CS 245 Homework 3
IntSet

1 Overview

In this homework, you will create a set data structure designed specifically to hold integers. A naive implementation might store the set as an array of booleans. If i is in the set, then the element at index i is true. If there are many numbers and they are spread out, the backing array will waste space.

A more efficient solution stores the set as a series of intervals. Each contiguous string of numbers will be represented with just two `int`s. For example, if we add numbers 10, 8, 3, 7, 2, 9, 3, 2, 10, 11 and 5, instead of storing 8 distinct numbers, we'll store just three intervals: 2-3, 5-5, and 7-11.

As you complete this homework, you will learn about the following topics: linked structures and iterators.

2 Requirements

Specifications do not tell you how to solve a problem—just what pieces may be used. The classes and methods described below will need to be thought about and pieced together using your own good mind. You will likely need to read this section many times.

Your solution is to meet the following specification:

1. Write all code in your fork of the class Bitbucket project, in package `hw3`.
2. Write a class `Interval` that encapsulates a bounded number range. It has the following specification:
   (a) Has a constructor that accepts two `int` parameters for the lower and upper bound of the interval. The bounds may be equal, but if the lower exceeds the higher, throw an `IllegalArgumentException`.
   (b) Has methods `getLowerBound` and `getUpperBound`, which are getters for the bounds.
   (c) Has a method `size` that returns the number of `int`s in the span of the bounds, inclusive.
   (d) Has a method `enclose` that accepts an `int` parameter. The interval's bounds are adjusted, if necessary, to include the parameter in the interval.
   (e) Has a method `setLowerBound` that accepts an `int` parameter, which is made the new lower bound. If the parameter exceeds the upper bound, throw an `IllegalArgumentException`.
   (f) Has a method `setUpperBound` that accepts an `int` parameter, which is made the new upper bound. If the parameter is less than the lower bound, throw an `IllegalArgumentException`.
   (g) Has a method `contains` that accepts an `int` parameter. If the parameter falls within the interval's bounds, inclusive, return true. Otherwise, return false.
   (h) Has a `String` method that returns a textual representation of this interval. If the interval contains just one number, return that number as a `String`. Otherwise, return the hyphen-separated bounds. For example, `new Interval(101, 101).toString()` → "101" and `new Interval(45, 57).toString()` → "45-57".

3. Write a class `IntSet` that encapsulates a set of integers. It stores the integers using a sequence of `Intervals`. No part of the public specification requires that you use a linked structure, but this will be checked by a human grader. Please use a doubly-linked `Node` class, and store both dummy head and tail nodes. Do not use any builtin `List` class. `IntSet` has the following specification:

   (a) Has a default constructor that initializes the set to contain no integers.

   (b) Has a method `isEmpty` that returns a `boolean` indicating the emptiness of the set.

   (c) Has a method `size` that returns as an `int` the number of integers in the set.

   (d) Has a method `getIntervalCount` that returns as an `int` the number of intervals used to hold the numbers added to the set.

   (e) Has a method `getInterval` that accepts an `int` parameter i and returns the i^{th} `Interval` in the set.

   (f) Has a method `contains` that accepts an `int` parameter. It returns true if the parameter is in the set and false otherwise.

   (g) Has a method `add` that accepts an `int` parameter. It adds the parameter to the set. What is done to add the new number depends on the existing intervals:

      i. If some interval already contains the `int`, nothing is done. For example, adding 3 to set 2-3,6 yields 2-3,6.

      ii. If the `int` is adjacent to just one existing interval, then the interval is adjusted to include the new number. For example, adding 8 to 9-20,25-29 yields 8-20,25-29.

      iii. If the `int` falls between two intervals, the two intervals are coalesced into one. For example, adding 5 to 1-4,6-7 yields 1-7.

      iv. Otherwise, a new interval is formed and inserted between any existing intervals so that intervals are in sorted order. For example, adding 6 to 1,9-15 yields 1,6,9-15.

   (h) Has a method `remove` that accepts an `int` parameter. It removes the parameter from the set. What is done to remove the number depends on the existing intervals:

      i. If no interval contains the `int`, a `NoSuchElementException` is thrown.

      ii. If the `int` is the sole member of its interval, the interval is removed. For example, removing 5 from 1,5,7-9 yields 1,7-9.

      iii. If the `int` is the bounds of its interval, the interval is adjusted to not include the number. For example, removing 3 from 1-3 yields 1-2.

      iv. Otherwise, the number lies within an interval, and the interval must be broken in two. For example, remove 7 from 2,5-10 yields 2,5-6,8-10.

   (i) Has a `toString` method that returns a textual representation of the set. If the set is empty, "{}" is returned. Otherwise, a comma-separated concatenation of the set's intervals is returned. For example, if we add numbers 1, 5, 8, 6, 7 to a set, its `String` representation is "1,5-8".

   (j) Has a method `getIterator` that returns a new `IntSet.Iterator` that may be used to traverse the set.

4. Write an inner class of `IntSet` named `Iterator` with the following specification:

(a) Is not `static`, i.e., it can access instance variables of `IntSet`. This is typical of iterators, which are closely tied to their containing instances.

(b) Has a default constructor, which sets up the iterator to point to the first `int` in the set. (Hint: you will likely need to track two instance variables to support iteration: one for the current interval node you are in and one for the number within the interval you are visiting.)

(c) Has a method `hasNext` that returns `true` if there's an next `int` to visit and false otherwise.

(d) Has a method `next` that returns the next visited `int`.

5. Write a class `Main` with the following specification:

   (a) Has a `main` method. What it does is not specified, but you are suggested to use it to test your code. Relying exclusively on the SpecChecker to test things will rob your brain of some neurons that are in your best interest to grow.

3 Submission

This homework is a regular assignment and is graded by hand and with help from the SpecChecker. To submit your work for grading:

1. Put the SpecChecker for this homework in your Build Path.
2. Run the SpecChecker as a Java Application (not a JUnit Test) and fix problems until all tests pass.
3. Commit and push your work to your repository. If you are resubmitting an earlier assignment, email me. The time of your email will determine the submission week.

The SpecChecker cannot check everything. Your assignment is also expected to fully meet the requirements above and the following:

- Variable names should be meaningful and accurate.
- Non-obvious parts of your code should be commented.
- Code should be cleanly formatted and indented.
- You must not plagiarize. Write your own code. Talk about code with your classmates. Ask questions of your instructor or TA. Do not look at others' code. Do not ask questions specific to your homework anywhere online but Piazza. (If you find violators of this rule, please let me know.)
- Work must be submitted according to course policies on deadlines. To be eligible for later-week submission, you must have at minimum the skeletons for all specified classes and methods in your repository by the homework deadline—without compilation errors (no red in Eclipse).