Project 2

Fregona Giacomo

April 8, 2022

Task 1. In order to work out the first task, we need to store all the frequency of every k-mer that appears in the genomic sequence. We choose to represent it in a (virtual) data structure that can be thought as a quaternary tree. So we implement the class *Tree* that we use to define the quaternary tree and also its nodes (each branch of the tree can be seen as a tree itself).
For our purposes, each node of the tree has a value, i.e. a letter A, C, G or T and can have four childs, each one corresponding to one of the four letters. We define the *depth* of each node of the tree as the number of nodes we pass throught in a path from the root of the tree (that is, the first node we insert in the tree) to the node itself. We also define the *leafs* as the only nodes that have no childs. In our strategy each leaf represents a k-mer, that is the k-mer obtained reading the node's values through the path from the leaf itself to the root in reverse order. So every leaf has depth k. In each leaf is saved an additional parameter, called *frequency*, that collects the number of times the leaf has already been visited.
In order to manipulate our tree, we have defined the following methods for our class. *add* is a method that updates the frequency of the kmers in the three. If the k-mer is already stored in the tree, it updates its frequency and adds the leaf to the list of already discovered leafs. If it is the first time the kmer appears, it creates a new branch of the tree. Notice that this strategy allows to save storage space since two kmers that differs only for a few final letters are stored in the same branch of the tree with only a few levels ramification in the end. *leaftokmer* is a method that returns the k-mer corresponding to a given leaf of the three.

So the procedure followed by our function *kmer hist* can be described as follows:

- scan each infix of the genomic sequence applying it the method *add*, therefore obtaining the list of every leaf of the tree with its frequency. Each infix takes k iterations and, if we suppose $n >> k$ (as it is in relevant cases), the total complexity for this step is about $O(nk)$.
- Find the maximal(s) leaf(s) with respect to the frequencies. If we denote with $|leafs|$ the number of leafs of the tree, it is reasonable to suppose that the built-in function max takes $O(|leafs|) = O(n)$ iterations to fulfill the task.
- Construct the desired output h with a for loop on the leaflist. This also takes $O(|leafs|) = O(n)$ iterations.
- Obtain the second part of the output applying the method *leaftokmer* to the most frequent leaf(s). This takes $O(k)$ operations.

Then we see that the overall complexity is $O(nk)$.

k	Most frequent k-mer(s)	Maximal frequency
3	TTT	1565770
8	TTTTTTTT	55795
15	TTTTTTTTTTTTTTT	14580
25	TGTGTGTGTGTGTGTGTGTGTGTGT	2380
35	TGGATATTTGGATAGCTTGGAGGATTTCGTTGGAA	1043

Table 1: human cromosome 21

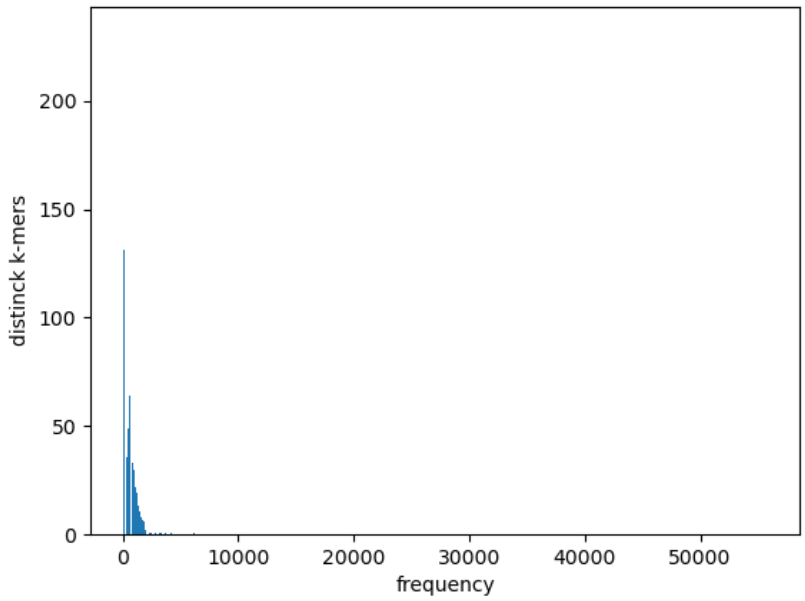

Figure 1: 8-spectrum for human cromosome 21

k	Most frequent k-mer(s)	Maximal frequency
3	CGC	115734
8	CGCTGGCG	778
15	ACGCCGCATCCGGCA	71
25	GGATAAGGCGTTCACGCCGCATCCG	39
35	GTAGGCCGGATAAGGCGTTTACGCCGCATCCGGCA	22

