
  - Nearest neighbor search
- Variable length codes for VQ.
  - PTSVQ - pruned tree structured VQ (Chou, Lookabaugh and Gray, 1989)
  - ECVQ - entropy constrained VQ (Chou, Lookabaugh and Gray, 1989)