CS 240 Programming Assignment 2

IntSet

1 Overview

In this assignment, you will create a set data structure that compactly stores integers. A naive implementation of an integer set might store the set as an array of booleans. If integer i is in the set, then the element at index i is true. An implementation based on a full array supports fast lookup and insertion, but it uses a lot of space. Additionally, advancing an iterator may be costly if the set is sparsely populated. There's also the problem of ensuring that the array is large enough to hold all possible integers that might be added to the set.

An alternative solution is to store the set as a series of intervals. Each sequence of contiguous numbers is represented with just two `int`s. For example, if you add numbers 10, 8, 3, 7, 2, 9, 3, 2, 10, 11 and 5, the set will store just three intervals: 2-3, 5-5, and 7-11. The intervals themselves are linked together in sequence. This implementation uses little space, provides fast iteration, and supports wide and sparse ranges of numbers. It's not as fast at lookup and insertion.

2 Requirements

Your solution is to meet the following specification:

1. Write all code in the default package.
2. Write class `Interval` to encapsulate a bounded number range. It has the following public interface:
   (a) A constructor that accepts two `int` parameters for the lower and upper bound of the interval. The bounds are inclusive. If the lower bound exceeds the upper bound, an `IllegalArgumentException` is thrown.
   (b) Methods `getLowerBound` and `getUpperBound`, which are getters for the bounds.
   (c) A method `size` that returns the number of integers in the span of the bounds.
   (d) A method `enclose` that accepts an `int` parameter. The interval's bounds are adjusted, if necessary, to include the parameter in the interval.
   (e) A method `setLowerBound` that accepts an `int` parameter, which is made the new lower bound. If the parameter exceeds the upper bound, it throws an `IllegalArgumentException`.
   (f) A method `setUpperBound` that accepts an `int` parameter, which is made the new upper bound. If the parameter is less than the lower bound, it throws an `IllegalArgumentException`.
   (g) A method `contains` that accepts an `int` parameter. It returns true if and only if the parameter falls within the interval's bounds.
   (h) A `toString` method that returns a textual representation of this interval. The textual representation has one of two forms, depending on its size, as shown in these examples:
   - `new Interval(101, 101).toString()` → `"101"`
   - `new Interval(45, 57).toString()` → `"45-57"`

3. Write class `TestInterval` to test the public interface of the `Interval` class. Your tests must achieve 100% code coverage of the `Interval` class. Additionally, all tests should pass before you move on to the `IntSet` class.

4. Write interface `IntSet` to describe these operations of a set of integers:

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

   (b) A method `size` that returns as an `int` the number of integers in the set.

   (c) A method `getIntervalCount` that returns as an `int` the number of intervals used to represent the set. This method really should not be in the interface of a set. It is included only to make your set easier to test.

   (d) A method `getInterval` that accepts an `int` parameter i and returns the i^{th} interval in the set. If no such interval exists, an `IndexOutOfBoundsException` is thrown. This method really should not be in the interface of a set. It is included only to make your set easier to test.

   (e) A method `contains` that accepts an `int` parameter. It returns true if and only if the parameter is in the set.

   (f) 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 1-4,6-7,99 yields 1-7,99.

      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.

   The method returns true if the integer was added to the set and false if it was already a member.

   (g) 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.

   (h) A `toString` method that returns a textual representation of the set. If the set is empty, `"{}"` is returned. Otherwise, a brace-enclosed and comma-separated sequence of the intervals' textual representations is returned. For example, if you add numbers 1, 5, 8, 6, 7 to a set, its `String` representation is `"{1,5-8}"`.

(i) An `equals` method that accepts an `Object` as a parameter. It returns true if and only if the parameter is an `IntSet` containing the same integers.

(j) Extends `Iterable<Integer>`. Members of the set are visited in ascending order. The iterator may support removal, but it does not need to. The `remove` method will not be tested.

5. Write class `LinkedIntSet`. It maintains a set integers using a linked sequence of `Interval`. It has the following public interface:

   (a) Implements `IntSet`.

   (b) A default constructor that initializes the set to contain no integers.

   No part of the public specification requires that you use a linked structure, but this will be checked by a human grader. Use a doubly-linked `Node` class, and store both sentinel head and tail nodes. Do not use any builtin `List` class.

3 Testing Code

You are responsible for writing your own unit tests for `Interval`. Tests for `IntSet` are provided here:

- `TestIntSet.java`

As you well know, running someone else's tests is demoralizing. Even though these tests are provided, you are encouraged to test each method on your own as you write it. If writing your own JUnit tests seems like too much work, at least create a `main` method that creates a set, manipulates it, and shows the output.

4 Scoring

In order to submit, create a .zip file named `intset.zip` and upload it to Autolab. The .zip file should contain: `Interval.java`, `TestInterval.java`, `IntSet.java`, and `LinkedIntSet.java`.

The project will be graded as follows:

Autolab Functionality Tests	60%
Autolab Test Coverage Checks	10%
Autolab Style Checks	10%
Instructor Style Points	20%

Instructor style points will be based on the quality and efficiency of your code and the quality of your documentation. However, this score may be impacted by problems with functionality. For example, submitting a single file with just one method will not be enough to get full credit for style, even if that method is elegant, well-documented and efficient.