Bill Manaris : Spring 2016 / CSCI 230 Homework 3

Assigned Date: Thursday, Feb. 18, 2016
Due Date: Tuesday, Mar. 1, 2016
Due Time: 5 mins before beginning of class

Last modified on February 27, 2016, at 09:02 PM (see updates)

Purpose

This assignment focuses on:

- designing and implementing a linked-list ADT in Java

This is a **solo** assignment. **You may discuss the assignment only with the TA or the instructor.**

Background

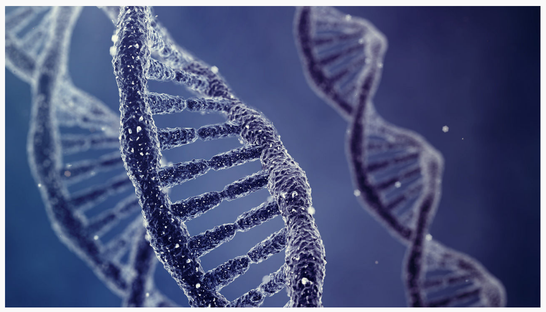


Fig 1. DNA Molecules

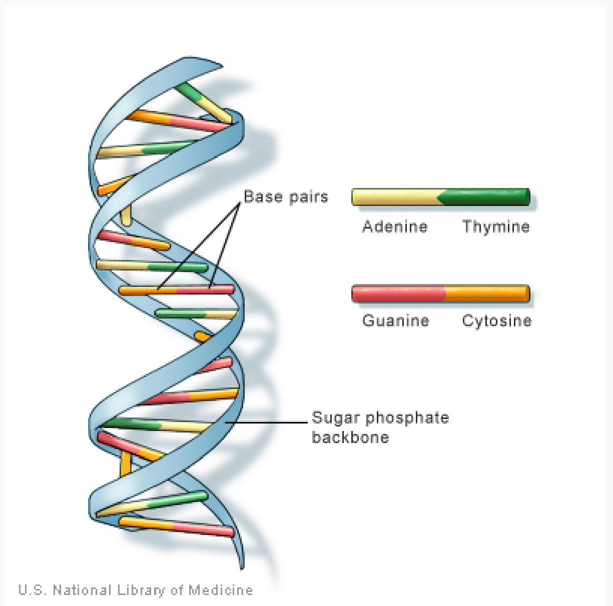


Fig 2. DNA Structure

DNA, or deoxyribonucleic acid, is the hereditary material in humans and almost all other organisms (see Fig. 1). Nearly every cell in a person's body has the same DNA. Most DNA is located in the cell nucleus (where it is called nuclear DNA), but a small amount of DNA can also be found in the mitochondria (where it is called mitochondrial DNA or mtDNA).

The information in DNA is stored as a code made up of four chemical bases: **adenine (A)**, **guanine (G)**, **cytosine (C)**, and **thymine (T)**. Human DNA consists of about 3 billion bases, and more than 99 percent of those bases are the same in all people. The order, or sequence, of these bases determines the information available for building and maintaining an organism, similar to the way in which letters of the alphabet appear in a certain order to form words and sentences.

DNA bases pair up with each other, **A with T and C with G**, to form units called **base pairs**. Each base is also attached to a sugar molecule and a phosphate molecule. Together, a base, sugar, and phosphate are called a **nucleotide**.

Nucleotides are arranged in two long strands that form a spiral called a double helix (see Fig. 2). The structure of the double helix is somewhat like a ladder, with the base pairs forming the ladder's rungs and the sugar and phosphate molecules forming the vertical sidepieces of the ladder.

(Quoted from NIH genetics reference site).

