CSC378 Assignment 2

Due March 14 at 6:05 pm in tutorial

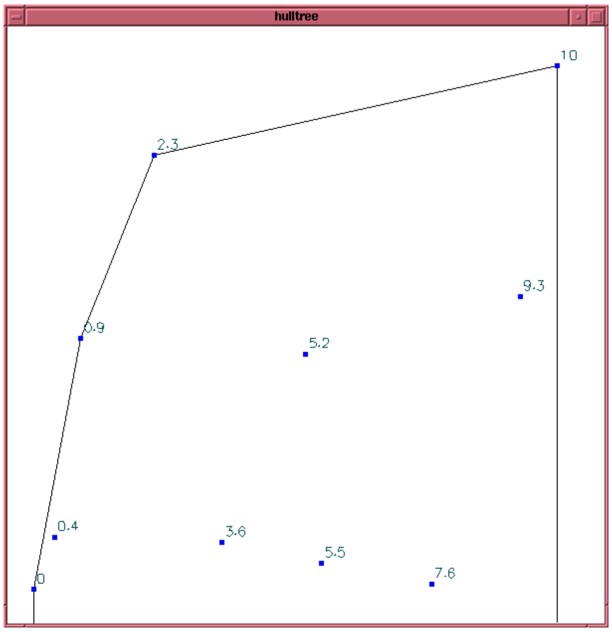
Convex Hulls

Consider a piece of wood, and a set of nails sticking vertically out of the wood. Stretch a rubber band around the nails so that it encloses all of them, and release the rubber band. Then the rubber band is the "convex hull" of the set of nails.

More formally, consider a set S of points in the plane. The convex hull is the smallest convex polygon that encloses all of the points. Note that the vertices of the polygon will be points of S.

A convex hull can be broken into two parts: the "upper hull" and the "lower hull". The upper hull is the part of the polygon that is above S. The lower hull is the part of the polygon that is below S.

The figure below shows the upper hull of a set of 10 points. There are two vertical edges added to the left and right sides of the upper hull, just to show how the points are "enclosed" by the upper hull. Each point is labelled with its x coordinate (the y coordinate is not shown).



The Operations

In this assignment you'll work on a data structure that stores the upper hull of a set S of points. There are two operations that you will provide for this data structure:

- insert point (x, y) into S
- list the points on the upper hull of S in left–to–right order

There are algorithms to build the convex hull in $\mathcal{O}(n \log n)$ time for a set of n points. If you used such an algorithm, you could insert in $\mathcal{O}(1)$ time (just add your point to a list) but it would take $\mathcal{O}(n \log n)$ time to list the points on the hull.

The data structure used in this assignment allows us to perform both insertion and hull–listing operations in polylogarithmic time (i.e. $\mathcal{O}(\log^k n)$ for some k) which is *much* better than $\mathcal{O}(n \log n)$.

The Data Structure

For a set S of points, a binary search tree is built. Each leaf stores one point of S. The leaves are ordered by x coordinate of the points: point (x_i, y_i) occurs to the left of point (x_j, y_j) if and only if $x_i < x_j$. We'll assume that no x values are duplicated. By "left of" we mean "earlier in the BST ordering."

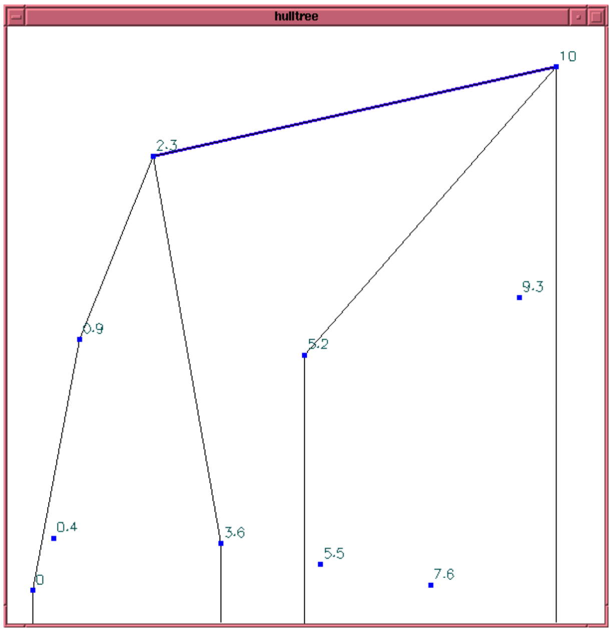
Here's the BST that stores the ten points shown above. Each leaf is labelled with the x coordinate of its point, as in the figure above. The internal nodes will be described below.

```
                                                 2.3-10
                                 +----------------------+     +----------------------+
                            0.9-2.3                                                       5.2-10
                    +---------+      +---------+                              +---------+      +---------+
                 0-0.9                         2.3-3.6                     5.2-5.5                    7.6-10
        +---------+      +-+               +-+      +-+                +-+      +-+              +-+      +---------+
     0-0.4                  0.9          2.3           3.6          5.2          5.5          7.6               9.3-10
  +-+      +-+                                                                                             +-+      +-+
0             0.4                                                                                        9.3           10
```

Each subtree of the BST represents a subset S' of S: The subset S' consists of those points stored in the leaves of the subtree. This subset S' has its own upper hull, just as the whole set has an upper hull.

