

# ITI 1121. Introduction to Computer Science II

## Winter 2014

### Assignment 4

Deadline: Sunday April 6, 2014, 18:00

[ [PDF](#) ]

## Solution

- [a4-solution.jar](#)

## Learning objectives

- Further understanding of linked structures
- Learn to implement and use iterators
- Introduction to recursive list processing
- A basic introduction to tree structures

## Introduction

The comparison of genomic sequences has many applications, such as understanding gene function, phylogenetic studies, and the development of DNA diagnostics for pathogen detection, just to name a few. The size of DNA sequences (strings) vary greatly, as illustrated by the table below.

Species	Size
Potato spindle tuber viroid (PSTVd)	360
Human immunodeficiency virus (HIV)	9,700
Bacteriophage lambda ( $\lambda$ )	48,500
<i>Mycoplasma genitalium</i> (bacterium)	580,000
<i>Escherichia coli</i> (bacterium)	4,600,000
<i>Drosophila melanogaster</i> (fruit fly)	120,000,000
<i>Homo sapiens</i> (human)	3,000 000,000
<i>Lilium longiflorum</i> (easter lily)	90,000,000,000
<i>Amoeba dubia</i> (amoeba)	670,000,000,000

One of the widely used measure of distance for comparing genomic sequences (Levinstein distance) requires computing time increasing as the square of the size of the sequences being compared. This limits its application to sequence fragments (sub-strings)

A publication by Yang suggests that a simple measure called the  $k$ -tuple distance performs well for evolutionary tree reconstruction [Yang et al. 2008]. Here, we implement a tool for measuring the  $k$ -tuple distance, and apply this measure to a variety of sequences. As computer scientists, we want to compare the efficiency of two implementations: linked list-based versus binary search tree-based.

## Part I

# Iterator

### 1 equals( LinkedList<T> xs, LinkedList<T> ys ) (10 marks)

In a class named **A4Q1**, implement the static method **equals**.

- Compares two linked lists for equality;
- Returns **true** if and only if both lists are of the same size, and all the corresponding pairs of elements in the two lists are equal. Otherwise, the method returns **false**;
- You must use iterators to implement the method.

The following **JUnit** test cases can be used to validate the method **equals**: **TestA4Q1.java**. Do not assume that the tests are exhaustive. In particular, do not assume that a method that passes these tests gets a perfect score.

### 2 hasPrevious() (2 marks)

In the class **LinkedList**, implement the instance method **hasPrevious()** of the iterator. It returns **true** if the iteration has a previous element, and **false** otherwise. In other words, it returns **true** if a call to the method **previous** would succeed, and **false** otherwise.

**Note:** the class **LinkedList** uses doubly linked nodes to store the elements of the list. The list starts off with a dummy node and the list is circular in both directions. This implementation technique, seen in class, facilitates the implementation of the methods since it eliminates many special cases.

### 3 previous() (5 marks)

In the class **LinkedList**, implement the instance method **previous()** of the iterator. It returns the previous element of the iteration. This method can be called repeatedly to iterate backwards, or interleaved with calls to **next()** to go back and forth. A call to **next()** followed by a call to **previous()** will return the same element. As the other methods of the iterator, the method **previous()** must be implemented using the technique called “fail-fast” presented in class.

The following **JUnit** test cases can be used to validate the methods **hasPrevious** and **previous**: **TestIteratorPrevious.java**. Do not assume that the tests are exhaustive. In particular, do not assume that a method that passes these tests gets a perfect score.

### 4 remove() (10 marks)

In the class **LinkedList**, implement the instance method **remove** of the iterator. It removes from the list the last element that was returned by **next()**. As the other methods of the iterator, the method **remove()** must be implemented using the technique called “fail-fast” presented in class. As well, the method throws an **IllegalStateException** upon a failure to remove an element (for instance, calling the method **remove()** when the iterator is positioned before the start of the list).

The following **JUnit** test cases can be used to validate the method **remove**: **TestIteratorRemove.java**. Do not assume that the tests are exhaustive. In particular, do not assume that a method that passes these tests gets a perfect score.

## Part II

# Recursion

### 5 take (bonus question: 10 marks)

**Note:** We decided to make this question optional. It now counts as bonus points.

In the class **SinglyLinkedList**, write a **recursive** (instance) method that returns a **new** linked list consisting of the first **n** elements of this list. This instance must remain unchanged. The method **public LinkedList<E> take( int n )** must be implemented following the technique presented in class for implementing recursive methods inside the class, i.e. where a recursive method is made of a public part and a private recursive part. The public method initiates the first call to the recursive method.

