# Sparse Vector Assignment

## Overview

A vector is a one-dimensional array of n elements: $(a_1, a_2, \dots, a_n)$. The natural implementation of a vector would use a one-dimensional array to store the vector. However, in many applications, most of the elements of a vector are zero and the remaining elements are non-zero - a sparse vector.

An example vector: (76.02, 0, 0, -36.4, 7.537, 0, 0, 0, 0, -2.19)
stored in an array:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 76.02 | 0 | 0 | -36.4 | 7.537 | 0 | 0 | 0 | 0 | -2.19 |

It is inefficient to use a one-dimensional array to store a sparse vector. Consequently, a linked list stores the elements of a sparse vector with the values of the list being the non-zero elements of the vector. An index and value pair represents each non-zero element of a sparse vector.

The above example vector written as a sparse vector:
([1, 76.02], [4, -36.4], [5, 7.537], [10, -2.19])

Vector addition produces the sum of two vectors, the sum vector. If an element with a particular index exists in both vectors then add the values of the elements together to form the element with that index in the sum vector. If the sum of the two values is zero then that element does not appear in the sum vector. If an element with a particular index exists in only one vector then copy that element to the sum vector.

An example:
A is ([3, 1.0], [2500, 6.3], [5000, 10.0], [60000, 5.7])
B is ([1, 7.5], [3, 5.7], [2500, -6.3])
A + B is ([1, 7.5], [3, 6.7], [5000, 10.0], [60000, 5.7])

Vector subtraction produces the difference of two vectors, the difference vector. If an element with a particular index exists in both vectors then subtract the value of the element in the second vector from the value of the element in the first vector to form the element with that index in the difference vector. If the difference of the two values is zero then that element does not appear in the difference vector. If an element with a particular index exists in the first vector but not the second vector then copy that element to the difference vector. If an element with a particular index exists in the second vector but not the first vector then copy that element to the difference vector and negate its value.

An example:
A is ([3, 1.0], [2500, 6.3], [5000, 10.0], [60000, 5.7])
B is ([1, 7.5], [3, 5.7], [2500, -6.3])
A - B is ([1, -7.5], [3, -4.7], [2500, 12.6], [5000, 10.0], [60000, 5.7])

Vector dot product produces the dot product of two vectors, a scalar value. If an element with a particular index exists in only one vector then that element does not contribute to the dot product. Initialize the dot product answer to zero. If an element with a particular index exists in both vectors then multiply the values of the elements together and add the result to the dot product answer.

An example:
A is ([3, 1.0], [2500, 6.3], [5000, 10.0], [60000, 5.7])
B is ([1, 7.5], [3, 5.7], [2500, -6.3])
A · B is (1.0 * 5.7) + (6.3 * -6.3) = 5.7 + -39.69 = 33.99

# Design

1. Since the elements of a sparse vector are a pair of numbers, it is logical to design a structure to store an element of a vector. Write a class called **Element** which stores the index and value of an element of a vector. The data type of the index is int and the data type of the value is double.

   The following are the **only** methods of the Element class:
   a. The constructor for the class.
   b. The get and set methods for the index.
   c. The get and set methods for the value.
   d. The Element class implements the Comparable interface:
   `public class Element implements Comparable<Element>`
   which would allow the Element class to implement the **compareTo** method to compare one element with another element in order to easily sort the elements of a sparse vector in the proper order, the element with the smallest index up to the element with the largest exponent.
   e. The **toString** method to properly print an element. For example, if the index of an element is 4 and the value is -36.4 then the toString method would print [4, -36.4].

2. Write a class called **SparseVector** which stores a sparse vector. A SparseVector is **NOT** a linked list. The SparseVector class has a data member which is a linked list storing the elements of a single sparse vector. A SparseVector does **NOT** store more than one sparse vector. Therefore, each instance of a SparseVector stores a single unique sparse vector. Store the elements of the sparse vector in increasing index order.

   The following are the methods of the SparseVector class:
   a. The constructor for the class.
   b. The **add** method: `public SparseVector add(SparseVector sv)`, which performs the addition of two sparse vectors. For example, if A and B are sparse vectors then A.add(B) returns a SparseVector which is A+B.
   c. The **subtract** method: `public SparseVector subtract(SparseVector sv)`, which performs the subtraction of two sparse vectors. For example, if A and B are sparse vectors then A.subtract(B) returns a SparseVector which is A-B.
   d. The **dot** method: `public double dot(SparseVector sv)`, which performs the dot product of two sparse vectors. For example, if A and B are sparse vectors then A.dot(B) returns a double which is A·B.
   e. The **toString** method to properly print a sparse vector. The elements of the sparse vector must appear in increasing index order. A sparse vector with no elements prints the string **empty vector**. For example, if the sparse vector has the elements [1, 76.02], [4, -36.4], [5, 7.537], and [10, -2.19] then the toString method would print
   ([1, 76.02], [4, -36.4], [5, 7.537], [10, -2.19]).
   f. You may add any additional methods as you see fit.

