CS 5040: Intermediate Data Structures and Algorithms
Virginia Tech
Fall 2020
Dr. Edmison

Programming Assignment 2

Last Updated: 09-18-2020 17:00:27-04:00

Due Friday, October 16 @ 11:00 PM ET for 125 points
Due Wednesday, October 14 @ 11:00 PM ET for a 10 point bonus
Note: This project also has three intermediate milestones.
See the Piazza forum for details.

Assignment

Many applications areas such as computer graphics, geographic information systems, and VLSI design require the ability to store and query a collection of rectangles. In 2D, typical queries include the ability to find all rectangles that cover a query point or query rectangle, and to report all intersections from among the set of rectangles. Adding and removing rectangles from the collection are also fundamental operations.

For this project, you will create a simple spatial database for handling inserting, deleting, and performing queries on a collection of rectangles. The data structure used to store the collection will be a Binary Search Tree (BST, see Section 7.11 of the OpenDSA textbook for more information about BST).

Invocation and I/O Files

The name of your executable must be `RectangleDB`. There will be one input parameter to the program: the name of the command file to read. Your program should read commands from the specified command file, and ***write its output to the standard output***.

The program will be invoked from the command-line as:

```
java RectangleDB <command-file>
```

where:

- `RectangleDB` is the name of the program. The file where you have your main() method must be called `RectangleDB.java`
- `command-file` is the name of the command file to read.

The command file contains one command per line. All rectangles must fit within a box 1024 by 1024 units in size (any that fall outside this box will be rejected) and all coordinates for rectangles are in the range 0 to 1023. Note that the coordinates for query rectangles (i.e., the search command) are permitted to go outside of the "world box" of size 1024. The commands are as follows:

The command file may contain any mix of the following commands. In the following description, terms in *italics* are parameters to the command.

- `insert <name> <x> <y> <w> <h>`

  Insert a rectangle named `<name>` with upper left corner (`<x>`, `<y>`), width `<w>` and height `<h>`. It is permissible for two or more rectangles to have the same name, and it is permissible for two or more rectangles to have the same spatial dimensions and position. But it is not permitted to insert multiple copies of a rectangle with a given name and spacial dimensions/position. The name must begin with a letter, and may contain letters, digits, and underscore characters. Names are case sensitive. A rectangle should be rejected for insertion if its height or width are not greater than 0. All rectangles must fit within the "world box" that is 1024 by 1024 units in size and has upper left corner at (0, 0). If a rectangle is all or partly out of this box, it should be rejected for insertion.

- `remove <name>`

  Remove the rectangle with name <name>. If two or more rectangles have the same name, then any one such rectangle may be removed. If no rectangle exists with this name, it should be so reported.

- `remove <x> <y> <w> <h>`

  Remove the rectangle with the specified dimensions. If two or more rectangles have the same dimensions, then any one such rectangle may be removed. If no rectangle exists with these dimensions, it should be so reported.

- `regionsearch <x> <y> <w> <h>`

  Report all rectangles currently in the database that intersect the query rectangle specified by the `regionsearch` parameters. For each such rectangle, list out its name and coordinates. A `regionsearch` command should be rejected if the height or width is not greater than 0. However, it is (syntactically) acceptable for the `regionsearch` rectangle to be all or partly outside of the 1024 by 1024 world box.

- `intersections`

  Report all pairs of rectangles within the database that intersect.

- `search <name>`

  Return the information about the rectangle(s), if any, that have name <name>. If there are more than one rectangle with the same name, you should return the info for all of them.

- `dump`

  Return a "dump" of the BST. The BST dump should print out each BST node (*use the in-order traversal*). For each BST node, print that node's depth and value (rectangle info). At the end, please print out the size of the BST.

Design Considerations

The rectangles will be maintained in a BST, sorted by the name. Use `compareTo()` to determine the relative ordering of two names, and to determine if two names are identical. You are using the BST to maintain your list of rectangles, but the BST is a general container class. Therefore, it should not be implemented to know anything about rectangles.

Be aware that for this project, the BST is being asked to do two things. First, the BST will handle searches on rectangle name, which acts as the record's key value. The BST can do this efficiently, as it will organize its records using the name as the search key. But you also need to do several things that the BST cannot handle well, including removing by rectangle shape, doing a region search, and computing rectangle intersections. So you will need to add functions to the BST to handle these actions. You should design in anticipation of adding a second data structure in Project 2 to handle these actions. Make sure you handle these actions in a general way that does not require the BST to understand its data type.

The biggest implementation difficulty that you are likely to encounter relates to traversing the BST during the intersections command. The problem is that you need to make a complete traversal of the BST for each rectangle in the BST (comparing it to all of the other 2 rectangles). This leads to the question of how do you remember where you are in the "outer loop" of the operation during the processing of the "inner loop" of the operation. One design choice is to augment the BST with an iterator class. An iterator object tracks a current position within the BST, and has a method that permits the position of the iterator object within the BST to move forward. In this way, one iterator object can be tracking the current rectangle in the "outer loop" of the process, while a second iterator can be used to track the current rectangle for the "inner loop". For the `regionsearch` and `intersections` commands, you need to determine intersections between two rectangles. Rectangles whose edges abut one another, but which do not overlap, are not considered to intersect. For example, (10, 10, 5, 5) and (15, 10, 5, 5) do NOT overlap, while (10, 10, 5, 5) and (14, 10, 5 5) do overlap. Note that rectangles (10, 10, 5, 5) and (11, 11, 1, 1) also overlap.

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 be under 600 lines.
- 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 may, however, provide a useful starting point.

Allowed Java Data Structure 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 use the standard array operators. You may use typical classes for string processing, byte array manipulation, parsing, etc. Also, you may use `java.awt.Rectangle`.

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 should implement your project using Eclipse, and you should submit your project using the Eclipse plugin to Web-CAT. Links to the Web-CAT client are posted at the class website. If you make multiple submissions, ***only your last submission will be evaluated***. 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 code coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by test cases that are provided by the graders. 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 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 a project unless you get special permission from the course instructor. When you work with a partner, then ***only one member of the pair will make a submission***. Make sure that you declare your partner in Web-CAT when submitting. Be sure both names are included in the documentation. Whatever is the final submission from either of the pair members is what we will grade unless you arrange otherwise with the GTA.

Honor Code Pledge

Your project submission must include this statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement ***near the beginning of the file containing the function main()*** in your program. The text of the pledge must include:

```
// On my honor:
// - I have not used source code obtained from another student,
//   or any other unauthorized source, either modified or
//   unmodified.
//
// - All source code and documentation used in my program is
//   either my original work, or was derived by me from the
//   source code published in the textbook for this course.
//
// - I have not discussed coding details about this project
//   with anyone other than my partner (in the case of a joint
//   submission), instructor, ACM/UPE tutors or the TAs assigned
//   to this course. I understand that I may discuss the concepts
//   of this program with other students, and that another student
//   may help me debug my program so long as neither of us writes
//   anything during the discussion or modifies any computer file
//   during the discussion. I have violated neither the spirit nor
//   letter of this restriction.
```

Programs that do not contain this pledge will not be graded.