Improving Efficiency with Linked Lists

COMP 2210 Assignment

Problem Overview

The Nobel Prize announcements were made earlier this month, and this assignment will have a nice tie-in to these prestigious awards and to what we've been talking about in class. The 1978 Nobel Prize in physiology and medicine was awarded to Werner Arber, Dan Nathans, and Hamilton Smith for their discovery, study, and application of *restriction enzymes*. Quoting from the press release[1]:

> Restriction enzymes provide the "chemical knives" to cut genes (= DNA) into defined fragments. These may then be used (1) to determine the order of genes on chromosomes, (2) to analyze the chemical structure of genes and of regions of DNA which regulate the function of genes, and (3) to create new combinations of genes. These techniques open up new avenues to study the organization and expression of genes of higher animals and to solve basic problems in developmental biology. In medicine, increased knowledge in this area should help in the prevention and treatment of malformations, hereditary diseases and cancer.

This assignment will simulate the use of restriction enzymes to cut a DNA strand ("DNA cleavage") and splice in new genetic material to form *recombinant DNA*. The science behind this is fascinating but mostly irrelevant to the assignment itself. None of us have to understand the science in order to complete the assignment. The main points of the assignment are (1) to provide experience in developing a software solution in the context of a real problem, (2) to provide experience in developing programs that build linked lists, and (3) to provide insight into how linked lists can offer efficiency advantages over arrays in certain circumstances.

The simulation that we're implementing is a simplification of one part of the chemical process known as *molecular cloning*. We've provided files for you to work with:

- `DnaStrand.java` – This is an interface that describes the abstract behavior necessary to simulate DNA being cleaved by a restriction enzyme and having new genetic material spliced in.
- `ArrayStrand.java` – This is a class that implements `DnaStrand` and uses an array (a `String`, specifically) to store the DNA "information" (the nucleotides). Do not change this class in any way. It is completely implemented for you and provides a reference for the expected behavior of the class you do have to implement.
- `LinkedStrand.java` – Only the shell of this class is provided for you, and it is the implementation of this class that comprises most of the work for the assignment. You must complete the implementation of this class, and its behavior must be equivalent to that of `ArrayStrand`.
- `DnaSimulation.java` – This is a class that simulates the repeated cleaving and splicing of DNA strands using restriction enzymes. Its output displays various statistics about the simulation to help you gauge the efficiency of the underlying `DnaStrand` implementations.

[1] `http://www.nobelprize.org/nobel_prizes/medicine/laureates/1978/press.html`

- LabReport.txt – This is a plain text file that contains the shell of a "lab report" for this assignment. You must provide good answers to each of the questions/prompts provided. Make sure you keep this a plain text file. Using Markdown[2] is fine (if you want), but keep it text.

- ecoli.dat – This is a plain text file that contains the DNA sequence[3] for a strain of *Escherichia coli*[4] (*E. coli*) bacteria (4,639,201 base pairs).

- ecolimed.dat – This is a plain text file that contains a subset of the DNA sequence for a strain of *E. coli* (320,101 base pairs).

To complete the assignment, you must do the following things.

(1) **Benchmark the array-based implementation.** You will use the DnaSimulation class to benchmark the ArrayStrand implementation of DnaStrand, and document this in LabReport.txt. Specifically:

  a. Show that ArrayStrand.cutAndSplice() is $O(N)$ where N is the size of the recombined strand returned by the method. Document your work in the appropriate section of LabReport.txt. You can use either ecoli.dat or ecolimed.dat for the simulation.

  b. Determine the largest power-of-two sized splice (the string spliced into the DNA strand) that the simulation supports without generating a java.lang.OutOfMemoryError when run using the java -Xms512M -Xmx512M heap size request. Document your work in the appropriate section of LabReport.txt. You must use ecoli.dat with the restriction enzyme EcoRI ("gaattc") for the simulation, and the length of the splice string must be a power of two.

  c. Double the size of heap memory available to the simulation (i.e., java -Xms1024M -Xmx1024M). With twice as much memory available, can your machine support the large splice that it couldn't handle before? If so, how long did it take to produce this next-larger splice? Document your work in the appropriate section of LabReport.txt. You must use ecoli.dat with the restriction enzyme EcoRI ("gaattc") for the simulation, and the length of the splice string must be a power of two.

  d. Keep doubling the size of heap memory until your machine can no longer support the request. What splice sizes and running times do you observe for each heap doubling? Document your work in the appropriate section of LabReport.txt. You must use ecoli.dat with the restriction enzyme EcoRI ("gaattc") for the simulation, and the length of the splice string must be a power of two.

