Summer Smith
CS 201
November 10, 2016

DNA Analysis

Non-Linked List Hypotheses

Creating a Strand of DNA

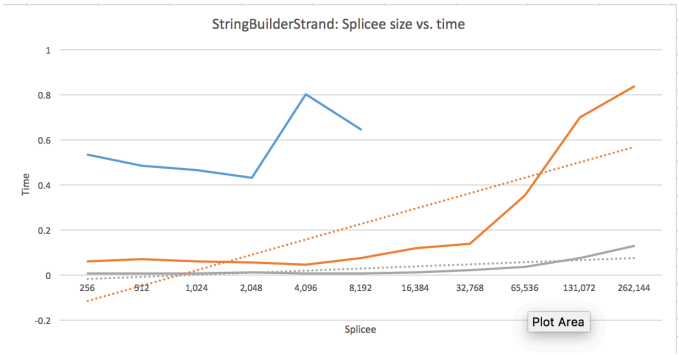
In order to generate a new strand of DNA, I created a class which randomly printed either an 'a,' 'c,' 'g,' or 't,' until the strand was n characters long. This file was saved into my data file for the DNA project. I ran these files with the pre-selected enzyme in the benchmark class, but changed the length of the enzyme when testing the number of appends versus time.

String Builder Strand hypothesis

To test the hypothesis that the StringBuilderStrand cutAndSplice method runs in O(bs) time, where b is the number of times the enzyme appears (breaks/additions) and s in the length of the splicee, I made three graphs to compare Splicee versus time, DNA length versus time, and number of appends versus time.

Splicee v. Time

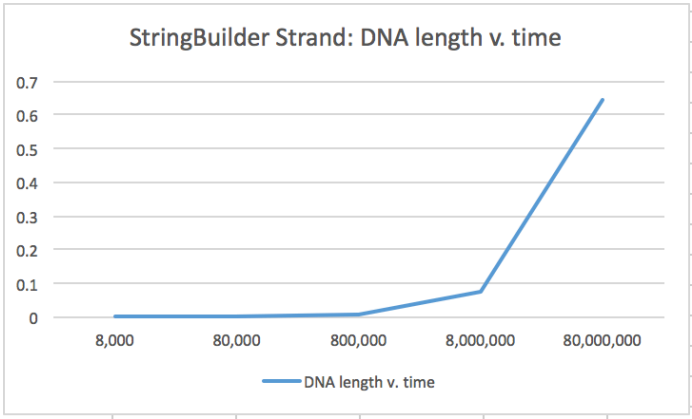
Splicee	Time
256	0.06
512	0.067
1,024	0.059
2,048	0.052
4,096	0.042
8,192	0.073
16,384	0.117



This graph shows the relationship between splicee and the time. As splicee increases, so does the time. While the relationship is not exactly linear, it is somewhat close as can be seen when splicee increases from 8,192 to 16,384 and time increases from .073 to .117. This relationship is caused by the line of code `ret.append(search.substring(start, pos));` which adds the new DNA strand to the StringBuilder ret. The bigger splicee is, the more letters that have to be appended to ret. Each append takes O(1) time, with s appends, the runtime is O(s).

DNA length v. Time

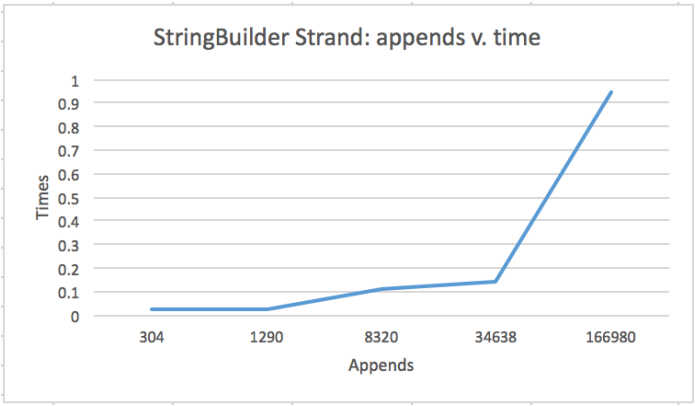
DNA length	Time
8,000	0
80,000	0.001
800,000	0.007
8,000,000	0.073
80,000,000	0.642



