National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2022/2023

Mission 11
DNA Translation

Release date: 12th October 2022
Due: 20th October 2022, 23:59

Required Files

- codon_mapping.csv
- dna.txt
- mission11-template.py

Information

In this mission, we will investigate how dictionaries can help us act as a look-up table to handle a series of DNA sequencing tasks. We will study some simple DNA translation techniques and how using dictionaries can help speed-up certain operations.

We start by introducing some terminology in Molecular Biology, and then proceed to attempt to replicate DNA, transcribe DNA to RNA, and translate RNA into protein. Finally, explore different implementations of representing mapping tables, and the effect on time and space complexities.

This mission consists of **five** tasks.

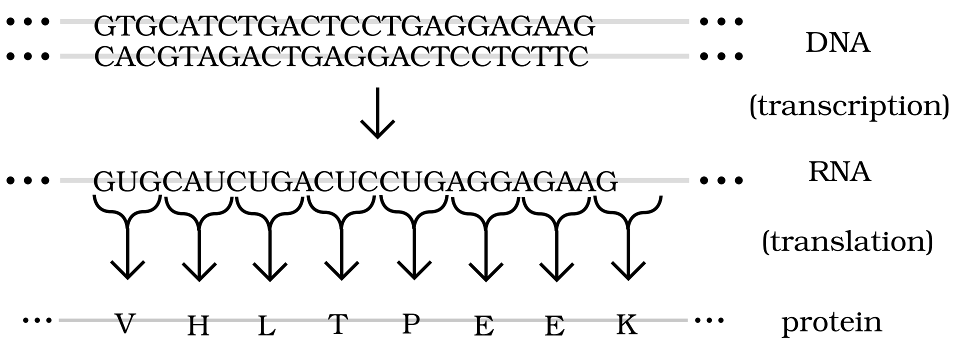
Background

The central dogma of molecular biology states that DNA encodes genetic information to produce protein. This information is first transcribed into RNA. RNA is then translated into a protein. In short, the flow of information is DNA $\rightarrow$ RNA $\rightarrow$ protein (Figure 1).

DNA is made from FOUR different chemical building blocks called nucleotides. Each nucleotide consists of a sugar molecule attached to a phosphate group and a nitrogen-containing base. The bases used in DNA are: adenine (A), cytosine (C), guanine (G), and thymine (T). To form a strand of DNA, nucleotides are linked into chains. A strand of DNA contains genetic instructions to produce proteins in living things.

RNA also contains FOUR different bases. Three of these are the same as in DNA: adenine, guanine, and cytosine. RNA contains uracil (U) instead of thymine (T).

Protein is made out of TWENTY amino acids, forming a polymer chain. Each amino acid is encoded by a sequence of THREE RNA bases, also called a codon. This leads to 64 possible combinations of codons. Of these 64 codons, 61 represent amino acids, and the remaining 3 represent stop signals, which trigger the end of protein synthesis. The full list of codons and their respective amino acids is given in the file `codon_mapping.csv`.



• • • GTGCATCTGACTCCTGAGGAGAAG • • •
• • • CACGTAGACTGAGGACTCCTCTTC • • •
DNA
(transcription)
• • • GUGCAUCUGACUCCUGAGGAGAAG • • •
RNA
(translation)
··· V H L T P E E K ···
protein

Figure 1: Central Dogma of molecular Biology
commons.wikimedia.org/wiki/File:Genetic_code.svg

Task 1: DNA Replication (3 marks)

Two DNA strands forms a double helix structure, and each nucleotide within each strand of DNA is paired with a complementary nucleotide. Complementary bases attach to one another (A-T and C-G). This means a single strand contains the information required to synthesize a new complementary strand during replication.

One point to note is that each strand of DNA has two ends, a 5' end and a 3' end. Paired DNA strands have an antiparallel structure—one strand of the helix runs in the 5' to 3' direction while the other complementary strand runs in the 3' to 5' direction. (Figure 2)

Complete the function `replicate`, that takes in an uppercase string representing a DNA strand in the 5' to 3' direction, and return the complementary DNA strand, also represented in the 5' to 3' direction. You should make use of the dictionary `dna_base_pairings` in your solution.

```
def replicate(dna_strand):
    dna_base_pairings = {
        "A": "T",
        "T": "A",
        "G": "C",
        "C": "G"
    }
    """Your code here"""

replicate("AAATGC")     # 'GCATTT'
replicate("ATTGGGCCCC") # 'GGGGCCCAAT'

with open("dna.txt") as f:
    dna = f.read()
replicate(dna)[:10]     # 'AATAGTTTCT'
```

