EN 600.439/639: Computational Genomics

Homework 3

Fall 2014, Prof. Langmead

This homework is out of 49 points:

1. (2 pts) Do rosalind.info problem "RNA splicing":

   http://rosalind.info/problems/splc/

2. (2 pts) Do rosalind.info problem "Enumerating Gene Orders":

   http://rosalind.info/problems/perm/

3. (3 pts) Do rosalind.info problem "Finding a Shared Motif":

   http://rosalind.info/problems/lcsm/

   Note: although we discussed how to do this with a suffix tree, I recommend you do something simpler here, even if its worst-case running time is worse.

4. (2 pts) Do rosalind.info problem "Finding a Spliced Motif"

   http://rosalind.info/problems/sseq/

5. (2 pts) Do rosalind.info problem "Point Mutations Include Insertions and Deletions"

   http://rosalind.info/problems/edit/

6. (2 pts) Do rosalind.info problem "Edit Distance Alignment"

   http://rosalind.info/problems/edta/

7. Let $T =$ `ggtaacc$`.

   (a) (4 pts) Draw the suffix tree of T. Label edges with the appropriate substrings of T rather than with integer pairs. In each leaf node, write the offset of the corresponding suffix. You can do this by hand, or you can use software like `dot` to draw it; see: http://www.graphviz.org/Documentation/dotguide.pdf.

   (b) (2 pts) Write the suffix array of T.

   (c) (2 pts) Write the LCP1 array of T.

   (d) (2 pts) Write the Burrows-Wheeler Transform of T.

8. (5 pts) Given a set S of k strings, we want to find every string in S that is a substring of some other string in S. Assuming the total length of the strings is n, describe an $O(k+n)$ worst-case-time algorithm to solve this problem, and argue why it is $O(k+n)$ worst-case-time. A $O(kn)$ solution won't get full credit. Hint: look back at our discussion of how to find the longest common substring of two strings.

9. (4 pts) Describe what kind of string T has the following property: when we build a suffix tree of $T\$$, the number of tree nodes equals $m+2$ where $m = |T|$. (Be general; don't assume anything about m.)

10. Say you are given the last column of a Burrows-Wheeler Matrix for a non-empty string.

    (a) (2 pts) Describe how to recreate the first column from the last column

    (b) (1 pts) Is it possible to recreate the first column from any column of the matrix? If so, how?

    (c) (1 pts) Is it possible to recreate the first column from any row of the matrix? If so, how?

11. Let $T =$ `TAGACA`. Recall our discussion of the *B-ranking*, which we found useful for algorithms that use the LF Mapping property.

    (a) (2 pts) Re-write T but including B-ranks as subscripts.

    (b) (2 pts) Why is the B-ranking a useful ranking for algorithms that make use of the LF Mapping property?

12. Here is a function that mutates a DNA string X given a parameter n:

```
import random
def mutate(x, n):
    # x is a list, so it's mutable
    y = x[:] # y = a copy of x
    for _ in xrange(n):
        randompos = random.randint(0, len(x)-1)
        y[randompos] = random.choice('ACGT')
    return y
```

    Say I execute this function for some X and n (where $n > 0$), getting result Y. Can the Hamming distance between X and Y be:

    (a) (2 pts) Less than n? Explain.

    (b) (2 pts) Equal to n? Explain.

    (c) (2 pts) Greater than n? Explain.