This graph and table show that as the length of the DNA strand increases, so does the time it takes the program to run while the number of splicee remains constant. The code that takes n time is iterating through the DNA strand in order to find all occurrences of enzyme. This time is added on to O(sb) to make the runtime O(n+sb); because sb is larger than n, we ignore n.

Appends v. Time

Appends	Time
304	0.022
1290	0.024
8320	0.113
34638	0.145
166980	0.943



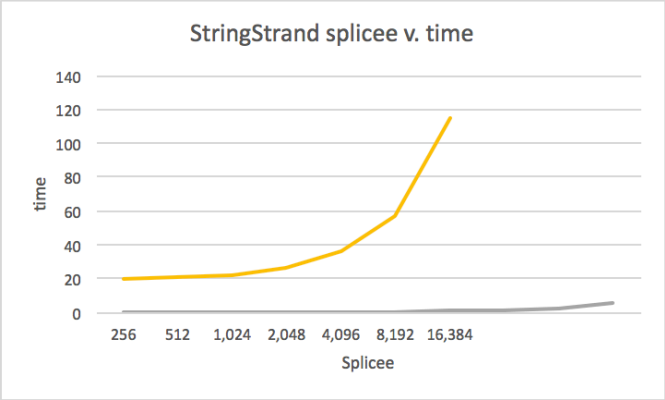
To graph the number of appends versus the time, I used the same txt file so the DNA length was the same and changed the length of the enzyme that the program was searching for. The longer the enzyme, the less it showed up in the DNA txt file. The line in the code that is responsible for the O(b) time is: `while ((pos = search.indexOf(enzyme, pos)) >= 0).` This line looks through the strand searching for incidences of the specified enzyme, which appears b times.

String Strand hypothesis

To test the hypothesis that the StringStrand cutAndSplice method runs in $O(b^2s)$ time, where b is the number of time the enzyme appears (breaks/additions) and s in the length of the splicee, I made three graphs to compare Splicee versus time, DNA length versus time, and number of appends versus time.

Splicee v. Time

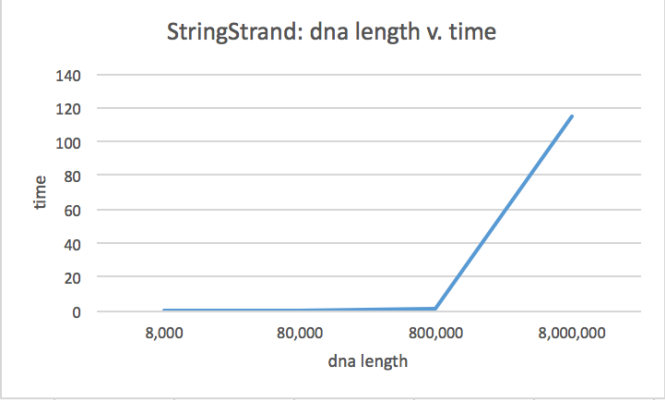
Splicee	Time
256	19.596
512	20.379
1,024	21.804
2,048	26.493
4,096	36.122
8,192	56.989
16,384	114.712



This graph shows that the relationship between splicee and time is close to linear. While the values in the table are not perfectly linear, the times increase close to a linear relationship. This relationship is linear because adding each new part of the DNA to a string (rather than a stringBuilder as in the StringBuilder implementation) takes O(sb) time. It takes sb time because it must create a new strand with s characters b times. This occurs in line 48: `ret.append(search.substring(start, pos));`.