(2) **Implement LinkedStrand.** After completing all the analyses in step 1, you must design, code, and test the LinkedStrand class. A description of the approach to the implementation is given a later section, and you must adhere to this description.

(3) **Empirically Verify LinkedStrand's Efficiency.** Once you have implemented LinkedStrand correctly, you must empirically verify that the LinkedStrand implementation is more efficient than ArrayStrand, and document this in the appropriate section of LabReport.txt. Specifically:

  a. Show that the time complexity of LinkedStrand.cutAndSplice() is independent of N where N is the size of the recombined strand returned by the method. That is, you will show that the new implementation is $O(1)$ *with respect to the size of the recombinant DNA*. Document your work in the appropriate section of LabReport.txt. You can use either ecoli.dat or ecolimed.dat for the simulation.

  b. Determine the largest power-of-two sized splice (the string spliced into the DNA strand) that the simulation supports without generating a java.lang.OutOfMemoryError when run using the java -Xms5120M -Xmx512M heap size request. Document your work in the appropriate section of LabReport.txt. You must use ecoli.dat with the restriction enzyme EcoRI ("gaattc") for the simulation, and the length of the splice string must be a power of two.

[2] https://help.github.com/articles/markdown-basics/

[3] http://www.genome.wisc.edu/sequencing.htm

[4] http://en.wikipedia.org/wiki/Escherichia_coli

c. Double the size of heap memory available to the simulation (i.e., `java -Xms1024M -Xmx1024M`). With twice as much memory available, can your machine support the large splice that it couldn't handle before? If so, how long did it take to produce this next-larger splice? Document your work in the appropriate section of `LabReport.txt`. You must use `ecoli.dat` with the restriction enzyme EcoRI ("gaattc") for the simulation, and the length of the splice string must be a power of two.

d. Keep doubling the size of heap memory until your machine can no longer support the request. What splice sizes and running times do you observe for each heap doubling? Document your work in the appropriate section of `LabReport.txt`. You must use `ecoli.dat` with the restriction enzyme EcoRI ("gaattc") for the simulation, and the length of the splice string must be a power of two.

Restriction Enzyme Cleaving

We'll need a little background information on DNA and restriction enzyme cleaving to understand the code you'll be writing, modifying, and analyzing.

DNA is formed by two biopolymer strands coiled around each other in the shape of a double helix. A strand of DNA is composed of nucleotides, which in turn are composed (in part) of four protein bases – guanine (G), adenine (A), thymine (T), and cytosine (C). As a simplification, we will model DNA as having only one strand instead of two, and we will represent this single strand as simply a sequence of bases, like this: "`AGTCGAATTCAAGTCGAATTCAGTCA`."

Enzymes[5] are biological catalysts responsible for many biochemical processes. Restriction enzymes[6] are enzymes that cut a DNA strand at particular locations. An example restriction enzyme is EcoRI[7]: "`GAATTC`." A restriction enzymes locates each occurrence of its pattern in the DNA strand and cuts the strand into two pieces at each of those points (the *binding sites or restriction sites*), leaving either "blunt" or "sticky" ends. Other chemical processes can be used to bind genetic material to these blunt or sticky ends and produce recombinant DNA. As a simplification, we will ignore the distinction of blunt and sticky ends, and we will model the cuts and splices as simply string deletion/replacement.

There are two distinct cut-related `DnaStrand` methods: `cutWith(enzyme)` and `cutAndSplice(enzyme, splice)`. The `cutWith` method cuts the strand of DNA at the *first* binding site of the specified enzyme. This method replaces its strand with the sequence that precedes the binding site and returns the sequence after the binding site. The `cutAndSplice` method cuts the strand of DNA at *every* binding site of the specified enzyme, replacing the enzyme sequence with the specified splice sequence. This method leaves the original DNA strand unchanged and returns the new recombinant DNA.

As examples of these two operations, consider the DNA sequence shown in Figure 1 and the restriction enzyme shown in Figure 2. Note that there are two occurrences of the restriction enzyme (i.e., two binding sites) in the DNA, as shown in Figure 3.

A	G	T	C	G	A	A	T	T	C	A	A	G	T	C	G	A	A	T	T	C	A	G	T	C	A

Figure 1: A sample DNA strand.

G	A	A	T	T	C

Figure 2: The restriction enzyme EcoRI

Let `dna` be the DNA strand in Figure 1, let `enzyme` be the restriction enzyme in Figure 2, and let `splice` be the DNA strand "XXYY." The behavior of the two cut methods is as follows:

- The effect on `dna` of `dna.cutWith(enzyme)` is shown in Figure 4.