Consider an internal node n in the BST. Let L be the subset of S represented by n's left subtree. Let R be the subset of S represented by n's right subtree. Each of L and R has its own upper hull. Since the leaves are ordered left–to–right by increasing x coordinate, all the points in L are left of all the points in R. That means that the upper hull of L is completely to the left of the upper hull of R.

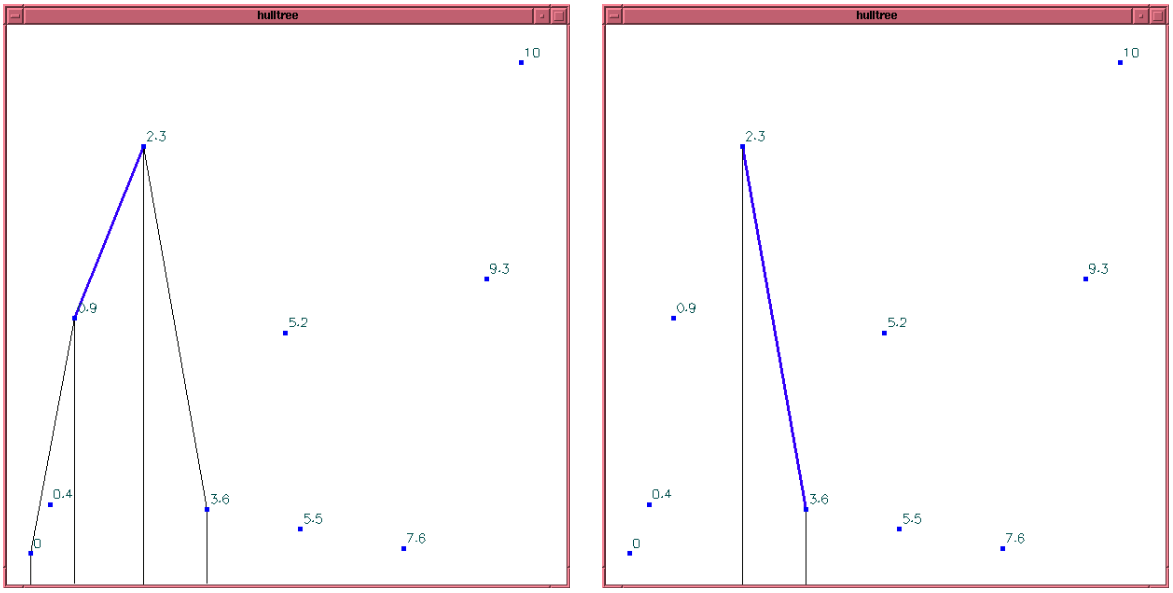
The figure below shows the upper hulls for two sets of points: In this example, n is the root, $L = \{0, 0.4, 0.9, 2.3, 3.6\}$, and $R = \{5.2, 5.5, 7.6, 9.3, 10\}$. See how these points are in the left and right subtrees of the BST shown above.



The upper hulls of L and R are joined by a "bridge" between the point 2.3 on the left and the point 10 on the right. This bridge is chosen to lie on the convex hull of the *union of the sets L and R*. For any two sets L and R, there's only one such bridge.

Each internal node of the BST stores the bridge between its two subtrees. Note that `2.3-10` is stored in the root of the BST above.

We can look farther down into the tree. Consider the internal node labelled `0.9-2.3` in the BST above. It stores the bridge between the upper hulls of $\{0, 0.4, 0.9\}$ on the left and $\{2.3, 3.6\}$ on the right. This is shown in the left figure below. In the right figure is shown the bridge `2.3-3.6` between the singleton sets $\{2.3\}$ on the left and $\{3.6\}$ on the right. There's nothing wrong with a singleton set: Its upper hull consists of the single point in the set (and is drawn with a vertical line downward just to show that it's an upper hull).



