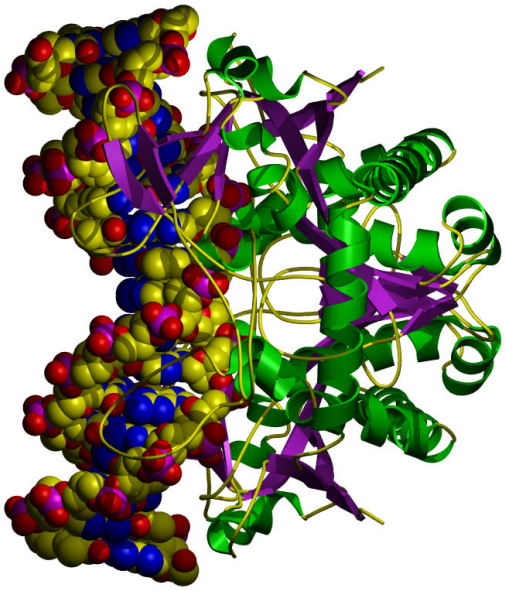
CompSci 201

Data Structures and Algorithms

DNA

You should snarf
assignment *dna* from `http://www.cs.duke.edu/courses/fall13/compsci201/snarf` or browse the code directory for the code files provided.

See the how to file for help and more instructions.



(source: http://www.astbury.leeds.ac.uk/gallery/leedspix.html)

Genesis of Assignments Linked to DNA/Genomics

Rich Pattis created an assignment based on the whole genome shotgun reassembly algorithm as part of the Nifty Assignments Archive, but there's nothing in the archive about the assignment and he hasn't used it in a while. We used that assignment at Duke in Compsci 4G for several years.This current assignment was developed in 2002, used for two years, then resurrected in the Spring of 2008 and used until 2010. It's back in a very different form though it has evolved over time. These differences leverage discussion of tradeoffs more than the original assignment, and were motivated in part by opportunities (and differences) provided by Java compared to C++. The assignment is meant to illustrate pragmatically the benefits of using a linked list. This version became a nifty assignment in 2009.

You can find more information about DNA, Restriction Enzymes, and PCR here.

The Assignment

In this assignment you're given an interface that represents a strand of DNA. You're also given a simple implementation of the interface that is based on the Java classes `String` and `StringBuilder`. You'll test and implement a new class, described below and in the howto, that uses a linked-list rather than strings/stringbuilders.

The interface and your classes will simulate two genomic ideas: reversing a strand of DNA and creating a recombinant strand by inserting new DNA where a restriction enzyme cuts/cleaves the existing DNA. This process is explained conceptually in the howto – the code models the process. You'll experiment with the existing strand class and then create a new strand class that's much more efficient in simulating cleaving/cutting/joining DNA. You'll run experiments to understand the tradeoffs in the implementation of these classes. In summary, you'll run simulated experiments to reason about the tradeoffs in alternative implementations.

In particular you'll need to do four things. These are described in more detail in the howto – be sure to look there for details.

1. Benchmark the code you're given in `SimpleStrand.cutAndSplice` that correctly finds all occurrences of an enzyme and replaces the enzyme with another strand of DNA. The benchmarking is done by the class `DNABenchMark`. Your benchmarking report must show that the algorithm/code in `SimpleStrand.cutAndSplice` is O(N) where N is the size of the recombined strand returned.
2. Test your benchmarking and O(N) simulation by running out of memory and then re-running the simulation with more memory.

3. Design, code, and test `LinkStrand` that implements the `IDnaStrand` interface so that your implementation passes the JUnit tests in `TestStrand`.
4. You must run *virtual experiments* to show that your `LinkStrand` linked-list implementation isO(B) for a strand with B breaks as described below.

Grading

Part A. 4 points Part B. 4 points Part C. 8 points Part D. 4 points

Submit your project using the assignment name *dna*.