Project 2: Sorted Lists!

Tests Due: Monday 10/10/22 Before 11:59 PM
Project Due: Monday 10/17/22 Before 11:59 PM

Advising (e.g., Piazza, Office Hours) may not be available over the weekend after 4:00PM on the Friday preceding the deadline. Plan accordingly

GitHub Classroom Invitation: https://classroom.github.com/a/h6eiqp3l

Total Points = 30

Objectives

In this assignment, you will:

- Implement a data structure based on linked lists
- Contrast algorithms with (nearly) constant and linear runtimes.
- See one application of access-by-reference (in contrast with access-by-index) in linked lists.

Useful Resources

Review the lecture notes and provided example code for some insight into the Scala syntax. You will also want to read the Scala references provided below:

- The Scala API
    - Scala Collections
    - scala.collection.mutable.Seq
    - scala.collection.Iterator
    - scala.Option
    - scala.math.Ordering
- Scala Tour
- Scala Resources
- ScalaTest: Writing your first test
- Scala Exercises

Late Policy

The policy for late submissions on assignments is as follows. Your project grade is the grade assigned to the latest (most recent) submission you make to autolab (or 0 if no submissions are made).

If your submission is made...

- ... prior to the deadline, your submission is assigned a grade of 100% of the points it earns.
- ... up to 24 hours after the deadline, your submission is assigned a grade of 75% of the points it earns.
- ... more than 24 hours after the deadline, but within 48 hours of the deadline, your submission is assigned a grade of 50% of the points it earns.
- ... more than 48 hours after the deadline, it will not be accepted.

Starting with this assignment there are no bonus points will be awarded for early submissions. (Early submissions are still encouraged, as Autolab gets congested close to deadlines)

You will have the ability to use three grace days throughout the semester, and at most two per assignment (since submissions are not accepted after two days). Using a grace day will negate the 25% penalty per day, but will not allow you to submit more than two days late. Please plan accordingly. You will not be able to recover a grace day if you decide to work late and your score was not sufficiently higher. **Grace days are automatically applied** to the first instances of late submissions, **and are non-refundable**. For example, if an assignment is due on a Friday and you make a submission on Saturday, you will automatically use a grace day, regardless of whether you perform better or not. **Be sure to test your code before submitting**, especially with late submissions **in order to avoid wasting grace days**.

Keep track of the time if you are working up until the deadline. Submissions become late after the set deadline. Keep in mind that **submissions will close 48 hours after hte original deadline** and you will not be able to submit your code after that time.

AI Policy Overview

As a gentle reminder, please re-read the academic integrity policy of the course. I will continue to remind you throughout the semester and hope to avoid any incidents.

What constitutes a violation of academic integrity?

These bullets should be obvious things not to do (but commonly occur): * Turning in your friend's code/write-up (obvious). * Turning in solutions you found on Google with all the variable names changed (should be obvious). This is a copyright violation, in addition to an AI violation. * Turning in solutions you found on Google with all the variable names changed and 2 lines added (should be obvious). This is also a copyright violation. * Use of Github Autopilot (should be obvious). This is still in murky legal water, and may be a copyright violation, in addition to being an AI violation. * Paying someone to do your work. You may as well not submit the work, since you will fail the exams and the course. * Posting to forums asking someone to solve assignment problems *(even if you do not receive the solution)* * Accessing solutions to assignment problems.

Note: Aggregating every { stack overflow answer, result from google, other source } because you "understand it" will likely result in full credit on assignments (if you are not caught), and then failure on every exam. Exams don't test if you know how to use Google, but rather test your understanding (i.e., do you understand the problem and material well enough to arrive at a solution on your own). Also, other students are likely doing the same thing, and then you will be wondering why 10 people that you don't know have your exact solution.

What collaboration is allowed?

There is a grey area when it comes to discussing the problems with your peers, and I do encourage you to work with one another to discuss course *concepts* related to an assignment. That is the best way to learn and to overcome obstacles. At the same time, you need to be sure you do not overstep and not plagiarize. Discussions pointing to relevant course materials are OK. For example, the following is acceptable advice:

It would be helpful to review the usage of the stack in the recitation slides from week XX.

When working with your peers, we ask that you include attribution; In the header comment of the Main function of your submission, please list all peers who you have discussed the project with.

Explaining every step in detail and/or giving pseudocode that solves the problem is **not ok**. For example, the following is **not acceptable** advice:

I copied the algorithm from the week XX notes into my code at the start of the function, created a function that went through the given data and put it into a list, called that function, and then sorted the results.

The first example is OK. The second example, however, is a summary of your code and is not acceptable. Remember that you should never show any of your code to other students prior to any deadlines. Regardless of where you are working, you must always follow this rule: **Never come away from discussions with your peers with any written work, either typed or photographed, and especially do not share or allow viewing of your written code**.

If you have concerns that you may have overstepped or worked too closely with someone, please address this with me prior to deadlines for the assignment. **Even if you have submitted an assignment that may have violated the course academic integrity policy, if you approach me *prior to detection* you will not face academic integrity proceedings**. We will address options at that point.