The following **JUnit** test cases can be used to validate the method **take**: **TestTake.java**. Do not assume that the tests are exhaustive. In particular, do not assume that a method that passes these tests gets a perfect score.

## Part III

# Binary search trees

The implementation of the class **BinarySearchTree** for this assignment differs slightly than that presented in class. Herein, **BinarySearchTree** implements the interface **Associative**.

An associative structure defines associations between keys and values. It provides operations for defining an association (**update**) and retrieving the value associated with a key (**get**).

The interface **Associative** has two type parameters. The first type parameter (**K**) specifies the type of the key objects. Keys must be **Comparable** one with another. The second type parameter (**V**) specifies the type of the value objects.

- **V get( K key )**: returns the value associated with the key, or null if the key is not found;
- **V update( K key, Functor<V> fun )**: the method is used to define a new association (key, value), or to update the value associated with an existing key. Further information will be presented in class;
- **LinkedList<K> keys()**: returns all the keys in order, as defined by the method **compareTo** of the key objects;
- **LinkedList<V> values()**: returns all the values in the order defined by the method **compareTo** of the key objects!

### 6 keys() and values() (15 marks)

In the class **BinarySearchTree** implement the methods **keys()** and **values()**.

The following **JUnit** test cases can be used to validate the methods **keys** and **values**: **TestKeysAndValues.java**. Do not assume that the tests are exhaustive. In particular, do not assume that a method that passes these tests gets a perfect score.

### 7 getPathLength (10 marks)

Let the **path length** of a node be the number of links starting from the root that must be followed to reach that node. The path length of the root is 0. Implement the method **int getPathLength( K key )** that returns the path length of the node where **key** is found or **-1** is **key** if not found in that tree.

# Part IV

## Application

$$d(X, Y) = \sum_{i=1}^{4^k} (X_i - Y_i)^2$$

For our implementation, we will be using an associative structure (here an associative list or tree) for computing the counts and frequencies, hopefully, saving time and space. Here is a worked out example. Given two input strings  $X$  and  $Y$ , over a three letter alphabet: A, B and C,

Let  $k = 5$ ,  $X$  consists of two distinct (overlapping) 5-tuples **ABABA** and **BABAB**. There are 18 occurrences of each tuple, therefore  $\text{fr}(\text{ABABA}) = \text{fr}(\text{BABAB}) = \frac{18}{40-5+1}$ .  $Y$  consists of 7 distinct (overlapping) 5-tuples: **ABABA**, **ABABC**, **ABCAB**, **BABAB**, **BABCA**, **BCABA** and **CABAB**. The tuples **ABABA** and **BABAB** occur 16 times, the other tuples occur only once each. Consequently,  $\text{fr}(\text{ABABA}) = \text{fr}(\text{BABAB}) = \frac{16}{41-5+1}$ ,  $\text{fr}(\text{ABABC}) = \text{fr}(\text{ABCAB}) = \text{fr}(\text{BABCA}) = \text{fr}(\text{BCABA}) = \text{fr}(\text{CABAB}) = \frac{1}{41-5+1}$ .

$$d(X, Y) = 2(\frac{18}{36} - \frac{16}{37})^2 + 5(0 - \frac{1}{37})^2 \simeq 0.013$$

8 compare (23 marks)

The short strings used for the above test cases create no problems for either associative structures, **AList** or **BinarySearchTree**. Here are 4 bacterial genomes that you can use for further testing.

<b>Id</b>	<b>Species</b>
NC_000913	<i>Escherichia coli</i> K12
NC_000907	<i>Haemophilus influenzae</i> Rd KW20
NC_000908	<i>Mycoplasma genitalium</i> G37
NC_000964	<i>Bacillus subtilis subsp. subtilis str. 168</i>

## Directives (15 marks)

You must preferably do the assignment in teams of two, but you can also do the assignment individually. Follow all the directives available on the [assignment directives web page](#). Assignments must be submitted through the on-line submission system [Blackboard Learn](#).

