CS3114/5040 (Fall 2025)
PROGRAMMING ASSIGNMENT #2

Due Wednesday, October 15 @ 11:00 PM for 100 points

Due Tuesday, October 14 @ 11:00 PM for 10 point bonus

Note: This project also has three intermediate milestones. See the Piazza forum for details.

Assignment:

This project will support a database of city records (containing a city name and two dimensional coordinates). You will use two tree structures as indices, one to support search by name and the other to support search by location.

The Trees:

To search for city records that match a given name you will use a Binary Search Tree (BST). You may use the code from OpenDSA (Module 7.11) to help you write your BST, though it will require a fair amount of revision to satisfy the needs of this project. You may use BST code that you have previously written, such as for another course — but make sure that it generates the same tree as the code in Module 7.11 would (or you won't pass the reference tests). Note that your BST must insert equal values to the **left**. On deletion, if the deleted node has two non-null children, then it is replaced by the node with **maximum** value from the **left** subtree.

To support searches by 2D locations, you will use a kd tree. A binary search tree gives expected O(log n) performance for insert, delete, and search operations (if you ignore the possibility that it is unbalanced). While this allows you to quickly insert and delete records, and quickly locate them by key value or within a key range, the BST does not help when doing a two-dimensional coordinate search. You could combine the (x, y) coordinates into a single key and store cities using this key in a second BST. That would allow search by coordinate, but would not allow for efficient **range queries** – searching for cities within a given distance of a point. The problem is that the BST only works well for one-dimensional keys, while our coordinates are two-dimensional keys.

The kd tree (see Module 15.5 of OpenDSA) is one of many **hierarchical data structures** commonly used to store **spatial** data such as 2D locations. It allows for efficient insertion, deletion based on 2D coordinates (these operations cost $O(\log n)$ time, similar to a BST). They also allow for efficient region search queries, as explained in Module 15.5. You may use the code sketches given in OpenDSA to help you write your kd tree implementation.

Invocation and I/O Files:

The name of the program is `GISProj` (for Geographic Information System). Your program will implement an interface called `GIS`, with this interface implemented by a class named `GISDB`. Graded reference tests for the project will work by making calls to the `GIS` interface. Note that this means none of the reference tests will read from or write to `System.out`, so reference tests will not conflict with whatever debug print statements you happen to implement. You may add other test files, but all tests that you add to the file `GISTest.java` should not call any internal methods of your project that were not specified here. One service that Web-CAT will provide for this project is to run any tests that you place in `GISTest.java` against the project reference implementation. If any of your tests fail against the reference implementation, that will indicate some mistake that you have made in your understanding of the project requirements.

The interface, example tests, and other starter code will be made available in the Starter Code available from Eclipse under `Project -> Download Assignment`. The interface, example tests, and other starter code will be made available in the Project 1 Starter Code available from Eclipse under `Project -> Download Assignment`.

Design Considerations:

Your main design concern for this project will be how to implement the BST and kd tree to work together. You probably will want to create a City class to store the data for a city record (and to support methods like `toString` and `equals`). There is no reason for the BST to understand City records (you can use Java generics to handle processing the records). However, your kd tree implementation is free to use City records directly if you like (this would make coordinate processing easier).

Your BST and kd tree design must meet the following requirements.

1. No BST or kd tree node may store a pointer to its parent.
2. Kd tree nodes may not store position, size, or depth information (aside from what is in their City record). If you need anything like that, it should be passed into the method as a parameter.

Your project will need to have at least the following classes: `GISProj` to include the main routine, the `GIS` interface that we give to you, `GISDB` to implement the `GIS` interface, your test classes (including `GISTest`) and one or more classes to implement the BST and kd tree.

Programming Standards:

You must conform to good programming/documentation standards. Web-CAT will provide feedback on its evaluation of your coding style, and be used for style grading. Beyond meeting Web-CAT's checkstyle requirements, here are some additional requirements regarding programming standards.

- You should include a comment explaining the purpose of every variable or named constant you use in your program.
- You should use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc. Use a consistent convention for how identifier names appear, such as "camel casing".
- Always use named constants or enumerated types instead of literal constants in the code.
- Source files should each be under 600 lines. Test files may be longer if needed.
- There should be a single class in each source file. You can make an exception for small inner classes (less than 100 lines including comments) if the total file length is less than 600 lines.

We can't help you with your code unless we can understand it. Therefore, you should no bring your code to the GTAs or the instructors for debugging help unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or code provided by the instructor. Note that the OpenDSA code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It might, however, provide a useful starting point.

Java Standard Data Structures Classes:

You are not permitted to use Java classes that implement complex data structures. This includes `ArrayList`, `HashMap`, `Vector`, or any other classes that implement lists, hash tables, or extensible arrays. (You may of course use the standard array operators.) You may use typical classes for string processing, byte array manipulation, parsing, etc.

If in doubt about which classes are permitted and which are not, you should ask. There will be penalties for using classes that are considered off limits.

Deliverables:

You will implement your project using Eclipse (be sure that you have updated to a 2025 distribution) that has been set to use the Java 11 compiler (see the instructions in Module 2.4 of the OpenDSA course textbook) and you will submit your project using the Eclipse plugin to Web-CAT. Links to Web-CAT client are posted at the class website. If you make multiple submissions, only your last submission will be evaluated unless you arrange otherwise with the GTA. There is no limit to the number of submissions that you may make.

You are required to submit your own test cases with your program, and part of your grade will be determined by how well your test cases test your program, as defined by Web-CAT's evaluation of mutation testing coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by reference test cases that are provided executed by Web-CAT. Web-CAT will report to you which test files have passed correctly, and which have not. Note that you will **not** be given a copy of these test files, only a brief description of what each accomplished in order to guide your own testing process in case you did not pass one of our tests.

When structuring the source files of your project, use a flat directory structure; that is, your source files will all be contained in the project "src" directory. Any subdirectories in the project will be ignored.

You must include in your submission (at the top level of the project) a file named `Transcript.txt` that contains the transcript of any use of an LLM for the purpose of writing or debugging this project. If you did not use an LLM, then `Transcript.txt` should include a statement to that effect. Projects will not be graded without this. It is considered an honor code violation to use an LLM for this project without providing the transcript file.

You must also submit a short video presentation about the project, to help the grader insure your conceptual understanding of the project implementation. Detailed instructions for this video requirement will be provided separately.

You are permitted to work with a partner on this project. While the partner need not be the same as who you worked with on any other projects this semester, you may only work with a single partner during the course of one project unless you get special permission from the course instructor. When you work with a partner, then **only one member of the pair** need make a Web-CAT submission. Be sure both names are included in the javadoc documentation. Whatever is the final submission from either of the pair members is what we will grade unless you arrange otherwise with the GTA. Note that if you work with a partner, each individual is required to submit their own video.

Pledge and Honor Code:

Your project submission must include a statement pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the pledge statement that is provided with the project starter code. The text of the pledge will also be posted online. Programs that do not contain the pledge will not be graded. Use of an LLM without submitting the transcript as required is considered an honor code violation.