Homework 9

Due: Monday, December 2nd, 11pm

This worksheet contains only the *written* parts of HW9. The coding instructions will be released Thursday and be due Friday (12/06). The coding portion of this assignment is long, so you should **aim to finish the written part early (Friday night (11/29) is a good goal)** to have plenty of time to work on the code.

Submission

After completing all parts below, submit your solutions as a PDF on Gradescope under "**HW9 Written**".

Don't forget to check that the submitted file is up-to-date with all written work you completed!

Make sure your work is legible and scanned clearly if you handwrite it, or compiled correctly if you choose to use LaTeX. Match each HW problem to the page with your work when you turn in. If your work is not readable or pages are not assigned correctly, you will receive a point deduction.

The next two pages contain the same definitions that we covered in section. Feel free to just skim/refer back to for reference when you start the tasks.

In this assignment, we will work with locations on a 2D map. For the sake of variety, rather than representing these as pairs, we will represent them as records as follows:

$$\textbf{type } \text{Location} := \{x : \mathbb{R},\ y : \mathbb{R}\}$$

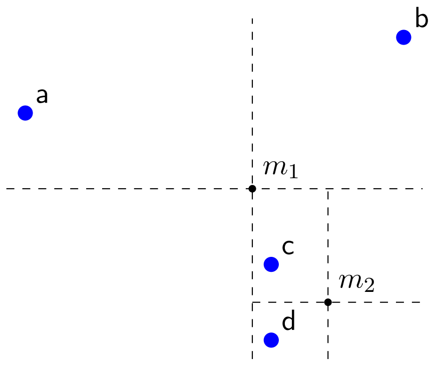
A basic operation on locations is calculating the distance between them. This can be done using the function dist : (Location, Location) $\to \mathbb{R}$, which is defined as follows:

$$\text{dist}(\ell, r) := \sqrt{(\ell.x - r.x)^2 + (\ell.y - r.y)^2}$$

We will be interested in finding the location, from amongst a list of possibilities, that is closest to a given target location. The simplest way to do this would be to calculate the distance of each location in the list to the target location. However, we can do this more efficiently by using a better data structure, namely, a tree, similar to the one we used in HW8, that recursively break up the points into smaller rectangles until each rectangle is either empty or contains a single point.

Trees of Locations

For example, with the four blue points below, we would start by splitting them at point "m_1" (the centroid of those locations). The NW and NE regions then contain only a single point and the SW region is empty. The SE region, however, still contains two points, so we would split them again at point "m_2", which leaves four regions containing at most one point.



We can represent this tree with the following inductive data type:

$$\begin{aligned}\textbf{type } \text{LocTree} \ :=\ & \text{empty} \\ |\ & \text{single}(\text{loc} : \text{Location}) \\ |\ & \text{split}(\text{at} : \text{Location},\ \text{nw} : \text{LocTree},\ \text{ne} : \text{LocTree},\ \text{sw} : \text{LocTree},\ \text{se} : \text{LocTree})\end{aligned}$$

The constructor "empty" represents a region that is empty, while "single(s)" represents a region containing a single location at location s. The final constructor "split(m, nw, ne, sw, se)" represents a region that splits at location m into the four regions to the NW, NE, SW, and SE, respectively.

With those definitions, the example above would be represented with the following tree:

$$\begin{aligned}&\text{split}(m_1, \text{single}(a), \text{single}(b), \text{empty}, \\ &\quad \text{split}(m_2, \text{single}(c), \text{empty}, \text{single}(d), \text{empty}))\end{aligned}$$

Regions

Note that, while each node represents some rectangular region in space, the bounds of that region are not recorded in any field. That said, we can easily calculate the bounds of the region as we work down to that node, recursively, through the tree.

