CSC 225 - SUMMER 2025
ALGORITHMS AND DATA STRUCTURES I
PROGRAMMING ASSIGNMENT 2
UNIVERSITY OF VICTORIA

Due: Tuesday, July 15th, 2025 before 11:55pm. **Late assignments will not be accepted.**

1 Assignment Overview

On this programming assignment, you will implement the FIND and INSERT operations for a left-leaning red-black tree. Since a red-black tree is a binary search tree, the FIND operation of a red-black tree is identical to the FIND operation for a normal binary search tree. To maintain a $\Theta(\log_2 n)$ height, the INSERT operation of a red-black tree must perform various restructuring operations to ensure that the constraints of the red-black tree are satisfied after every operation. A description of the various insertion cases is included in the lecture slides, but you may find it easier to derive the insertion operations directly using your knowledge of 2-3 Trees.

Unlike Programming Assignment 1, this assignment will not be evaluated by an in-person demo. Instead, we will use a combination of automated validation (using the same tools that we have provided for your own testing) and human inspection to mark your submission. As usual, correctness is critical, and a submission which is not substantially correct on our tested inputs will not receive any marks for achieving an efficient running time.

2 Starter Code

Four code files have been provided.

- `RedBlackNode.java` contains the specification of an interface for tree nodes. Your implementation must include a class that implements this interface (but a basic implementation has been provided as part of `RedBlackTree.java`).
- `RedBlackTester.java` contains a tester program that help with testing your implementation (but you are encouraged to write your own testing code as well). See Section 2.1 for more information.
- `RedBlackSVGRenderer.java` contains code to generate a graphical representation of a red-black tree in SVG format and is used by the `RedBlackTester` program. You do not need to read or understand this file at all to complete the assignment.
- `RedBlackTree.java` contains the basic definitions for a red-black tree with FIND and INSERT operations. Your task is to implement both operations. All of the code required for your implementation must be contained in `RedBlackTree.java`. You are not permitted to modify the signatures of the `find`, `getRoot` or `insert` in any way, and you must ensure that the `RedBlackTree` class can be instantiated correctly using the default constructor (although you are permitted to add whatever other methods or constructors you consider necessary).

- `RedBlackValidator.java` contains code to validate the various constraints on red-black trees and is used by the `RedBlackTester` program. You do not need to read or understand this file at all to complete the assignment.

Your submission must be a completed version of the `RedBlackTree.java` file. Only one file will be accepted, so you must ensure that all of your code is contained in `RedBlackTree.java`.

2.1 Tester Programs

The provided `RedBlackTester.java` file contains a program to help test your implementation. The `RedBlackTester` program creates a red-black tree and performs various operations specified on the command line. It also contains functionality to check the red-black tree constraints and generate a graphical representation of the tree to assist with debugging.

The command line options provided to `RedBlackTester` specify a set of operations to perform on the red-black tree. For example, the command

```
java RedBlackTester i6,10,17 f187 i225,226 v f10
```

performs the following operations: INSERT(6), INSERT(10), INSERT(17), FIND(187), INSERT(225), INSERT(226), Verify constraints, FIND(10).

The supported operations are described in the table below.

Operation	Directive	Example	Description
Find	`f`	`f6,10,17,187`	Call the `find` method for a sequence of integer elements, specified as a comma-separated list (with no spaces between items).
Insert	`i`	`i6,10,17,187`	Call the `insert` method for a sequence of integer elements, specified as a comma-separated list (with no spaces between items).
Save to SVG	`s`	`sfilename.svg`	Save a graphical representation of the tree in SVG format using the provided filename.
Verify	`v`	`v`	Verify that the tree satisfies the various binary search tree and red-black tree constraints.

Operations are performed in left to right order. For example, the command

```
java RedBlackTester i6,10,17,187,225,10,226 v stree1.svg i20,25,226,445,485 v stree2.svg
```

performs the following operations.

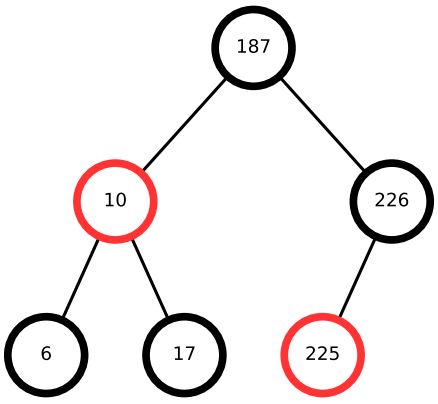
- `i6,10,17` calls the `insert` method on the keys $6, 10, 17, 187, 225, 10$ and 226. Note that the second insertion of the key 10 will result in no change to the tree, since the `insert` method will refuse to insert any duplicate keys.
- `v` verifies the tree constraints.
- `stree1.svg` saves the current state of the tree (after the insertions above) to the file '`tree1.svg`'.
- `i20,25,226,445,485` calls the `insert` method on the keys $20, 25, 226, 445$ and 485.
- `v` verifies the tree constraints.
- `stree2.svg` saves the current state of the tree (after the insertions above) to the file '`tree2.svg`'.

