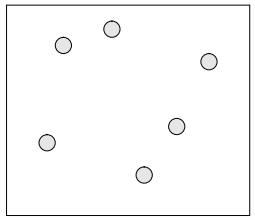
CSE 326
Lecture 13

Spring Quarter 1998

Multidimensional Search Trees

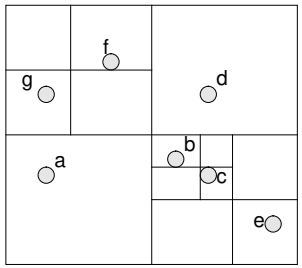
- Example: two dimensions. We want to store a set of ordered pairs (x_1,y_1), (x_2,y_2), (x_3,y_3), ... , (x_n,y_n). There is no specific order to the points as there is with say integers or character strings.

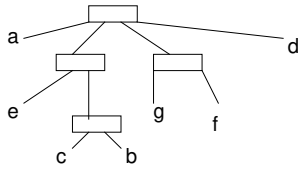


Application of Multidimensional Search

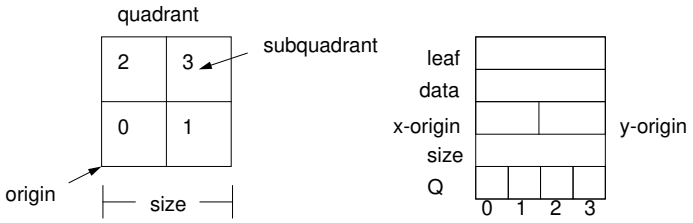
- Astronomy - simulation of galaxies, 3 dimensions.
- Protein folding in molecular biology - 3 dimensions.
- Lossy data compression - 4 to 16 dimensions.
- Image processing - 2 dimensions
- Graphics - 2 or 3 dimensions
- Animation-video - 3 to 4 dimensions
- Geographical Databases - 2 or 3 dimensions
- Web searching - 200 dimensions or more

Quad Trees





Quad Tree Node



- leaf is boolean. If leaf is true then data holds data or pointer to data.
- origin is the bottom left corner of the quadrant.
- size is the width and height of the quadrant.
- Q is an array that contains pointers to other active subquadrants.

Find in Quad Trees

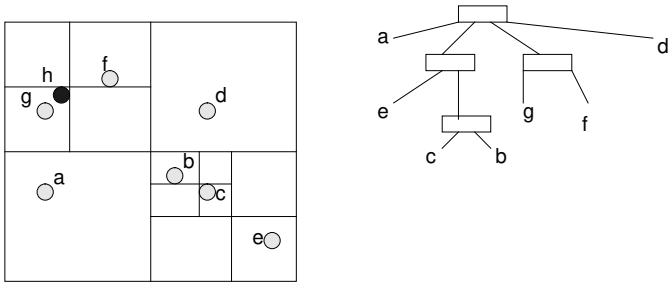
- Find(x,y,n) finds the node which has the pair (x,y) in it and returns null if there is no such node.
- Find(x: real, y: real, n: node): node {

  case
  n = null: return null;
  n.leaf = true: if n.data = (x,y) then return(n) else return null
  otherwise: return(find(x,y,Q[quadrant(x,y,n)])
  endcase
- quadrant(x: real, y: real, n: node) : integer{

  return((x ≥ n.x_origin + size/2) + 2(y ≥ n.y_origin + size/2)) }

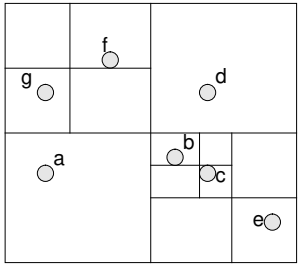
Insert

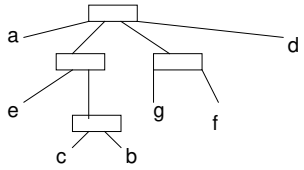
- Insert - Find the leaf quadrant where point belongs.
  If quadrant occupied then split into new quadrants, this may propagate



Delete

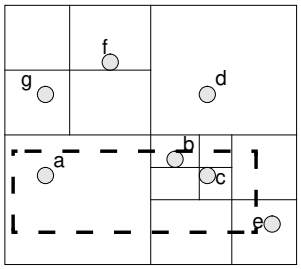
- Delete - Find the point in its leaf quadrant.
  Delete the point.
  If the parent has one point, then delete the parent.
  This may propagate.

