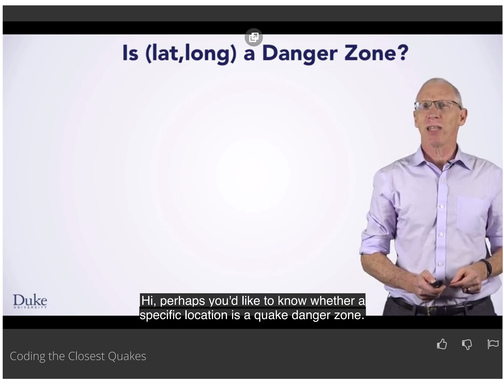

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
Hi, perhaps you'd like to know whether a specific location is a quake danger zone. In this coding demo, we'll describe how to find the closet ten earthquakes to where you live, or to a city where a friend lives. We'll write code for a specific location, like New York City or Jakarta, Indonesia but the code will work for any location. Our code will find the ten closest quakes, but we'll be able to replace ten with any number. You've solved a similar problem before. Find the minimal or maximal element in a list of elements. This is similar. We'll find the closest quake to a specific location.

0:42
We'll then repeat the process but first, we'll remove the closest quake from the data set. In the same algorithm, we'll find the second closest quake. We need to remove the closest quake but we don't want to change the list of quakes. We need to make a copy to avoid removing quake data. And we'll use 10 and Jakarta, Indonesia in the demo but you'll be able to find the 57 closest quakes to any city you choose.

1:09
The seven step process is implicit in what we're doing here. Can you find the closest quake to a specific location? You've seen this algorithm before, finding the minimal or maximal element in a list. We'll use the same steps here. Or keep track of the index in to an array list of the closest quake. The one with the minimal distance to Jakarta and each time we examine a quake entry, we'll update the closest index value if we need to. In solving the problem with paper and pencil, we'd cross out the minimal closest quake entry after finding it, so that we could repeat the process and be sure we wouldn't find the same quake again. In programming, we'll remove the closest element to avoid finding it as we repeat the process of finding the closest quake. Let's write some code. As we look through the find closest quakes method here, we see that we've created an earthquake parser, an array list of quake entries that we fill with data using the parser. Then I've created a location variable named Jakarta and I've used the latitude and longitude of Jakarta, Indonesia. I've called the get closest method which is the one we're going to write during this lesson. I've passed at the list of data that I read from the parser. The location that I'm interested in which is Jakarta and a number 10, so that I can find the 10 closest earthquakes to Jakarta, Indonesia. The rest of the method here is based on printing out that data. If I run the program now by right-clicking and saying find closest quakes, we can see that data was read for 1,584 earthquakes from the USGS source, but none were found, and that's because I haven't yet written the method that we're gonna write now. So let's look at that method. The method that we're gonna write is getClosest. We can see in getClosest, that it has three parameters, an array list of quake data. That's the source of the data about earthquakes that we're going to use. The current location that in our demo is Jakarta, but could be any variable that you pass. And how many, which is ten that you've seen before, just to find the single closest entry. We've done that before. And we did that by tracking the minimal index. So i'm going to have a variable named minindex which I initialized to zero. That's the location of the closest earthquake to where I am. And then I'm gonna loop over all the other data by starting at one, going up to two. The parameter quakeData, I will use its size.

3:53
And I'm gonna look at each one of those entries.

4:00
By obtaining the quakeData from the array list

4:08
Using the .get method. And then I'm going to see if the current quake distance is.

4:20
Location, which is Jakarta.

4:25
So, this is current, The Jakarta reference, so I've used current here, which is not a good one so we'll just call this quake. If quake is, Close to current, and if that's less than the distance from.

4:49
My minimal index which is quakeData.get(minIndex), I'm working hard here to write this, .distanceTo(current). That's a long list to fit on that line, so I'm gonna break the line to make it easier to read. If the distance from the current quake here that I've read, quake. If the distance from that to Jakarta is less than the distance from the smallest location to Jakarta, then I need to remember a new minimal distance. And so I will reset the minimal index variable to K. When I'm done I need to make sure that I store

5:37
in my return array the location that's closest to Jakarta, which I get from saying quakeData.get(minIndex).

