MSCBIO 2070/02-710: Computational Genomics, Spring 2018

HW1: Sequence alignment

Version: 1
Due: 23:59 EST, Feb 12, 2018 by autolab

Topics in this assignment:

1. Sequence alignment
2. Burrows Wheeler Transform
3. Multiple alignments
4. Suffix tries

What to hand in.

- One write-up (**in pdf format**) addressing each of following questions.
- All source code. If the skeleton is provided, you just need to complete the script and send it back. Your code is tested by autograder, please be careful with your main script name and output format.

It is highly recommended that you typeset your write-up. Illegible handwriting will not be graded.

Submit a zip file containing the completed code and the pdf file to autolab. The zip file should have the following structure (flat, three files)

./S2018HW1.pdf
./NW.py
./BWT.py

1. **[25 points] Sequence Alignment**

   Warning: You should implement the dynamic programming from scratch in Python.

   Complete the function `needleman_wunsch` in the python script `NW.py` to implement the Needleman-Wunsch algorithm with the scoring function below:

$$\begin{cases} \text{Match:} & 0 \\ \text{Mismatch:} & -1 \\ \text{Gap:} & -1 \end{cases}$$

Note: Your code is graded by an autograder. The script should be able to run with command line:

```
python NW.py example.fasta
```

The `needlman_wunsch` function should return a single value, a string with three lines with the following format:

```
alignment score
alignment in the first sequence
alignment in the second sequence
```

Each of the three items should be in separate lines, no space in front of or after each line (the box is not a part of the format). For example:

```
-2
AT-GC
ATTGA
```

2. **[25 points] Burrows-Wheeler Transform**

In this problem, you will complete several functions in `BWT.py`, related to the Burrows-Wheeler Transform (BWT).

(a) The haploid human genome has 3,234.83 Mega-basepairs. How many gigabytes would it take to store this information (Hint: first determine how many bits you need to encode a nucleotide)? *Paris japonica* has the longest plant genome sequenced so far, at 150 billion base pairs; how many gigabytes would you need to store it?

Genomes are quite large! Compressing the genome to be as small as possible, without losing any information, is vital for efficient storage and computation. Here we will explore a simple, reversible compression technique called Run Length Encoding (RLE). If a character is repeated more than once, we replace it with two instances of that character followed by the length of the run. If it only occurs once, we leave it alone. For example:

$$RLE(\text{“WOOOOOHOOOOHOOOO!”}) = \text{“WOO5HOO4HOO4!”}$$

(b) Complete the `rle` function in `BWT.py` to return the run-length-encoded version of an input string. **This must be your own code, you may not take code from elsewhere**. Use your function to encode the string:
`"scottytartanscottytartanscottytartanscottytartan"`
How long is the encoded string?

BWT is a technique to transform a sequence into one that has more repeated runs of characters. The transform is lossless, and easily reversible. As we discussed in class, this helps when trying to search for a substring. It can also help when performing lossless compression. See the Wikipedia article for more details. **Feel free to use their python implementation for your code below.**

(c) Complete the `bwt_encode` function in `BWT.py` to return the transformed version of an input string (You must as a first step insert “`{`” and “`}`” at the beginning and end of the input string, respectively. You may assume that these two characters will not appear in any raw input strings).

(d) Complete the `bwt_decode` function in `BWT.py` to return the original string given the BWT version as input (The input BWT string will contain the “`{`” and “`}`” delimiters, but your decoded output should not).

For questions 2e and 2f below, assume you are using the most straightforward, basic implementation of BWT, and that anytime you need to sort, you use Merge Sort.

(e) What is the worst case runtime complexity of `bwt_encode`?

(f) What is the worst case runtime complexity of `bwt_decode`?

(g) Now compute RLE(BWT(<same string from 2b>)). How long is the encoded string now?

(h) Explain in your own words how BWT makes it so that the transformed strings have long runs of identical characters.

(i) Finally, explore using the BWT followed by RLE for various strings (each of your examples should be at most 100 characters long, and you can use the full alphabet, save for “{” and “}”, and **must not be trivial copies of the two given strings in the file**):

i. Give an example string where the final string is longer than the original.

ii. List two example strings where the final strings are shorter.

iii. What type of strings seem to compress better with `rle(bwt(s))`? What are these called in DNA?

In practice, even more compression is applied after RLE(BWT(s)), and there are clever ways to search for matches against a string while it is compressed. BowTie (6961 citations) and BowTie2 (5680 citations) are a famous DNA alignment tools that make use of these techniques, making it possible to align millions of DNA reads to the genome within hours on a laptop.

3. **[25 points] Multiple Sequence Alignment (MSA)**

(a) We discussed in class (and in problem 1) a method to align two strings using dynamic programming. At each step, the algorithm needs to look at three neighbors to fill in a dynamic programming table cell. Consider now the problem of multiple sequence alignment. Assume we are attempting to align three strings simultaneously using a dynamic programming cube in a similar fashion. How many neighbors will the algorithm need to check to fill in each entry in this matrix? Explain.

(b) Extend this further to $K > 3$ strings, each of length roughly N. What is the runtime complexity of the alignment algorithm (Explain how you arrive at this answer)? Also name the complexity class to which the algorithm belongs. Would you consider this an efficient algorithm?

(c) Does this method result in the optimal multiple sequence alignment? (Yes or No is fine).

(d) Given the run time discussed above, multiple sequence alignment methods rely on heuristics. Consider the following method:
We are performing a progressive multiple sequence alignment. The scoring is (match $m = 2$, mismatch $n = 0$, gap $g = -2$, and gaps that align with each other do not contribute to the score) We first align Seq1 and Seq2. Now consider adding the sequence Seq3=`AGC` to the pairwise alignment. The total score for the MSA (which we are maximizing) is the sum of pairwise alignments between the sequences.

$Seq1$ `AT_C`
$Seq2$ `ATGC`

- What is the optimal alignment and its score for Seq3 under the sum-of-pairs MSA scoring method? Make sure to correctly score induced alignments. Show your work.
- What should the mismatch score be so that the two possible alignments (assigning the G of Seq3 to position 2 or position 3) have equal scores?
- Let's say we have the following progressive method for aligning K sequences: We first align two sequences. We treat this alignment of two sequences as a single object, A, and assuming we have an algorithm that can align a single sequence with an alignment object, we try aligning A with all remaining sequences and pick the best scoring alignment as the new A (which now is the alignment of three sequences). We do this until we have a multiple sequence alignment for all K sequences. What is the runtime complexity (Explain)?

(e) Propose a different heuristic method for multiple sequence alignment. Describe your method and give its runtime complexity.

4. **[25 points] Suffix Tries**

(a) Draw the **Suffix Trie** for the sequence "ATGATGC". (Include a picture of your trie in your writup)

(b) What is the longest repeat in the sequence? Explain how you are able to find this using the trie you drew. Draw a circle around the feature of your trie that indicates this (just draw it on the picture in part (a)).

(c) Why is the termination character "$" necessary? (*Hint:* try removing it from the leaves of your completed suffix trie, and explain any issues that arise).

(d) For a string of length N characters, what is the maximum number of nodes a suffix trie could have? Explain.