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 Euclidean 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 trees 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 (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 example

divide perpendicular to the widest spread.
k-d tree construction example
k-d tree construction example
k-d tree construction example
k-d tree construction example
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 dividing 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.
Codebook for 2-d vector

2-d vectors
(x,y)
Node Structure for k-d Tree

- 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)
Node Structure for k-d Tree

- 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)
Why does k-d tree work?

- **search left**
  - nearest codeword
  - \( q(n.\text{axis}) - w \leq n.\text{value} \) means the circle overlaps the left subtree.

- **search right**
  - \( q(n.\text{axis}) + w > n.\text{value} \) means the circle overlaps the right subtree.
k-d Tree Nearest Neighbor Search

\[ \text{NNS}(q: \text{point}, n: \text{node}, p: \text{ref point}, w: \text{ref distance}) \]

\[
\text{if } n.\text{left} = n.\text{right} = \text{null} \text{ then } \{\text{leaf case}\} \\
\quad w' := \|q - n.\text{point}\|; \\
\quad \text{if } w' < w \text{ then } w := w'; \ p := n.\text{point}; \\
\text{else} \\
\quad \text{if } q(n.\text{axis}) \leq n.\text{value} \text{ then} \\
\quad \quad \text{search}_1 := \text{left}; \\
\quad \text{else} \\
\quad \quad \text{search}_1 := \text{right}; \\
\text{if } (\text{search}_1 == \text{left}) \\
\quad \text{if } q(n.\text{axis}) - w \leq n.\text{value} \text{ then } \text{NNS}(q, n.\text{left}, p, w); \\
\quad \text{if } q(n.\text{axis}) + w > n.\text{value} \text{ then } \text{NNS}(q, n.\text{right}, p, w); \\
\text{else} // \text{search}_1 == \text{right} \\
\quad \text{if } q(n.\text{axis}) + w > n.\text{value} \text{ then } \text{NNS}(q, n.\text{right}, p, w); \\
\quad \text{if } q(n.\text{axis}) - w \leq n.\text{value} \text{ then } \text{NNS}(q, n.\text{left}, p, w); \\
\]

\[\text{initial call} \quad \text{NNS}(q, \text{root}, p, \text{infinity})\]
k-d Tree Nearest Neighbor Search
k-d Tree Nearest Neighbor Search
k-d Tree Nearest Neighbor Search
k-d Tree Nearest Neighbor Search
k-d Tree Nearest Neighbor Search
Notes on Nearest Neighbor Search

- Has been shown to run in $O(\log n)$ average time per search in a reasonable model. (Assuming $d$ 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.
Notes on Nearest Neighbor Search

- 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.

- Principal Component Partitioning (PCP)
  - Similar to k-d trees.
  - Also very fast.
PCP Tree
PCP Tree vs. k-d Tree
Search Time

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:
  - PTVQ - pruned tree structured VQ (Chou, Lookabaugh and Gray, 1989)
  - ECVQ - entropy constrained VQ (Chou, Lookabaugh and Gray, 1989)