

Rabin-Karp

N. Maillet

2023-2024

Q 1. Implement a naive algorithm of exact pattern matching in the function

```
naive_exact_pattern_matching(pattern, text):
```

This function returns the list of occurrence's position.

For example, with `pattern=AG` and `text=ATAGCTAGCAT`, the produced result must be `[2, 6]`

Q 2. Write a class `RollingHash` taking two arguments (`text` and `k`, the size of the pattern). **Please read the following before starting to code it.**

The constructor will:

- define the alphabet's mapping (A=0, C=1, G=2, T=3),
- store the length of the alphabet (`a`),
- compute the hash of the first portion of `k` characters (`k`-mer) of `text`,
- create two indexes, one for the starting position of current hash, and one for the ending position of current hash.

This class will also contains two functions:

- `next_hash(self)`, to compute the hash of the next `k`-mer,
- `get_string(self)` that return the current the `k`-mer.

Q 3. Duplicate and modify your naive algorithm to implement `rabin_karp(pattern, text)`.

Q 4. Create a function `generator_of_sequence(size)` that generates and returns a random sequence of `size` nucleotides.

Q 5. Compare the execution time of your two algorithms. For this, you will need to use the package: `time`. In this package, the function `time.time()` returns the current time when processed by Python. Use this function to determine how fast (or slow) are your algorithms, using your generator to generate texts with different size. Is everything ok?

Q 6. Install `pyahocorasick` (pip3 install pyahocorasick) and make it run using:

```
def aho_corasick(pattern, text):  
    """ Aho-Corasick implementation, requires pypi package pyahocorasick """  
    ahoc = ahocorasick.Automaton()  
    ahoc.add_word(pattern, (0, pattern))  
    ahoc.make_automaton()
```

```
matches = []
for item in ahoc.iter(text):
    matches.append(item[0]-len(pattern)+1)
return matches
```

Compare execution time with your algorithms.