DNA length v. Time

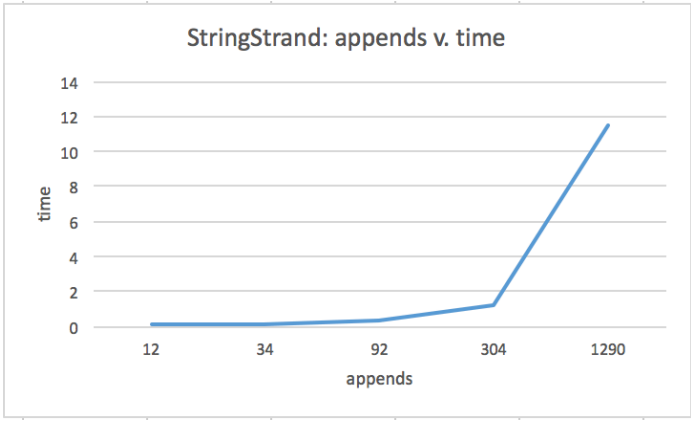
DNA length	Time
8,000	0
80,000	0.006
800,000	0.765
8,000,000	114.712



This graph and table shows that as the length of the DNA strand increases, so does the time it takes the program to run (while the number of splicee remains constant). This is because the longer the DNA strand, the longer it takes to traverse through it and find all occurrences of enzyme. This time is added on to $O(sb^2)$ to make the runtime $O(n+sb^2)$; because sb^2 is larger than n, we ignore n.

Appends v. Time

Appends	time
12	0.072
34	0.133
92	0.303
304	1.187
1290	11.528



The relationship between appends and time is close to quadratic. This can be seen when appends increases from 304 to 1290, a factor of 4. The time increases from 1.187 to 11.525, a factor of 11. This quadratic relationship is the case because it takes O(b) time to find every occurrence of enzyme in the strand. The for loop runs b times. Since StringStrand uses a string instead of a StringBuilder, the string must be remade every time an append is made, this takes O(sb) time. The nested O(sb) time within the O(b) for-loop makes the total time related to b $O(b^2)$.

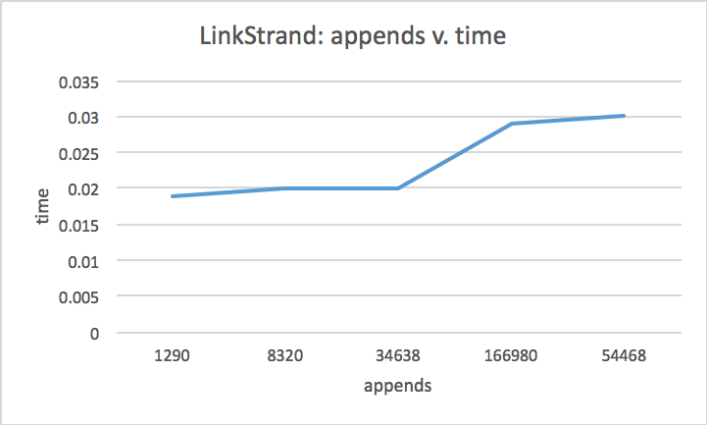
Linked List Hypothesis

Link Strand hypothesis

To test the hypothesis that the StringStrand cutAndSplice method runs in O(N+b) time, where b is the number of time the enzyme appears (breaks/additions) and s in the length of the splicee, I made a graph to compare the number of appends versus time.

Appends v. Time

appends	time
1290	0.019
8320	0.02
34638	0.02
166980	0.029
54468	0.03



This graph shows that the relationship between appends and time is somewhat linear. There is no factor of s in the LinkStrand runtime because the DNA strand is never being recreated as it is in the other two classes. The node containing the enzyme is merely being replaced with another node (replaced in the sense that the pointers pointing to it now point to a different node). It takes O(b) time because the for loop iterates every time the enzyme is found, which is b times. In order to iterate through the entire linked list to find the node that contains the enzyme, the time is O(n), with n being the number of nodes in the linked list. Everything else happens in O(1) time, this means that the final runtime is O(n+b).