The Program

There's a working program on CDF. To see the example above, type

```
% cd ~jstewart/378/a2
% ch < niceData
```

You must be in the `a2` directory for the `ch` command to work. Experiment a bit:

- Click on a line to show the upper hulls that it bridges
- Hold down the middle button on a line to show its label
- Press `b` to turn on or off the bridge drawing
- Press `p` to turn on or off the point labelling
- Press `t` to print the BST (only useful for very small BSTs)

You can also try some other data. There's a program called `gen` that generates data. Try the following to generate 15 data points:

```
% gen 15 | ch
```

Or try `gen 100` for a big tree. You might have to wait a short time for it to appear on the screen.

Assignment Part A

Describe in clear and concise English how to do these operations:

A.1 Given the root n of a subtree, output the points on the upper hull of that subtree in left–to–right order.

A.2 Insert a new point into the tree.

You may provide pseudocode where it helps your description. But the reader must be able to understand your algorithms without looking at the pseudocode. It's only there to make things a bit clearer; it's not there to describe the algorithm!

Each node of the tree stores the usual left, right, and parent pointers. There's also a colour (it's a red–black tree). In addition, each node n stores:

- `max`: the maximum x value of all points stored in the leaves of the subtree rooted at n.
- `bridge.left` and `bridge.right`: The (x, y) coordinates of the left and right points of the bridge. These are the actual coordinates and are *not* pointers to the leaves that store the points. A leaf node will have `bridge.left` = `bridge.right`.

Assignment Part B

Implement your two operations. The code in the `a2` directory is missing three functions:

`enumerateHullPointsFromRoot( n, enumFunc, min, max )`

This function is called with `n` pointing to the root of a subtree, with `enumFunc` the name of a function that is to be called with each point that you enumerate, and with `min` and `max` equal to minus infinity and plus infinity, respectively.

When you find a point `p` on the upper you should call `enumFunc`. So your code might look vaguely like:

```
Point p;

for each point p on hull, in left-to-right order
  enumFunc( p );
```

Your calls to `enumFunc` must be in the correct order: Call `enumFunc(p)` before `enumFunc(q)` if `p` is to the left of `q` on the upper hull.

You may find the `min` and `max` useful to bound the interval in which you're currently processing points of the upper hull. If your algorithm does not need these, just ignore them (but keep them in the argument list so you don't have to change the rest of the code).

In your code, do *not* make any calls to `findBridgeWithPointOn...()`. Such a solution will get zero marks.

`insert( p )`

This function is called to insert a new point, `p`. *Hint:* Recall that points are only stored in leaves of the tree. If you hang a new node n off of an already–existing leaf node m, then m is no longer a leaf node and can no longer store a point. So where should the point that used to be in m go?

When you create a new node, you'll have to fill in the fields `bridge.left`, `bridge.right`, `max`, and `colour`. If `n` points to your new node, just set these fields like this:

```
Node<Info> *n = new Node<Info>( NULL );

n->key->bridge.left  = ...
n->key->bridge.right = ...
n->key->max          = ...
n->key->colour       = RED;
```

(Recall that a new node in a red–black tree is red.) See `hulltree.h` for details of the classes. Your new code should be put in the three empty functions that are provided in `hulltree.C`.

There's a very useful function provided for you: `computeBridge( n, m )` returns the bridge between the upper hulls of the subtrees rooted at `n` on the left and `m` on the right. You'll need this when inserting.

`fixupAfterRotate( x, y )`

You'll also have to handle the rotations that occur in the red–black tree. This is actually not too difficult: The function `fixupAfterRotate( x, y )` is called after each rotation, with `x` the parent and `y` the child *after* the rotation. You only need to add a very little bit of code to this function.

A `Makefile` is provided, so all you have to do is copy the whole `a2` directory to your own directory and type `make`:

```
% cd
% cp -r ~jstewart/378/a2 .      <-- don't forget the period there!
% cd a2
% mv ch ch-working
% make
```

Of course, the code initially won't work. You can always test your own code against the output of the `ch-working` program. A good way is to generate a test file (like `niceData`) and pipe it in to your program and in to `ch-working`, and then compare the outputs. If the last line of your test file contains the single letter `p` then the BST will be printed after all points have been inserted. If the last line of your test file contains the single letter `e` then the upper hull of the entire point set will be printed after all points have been inserted.

What To Hand In

1. Hand in your written descriptions from Part A.
2. Hand in hardcopy of the code the three functions that you wrote: `enumerateHullPointsFromRoot()`, `insert()`, and `fixupAfterRotate()`.

   If you know that your code doesn't work in certain cases, attach a short description of what happens and what you think is wrong. This might get you more (partial) marks than otherwise.

   You shouldn't have needed to modify anything else. But if you *did*, then hand in hardcopy of the other parts that you modified. But don't hand in a listing of any part of the code that you didn't modify.
3. Submit your entire directory electronically: Assuming that it's called `a2` in your home directory, do the following:

   ```
   % submit csc378h ~/a2/*
   ```

   Ignore any message about directory `windows` not being submitted.