Table 2: Escherichia coli bacteria

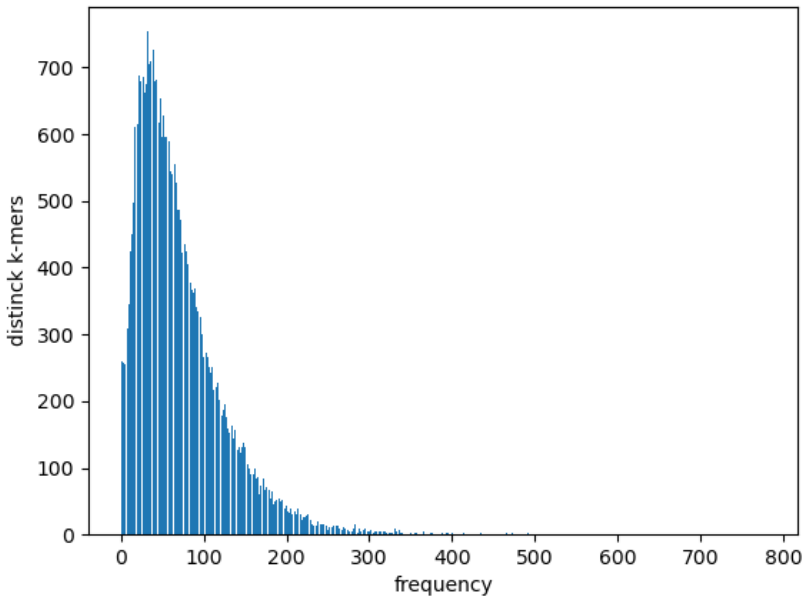

Figure 2: 8-spectrum for Escherichia coli bacteria

Task 2. Tables and figures 1,2 reports what is required by the assignment for the human chromosome 21. and the Escherichia coli bacteria

Task 3. As in Task 1, we make use of the class Tree. We construct the tree applying *add* to the k-mers in the given input list. Then we scan the genomic sequence, count how many times each k-mers already represented in the tree appears as infix of the sequence and store the first position in which it can be found. In order to do so we implemented the method *fill*, that starts from the root and follows a path througt the nodes of the three chosing at each level the branch associated to the letters of the infix. If the letters of the infix suggest a branch that does not already exist, then the path terminates and no frequency parameter is updated (this corresponds to k-mers not belonging to the given list). If the path reaches a leaf, then the algorithm changes the parameters that represent the frequency and the first apparison position.

We can describe the procedure followed by *kmer search* with the following steps:

- create the tree applying add to each k-mer in kmlist. If we call $|kmlist|$ the length of the list, since the method add requires k iterations, we can estimate the complexity of this step as $O(k|kmlist|)$.
- Apply the method *fill* to each subsequence of lenght k of the genomic sequence. There are $n - k + 1$ of that subsequences. The number of iterations of the call of *fill* depends on the input sequence, but considering the worst case (i.e. the case in which the subsequence is one of the sequences in kmlist) it can be described as $O(k)$. Then we conclude that the complexity of that step is $O(nk)$.
- Extract the maximum from the list of the leafs with respect to the frequency parameter. Both n and $|leaflist|$ are upper bounds for the lenght of that list and, since it is reasonable to suppose that the lenght of the built in munction max has linear complexity, the asymptotic complexity of this step turns out to be negligible with respects to the previous steps.
- Return the desired output that are stored in the parameters of the maximum frequency node. This step has complexity $O(1)$.

Then we see that the overall complexity is $O(nk + |leaflist|k) = O(k(n + |leaflist|))$.