[5] `http://en.wikipedia.org/wiki/Enzyme`

[6] `http://en.wikipedia.org/wiki/Restriction_enzyme`

[7] `http://en.wikipedia.org/wiki/EcoRI`

A	G	T	C	G	A	A	T	T	C	A	A	G	T	C	G	A	A	T	T	C	A	G	T	C	A

Figure 3: A DNA strand with EcoRI binding sites highlighted.

- The return value of `dna.cutWith(enzyme)` is shown in Figure 5.
- The effect on `dna` of `dna.cutAndSplice(enzyme, splice)` is shown in Figure 1; i.e., no effect.
- The return value of `dna.cutAndSplice(enzyme, splice)` is shown in Figure 6.

A	G	T	C

Figure 4: Effect on `dna` of `dna.cutWith(enzyme)`.

A	A	G	T	C	G	A	A	T	T	C	A	G	T	C	A

Figure 5: Return value of `dna.cutWith(enzyme)`.

A	G	T	C	X	X	Y	Y	A	A	G	T	C	X	X	Y	Y	A	G	T	C	A

Figure 6: Return value of `dna.cutAndSplice(enzyme, splice)`.

An Array-based Implementation

Given the discussion above, an array-based implementation of `DnaStrand` is an intuitive and natural fit. Specifically, the DNA strand information could be stored as an array of strings, and we can rely on methods from class `String` and `StringBuilder` to implement the `DnaStrand` methods. In this implementation, Figures 1 through 6 look identical to our underlying representations (minus the indexes).

The class `ArrayStrand` is a full implementation of the `DnaStrand` interface using strings as the underlying physical representation. **Your first step in the assignment will be to study this implementation.** Make sure you understand the implementation of each `DnaStrand` method, and make sure you understand each `String` and `StringBuilder` method that is used. **jGRASP viewers should be very helpful here.** For example, Figure 7 shows the source code for the `cutAndSplice` method, and Figure 8 shows a jGRASP viewer canvas depicting the execution of this method during a debug session.

Although this array-based implementation may be intuitive, it is inefficient with respect to both time and memory usage. For example, the operation that physically splices in new DNA (`recombined.append(splice)` in Figure 7) is $O(N)$ for a splice of length N. Thus the cost of creating a recombinant strand, that is, the time complexity of `cutAndSplice` with an N-length splice with B binding sites is $B \times N$, which is $O(N)$.

The amount of memory required is also linear in the size of the splice. Memory on the order of $B \times N + M$ is required to splice an N-length strand into a M-length strand with B binding sites. This more than the time complexity significantly limits the scalability of the array-based solution.

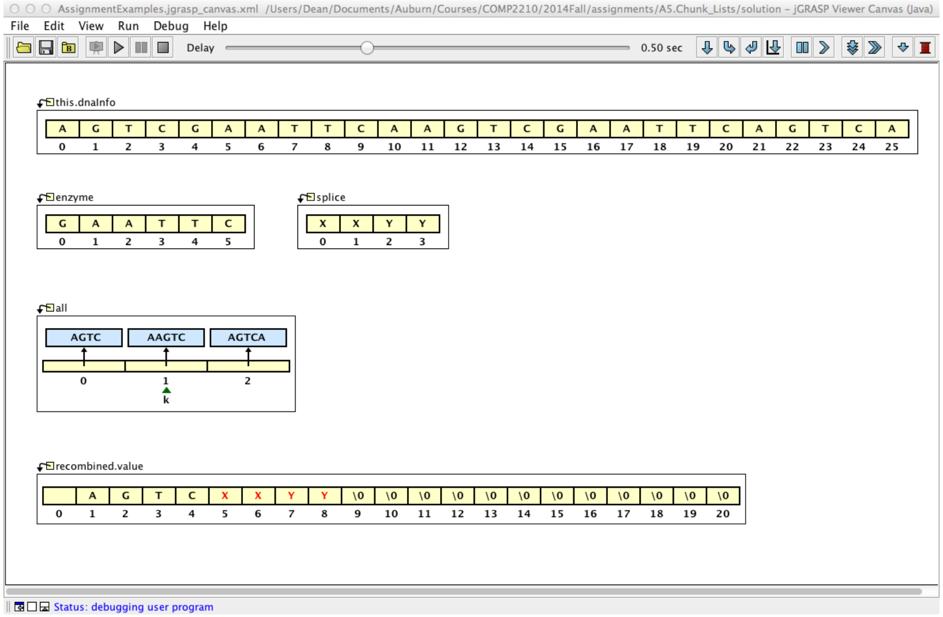
Alternate Implementation: Linked Lists

We can make a significant improvement in efficiency by basing our implementation on a linked list rather than a single array/string. A linked node representation will enable `cutAndSplice` and `append` to be implemented by adding nodes to a linked list rather than copying arrays.

