DSC 190 - Homework 06

Due: Thursday, February 25

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope on Thursday at 11:59 p.m.

Problem 1.

In this problem, consider an alphabet of the four characters that make up DNA: $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$. We'll use an encoding that maps $\texttt{A} \to 0$, $\texttt{C} \to 1$, $\texttt{G} \to 2$, and $\texttt{T} \to 3$.

a) Suppose we hash the first four letters of `"GATTACA"` by interpreting `"GATT"` as a number represented in base-4 using the above encoding. What is this number in base-10 (decimal)? Do not use modular arithmetic in computing the hash (that is, *do not* generate a random prime number q).

b) Now suppose we wish to hash letters `1:5` in `"GATTACA"` by interpreting `"ATTA"` as a number represented in base-4 using the above encoding. What is this number in base-10 (decimal)? Compute it directly by hand, without using the hash you found in the previous part.

c) Hash the letters `1:5` in `"GATTACA"` again, this time applying a *rolling* hash. That is, starting with the hash you found in part (a), apply corrections to arrive at the hash you found in part (b). Show your work.

Programming Problem 1.

In a file named `multiple_rabin_karp.py`, create a function named `multiple_rabin_karp(s, patterns)` which accepts a string `s` and a *list* k equal-length strings `patterns` which contains one or more patterns to search for. It should return a list-of-lists, where list 0 contains the indices where pattern 0 can be found in `s`, list 1 contains the indices where pattern 1 can be found, and so forth. Your function should be a modified version of Rabin-Karp. You can assume all strings contain only ASCII characters. Your implementation should use the rolling hash functions from lecture slide 50 without modifying them.

Hint: Rabin-Karp performs a single pass over the string `s`. A simple-but-inefficient way to solve this problem is to run Rabin-Karp over `s` k times, each time looking for another pattern. Don't do this! Instead, think of how to perform a single pass over the string `s` while simultaneously looking for all k patterns at once. How does the fact that all of the patterns have the same length help?

Example:

```
>>> multiple_rabin_karp("this is a test", ["this", "is"])
[[0], [2 5]]
```