3. Download the files List.java and DoublyLinkedList.java containing most of the implementation of the doubly linked list class. You cannot add data members to or modify the data members of the doubly linked list class. You cannot modify the nested private class **Node** of the doubly linked list class. You cannot modify the nested private class **LinkedListIterator** of the doubly linked list class. You cannot modify the doubly linked list class constructor and the **clear**, **size**, **isEmpty**, **add(AnyType newValue)**, **remove(int index)**, **iterator**, and **getNode(int index)** methods. You cannot add additional methods to the doubly linked list class.

You will write the code for the **getNode(int index, int lower, int upper)** method. This method returns the pointer to the node whose position in the list corresponds to the value of the parameter index. This method should ensure index is a value which is greater than or equal to the parameter lower and less than or equal to the parameter upper.

You will write the code for the **add(int index, AnyType newValue)**, **remove(Node<AnyType> currNode)**, **get**, and **set** methods. Each of these methods will use one of the getNode methods. Recall for the add method, the index can be a value from 0 to size().

4. The input to your program will read from a plain text file called **project1.txt**. This is the statement you'll use to open the file:

   ```
   FileInputStream fstream = new FileInputStream("project1.txt");
   ```

   Assuming you're using Eclipse to create your project, you'll store the input file project1.txt in the parent directory of your source code (**.java** files) which happens to be the main directory of your project in Eclipse. If you're using some other development environment, you'll have to figure out where to store the input file.

   The input consists of one or more sparse vector operations (add, subtract, dot) with each operation taking up 3 lines. A sparse vector operation consists of two sparse vectors and the operation to perform on the sparse vectors. The first sparse vector is on the first line, the second sparse vector is on the second line, and the operation is on the third line. A pair of numbers represents each element of the sparse vector, the element's index followed by the element's value. One or more spaces may separate each of the numbers of the sparse vector. The elements of the sparse vector should appear in increasing index order. The operation is one of the three words: add, subtract, dot. Your program should handle any erroneous input as best as possible.

   An example sparse vector operation:

   ```
   3 1.0 2500 6.3 5000 10.0 60000 5.7
   1 7.5 3 5.7 2500 -6.3
   add
   ```

5. The output of your program is each sparse vector operation along with the result of the operation on the two sparse vectors. For each sparse vector operation, print the first sparse vector on the first line, the operation on the second line, the second sparse vector on the third line, an equal sign on the fourth line, and the result of the operation on the two sparse vectors on the fifth line. For the operation in the output, use + for add, - for subtract, and · for dot. You can use this statement (`char dotCharacter = (char) 183;`) to be able to print the dot character. Have a blank line between each sparse vector operation processed by your program.

   The output for the example sparse vector operation in number 4 above:

   ```
   ([3, 1.0], [2500, 6.3], [5000, 10.0], [60000, 5.7])
   +
   ([1, 7.5], [3, 5.7], [2500, -6.3])
   =
   ([1, 7.5], [3, 6.7], [5000, 10.0], [60000, 5.7])
   ```

6. Make the name of the driver class **Project1** and it should only contain only one method: `public static void main(String args[]).` The main method will open the file **project1.txt** and have a loop to process each sparse vector operation in the file. The main method itself should be fairly short and will pass the operation to another class to perform the operation.

7. Do **NOT** use your own packages in your program. If you see the keyword **package** on the top line of any of your .java files then you created a package. Create every .java file in the **src** folder of your Eclipse project

8. Do **NOT** use any graphical user interface code in your program!

9. Make your program as modular as possible, not placing all your code in one .java file. You can create as many classes as you need in addition to the classes described above. Methods should be reasonably small following the guidance that "A function should do one thing, and do it well."

10. Document and comment your code thoroughly.

## Grading Criteria

The total project is worth 20 points, broken down as follows:

4 points: Program compiles without any errors.

4 points: Following good coding standards, good comments, concise main program, no significant code duplication (i.e., good method design), and following directions as specified in this assignment. Putting all or most of the code in the main method will cause you to lose points. You must write methods and classes.

4 points: Checking error conditions such as basic input exception handling, unknown commands, and anything else you can anticipate.

4 points: Ability to read the sparse vector operations from the file and process them properly.

4 points: Correct result for each sparse vector operation.