You must develop `LinkedStrand` as an alternate implementation of `DnaStrand` based on linked nodes, subject to the following items.

```
public DnaStrand cutAndSplice(String enzyme, String splice){
   String toSplit = " " + dnaInfo + " ";
   String[] all = toSplit.split(enzyme);
   if (all[0].trim().equals(dnaInfo)) {
      return EMPTY_STRAND;
   }
   StringBuilder recombined = new StringBuilder(all[0]);
   for (int k = 1; k < all.length; k++) {
      recombined.append(splice);
      recombined.append(all[k]);
   }
   return new ArrayStrand(recombined.toString().trim());
}
```

Figure 7: Source code for ArrayStrand.cutAndSplice.


AssignmentExamples.jgrasp_canvas.xml /Users/Dean/Documents/Auburn/Courses/COMP2210/2014Fall/assignments/A5.Chunk_Lists/solution – jGRASP Viewer Canvas (Java)
File
Edit
View
Run
Debug
Help
Delay
0.50 sec

this.dnaInfo

A	G	T	C	G	A	A	T	T	C	A	A	G	T	C	G	A	A	T	T	C	A	G	T	C	A
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

enzyme

G	A	A	T	T	C
0	1	2	3	4	5

splice

X	X	Y	Y
0	1	2	3

all

AGTC	AAGTC	AGTCA
0	1 k	2

recombined.value

	A	G	T	C	X	X	Y	Y	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Status: debugging user program

Figure 8: A jGRASP viewer canvas during the execution of ArrayStrand.cutAndSplice.

- The node class has been written for you as a nested class in LinkedStrand. **Do not make any changes to the Node class.**
- The LinkedStrand class must maintain two fields, front and rear that reference the first and last nodes respectively of the linked list. These fields have been declared for you. **Do not change the visibility of these fields. They must be accessible to the test cases.**
- An empty LinkedStrand object has front and rear set to null. This is embodied in the parameterless contructor provided for you. **Do not change this constructor.**
- A non-empty LinkedStrand object starts out with exactly one node, which front and rear both reference. For example, the effect of new LinkedStrand("AGTCGAATTCCATTG") is shown in Figure 9.
- The size method must be $O(1)$.

- The `cutWith` method must be $O(1)$.
- The `cutAndSplice` method must be $O(1)$ with respect to the size of the splice, but may be linear with respect to the number of binding cites of the enzyme.
- The `cutAndSplice` method must not call the `String.split()` method like the `ArrayStrand` implementation does. Instead, it should call the `String.indexOf(str, fromIndex)` method and one or more of the `String.substring()` methods.
- The `cutAndSplice` method must not create a larger string, append to a string, or use a `StringBuilder` as the `ArrayStrand` implementation does. Instead, it must create and link new nodes to model splicing material into several binding sites. For example, if `dna` is the `LinkedStrand` shown in Figure 9, the the return value of `dna.cutAndSplice("GAATTC", "XXYY")` is shown in Figure 10.
- The `cutWith` and `cutAndSplice` methods should only operate on newly initialized strands; that is, those containing a single node. These methods throw an `IllegalStateException` if the strand contains multiple nodes.
- The `append(DnaStrand)` and `append(String)` methods must be $O(1)$.

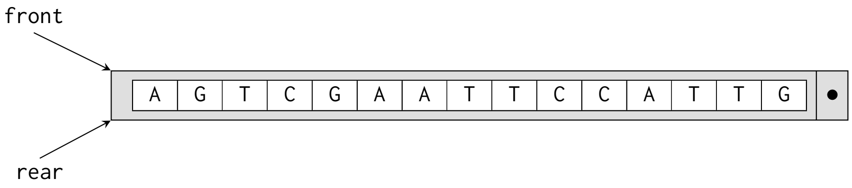


Figure 9: A newly initialized `LinkedStrand`

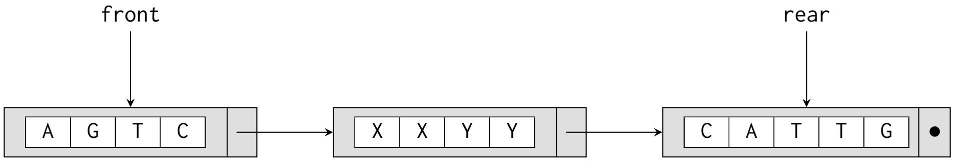


Figure 10: A `LinkedStrand` after a `cutAndSplice` operation.

Acknowledgment

This assignment was adapted from an idea and assignment originally developed by Owen Astrachan.