Specifically, we can store a rectangular region as an instance of the following type

$$\textbf{type}\ \text{Region} := \{\text{x1}\colon \mathbb{R},\ \text{x2}\colon \mathbb{R},\ \text{y1}\colon \mathbb{R},\ \text{y2}\colon \mathbb{R}\}$$

The region R contains the location ℓ iff $R.\text{x1} \leqslant \ell.x \leqslant R.\text{x2}$ and $R.\text{y1} \leqslant \ell.y \leqslant R.\text{y2}$. Thus, the following region includes every point in the plane:

$$\text{EVERYWHERE} := \{\text{x1}\colon -\infty,\ \text{x2}\colon \infty,\ \text{y1}\colon -\infty,\ \text{y2}\colon \infty\}$$

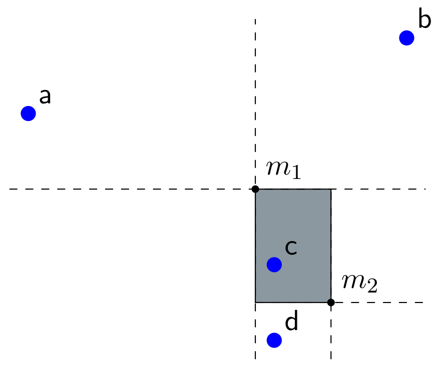
The following functions $\text{NW}, \ldots, \text{SE} : (\text{Location}, \text{Region}) \to \text{Region}$ return the intersection of the region passed in with one of the quadrants of a node that was split at the location passed in. For example, $\text{NW}(m, R)$ would return the region that includes only the area that is both inside of R and northwest of m (i.e., an x coordinate that is $\leqslant m.x$ and a y coordinate that is $\leqslant m.y$).

These functions assume that m falls within the region R, and they are defined as follows:

$$\begin{aligned}
\text{NW}(m, R) &:= \{\text{x1}\colon R.\text{x1},\ \text{x2}\colon m.x,\ \text{y1}\colon R.\text{y1},\ \text{y2}\colon m.y\} \\
\text{NE}(m, R) &:= \{\text{x1}\colon m.x,\ \text{x2}\colon R.\text{x2},\ \text{y1}\colon R.\text{y1},\ \text{y2}\colon m.y\} \\
\text{SW}(m, R) &:= \{\text{x1}\colon R.\text{x1},\ \text{x2}\colon m.x,\ \text{y1}\colon m.y,\ \text{y2}\colon R.\text{y2}\} \\
\text{SE}(m, R) &:= \{\text{x1}\colon m.x,\ \text{x2}\colon R.\text{x2},\ \text{y1}\colon m.y,\ \text{y2}\colon R.\text{y2}\}
\end{aligned}$$

For example, $NW(m, R)$ leaves left side of the region at $R.\text{x1}$, but it only allows the region to extend to the right as $m.x$. Any point further to the right than that is outside of the region to the NW of m.

Returning to our example from before



the shaded region, which is the region that lies in the SE quadrant of the split at m_1 and then, within that, the NW quadrant of the split at m_2, can be calculated as $\text{NW}(m_2, \text{SE}(m_1, \text{EVERYWHERE}))$.

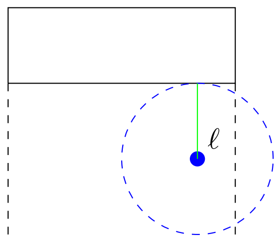
Task 1 – He's Making a Dist, and Checking It Twice [8 pts]

We defined a function "dist" before that calculates the distance between two locations. It will also be useful for us to have a function distTo : $(\text{Location}, \text{Region}) \to \mathbb{R}$ that calculates the distance from the given location to the *closest* location in the given region. This gives us a *lower bound* on the distance to any location within that region.

To define this function, $\text{distTo}(\ell, R)$, we will break our analysis into a few different cases. The first case is when ℓ is within R. In that case, the distance from ℓ to R is 0.

(a) Write an expression that is true iff location ℓ falls within the region R.