The output of the command above on a model solution is shown below.

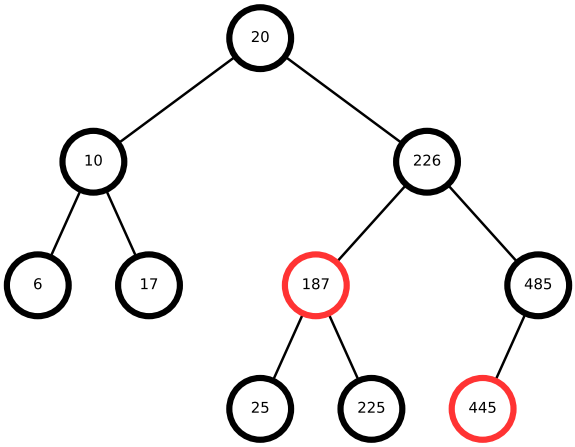
```
Inserting 6
  Insertion successful
```

```
Inserting 10
  Insertion successful
Inserting 17
  Insertion successful
Inserting 187
  Insertion successful
Inserting 225
  Insertion successful
Inserting 10
  Key already present
Inserting 226
  Insertion successful
Validating tree structure
  BST constraints are satisfied
  Red/Black constraints are satisfied
Saving current tree to tree1.svg
Inserting 226
  Key already present
Inserting 445
  Insertion successful
Inserting 485
  Insertion successful
Validating tree structure
  BST constraints are satisfied
  Red/Black constraints are satisfied
Saving current tree to tree2.svg
```

The image saved as `tree1.svg` by the command above is shown below. Note that there may be multiple valid red-black trees with a particular set of keys (and it is not required that the tree produced by your implementation exactly match the tree produced by a model solution, as long as all constraints are satisfied).



The image saved as `tree2.svg` by the command above is shown below.



3 Using Outside Resources

If you consult any outside resources (books, online sources, etc.), include citations to those resources as comments in your code. You are **not permitted** to incorporate code from any outside sources into your submission for this assignment (even if citation is given). As usual, you are not permitted to use generative AI tools for any aspect of this assignment, and providing any part of the assignment materials to generative AI tools is not permitted either.

You are encouraged to discuss the assignment with your peers, and even to share implementation advice or algorithm ideas. However, you are not permitted to use any code from another CSC 225 student under any circumstances, nor are you permitted to share your code in any way with any other student (or the internet). Sharing your code with others before marking is completed, or using another student's code for assistance in any way (even if you do not directly copy it) is plagiarism.

4 Evaluation

Submit your completed `RedBlackTree.java` file electronically via Brightspace. After finishing the submission process, download the file you submitted and verify that it is correct (to ensure that the correct version was submitted). Your code must compile and run correctly in the lab environment (ECS 250). If your code does not compile as submitted, you will receive a mark of zero.

This assignment is worth 7% of your final grade and will be marked out of 14 with a combination of automated validation and human inspection.

The marks are distributed among the components of the assignment as follows. Each item in the table depends on the item above it, so no marks will be given for a particular item in the table unless the item above was substantially correct.

Marks	Component
3	When the `insert` method is called with a key that is not already contained in the tree, the key is added and the resulting tree is a valid binary search tree (even if the red-black tree constraints are not satisifed). The return value of the `insert` method is consistent with the specification given in `RedBlackTree.java`.
3	The `find` method is implemented correctly and complies with the specification given in `RedBlackTree.java`.
5	After each call to `insert`, all of the red-black tree constraints are satisfied.
3	The running times of `find` and `insert` are both $\Theta(h)$ on a tree of height h.

You are permitted to delete and resubmit your assignment as many times as you want before the due date, but no submissions or resubmissions will be accepted after the due date has passed. You will receive a mark of zero if you have not officially submitted your assignment (and received a confirmation email) before the due date.

Your primary submitted file must be based on the provided template `RedBlackTree.java`. Ensure that your submission contains a comment with your name and student number.

The best way to make sure your submission is correct is to download it from Brightspace after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. After submitting your assignment, Brightspace will automatically send you a confirmation email. **If you do not receive such an email, you did not submit the assignment.** If you have problems with the submission process, contact the instructor **before** the due date.