What resources are allowed?

With all of this said, please feel free to use any { files | examples | tutorials } that course staff provides, directly in your code. Feel free to directly use any materials from lecture or recitations. You will never be penalized for doing so, but must always provide attribution/citation for where you retrieved code from. Just remember, if you are citing an algorithm that is not provided by us, then you are probably overstepping.

More explicitly, you may use any of the following resources (with proper citation/annotation in your code: * Any example files posted on the course webpage or Piazza (from lecture or recitation) * Any code that the instructor provides * Any code that the TAs provide * Any code from the Tour of Scala * Any code from Scala Collections * Any code from Scala API * Additional references may be provided as the semester progresses, but only those provided publicly by course staff are allowed for use. These will be listed on Piazza under Resources

Omitting citation/attribution will result in an AI violation (and lawsuits later in life at your job). This is true, **even if you are using provided resources**.

Again, **if you think you are going to violate/have violated this policy, please come talk to a member of the course staff ASAP so we can figure out how to get you on track to succeed in the course**. If you have a question about the validity of a resource, please ask a TA or your instructor prior to using it. If you have already used it, please discuss with the instructor to determine a workaround and, at the very least, avoid an academic integrity infraction. For example, you might send an email such as the following to the course instructor:

Clarus T Example
UBIT: ctexamp
Person #: 123456789

Dear Dr. Kennedy/Mikida,

I believe that I may have submitted work that is *{ not fully my own* | *not properly attributed }*. I wish to retract my submission to preserve academic integrity in the course.

Signed,
 Clarus T Example

This policy on assignments is here so that you learn the material and how to think for yourself. There is no cognitive benefit achieved by submitting solutions someone else has written (which likely already exist in some form).

Collaboration Policy

The policy for collaboration on assignments is as follows:

- All work for this course must be original individual work.
- You must follow the limits on collaboration as defined in the AI policy (i.e., no shared code/etc...)
- You must identify any collaborators (first and last name) on every assignment. This can be in a comment at the top of your code submissions or on the first page at the top of your written work, beside your name.

All references must be cited using a comment containing a direct link to the resource, as well as a brief description of what was used. For example, if you reference the textbook, a page number and description is sufficient. If you copy example code from the Scala Language API, then include the link to the class page within the API as well as where the example code resides.

Instructions

Answer the following questions by:

1. Accept The PA2 Assignment in GitHub Classroom.
2. Use the template repository and develop several test cases as detailed below.
3. Commit **and push** your answers to GitHub
4. Submit PA2-Tests in Autolab (note: Submissions open by September 30). Repeat 2-4 as needed.
5. Implement the `???` placeholders in `SortedLinkedList.scala`
6. Commit **and push** your answers to GitHub

7. Submit PA2-Implementation in Autolab (note: Submissions open by September 30). Repeat 5-7 as needed.

8. After the deadline, an additional round of tests will be conducted on your implementation for the final 5 points.

Make sure your submission is committed and pushed into your GitHub Classroom Git repository.
Seriously, make sure it's committed. Yeah you... the person who clicked submit without checking.

Expect this project to take 10-12 hours of setting up your environment, reading through documentation, and planning, coding, and testing your solution.

Once you have the project passing the provided tests, you are encouraged to invoke `sbt run` in the project directory from a unicode-capable terminal (e.g., the 220 VM's default terminal or most other Linux terminals, or the MacOS Terminal.app). For best results, use a font with color emoji glyphs (e.g., `noto-color`).

Overview

In this assignment you will implement `SortedLinkedList`, a doubly linked list-style data structure that (i) enforces a sorted order over its elements, and (ii) provides "hinted" variations of several methods.

Doubly Linked Lists

A doubly linked list is a collection data structure that stores each of its elements wrapped in a linked list node (`SortedLinkedListNode` for this project).

A doubly linked list contains a `headNode` reference to the first node in the list and a `lastNode` reference to the last node in the list. Each node contains a pointer to the following and preceding nodes (if they exist).

To insert an element into a doubly-linked list, all of the relevant references need to be updated. Make sure you handle all of the corner cases!

- `newNode.next` and `newNode.prev` need to be set appropriately
- `newNode.prev.next` needs to be updated if it exists
- `newNode.next.prev` needs to be updated if it exists
- `headNode` needs to be updated if `newNode` is inserted at the front.
- `lastNode` needs to be updated if `newNode` is inserted at the end.

Ordering

SortedLinkedList uses a scala provided class called `Ordering` to define a sort order. This class defines a method: `compare(a, b)` that behaves as follows:

- `compare(a, b) < 0` if $a < b$

- `compare(a, b) == 0` if $a = b$
- `compare(a, b) > 0` if $a > b$

There is a little scala magic involved how you call `Ordering.compare`, so `SortedLinkedList` defines a helper function `compare` to call it for you.