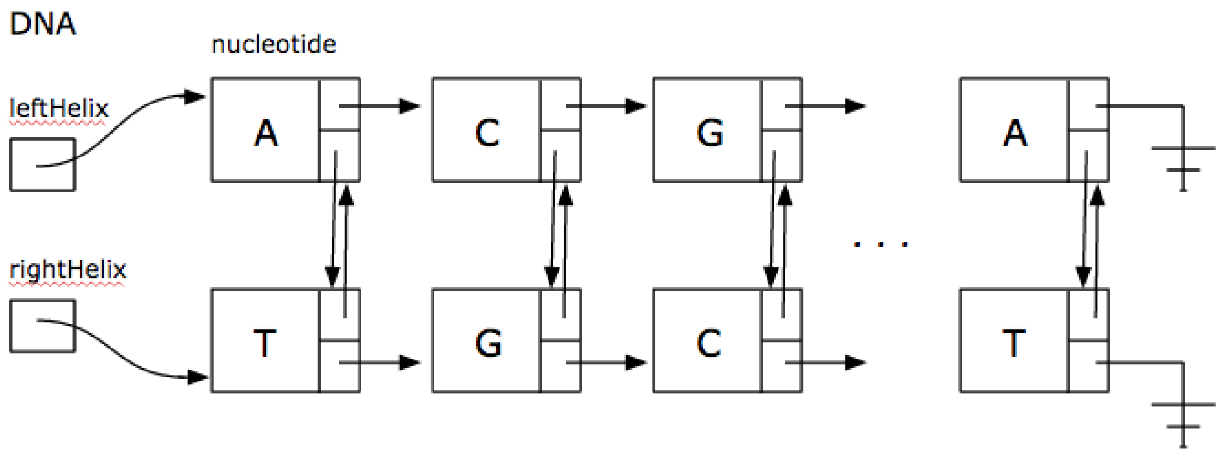
Assignment

Create a Java program, `DNA.java`, which stores a DNA sequence.

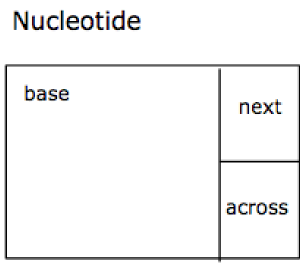
This program should contain two classes, a `DNA` class, and a (nested) `Nucleotide` class.

The `DNA` class should contain two pointers (Java references), among other things:

- `leftHelix`, which points to the first node in the left helix (see image below), and
- `rightHelix`, which points to the first node in the right helix (see image below).



Each of these references points to a `Nucleotide` object. A nucleotide is the genetic message carrying unit. As shown below, a nucleotide (or node) contains the following instance variables:



- `base` - a Java Character (this will store an 'A', 'C', 'G', or 'T'),
- `next` - a link pointing to the next nucleotide (if any) on the **same helix** (see image above), and
- `across` - a link pointing to the corresponding nucleotide (if any) on the **other helix**, i.e., the two nucleotides forming a base pair (see image above).

Details

The `DNA` class should implement the following API.

NOTE: Indices correspond to nucleotide base pairs, i.e., index 0 corresponds to nucleotide base pair "AT" (see image above), index 1 to "CG", index 2 to "GC", and so on.

1. `void insert(int index, String basePair)`, where `index` is 0-numbered from the beginning of the list, and `basePair` contains exactly two bases, "AT", "TA", "CG", or "GC".
2. `String remove(int index)`, which returns the corresponding base pair as a String. This should remove a nucleotide pair (one from the left helix and the other from the right helix), ensuring that the two helices remain properly connected (minus the two nodes, one from each).
3. `void print(int startIndex, int endIndex)`, which outputs the nucleotide base pairs (going from left to right, next left and next right, pair-by-pair, until the last one).

   - **For example**, given the DNA structure depicted above, `print(0, 3)` should output "ATCGGC".
   - In other words, `startIndex` is included, `endIndex` is excluded (as in Python lists).

4. `void clear()`, which clears the DNA structure. You do **not** need to worry about efficient garbage collection.

5. `bool isEmpty()`, which returns true if the DNA structure is empty, false otherwise.

6. `int getLength()`, which returns the number of nucleotide base pairs.

