Tuesday, November 03, 2009
3:28 PM

- with our PointAxisCompare functor, we can now call

  sort (x.begin(), x.end(), $\underbrace{\text{PointAxisCompare(axis)}}_{\text{instantiating object (call its constructor)}}$)

- we can make the kdTree object "more generic" by relegating this specific instance to the template
- the Tree is now

  template < typename T, typename P, typename C >

  type of function object

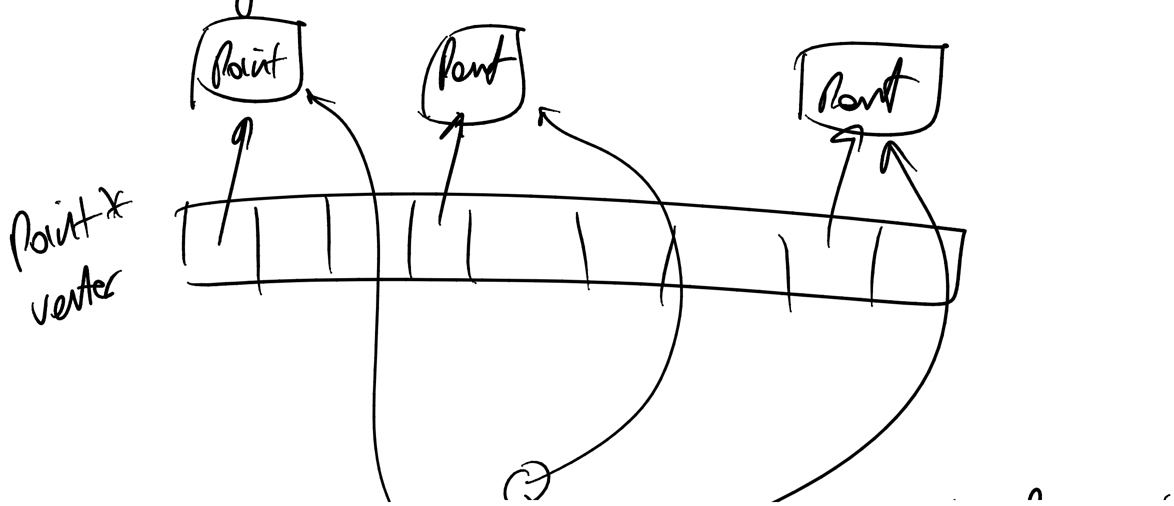
- and would be declared as

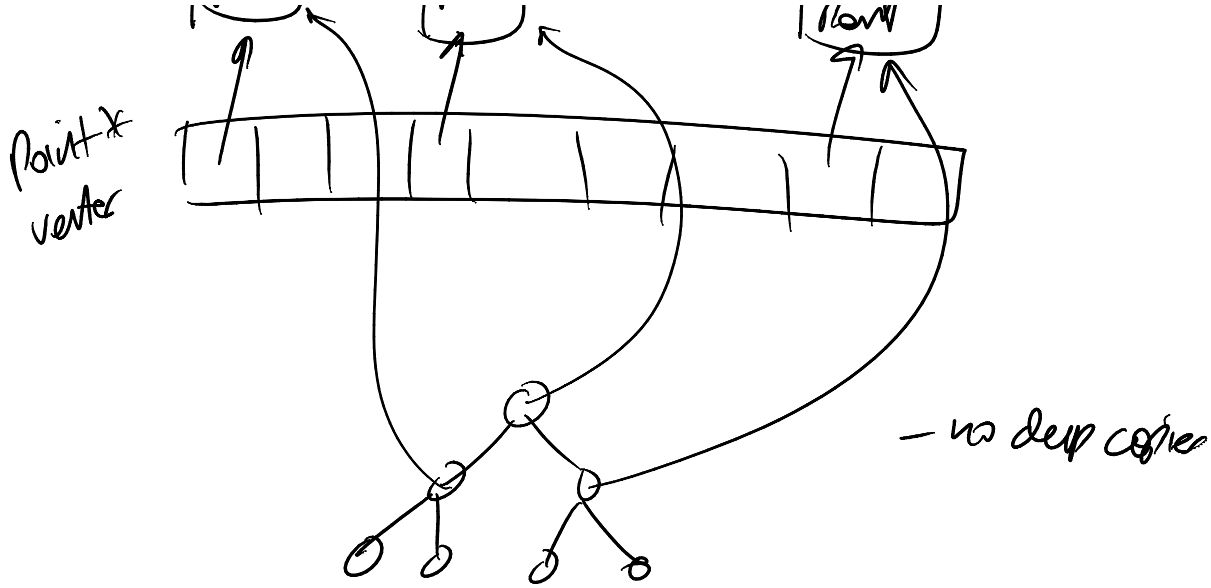
  kdTree < Point, Point*, PointAxisCompare > kdTree;

