ITI 1121. Introduction to Computing II
Winter 2017

Assignment #5

Deadline: April 10 2017, 11:59 PM

[PDF]

Learning objectives

- **Writing** instance methods for a doubly linked list
- **Modifying** the implementation of an iterator
- **Implementing** recursive methods for a singly linked list
- **Creating** methods for a binary search tree

Part I
Big Data

As a collective, (...) humanity produces five zettabytes of data every year: 40,000,000,000,000,000,000,000 (forty sextillion) bits. (...) If you wrote out all five zettabytes that humans produce each year by hand, you would reach the galactic core of the Milky Way.
The Predictive Power of Big Data
by Erez Aiden and Jean-Baptiste Michel
Newsweek, December 25, 2013

Introduction

Enormous amounts of data are generated by sensors, online transactions, scientific experiments, and social media to name but a few examples. Big Data [1] aims to extract meaningful information (knowledge) from large and complex data sets. This information helps business leaders make rapid and precise decisions, and scientists make discoveries.

This information is often represented linearly. For instance, one can think of texts, speech, or biological sequences. Web pages are a good example of texts available in large quantities. According to «*How Search Works*», the Web comprises 130 trillion individual pages!

Comparing documents is an important activity of analytics. When the data sets are small, one can use the edit distance to compare documents. This measure takes into account the position of each word and letter in the text. It is defined as the minimum number of edit operations (insertions, deletions, and substitutions) that are needed to transform one text into another. Unfortunately, the computation of this measure scales as the square, or even the cube, of the size of the input data. Consequently, it can only be applied for small data sets.

For larger data sets, the n-grams representation is particularly interesting. A n-gram is a subsequence of n contiguous elements from some input sequence. To compare efficiently large documents, the documents are represented as frequency vectors for all possible n-grams, where the value of n could be 2, 3, 4, or 5, for instance. It is these frequency vectors that are compared rather than the entire documents. Google uses this idea for their **Google Ngram Viewer**:

- `https://books.google.com/ngrams/info`

[1] A term akin to "Analytics" and "Data Science"

Data

In order to simplify this assignment, we will be using biological data. Specifically, we will be using DNA sequences. These sequences are made of only four letters, A, C, G or T.

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) varies greatly, as illustrated by the table below.

Species	**Size**
Potato spindle tuber viroid (PSTVd)	360
Human immunodeficiency virus (HIV)	9,700
Bacteriophage lambda (λ)	48,500
Mycoplasma genitalium (bactérie)	580,000
Escherichia coli (bactérie)	4,600,000
Drosophila melanogaster (mouche à fruits)	120,000,000
Homo sapiens (humain)	3,000 000,000
Bufo bufo (common toad)	6,900,000,000
Podisma pedestris (mountain grasshopper)	17,000,000,000
Lilium longiflorum (lice de Pâques)	90,000,000,000
Necturus lewisi (a salamander)	118,000,000,000
Amoeba dubia (amibe)	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 (or even cube) 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, which we will call the n-grams distance, performs well for evolutionary tree reconstruction [Yang et al. 2008]. Here, we implement a tool for measuring the n-grams 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.

Id	**Species**
NC_000913	*Escherichia coli* K12
NC_000907	*Haemophilus influenzae* Rd KW20
NC_000908	*Mycoplasma genitalium* G37
NC_000964	*Bacillus subtilis subsp. subtilis str. 168*

Implementation

This application comprises the following classes **Distance**, **Utils**, **LinearFrequencyTable**, **TreeFrequencyTable**, and **LinkedList**, as well as two interfaces **FrequencyTable** and **Iterator**.

Distance

The method **compare** of the class **Distance** returns n-grams distance between two strings X and Y:

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

where n is the length of the n-grams (substrings of length n, also called n-tuples), X_i is frequency of the n-gram i in X, that is $\frac{\text{count}}{m-n+1}$, count is the number of occurrences of the n-gram i in X, m is the length of the string X, X and Y are strings over a 4-letter alphabet: A, C, G and T.

The above formula suggests considering all 4^n n-grams. Clearly, for short input strings (or whenever $|X| << 4^n$ and $|Y| << 4^n$), many n-grams will not be observed, their frequency will be zero, and they will not contribute to the distance measure.

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, G and C,

```
X = ACACACACACACACACACACACACACACACACACACACAC
Y = ACACACACACACACACACACGACACACACACACACACACAC
```

Let $k = 5$, X consists of two distinct (overlapping) 5-grams `ACACA` and `CACAC`. There are 18 occurrences of each n-gram, therefore $\mathrm{fr}(\mathrm{ACACA}) = \mathrm{fr}(\mathrm{CACAC}) = \frac{18}{40-5+1}$. Y consists of 7 distinct (overlapping) 5-grams: ACACA, ACACG, ACGAC, CACAC, CACGA, CGACA and GACAC. The n-grams ACACA and CACAC occur 16 times, the other n-grams occur only once each. Consequently, $\mathrm{fr}(\mathrm{ACACA}) = \mathrm{fr}(\mathrm{ACACA}) = \frac{16}{41-5+1}$, $\mathrm{fr}(\mathrm{ACACG}) = \mathrm{fr}(\mathrm{ACGAC}) = \mathrm{fr}(\mathrm{CACGA}) = \mathrm{fr}(\mathrm{CGACA}) = \mathrm{fr}(\mathrm{GACAC}) = \frac{1}{41-5+1}$.