7. `int find(String basePair)` which returns the first index of the base pair (if any), or -1 if none.

8. `void printLeft()`, which prints the complete left helix (nothing from the right one).

9. `void printRight()`, which prints the complete right helix (nothing from the left one).

10. `void printBasePair(int index, int helix)`, which prints a base pair at the given `index`, where `side` refers which helix to traverse to get to this pair (0 means left, 1 means right).
    - For example, given the DNA structure depicted above, `printBasePair(0, 0)` should print "AT", whereas `printBasePair(0, 1)` should print "TA".
    - Note, this is our way to test that your `across` links work properly, so implement this as described (i.e., do not take any implementation shortcuts).

11. **(Bonus)** `void insertSequence(int index, String sequence)`, where `index` is 0-numbered from the beginning of the list, and `sequence` contains a sequence of bases, 'A', 'C', 'G', or 'T'.
    - **For example**, the sequence "ATCGGC ... AT" should generate the **exact** DNA structure depicted above.
    - In other words, the odd letters correspond to the left helix, and the even ones to the right.

12. **(Bonus)** `int findSequence(String subsequence)` which returns the first index of the base-pair subsequence (if any), or -1 if none.

13. **(Bonus)** `String removeSequence(int startIndex, int endIndex)`, which returns the corresponding base-pair subsequence as a String. This should leave the rest of the DNA sequence fully connected. Similarly, to `print()`, `startIndex` is included, `endIndex` is excluded (as in Python lists).

Error Checking

You may initially construct the above methods without worrying about error checking (i.e., assuming that they are properly called).

However, for full credit, you need to perform thorough error checking.

You do not need to worry about proper base pairing in the input (e.g., "AC" should **not be an error**). Your program may be used to study damaged DNA sequences (e.g., caused by radiation or other environmental conditions).

Input

The program should accept several lines of input, as follows:

The first line contains an `int` (e.g., 10), which specifies how many lines of input follow.

Each subsequent line of input contains an `int` (e.g., 1), which specifies which method to run (e.g., the first method above), followed by the proper parameters to this method (e.g., 0 AT)

You may assume that the input is formatted properly and is valid.

Your program should create an empty DNA structure, and operate on it accordingly based on this input.

Output

The program should output the value returned by the method called. Nothing more.

I/O Sample 1

For example, given this input:

```
2
1 0 AT
3 0 1
```

the output of your program should be:

```
AT
```

I/O Sample 2

For example, given this input:

```
4
1 0 AT
1 1 CG
1 0 TA
8
```

the output of your program should be:

```
TAC
```

I/O Sample 3

For example, given this input:

```
5
1 0 AT
1 1 CG
1 0 TA
2 1
7 AT
```

the output of your program should be:

```
AT
-1
```

I/O Sample 4

For example, given this input:

```
7
1 0 AT
1 1 CG
1 0 TA
2 1
2 1
10 0 0
10 0 1
```

the output of your program should be:

```
AT
CG
TA
AT
```

Documentation

Follow the Golden Rule of Style: "A program should be as easy for a human being to read and understand as it is for a computer to execute." [1]

In general, you should comment any variable, obscure statement, block of code, method, and class you create.

Your comments should express **why** something is being done, as opposed to **how** – the how is shown by the code.

Also see the course documentation standards in homework #1.

NOTE: You may use Javadoc formatting, if you prefer. If so, do it consistently.

Submissions

Submit via OAKS as follows:

1. Create a folder named 'First.Last', where 'First' is your first name, and 'Last' is your last name.
2. Place your Java program inside that folder.
3. Compress the folder as a .zip file.
4. Upload on OAKS.

Grading

Your grade will be based on how well you followed the above instructions, and the depth/quality of your work.

Reference

1. Cooper, D. and Clancy, M. (1985) "Oh! Pascal", 2nd ed., W.W. Norton & Company, New York, p. 42.

(Printable View of http://www.cs.cofc.edu/~manaris/?n=Spring2016.CSCI230Homework3)