Due Date: 9 Jan 2022

CENG113 - Programming Basics

Assignment 4: Genome Analysis Tool

1) Implement **read_genes(file_path)** function that reads genes from **input.txt** and returns **gene_dict** (a dictionary of genes) where headers are keys and sequences are values.

- **input.txt** is a FASTA formatted file that consists of a number of genes. Here, each gene is represented by two lines: a header and a DNA sequence.

  ```
  >header
  sequence
  ```

- **Each header is formatted as:**

  ```
  >chromosome_id|starting_index-ending_index
  ```

- **For example:**

  ```
  >chr1|28574200-28574230
  CAGTTTATTGAGCACTTCATCTGCATCAAG
  ```

  Represents a 30 base-pair (bp) DNA sequence in range [28574200, 28574230) of chromosome 1. In **gene_dict** the same gene looks like:

  ```
  "chr1|28574200-28574230":"CAGTTTATTGAGCACTTCATCTGCATCAAG"
  ```

2) Implement **get_fragments(gene_dict, frag_len = 50)** function that splits each gene into 50 bp fragments and returns **frag_dict** (a dictionary of fragments). Note that, the **get_fragments** function filters out the fragments shorter than 50 bp and resets keys (headers) according to starting and ending indices of fragments.

- **For example:**

  ```
  "chr5|12345000-12345174":"CTGCATCAAGCCTTGTTAAGGGGTT…"
  ```

  is a 174 bp long gene. It will be split into 3 fragments of 50 bp:

  ```
  "chr5|12345000-12345050":"CTGCATCAAGCCTTGTTAAGGGGTT…"
  "chr5|12345050-12345100":"GAGCACTTCATCTGCATCAAGAAGC…"
  "chr5|12345100-12345150":"CTAGCCATGTCAAGCGATGATAAGT…"
  ```

  and the last 24 bp will be filtered out.

3.a) Implement **get_similarity(s1, s2)** helper function that takes two sequences (strings) and returns the % of matching characters.

- **For example:**

  `get_similarity("CTGCATCAAG","CTGTATCAAT") → 0.8`

  Since the underlined base-pairs (8 out of 10) are matching in the example above.

3.b) Using **get_similarity(s1, s2)** implement **filter_frags(frag_dict, threshold = 0.7)** that filters out fragments with high similarity (with sequence matching ≥ 70%) by keeping only the first one (according to lexicographic order of the keys) and returns **dissimilar_frag_dict** (a dictionary of dissimilar fragments).

- **For example:**

  `"chr1|55555000-55555010":"CTGCATCAAG"`
  `"chr1|55555010-55555020":"CTGTATCAAT"`

  Since similarity ≥ threshold, **filter_frags** function keeps only the first fragment.

4.a) Implement **generate_kmers(seq, k)** helper function that generates k-mers (words) from a DNA sequence and returns a **sentence** (string).

- **What is a k-mer?** https://en.wikipedia.org/wiki/K-mer

- **For example:**

  `generate_kmers("ATGAGTC", 4) → "ATGA TGAG GAGT AGTC"`

4.b) Using **generate_kmers(seq, k)** implement **get_sentences(dissimilar_frag_dict)** function that generates sentences of 4-mers for each dissimilar fragment and returns **sentences_dict** (a dictionary of sentences).

- **For example:**

  `"chr5|12345000-12345050":"CTGCATCAA…"`

  from **dissimilar_frag_dict** will become

  `"chr5|12345000-12345050":"CTGC TGCA GCAT CATC ATCA…"`

  in **sentences_dict**.

5.a) Implement **clean_sentence(sentence)** helper function that filters out duplicate words in a sentence (by keeping the first occurence and preserving the order) and returns a sentence (string) with distinct words.

- Hint: Use **dict.fromkeys()** function to preserve the order of words.
- **For example:**

```
clean_sentence("ATAT TATA ATAT TATG") → "ATAT TATA TATG"
```

5.b) Using clean_sentence(sentence) implement **clean_dict(sentences_dict)** function that filters out duplicate 4-mers in each sentence (by keeping the first occurence) and returns **clean_sentences_dict** (a dictionary of sentences with distinct 4-mers).

- **For example:**

```
sentences_dict = {
      "chr5|1000-1007":"ATAT TATA ATAT TATG",
      "chr5|2000-2007":"GGGG GGGG GGGG GGGA"
}

clean_dict(sentences_dict) → {
            "chr5|1000-1007":"ATAT TATA TATG",
            "chr5|2000-2007":"GGGG GGGA"
      }
```

6) Implement **write_genes(file_path, clean_sentences_dict)** function that writes the **clean_sentences_dict** into a new CSV file (**output.csv**) which is formatted as:

fragment_id	sentence	sentence_length	number_of_words
chr5|1000-1007	ATAT TATA TATG	14	3
chr5|2000-2007	GGGG GGGA	9	2
...	...	...	...

7) Implement the **main** function where you will call functions implemented in step 1, 2, 3, 4, 5, and 6, respectively.

Important Note: You should implement **ONLY** the expected functionalities listed above. Your program **SHOULD NOT** include extra features like user inputs etc. It should automatically run the required steps, print data statistics given in **template.py**, and generate **output.csv** file.

SUBMISSION RULES

1. You should submit your homeworks through Microsoft Teams.
2. One submission per group is expected. **DO NOT** submit two times as a group.
3. Submit only one python file which should be named as **CENG113_HW4_G01.py**
4. Use comments in your code to explain important parts.
5. Write your student IDs – Names & Surnames at the beginning of your code.
6. You can use only the topics covered so far, including:
   Lists, functions, dictionaries, sets, string and file manipulation
7. Please note that your code will be checked against plagiarism.