(b) If ℓ is not within R, then our next case is when it is directly below, above, to the left, to the right. For example, if ℓ is directly below, then the picture looks like this:



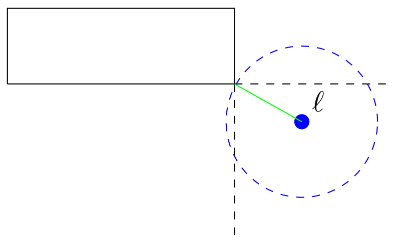
As we can see, the closest location in the region R is directly above ℓ.

Write an expression that is true iff location ℓ is directly below R.

(Note that larger y values are farther **down** on the page.)

(c) Write an expression that calculates the distance from ℓ to the closest location in R when ℓ is directly below R.

(d) If ℓ is not within R and not directly below, above, left, or right, then it is in one of the regions diagonally away from R (above and left, above and right, below and left, or below and right). For example, if ℓ is below and right, then the picture looks like this:



In this case, the closest point is the bottom-right corner of R.

Write an expression that is true iff location ℓ is below and right of R.

(e) Write an expression that calculates the distance from ℓ to the closest location in R when ℓ is below and right of R. Feel free to use the function dist defined before.

Finding the Closest Location

The function closest : (LocTree, Location, Location) → Location finds the closest location in the given tree to a given location. More specifically, when invoked as closest(T, ℓ, c), it returns either the closest location in the tree T to ℓ or the location c, whichever is closer to ℓ. It is defined as follows:

$$\begin{aligned}
\text{closest}(\text{empty}, \ell, c) &:= c \\
\text{closest}(\text{single(s)}, \ell, c) &:= c && \text{if } \text{dist}(c, \ell) \leqslant \text{dist}(s, \ell) \\
\text{closest}(\text{single(s)}, \ell, c) &:= s && \text{if } \text{dist}(c, \ell) > \text{dist}(s, \ell) \\
\text{closest}(\text{split}(m, \text{nw}, \text{ne}, \text{sw}, \text{se}), \ell, c) &:= \text{closest}(\text{nw}, \ell, \\
&\qquad \text{closest}(\text{sw}, \ell, \\
&\qquad\qquad \text{closest}(\text{se}, \ell, \\
&\qquad\qquad\qquad \text{closest}(\text{ne}, \ell, c))))
\end{aligned}$$

The case for split(. . .) performs a tree traversal. It first recurses into the NE quadrant. That call returns the closest location to ℓ within the NE quadrant or, if no location is closer, c. The location returned from the NE quadrant is passed as the third argument to the recursive call into the SE quadrant. That call returns the closest location to ℓ within the SE quadrant or, if no location is closer, the third argument passed to it, which is the closest location from the NE quadrant or, if no location is closer, c.

This process continues through the SW and NW quadrants, so that the location returned from the final recursive call is the closest to ℓ within the NW region or, if none is closer, the closest to ℓ within the SW region or, if none is closer, the closest to ℓ within the SE region or, if none is closer, the closest to ℓ within the NE region or, if none is closer, c. in other words, it returns the closest location within the entire subtree to ℓ or c, whichever is closer.

Note in particular that, if *every* location in the tree is farther away from ℓ than c is, then closest will return c. (This can be proven formally by induction.) We will need that fact later on!

The particular order chosen here — NE, SE, SW, NW — traverses the quadrants in *clockwise* order. We could traverse the quadrants counter-clockwise or in any order we want by switching the order of nw, . . . , ne in the recursive calls.

So far, our closest calculation still calculates the distance between the target location ℓ and every point in the tree. We can fix this by excluding any quadrant that is farther away from ℓ than c is. When this is true, we know immediately that a recursive call to closest on the subtree for that quadrant would return c, so there is no reason to traverse all of its locations. . .

Task 2 – Going Weak in the Trees [20 pts]

We will define a function cl : (LocTree, Location, Region, Location) → Location that looks like closest but takes an extra argument that is a region containing all the points in the tree. It will use that region to avoid recursing into quadrants of split nodes when they cannot contain a location closer than c.

