COMP 215 Assignment #7

Due November 26th (note this is a Monday) at 11:55PM. We've given you a lot of time on this assignment (since I didn't want to have it due the week before Thanksgiving) but it is challenging—start early! Plus, A8 will be assigned before A7 is due.

1 Description

In A7, you are asked to provide an implementation of the `IMTree` interface which allows for indexing of data embedded in an arbitrary metric space. In other words, you are asked to provide an implementation of the M-Tree data structure from class. Note that the `IMTree` interface is fully generic in the sense that the key type and the data type stored in the tree are type parameters. It will work with *any* key type that implements the `IPointInMetricSpace` interface. All `IPointInMetricSpace` requires is that you write a `getDistance()` method. So it is trivially easy to take an `IMTree` implementation and use it to index any data type for which you have a metric distance that makes sense: floating point numbers, text strings, high dimensional vectors, images, etc. For example, to use the `IMTree` interface to implement a dictionary, you could create a new class called `StringInMetricSpace` that implements the `IPointInMetricSpace` interface, and has a reference to a `String` object inside of it. You could then use the edit distance code from earlier this semester to implement `getDistance()` for the `StringInMetricSpace` class. Wrap up all of the words in the English language in `StringInMetricSpace` objects—several such lists are available on the web—and insert them into an M-Tree. With very little effort you've got yourself a dictionary that can quickly search for alternate word spellings.

Your own implementation class should be called `MTree`, and it should have a constructor that takes exactly two integers: the first is the max number of entries in an internal tree node, and the second is the max number of entries an a leaf node.

We are giving you a lot of freedom on this assignment. You may design your M-Tree as you see fit. You must, however, actually implement an M-Tree as described in class and your design decisions should be reasonable and defensible. We would also recommend that you do not ignore everything we've taught you in class so far and do not make decisions that go against previously taught principles! That will only make the assignment harder. As you work, think very carefully about all of the helper classes that you will need in order to get things working in the easiest and most error-free way possible. Some careful thought early on could go a long way towards reducing the amount of effort required on the assignment.

2 Grading

80% of the assignment comes from passing all of the test cases, and 20% is associated with documentation and design. We are publishing the test cases with the assignment.

Because we think the assignment is challenging, we will have four grading "tiers"—everyone should be able to get to the first tier and get a reasonable grade on the assignment. In fact, you don't even need to implement a tree to pass the first tier tests. We think most people should think about just taking the time to pass the first tier (without using a tree at all) as a warmup on the assignment, as this should help with understanding the ins and outs of how to handle spheres and data points in the generic metric space you are given.

Here is how the "tier-based" grading will work. We are publishing 32 test cases. The first eight should be considered relatively easy, since to pass them you don't have to handle any node splits. A "tree" (which

is not even a tree!) with a single leaf node and no internal nodes will get you all of those test cases. Those eight are worth 40% of the "test case" portion of your grade on this assignment.

The next eight test cases are "moderate". To pass those, you need to handle leaf node splits but no internal node splits. Those test cases are worth 20% of the "test case" portion of your grade on this assignment.

To pass the next eight, you need to handle both leaf node and internal node splits. Those eight together are worth 20%.

None of the first 24 cases test "top k" queries; they all test range (sphere) queries. The last eight tests run "top k" queries over trees that were constructed in such a way that they would have leaf and node splits. Passing these test cases is the final 20% of the "test case" portion of your grade on this assignment.

3 Getting Started

To start, you should download the following files:

1. `IMtree.java`: the interface that the M-Tree should implement.
2. `IPointInMetricSpace.java`: the interface that the keys stored in the M-Tree must implement. All keys must "live" in a metric space and support a distance operation; this interface specifies that distance operation.
3. `OneDimPoint.java` and `MultiDimPoint.java`: two classes that implement the `IPointInMetricSpace` interface. They are used in testing.
4. `DataWrapper.java`: a class that is used to wrap up the result of a query to the M-Tree. Specifically, when you issue a query over an M-Tree, it will return an array list of `DataWrapper` objects.
5. `MTreeTester.java`: contains the 32 test cases described above.
6. `doublevector.jar`: contains the two `IDoubleVector` implementations (used by the `MultiDimPoint` class and the test cases).
7. `mtree.jar`: contains a reference implementation of `IMTree` that is also used by the test cases.

Use all of these to create a `HW7.drjava` project and you are ready to go.

4 Turnin

On Owlspace, you should turn in the `HW7.drjava` project file that DrJava creates when you create your project, as well as any and all `.java` source files or `.jar` files that you create or use in your project. The goal here is for us to be able to simply fire up DrJava using your `.drjava` project, and everything should work without having to move files around or add files to the project. Please comment out any debugging output from your code before you turn it in. Please upload each file directly. Do not create an archive (zip, rar, etc.).

Be super-careful and double check after you submit. Make sure you have turned in exactly what you thought you have turned in. **Make sure you turn in your Java source files and NOT your class files!** On each and every assignment, one or two people neglect to turn in their Java source files.

5 Academic Honesty

Please keep the academic honesty policy in mind as you work on the assignment.