## Files

Create a **.zip** file containing all the **Java source files** for this assignment. Upload this .zip file to Blackboard Learn. Make sure that all the files are in a directory (folder) named a4.1234 or a4.1234.5678, where 1234 and 5678 are the student ids of the team members. The archive must comprise the following files: **README**, [StudentInfo.java](#), [A4Q1.java](#), [Iterator.java](#), [LinkedList.java](#), [SinglyLinkedList.java](#), [AList.java](#), [Associative.java](#), [BinarySearchTree.java](#), [Distance.java](#), [Functor.java](#), [Plus.java](#), [TestA4Q1.java](#), [TestIteratorPrevious.java](#), [TestIteratorRemove.java](#), [TestTake.java](#), [TestKeysAndValues.java](#), [TestGetPathLength.java](#), [TestDistance.java](#), [Run.java](#), [RunTests.java](#), [Utils.java](#), [a.txt](#), [b.txt](#), [c.txt](#) and [d.txt](#)

- The following **.jar** file contains the necessary files for your assignment: [a4.jar](#).

## Resources

- Yang et al. (2008) Performance comparison between  $k$ -tuple distance and four model-based distances in phylogenetic tree reconstruction. *Nucleic Acids Res.* 36(5):e33, [doi:10.1093/nar/gkn075](#).
- Gregory, T.R. (2008-03-23) Animal Genome Size Database — [www.genomesize.com](#).
- Genome biology (2008-03-23) — [www.ncbi.nlm.nih.gov/Genomes](#).

## A Frequently Asked Questions (FAQ)

1. In the class `TestIteratorPrevious`, when the iterator is positioned at the end of the list, I don't understand why the method "previous" should be returning the value "three" instead of "two".

The key element here is that the behavior of the iterator is defined with respect to the **iteration** and not the list.

When an iterator is created, it's positioned to the left of the list:

```
i = l.iterator()
```

```

A   B   C
~

```

After the first call to the method `next()`, the iterator is positioned as follows:

```

A   B   C
  ~

```

After the second call to the method `next()`, the iterator is positioned as follows:

A   B   C  
      ^

After the third call to the method `next()`, the iterator is positioned as follows:

A   B   C  
          ^

`hasNext()` is now false, the method `previous` returns the last element returned by the call to the method `next`, hence C (or “three” in the Junit test).

2. “Could you post the slides that you showed in class regarding this assignment?”

Of course.

- [iti1121-a4.pdf](#)

3. “I don’t understand the source of the error: Cannot find symbol T for the static method equals, can you help?”

Up to now, we have used generics in the context of parametric types. The signature of the class included a type parameter, and its value was used for the type of instance variables, or the parameters and local variables of instance method. The value of the type information was determined when a reference variable was declared, or when an instance was created. Here is a refresher. First, declaring a parametric type:

```
public class Pair <T> {  
    private T first;  
    private T second;  
    public Pair(T first, T second) {  
        this.first = first;  
        this.second = second;  
    }  
    ...  
}
```

Next, using the parametric type, this is when the value of **T** becomes known.

```
Pair<String> p;  
p = new Pair<String>("final", "exam");
```

For Assignment #4, Question #1, the situation is different. It is not about a generic class, it is about a generic method. Consider the following, where does the value of **T** comes from?

```
public class A4Q1 {  
  
    public static boolean equals( LinkedList<T> xs,  LinkedList<T> ys) {  
  
        throw new UnsupportedOperationException("not implemented yet!");  
  
    }  
  
}  
  
// > javac A4Q1.java  
// A4Q1.java:3: error: cannot find symbol  
//     public static boolean equals( LinkedList<T> xs,  LinkedList<T> ys) {
```

```
//
// symbol:   class T
// location: class A4Q1
// A4Q1.java:3: error: cannot find symbol
//     public static boolean equals( LinkedList<T> xs,  LinkedList<T> ys) {
//                                   ^
// symbol:   class T
// location: class A4Q1
// 2 errors
```

Generic methods can be created as follows:

```
public class A4Q1 {

    public static <T> boolean equals( LinkedList<T> xs,  LinkedList<T> ys) {

        throw new UnsupportedOperationException("not implemented yet!");
    }
}
```

Simply insert <T> before the return type, now equals is a generic method. This is needed because LinkedList is a parametric type (generics). The value of the type parameter is needed.

4. “Could you post the slides that you presented in class on Wednesday (Section B)”?

Voilà!

- [iti1121-1920.pdf](#)

**Last Modified: April 7, 2014**