- with the sort call get replaced with

  sort(x.begin(), x.end(), C(axis));

- now we have a generic kdTree
- and it's fast because all it does is "sort" pointers





- for main.cpp, see Asg. 6 web page

Tuesday, November 03, 2009
3:47 PM

- nn, knn queries

↳ nearest neighbor. basic alg:

input: q — query object (a Point that does not necessarily exist in the kdTree, we just want the closest point)

t — tree node (root to start with)

&r — distance thus far of q to closest node (∞ initially), reference to (to allow recursive calls to update)

*&p — pointer (reference to) nn (output)

Tuesday, November 03, 2009
3:52 PM

if(t == NULL) return; // to end recursion

// compute distance from q to point pointed to by t

$$d = \sqrt{(q[0] - (*t \to data)[0])^2 + (q[1] - (*t \to data)[1])^2}$$

($\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ Euclidean dist in 2D)

(what about 3D, 4D, etc?)

↳ better to write your own

Point::distance(Point* rhs)

q.distance(t→data)

```
if ( d < r) {
    r = d          // dist to current node
    p = t→data     // current node
}
```

// traverse "closer" side of the tree

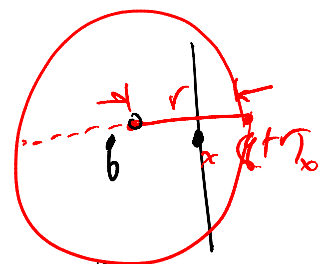
(recursive calls will override r, p, as intended)

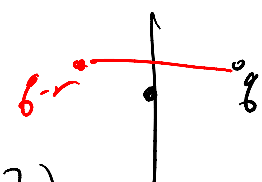
- the first approximation (leaf node) is not necessarily the closest, but from this descent-only search (thus far) we know that any potentially nearer neighbor must lie closer (than r) & so must lie within the circle defined by q & radius r

- as we return up the tree, need to check whether the current closest circle intersects the "farther" side of the tree, if so, search that subtree

```
axis = t->axis
if ( q[axis] <= (t->data)[axis] )
    nn(t->left, q, p, r)
    if ( q[axis] + r >=? (t->data)[axis] )
        nn(t->right, q, p, r)
else
    nn(t->right, q, p, r)
    if ( q[axis] - r <=? (t->data)[axis] )
        nn(t->left, q, p, r)
```

left subtree





Tuesday, November 03, 2009
4:12 PM

- knn :

k nearest neighbors : like nn()
but "~~returns~~" vector <point *>
(adds to it)
instead of just setting a single pointer

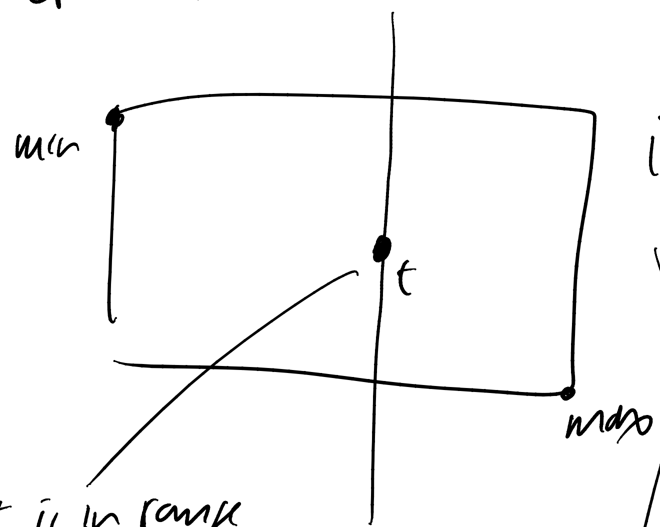
input   q is the same
p is now vector <point *>
k — how many neighbors you want

- instead of searching with a circle whose radius is closest distance yet found,
search within a circle whose radius is the k^{th} closest yet found.
UNTIL K POINTS HAVE BEEN FOUND
KEEP DISTANCE AT ∞

e.g. keep the vector in sorted order

Tuesday, November 03, 2009
4:19 PM

- range query:
  - similar to previous queries, just need to check whether range box straddles split axis



if (min[axis] < t[axis] && max[axis] > t[axis])
search both subtrees

else if on left
search left subtree

else if right
search right subtree

if t is in range
add it to <u>set of points</u>
(or vector<Point*>)