CSCI 1933 Lab 7

Linked Lists

Lab Rules

You may work individually or with *one* partner in lab. We suggest that you have a TA evaluate your progress on each milestone before you move onto the next. The labs are designed to be completable by the end of lab. If you are unable to complete all of the milestones by the end of lab, you have **until the last office hours on Monday** to get those checked off by **any** of the course TAs. We suggest you get your milestones checked off as soon as you complete them since Monday office hours tend to become crowded. If you worked with a partner, both of you must be present when getting checked off to receive credit. You will only receive credit for the milestones you have checked off by a TA. There is nothing to submit to Canvas for this lab.

Attendance

Per the syllabus, students with 4 unexcused lab absences over the course of the semester will automatically fail the course. Meaning if you have missed 3 labs without excuses you are OK, but if you miss a fourth lab (without a formal excuse) you would fail the course. The TAs will take attendance during lab; it is expected that students will be actively working on the lab material. *Your physical presence in lab is not sufficient to be marked as present for the purposes of attendance.*

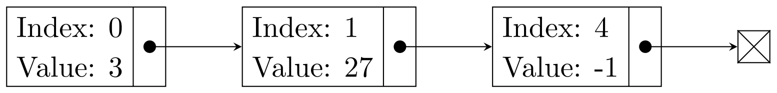
Note: Since this lab is being run remotely, you will need to submit milestones to your lab TAs electronically. Since communication will be more difficult than in previous in-person labs, when asking for assistance please explain what problem you are facing, what you have tried, and what you were expecting the output to be. Be sure to include code and show any relevant input/output.

Sparse vectors

A vector is a one-dimensional array of elements. When doing linear algebra operations (e.g. addition, cross product, dot product) in code, it's common to use an array. Sometimes, a large amount of the coefficients are zero, called *sparse*. It is a big waste of computational resources to not only store these zeros, but use them in linear algebra operations.

In this lab, you'll be creating a sparse vector with linked lists. You should recall that a linked list is a series of `Nodes` that are connected to one-another with a pointer. Iterating over a linked list amounts to traversing the nodes, one by one, until you've found what you're looking for.

For example, the figure below shows a diagram of a sparse vector that has length five, but only three non-zero elements.



It's important to note the length of a vector is separate from the number of nodes it has. In array form, the above vector would be: $\boxed{3}\boxed{27}\boxed{0}\boxed{0}\boxed{-1}$

1 Sparse vectors as linked lists

Your first step is to download the `Node` and `SparseVector` classes from Canvas. Some of the methods are already implemented, such as `toString` and trivial getter/setters.

In milestone one, you will implement `SparseVector::addElement`. **For now, we'll assume that addElement is valid for increasingly larger indices.** That is, every time you add a node to the `SparseVector`, it will be added to the end of the linked list (so long as it does not exceed the vector's length). You should check the index is valid before adding a node, and alert the user if they've tried to add an element incorrectly.

Implement the `SparseVector::addElement` method so that the following code

```
SparseVector vec = new SparseVector(6);
vec.addElement(0, 10.0);
vec.addElement(3, -1.1);
vec.addElement(5, 32.0);
System.out.println(vec);
```

outputs the result:

```
10.0, 0, 0, -1.1, 0, 32.0
```

Milestone 1:
Implement `SparseVector::addElement` and send your TA the result of running the code above. What happens if you add one more element at index 1? What about index 10?

Hint: There are three important cases to account for: (1) invalid index, (2) `head` is null, (3), `head` is not null. Be aware that this:

```
Node a = b.getNext();
a = new Node(index,value);
```

does not set `b`'s `next` pointer. Why not?

2 Debugging

By now, you should have a working `SparseVector` class that contains the `addElement` method. To get to this point, you likely had to do some debugging in order for your code to work correctly. For most of you, this debugging likely consisted of calls to `System.out.println()` to print out the value of a variable at different points in your program. However, IntelliJ has many built in debugging features that allow for easier debugging.

For milestone two, you will need to use some of these debugging tools to go through your program. Take some time to practice using the built-in debugger and familiarize yourself with Step Into, Step Over, and Step Out. The PDF on IntelliJ debugging will likely be a helpful resource.

Hint: Learning how to use the IntelliJ debugger will allow you to more quickly debug projects and labs going forward, so it is in your best interest to begin familiarizing yourself with it.

Milestone 2:
Go through your `main` method in `SparseVector` using the IntelliJ debugger and send screenshots to your TA to show your understanding of IntelliJ's debugging functionality. Also write out the following and send to your TA:

- Explain the difference between Step Over and Step Into.
- Give a situation where using Step Over instead of Step Into would be desirable.
- Explain what does Step Out does.

3 Dot product

Now that we can create sparse vectors and debug effectively, let's do some linear algebra. The dot product of two vectors is the scalar sum of coefficient-wise multiplications. Both vectors must be the same length for the dot product to be valid. For example, if $\mathbf{x} = [a, b, c]$ and $\mathbf{y} = [d, e, f]$, then $\mathbf{x} \cdot \mathbf{y} = a * d + b * e + c * f$. If your vector was stored as an array, it would look like

```
for (int i=0; i<length; ++i){
        result += a[i] * b[i];
}
```

Implement the following static method in your your `SparseVector` class:

```
public static double dot( SparseVector x, SparseVector y )
```

Do **not** buffer the results in a temporary array, as tempting as it sounds. The method should iterate over **x** and **y**'s elements, and add them to the result if their indices are equal. Using an array to help with debugging might be useful, however!

Hint: Before traversing the linked list, check that the lengths of the two vectors are equal, and neither are empty. Iterate over both sparse vectors at the same time, and stop iterating when you've reached the end. Each iteration, move the lower-index node forward (or both, if they are equal).
With every piece of code you write, start with simple, easy cases. First, try $\mathbf{x} = [1]$ and $\mathbf{y} = [2]$. Then, try $\mathbf{x} = [0, 1]$ and $\mathbf{y} = [1, 0]$. Once those work, progressively add more elements to the vectors.

When you have a working solution, the following code

```
SparseVector x = new SparseVector(100000000);
x.addElement(0, 1.0);
x.addElement(10000000, 3.0);
x.addElement(10000001, -2.0);
SparseVector y = new SparseVector(100000000);
y.addElement(0, 2.0);
y.addElement(10000001, -4.0);
double result = dot(x, y);
System.out.println(result);
```

should output the result `10.0`.

Milestone 3:
Implement `SparseVector::dot` and send your TA a picture of the result from running the code above. Also show the following:

- What happens when you take the dot product of two different-length vectors?
- What happens if one vector is empty?

4 Honors Section

Note: This section and milestone are only required for students in the honors section. *Students in other sections do not need to do this, but are still encouraged to work on it if interested and time permits.*

Implement the following method:

```
public void removeElement(int index)
```

As its name suggests, this method should remove a node at the specified index from the sparse vector. Be sure to properly update the linked list and handle special cases (e.g., empty list, bad index, etc...). Test your method with the following code

```java
SparseVector x = new SparseVector(100000000);
x.addElement(0, 1.0);
x.addElement(10000000, 3.0);
x.addElement(10000001, -2.0);
SparseVector y = new SparseVector(100000000);
y.addElement(0, 2.0);
y.addElement(10000001, -4.0);
y.removeElement(0);
double result = dot(x, y);
System.out.println(result);
```

What should the result be?

Milestone 4:
Implement `SparseVector::removeElement` and send your TA the result of running the code above. Try removing other elements, both ones that exist in the sparse vector and ones that do not, and show what happens.