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

Files

- Distance.java
- TestDistance.java

Utils

The class **Utils** provides utilities for this application:

- The method **setType** is used to select between the two implementations of the interface **FrequencyTable**, **LINEAR** or **TREE**.
- The (factory) method **getFrequencyTable** returns an object implementing the interface **FrequencyTable**. The type of the object depends on the current selection (see **setType**).
- The method **readFile** reads and returns the content of a file into a string.

File

- Utils.java

FrequencyTable

For this assignment, there will be two implementations of a frequency table. The interface **FrequencyTable** declares the methods that are common to both implementations. In order to simplify the assignment, we consider entries (keys) that are strings and counts that are of type **long**.

- **void init(String key:** add an entry to the frequency table. The initial value associated with the key is 0.
- **void update(String key:** increments by one the value associated with the key. Throws **NoSuchElementException** if the key is not found.
- **long get(String key):** returns the count associated with the key. Throws **NoSuchElementException** if the key is not found.
- **LinkedList<String> keys()**: returns the list of all the keys. The keys are in increasing order, based on their method **compareTo**.
- **long[] values()**: returns an array of all the counts, in increasing order of keys, using the method **compareTo**.

File

- FrequencyTable.java

1 LinearFrequencyTable [35 points]

The class **LinearFrequencyTable** implements the interface **FrequencyTable**. The information is stored in a circular doubly-linked list that also uses a dummy node. You must complete the implementation of the methods: **init**, **update**, **get**, **keys**, **values**.

Fichier

- LinearFrequencyTable.java

2 TreeFrequencyTable [35 points]

The class **TreeFrequencyTable** implements the interface **FrequencyTable**. The information is stored in a binary search tree. You must complete the implementation of the methods: **init**, **update**, **get**, **keys**, **values**.

File

- TreeFrequencyTable.java

Part II
Iterators

3 LinkedList [15 points]

Make all the necessary changes to the class **LinkedList** in order to implement the following methods.

- **Iterator<E> iterator(stop)**. Returns an iterator for this list stopping at a specified position. When the element at the specified position has been returned, the method **hasNext** returns **false**, call to the method **next** will cause **NoSuchElementException** to be thrown.
- **Iterator<E> iterator(start, stop)**. Returns an iterator for this list that starts at a specified position and stops at a specified position.

Files

- Iterator.java
- LinkedList.java
- TestLinkedList.java

4 LinkedList [5 bonus points]

- **Iterator<E> iterator(start, stop, step)**. The method returns an iterator with the same behaviour as above, except that it moves by more than one position at a time if **step** is larger than one.

Part III
Recursive list processing

5 LinkedStack [15 points]

The class **LinkedStack** implements the interface **Stack** using singly linked elements.

```
public interface Stack<E> {
    boolean isEmpty();
    E peek();
    E pop();
    void push(E elem);
}
```

For this question, you must implement the following two methods **roll()** and **unroll** using the technique presented in class to implement recursive methods:

- The method **roll** removes the top element of the stack then adds to the bottom of the stack.
- The method **unroll** removes the bottom element of the stack and adds it to the top.

```
Stack<String> s;
s = new LinkedStack<String>();

s.push("a");
s.push("b");
s.push("c");
s.push("d");
s.push("e");

System.out.println(s);

s.roll();
s.roll();

System.out.println(s);
```

Running the programme above prints the following:

```
[a, b, c, d, e]
[c, d, e, a, b]
```

Files

- Stack.java
- LinkedStack.java
- TestLinkedStack.java

Resources

- The documentation of all the classes
- `https://www.google.com/insidesearch/howsearchworks/thestory/`
- `https://www.google.com/insidesearch/howsearchworks/crawling-indexing.html`
- 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.

Rules and regulation

Follow all the directives available on the assignment directives web page, and submit your assignment through the on-line submission system Blackboard Learn.

You must preferably do the assignment in teams of two, but you can also do the assignment individually. Pay attention to the directives and answer all the following questions.

Files

You must hand in a zip file containing the following files.

- A text file README.txt which contains the names of the two partners for the assignments, their student ids, section, and a short description of the assignment (one or two lines).
- The source code of **all** your classes
  - Distance.java

  - FrequencyTable.java
  - Iterator.java
  - LinearFrequencyTable.java
  - LinkedList.java
  - LinkedStack.java
  - Stack.java
  - StudentInfo.java, properly completed and properly called from your main methods.
  - Test.java
  - TestDistance.java
  - TestFrequencyTable.java
  - TestLinkedList.java
  - TestLinkedStack.java
  - TreeFrequencyTable.java
  - Utils.java
- The corresponding JavaDoc doc directory.

Do not include any other file. In particular, do not include the directory **data** (or **data.zip**) with your submission. The archive a5.zip contains all the necessary files.

WARNINGS

- Failing to strictly follow the submission instructions will cause automated test tools to fail on your submission. Consequently, your submission will **not** get marked.
- A tool will be used to detect similarities between submissions. We will run that tool on all submissions, across all the sections of the course (including French sections). Submissions that are flagged by the tool will receive a mark of 0.
- It is your responsibility to ensure that your submission is indeed received by the back-end software, blackboard. If you submission is not there by the deadline, it will obviously **not** get marked.
- Late submission will not be accepted.

Last Modified: March 28, 2017