Thursday, October 29, 2009
3:23 PM

kd-tree insertion

will be Point

will be Point*

- with a templated kdNode defined as follows:

```
template <typename T, typename P>
class kdTree {
private:
struct kdNode {
    P data;  // used to store Point* pointers to data
    T min, max;  // the min, max of type Point
                 // (node's local copies of
                 //    bounding box extents)
    kdNode *left, *right;  // subtrees
    int axis;  // axis this node splits on (x, y)
    kdNode (const P& d = P(), \
            const T& in = T(), \
            const T& ix = T(), \
            kdNode* l = NULL, \
            kdNode* r = NULL, \
            int a = 0): \
    data(d), min(in), max(ix), left(l), right(r),
            axis(a)  { };
};
}
```

Thursday, October 29, 2009
3:44 PM

- Note: template <typename T, typename P>
  Lets the operator on generic type & pointer to generic type
- If you find templates confusing, you can define everything in terms of Point: Point*

e.g., class kdTree {
  struct kdNode {
    Point *data,
    Point min, max
    ⋮
  }
}

- if using templates, don't forget to specialize the kdTree class (at bottom of implementation, kdtree.cpp):

template class kdTree<Point, Point*>;
template std::ostream& operator<<(std::ostream&,
  const kdTree<Point, Point*>&);

- don't forget forward declaration at top of kdtree.h as well.

---

friend ostream& operator<< <>(...)

this says that the data type in arguments is actually templated

Thursday, October 29, 2009
3:53 PM

- our Point class is just a simple 2D point;
  (no really)

```
class Point {
  private:
    vector<double>   point;
  public:
    Point (double x=0.0, double y=0.0)
        { point.push_back(x);
          point.push_back(y); }
    // big 3
const double& operator[](int k) const
        { return point[k]; }      //accessor
    double& operator[](int k)
        { return point[k]; }      //mutator
    // no friends
    // plain method
    int dim() const { return point.size(); }
}
```

in its own point.h & point.cpp files

Thursday, October 29, 2009
4:00 PM

- in main.cpp, declare the kdtree

```
kdTree<Point, PointTo>  kdtree;
```

- in kdtree.h:

```
class kdTree {
  public:
    kdTree(): root(NULL) {}; //constructor
    :
  private:
    kdNode *root;
};
```

Thursday, October 29, 2009
4:02 PM

the basic insert mechanism (p. 551)

↑ mine differs

- in kdTree.h:

public:

```
void insert(std::vector<P>& x,
            const T& min,
            const T& max)
{ root = insert(root, x, min, max, 0); }
```

↑ depth level

public insert returns node (diff't from root)

private:

```
kdNode* insert(kdNode* p,
               std::vector<P>& x,
               const T&,
               const T&,
               int);
```

Thursday, October 29, 2009
4:07 PM

- in kdtree.cpp (among other things), we have:

```
template <typename T, typename P>
typename kdTree<T,P>::kdNode *
kdTree<T,P>::insert(kdNode *&t,
                    std::vector<P>& x,
                    const T& min,
                    const T& max,
                    int d)
{
    P      median; // median point (points to)
    T      _min, _max; // bounding box corners —
                       each is a point,
                       so _min has {x,y}
                          _max has {x,y}

    typename std::vector<P>  left, right;
    int    axis = d % x[0]->dim();
                                  don't do this if x.empty()!!
    if(x.empty()) return NULL; // list empty, end of recursion

    // find median — can use Geist's code
    // on std::vector use STL's sort routine
    // need #include <algorithm>
    sort(x.begin(), x.end(), PointAxisCompare(axis));
```

(Iterators) — what's this? a Functor, a function object

can't simply use operator< here because we need to toggle coordinate:

a functor, a function object (not a pointer to a fn, but sort of similar)

sometimes we want
```
point1[0] < point2[0]   // sort on x
```
sometimes
```
point1[1] < point2[1]   // sort on y
```
/// return to this, functor below

```
int m = x.size()/2;   index to median
// now construct left & right lists
for(int i=0; i<(int)x.size(); i++)
    if(i<m)   left.push_back(x[i]);
    else if(i>m)   right.push_back(x[i]);
    else   median = x[m];   // (i==m)
```

Note: dealing with pointers — if we had point values, we'd have a lot of copies going on here

```
// create new node
kdNode *onode = new kdNode(median, min, max,
                           NULL, NULL, axis);

// add left subtree
[set values of _min, _max]   !!
node->left = insert(node, left, _min, _max, d+1);
```

```
    [left ... U
    node->left = insert(node, left, _min, _max, d+1);

    // added right subtree
    [set values _min, _max]   !!
    node->right = insert(node, right, _min, _max, d+1);

    return node;
}
```

Thursday, October 29, 2009
4:37 PM

- what is the PointAxisCompare?
- it's function object, a class that has:

```
bool operator()
```

- where to define it?

at bottom of point.h file, below

```
class Point {...};
```

```
class PointAxisCompare
{
public:
    PointAxisCompare(int inaxis = 0) : axis(inaxis) { };   // functor constructor
    bool operator()(const Point& p1,
                    const Point& p2) const
    { return (p1[axis] < p2[axis]); }

    bool operator()(const Point* p1,
                    const Point* p2) const
    { return ((*p1)[axis] < (*p2)[axis]); }

private:
    int axis;
};
```