More precisely, cl is defined as follows:

$$\begin{aligned}
\mathsf{cl}(\mathsf{empty}, \ell, R, c) &:= c \\
\mathsf{cl}(\mathsf{single(s)}, \ell, R, c) &:= c && \text{if } \mathsf{dist}(c, \ell) \leqslant \mathsf{dist}(s, \ell) \\
\mathsf{cl}(\mathsf{single(s)}, \ell, R, c) &:= s && \text{if } \mathsf{dist}(c, \ell) > \mathsf{dist}(s, \ell) \\
\mathsf{cl}(\mathsf{split}(m, \mathsf{nw}, \mathsf{ne}, \mathsf{sw}, \mathsf{se}), \ell, R, c) &:= c && \text{if } \mathsf{dist}(\ell, c) \leqslant \mathsf{distTo}(\ell, R) \\
\mathsf{cl}(\mathsf{split}(m, \mathsf{nw}, \mathsf{ne}, \mathsf{sw}, \mathsf{se}), \ell, R, c) &:= \mathsf{cl}(\mathsf{nw}, \ell, \mathsf{NW}(m, R), && \text{if } \mathsf{dist}(\ell, c) > \mathsf{distTo}(\ell, R) \\
&\qquad \mathsf{cl}(\mathsf{sw}, \ell, \mathsf{SW}(m, R), \\
&\qquad\quad \mathsf{cl}(\mathsf{se}, \ell, \mathsf{SE}(m, R), \\
&\qquad\qquad \mathsf{cl}(\mathsf{ne}, \ell, \mathsf{NE}(m, R), c))))
\end{aligned}$$

Here, the case for split nodes from closest has become two cases. In the first case, when $\mathsf{dist}(\ell, c) \leqslant \mathsf{distTo}(\ell, R)$, we know that all the locations within this subtree are farther away than c is from ℓ, so we skip the recursive calls and simply return c. It is only in the second case, when $\mathsf{dist}(\ell, c) > \mathsf{distTo}(\ell, R)$, that we perform the recursive calls.

Note how we are using the functions NW, NE, SW, SE to calculate a smaller region that only includes the part of R that falls within that quadrant and passing that region in the recursive call. (This implements the procedure we described earlier for calculating the region for a given node in the tree.)

(a) Prove that, if the region R contains all locations in the tree T, then $\mathsf{cl}(T, \ell, R, c) = \mathsf{closest}(T, \ell, c)$. Your proof should be by structural induction on T.

Feel free to use the fact that, if R contains all the locations in $\mathsf{split}(m, \mathsf{nw}, \mathsf{ne}, \mathsf{sw}, \mathsf{se})$, then $\mathsf{NW}(m, \mathsf{nw})$ contains all the locations in nw and likewise for ne, sw, and se. (This follows from the representation invariant for split nodes and the definitions of these functions.)

Remember that, because $\mathsf{distTo}(\ell, R)$ is the distance from ℓ to the closest location in R, any other location s in R must satisfy $\mathsf{dist}(\ell, s) \geqslant \mathsf{distTo}(\ell, R)$. Hence, if $\mathsf{dist}(\ell, c) \leqslant \mathsf{distTo}(\ell, R)$ $(\leqslant \mathsf{dist}(\ell, s))$, then we know already that s cannot be closer to ℓ than c. Thus, if R contains all the locations in the tree, then none of them can be closer to ℓ than c, and we can simply return c immediately as the closest, which is exactly what cl does in its definition above.

Hints:

- Do not forget about the section materials! There is a suspiciously similar problem we covered there that you should use as a direct reference for how to organize your proof.
- Your base case and inductive step will need to contain a proof by cases. Don't forget the to state your cases and write your conclusion for these sub-proofs!
- Our claim states "if the region R contains all locations in the tree...," but you still need to state that the claim holds in cases when the region R does not contain all the locations.

(b) Explain, in your words, why your proof idea would still work if we defined cl to traverse the quadrants in counter-clockwise order or any other order we choose.