In general, the following condition should hold for all nodes of the linked list, excluding the final one: `compare(node.value, node.next.get.value) <= 0` Operations that modify the contents of the linked list (`insert` and `update`) should ensure that the inserted/modified node is moved to the correct location.

Part 1. Testing (5 points)

For this part, modify the file `SortedListTests.scala` to include new test cases.

Your test cases will first be run against a correct implementation of `SortedLinkedList`. To get any points on this part, your tests will need to pass on this implementation.

Your test cases will then be run against several broken implementations of `SortedLinkedList`. You will get points for each broken implementation that you fail.

Part 2. Implementation (20+5 points)

For this project, you will need to implement the following methods:

```scala
def findRefBefore(elem: T, hint: SortedListNode[T]): Option[SortedListNode[T]]
```

If the list contains `elem`, this function should return a reference to the **first** occurrence of it in the list.

If the list does not contain `elem`, this function should return a reference to the element that would precede it if it were inserted into the list, or `None` if `elem` is lower than the lowest element in the list (i.e., if it would be inserted at the head).

The search for the element should be started at the node referenced by `hint`, moving forward or backward as needed. Thus, if `hint` is at position `i` and `elem` is at position `j`, then this function should run in $O(|\mathtt{i} - \mathtt{j}|)$

```scala
def findRefBefore(elem: T): Option[SortedListNode[T]]
```

If the list contains `elem`, this function should return the **first** occurrence of it in the list.

If the list does not contain `elem`, this function should return a reference to the element that would precede it if it were inserted into the list, or `None` if `elem` is lower than the lowest element in the list (i.e., if it would be inserted at the head).

This function should run in $O(\mathtt{length})$

```
def insert(elem: T, hint: SortedListNode[T]): SortedListNode[T]
```

The value should be placed so that the list remains in sorted order. After the insertion, the inserted element's `next` method should return a reference to the next greatest element, and the `prev` method should return a reference to the next least element.

If elem is already in the list, the ordering of the new node is up to the implementation, it may be created before or after the existing value(s).

The function should return a reference to the newly inserted node.

As above, the search for an insertion point should start at the node referenced by `hint`. If `hint` is at position `i` and `elem` should be inserted at position `j`, then this function should run in $O(|\mathtt{i} - \mathtt{j}|)$

```
def insert(elem: T): SortedListNode[T]
```

The value should be placed so that the list remains in sorted order. After the insertion, the inserted element's `next` method should return a reference to the next greatest element, and the `prev` method should return a reference to the next least element.

If elem is already in the list, the ordering of the new node is up to the implementation, it may be created before or after the existing value(s).

The function should return a reference to the newly inserted node.

This function should run in $O(\mathtt{length})$

```
def findRef(elem: T): Option[SortedListNode[T]]
```

If the list contains `elem`, this function should return the **first** occurrence of it in the list.

If the list does not contain `elem`, this function should return None

This function should run in $O(\mathtt{length})$

```
def getRef(idx: Int): SortedListNode[T]
```

Return a reference to the node at the provided index. Throw an `IndexOutOfBoundsException` if the provided index is invalid (i.e., `idx < 0` or `idx >= length`

This function should run in $O(\mathtt{length})$

```
def apply(idx: Int): T
```

Return the value contained in the node at the provided index. Throw an

`IndexOutOfBoundsException` if the provided index is invalid (i.e., `idx < 0` or `idx >= length`

This function should run in $O(\texttt{length})$

```
def remove(ref: SortedListNode[T]): T
```

Remove the node referenced by `ref` from the linked list.

This function should run in $O(1)$

The following functions have already been defined for you, but feel free to modify them if you want:

```
def update(ref: SortedListNode[T], elem: T): SortedListNode[T]
def update(idx: Int, elem: T): Unit
def iterator: Iterator[T]
override def last = lastNode.get.value
```

Suggested Approach

Set up your GitHub Classroom repository as detailed above and set up your programming environment as in PA1.

Review the **Implementation Objectives** section and identify ways in which the code might fail. Several test cases are provided. These tests are meant to cover several basic cases, but are not comprehensive. Define test cases that test for these and submit them for Part 1.

Start with `findRefBefore` and `insert`. You may find that `findRefBefore` is helpful in your implementation of `insert`. You may find that several of the methods that you are implementing for Part 2 will be helpful in implementing others.

Seq defines a `mkString` method that will generate a string representation of the nodes of the linked list (that can be printed). This can be helpful for debugging problems with your list.

Once you are passing all of the test cases, you are encouraged to try running the provided `Main` function with `sbt run`. For best results, use a TTY- and Unicode-capable terminal application.

Allowed library/container usage

- You may not use any external containers in the implementation of this assignment, other than those already defined in the implementation.

Submission

For **Project 2** you may submit as many times as you want. Your final score will be the last (most recent) submission

Revision History

- Fall 2022 - Oliver Kennedy (okennedy@buffalo.edu), Eric Mikida (epmikida@buffalo.edu)