WSU CPTS 471 Programming Project 2: Suffix Trees
Kyler Little, Stacy Schauls

1. System Configuration:
    a) Stacy: Intel i7 - 8550U @ 4Ghz, 8GB RAM, 8MB Cache
    b) Kyler: Intel i7 - 5250U @ 2.2Ghz, 8GB RAM, 1.5MB Cache

2) Construction Time (seconds):

	Construction Time (seconds)	
DNA Sequence	**Kyler**	**Stacy**
String S1	5.9e-05	3.1e-05
String S2	4.4e-05	6.1e-05
Ospin Gene in human	0.007478	0.008949
Ospin Gene in mouse	0.009978	0.016872
Human BRCA2 gene	0.037985	0.07798
Tomato's Chloroplast genome	5.85721	6.44438
Yeast Chromosome	868.667	251.623

Peak memory usage was 103 MB during the construction of the Suffix Tree for the Yeast Chromosome.

3) Justification

Performance optimizations did meet our expectations. Originally, we stored a "string" data type in each node *n* to represent the edge label from *n*'s parent to *n*. This was done for ease of development, but it disallowed us from running our program on the Yeast Chromosome due to the memory requirements (estimated 500+ GB). To optimize this, we use two integers – a start index and a length – which represented the starting location within the input string of the edge label and the number of characters in the edge label. This improved the memory performance by approximately 5000x. Additionally, it improved the time performance significantly as there were less string copies.

Interestingly, Stacy's computer performed better for the larger sequences. This is likely due to his computer cache size being significantly larger. Memory seemed to be the primary performance bottleneck, so this makes sense.

4) Implementation Constant (How many bytes does the code consume for every input byte? Rough estimate)

If there are *n* characters in the input and we assume one byte per character (ASCII), then there are *n* input bytes. For n input characters, we create 2*n* nodes in the worst case. Every node has the following fields:

- int _id
- int _startIndex
- int _parentEdgeLabelLength
- int _stringDepth
- Node * _sibling
- Node * _parent

Internal nodes additionally contain a “suffix link” pointer and and “child” pointer. In a 64-bit computer architecture, these addresses are 8 bytes each. Let’s assume the worst case of every node having 4 pointers.

The implementation constant is then simply:

$$((4\ ints * 4) + (4\ pointers * 4)) * 2n\ nodes / n\ input\ bytes = 64$$

5) BWT index(files)
   a) Named: Tomato_BWT.txt and Yeast_BWT.txt

6) Exact Matching repeat:

We know that finding the longest repeated substring boils down to finding the deepest node in the suffix tree. So, to find the LRS we used a DFS, in which we kept track of the deepest internal node. Once we finish the DFS, we simply grab the path label from the root to this deepest internal node, which provides us with the LRS.

	Longest Exact Matching Repeat	
DNA Sequence	**Length**	**Coordinates**
String S1	3	1, 3
String S2	4	1, 4
Ospin Gene in human	18	1810, 1817
Ospin Gene in mouse	14	2838, 2839
Human BRCA2 gene	14	7251, 9236
Tomato’s Chloroplast genome	48	88130, 88150

Yeast Chromosome	8375	451418, 460555