Project 2: Scientific Computing

Gianmaria Lucca
MAT: 241440

In this brief report we analyze and explain the functions used in **project2.py**.

- **kmer_search(S, k, freq, x)**: function that, given the string S and the integers k, $freq$ and x, returns the lists $L1$ and $L2$ such that:
  - L1 is a list of lists, where the i–th entry is a list of integers, i.e. the locations of the i–th k–mer in the string S
  - L2 is a list where, at its i–th position, it contains the total number of point–x mutation of the k–mer substring corresponding to the i–th entry of $L1$

  The implementation is based on this procedure:

  1. Given a length k, we consider iteratively all the possible substrings of length k in S (we call them *sub*): for each one of them we use the Rabin–Karp algorithm in order to determine all the positions of its occurencies. If the substring has been already evaluated, we skip to the next substring (based on the list *sub_list*).
  2. If the number of occurencies $\geq freq$, then we have a k–mer: we append the element in the list $L1$.
  3. Given the i–th k–mer, we call the function **x_mutation_counter(S, sub, x, L2)**: it appends in $L2$ the number of point–x mutations of the i–th k–mer in L1. We generate all the possible mutations of the substring *sub* and we search their positions in the string S using the Rabin–Karp algorithm.

  Asymptotic Complexity

  Let $s = len(S)$.

  - Since we look for substrings of length k, the for loop is repeated $s - k + 1$ times
  - Suppose that $\forall i \ i \notin sub\_list$, then the cost of the check in the if statement is proportional to $len\,(sub\_list)$
  - The cost of obtaining from the string S a substring of length k is k
  - Then we use the *Rabin–Karp algorithm*, searching for the occurencies of a substring of length k in a string of length s: the cost is $(s + k)$

– Now, we look for the x–mutations: since we have three possible mutations and we search the occurencies for each one of them, we have that the cost is $3\,(s+k)$

Overall, the complexity is

$$\begin{aligned}&\sum_{i=0}^{s-k}\left[\left(\sum_{j=0}^{i} j\right)+k+(s+k)+3(s+k)\right]= \\ &=\sum_{i=0}^{s-k}\frac{i\,(i+1)}{2}+4s+5k\sim\mathcal{O}\left(s^3\right).\end{aligned} \tag{1}$$

- **spet_location(S, k, p)**: function that returns the starting location q of a k–mer K in S such that:
  - $p-2k\leq q<p-k$
  - the k–mer can't appear more than 5 times in S
  - $0.35*k\leq\#\{c\in K: c=C\vee c=G\}\leq 0.65*k$

  The implementation is based on this procedure:

  1. for $i\in[p-2k,\ p-k)$, we select the substring that starts at position i, with length k. Then we check if the proportion of Cytosine and Guanine satisfies the third constraint.
  2. If so, we check the number of occurencies in S of the k–mer using the Rabin–Karp algorithm: if $frequency\leq 5$, we pass to the last step.
  3. Now, we compare our current k–mer with the previous k–mer (with initial position q) that had the lowest frequency $q\_frequency$: if $frequency\leq q\_frequency$, then we update the value of q with i (in case of equivalence, we update the value anyway, since the index i is always nearer to p than q).

  Asymptotic Complexity

  - The first for loop is repeated $k-1$ times
  - The cost of obtaining from the string S a substring of length k is k
  - The Rabin–Karp method has a cost of $(s+k)$, where $s=len(S)$

  So, the overall complexity is $\sim\mathcal{O}\left(k^2 s\right)$.

- **get_my_pqs()**: function that returns a list of three pairs $(p,\ q)$ determined experimentally using the function **spet_location(S, k, p)**, where S is the DNA of the chloroplast of the tomato plant. Let $k=40$.
  - for $p=156$, we have no admissible q

- for $p = 999$, we have $q = 958$ and the k–mer is
  TAGCACTGAATAGGGAGCCGCCGAATACCCCAGCTACGCC
- for $p = 130800$, we have $q = 130759$ and the k–mer is
  CATGAGCTGTCCCGTAATAAAACCAGTTGTTGCTGATACC

Extra

Determining the position of q within long strings of DNA can be quite hard: the main problem might be the fact that for long strings of DNA the *hash collision* can be quite common, so the Rabin–Karp algorithm isn't very efficient in finding all the positions of a given k–mer in the DNA string. Moreover, it might be possible that the proportion's constraint of Cytosine and Guanine is satisfied more frequently when the DNA string is longer.
For example, the tomato chromosome 6 has length equal to 43960405 and for comparison the chloroplast of the tomato plant is made of only 155461 elements: so in general it might be easier to determine the positions q in the latter than in the longer DNA of the whole chromosome.