5:50
Let's see if this code compiles.

5:54
There is no distanceTo method. That's because I've forgotten the name of the method, so I will use the handy Ctrl+Space, which tells me the names of all the different functions that I can use on a location variable. And we can see here that that is the quake entry class, and I've made the canonical mistake of forgetting that I need the location variable. Not just the current quake, but I need the quake location. So I will set location to quake.getLocation, and then I will use loc, .distanceTo. And getLocation().distanceTo. That's a mouthful, so I'll go over it to make sure once it's compiled. The program has compiled, and I will now run through it very quickly to make sure we're all on the same page.

6:52
My minimal index will remember where in

6:56
array list quake data my closest location is. As I loop over all the data I get the quake entry, stored in variable quake. I use quake.getLocation, to get the location of the current one I'm iterating over. And then I'm comparing the distance from that location, which is the one I'm iterating over in this loop.

7:22
I compare it to the closest quake I've seen so far. That's what minIndex is, the closest quake I've seen so far. And I do that by asking, if this distance is less than the distance to current, which is Jakarta then I will remember, and reset minIndex. If I run this program, by coming out here, right clicking, creating a new object.

7:49
And then invoking the findClosestQuake.

7:54
I will see that the closest quake found is 223 kilometers from Jakarta. And this is where it is, it found one. If I want to find the ten closests quakes, I simply need to take this loop that I've done here, and put that in another loop, that iterates how many times?

8:17
I can do that simply, by putting a for loop around this.

8:35
Let me make sure that is down here.

8:38
And taking all this code and indenting it.

8:43
Let me just make clear what I've done.

8:46
This is the code I'd already tested. That looped through all the arrays, quake entry objects, finds the smallest one and adds it to the variable I'm returning. If I repeat this over and over again I'll find the closest quake. Then the next closest quake, and then the next closest quake. The only problem is once I found the closest quake, I'll keep finding it over and over again. We can see this by running the program.

9:18
Find the closest quakes. And we see we've found the same one over and over again. Not surprising we've found ten but we kept finding the closest one. That's a simple thing to fix. After we find it we simply remove it from quakeData. So our array list have a remove element so I will simply remove the element after I get it. So, quakeData.remove(minIndex). After I've obtained the closest quake, I'll remove it so I never find it again. Compile that program.

9:56
Create a new, ClosestQuake object and, find the closest ten quakes. I can see here, I found one quake that's 223 kilometers away. And the tenth closest quake, which is more than 2000 kilometers away. I'm reasonably confident that my program works correctly. I'd like to verify that by running it through some real data. The only problem I have, is that, right now, after finding the ten closest quakes, I've removed them from quakeData. That would be a simple fix to make. Create a copy of quakeData, and use that copy to show you how that would work, I'd simply say, create a QuakeEntry and copy, make a new array list, and construct that from quakeData. Now simply use copy everywhere in my code, instead of using quakeData, so that, no, I'm not moderating my parameter. Instead of quakeData.size, I'll use copy.size. Instead of quakeData.get, that one's fine. Down here I'll use copy, and here I'll use copy.

11:13
When I compile that program, there are no syntax errors, I wanna make sure I'm not using quakeData anywhere. I'm looking through my code, I've used copy, I've used copy, quakeData, I need copy there.

11:27
And now I'm set to go.

11:29
One last run, Find the closest, and there they are. I found ten closest quakes to Jakarta, Indonesia. Let me just remind you of what we did there. I took my standard code for finding the closest quake to one that was specified by the parameter current.

11:53
And once I tested that, and found one, I took that code and added this loop around it, so that I could repeat it this many times, how many. I then realized, I was modifying my parameter, and so I made a copy.

12:07
Happy programming, don't get too close to an earthquake.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?