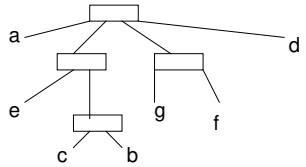




Rectangular Range Query

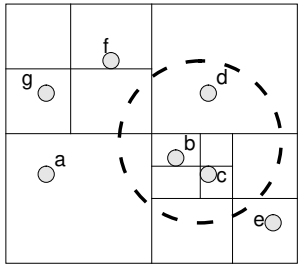
- Given a rectangle, find all the points inside the rectangle

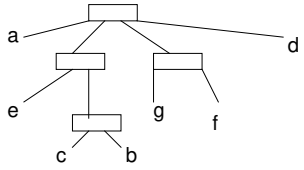




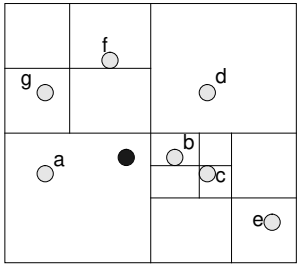
Circular Range Query

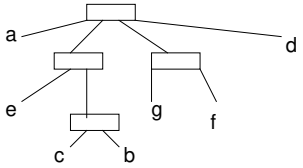
- Given a circle, find all the points inside the circle.





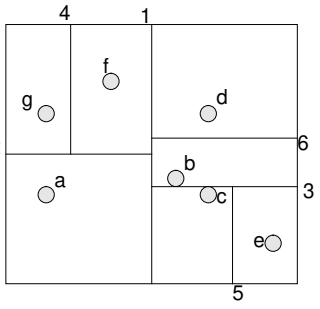
Nearest Neighbor Search

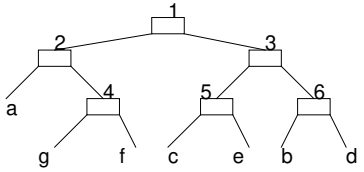




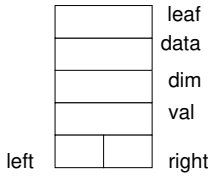
- Find a nearby point by doing a find
- Then do a circular range query to find the nearest point.

k-d Trees





K-d Trees



- Unlike Quad Trees, the decision has to be made in which dimension to split the data, and where within that dimension the split line should lie. This depends on the application somewhat but a good policy is:
  1. Choose the dimension with the widest spread.
  2. Split the points approximately in half.

Find in K-d Trees

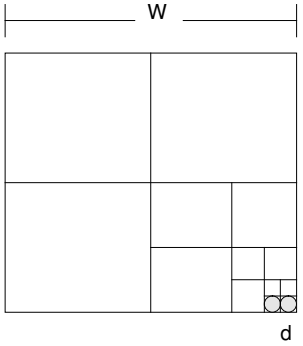
- Find(x,y,n) finds the node which has the pair (x,y) in it and returns null if there is no such node.
- Find(x: real, y: real, n:node): node {

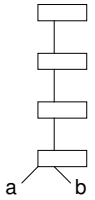
```
case
  n = null: return null
  n.leaf = true:  if n.data = (x,y) then return(n) else return null
  otherwise:
  case
    n.dim =1: if x < n.val then return(Find(x,y,n.left))
              else return(Find(x,y,n.right))
    n.dim =2: if y < n.val then return(Find(x,y,n.left))
              else return(Find(x,y,n.right))
  endcase
endcase
```


Quad Trees vs. K-d Trees

- Quad trees - spatially balanced trees
  - Number of nodes is O(n(1 + log Δ/n)) where n is the number of points and Δ is the ratio of the width original quadrant and the smallest distance between two points.
  - Height of the tree is O(log n + log Δ).
  - Supports insert, delete, find, nearest neighbor, range queries
  - For higher dimensions are somewhat bulky
  - Alternative - binary space partitioning
- K-d trees - density balanced trees
  - Number of nodes is O(n) where n is the number of points
  - Height of the tree is O(log n)
  - Supports insert, delete, find, nearest neighbor, range queries.
  - Leaner in higher dimensions

Pathological Example





$\Delta = W/d$

Let d be the distance between the closest two points. If $\Delta = W/d$ is large then potentially there is a non-branching path of length $\log_2 \Delta$ in the quad tree.