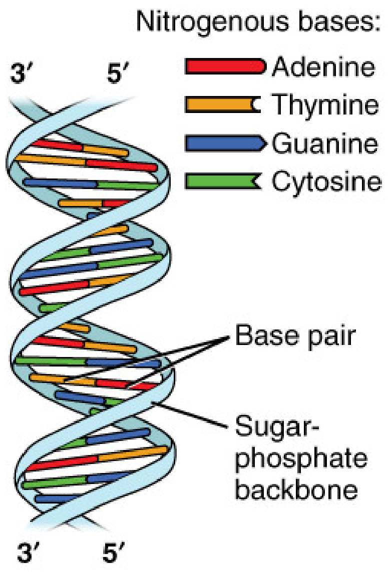


Figure 2: Antiparallel DNA strands form a double helix.
commons.wikimedia.org/wiki/File:DNA_Nucleotides.jpg

Task 2: DNA to RNA transcription (4 marks)

RNA is transcribed from DNA. During the transcription process, an RNA polymerase binds to a sequence of DNA. The bound RNA polymerase separates the DNA strands, providing the single-stranded template needed for transcription. The transcription process reads the template DNA strand from 3' to 5' direction. This results in a new RNA molecule that grows from 5' to 3' direction. The RNA transcript carries the same information as the complementary strand of DNA, but contains the base U instead of T.

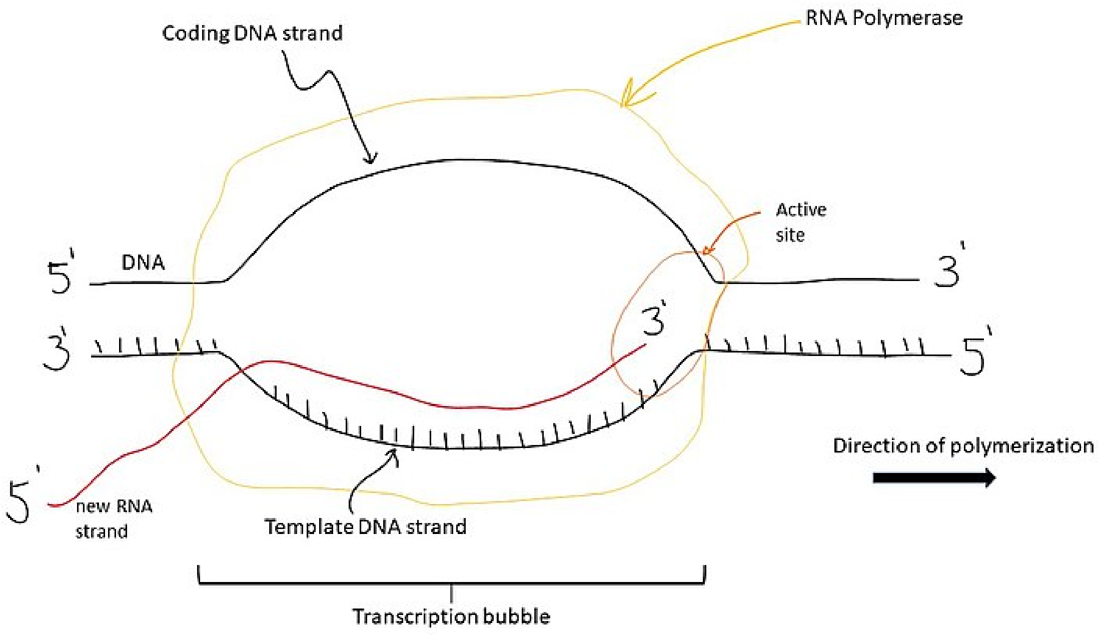


Figure 3: Simplistic view of a template DNA strand undergoing transcription to RNA
`commons.wikimedia.org/wiki/File:Transcription_bubble.jpg`

Define the function `transcribe`, that takes in an uppercase string representing a DNA template strand in the 5' to 3' direction, and return new RNA strand produced by the transcription process, also in the 5' to 3' direction.

```
def transcribe(dna_strand):
    """Your code here"""
```

Then, define the function `reverse_transcribe` which reverses this process.

```
def reverse_transcribe(rna_strand):
    """Your code here"""

transcribe("AAATGC")      # 'GCAUUU'
transcribe("ATTGGGCCCC")  # 'GGGGCCCAAU'

reverse_transcribe(transcribe("AAATGC")) # 'AAATGC'
reverse_transcribe("GGGGCCCAAU")         # 'ATTGGGCCCC'

rna = transcribe(dna)
rna[-10:]                 # 'GAAUAUGUGA'
```

Task 3: Codon-Amino acid mapping (3 marks)

Now that we have the transcribed RNA strand, we can begin translation. The message encoded in RNA is read in non-overlapping three-character groups called codons. This gives us a total of 64 possible codon combinations.

Since there are only 20 different amino acids but 64 possible codons, most amino acids are indicated by more than one codon. This phenomenon is known as redundancy or degeneracy, and it is important to the genetic code because it minimizes the harmful effects that incorrectly placed nucleotides can have on protein synthesis.

Each codon maps to a specific amino acid, which then forms the building blocks of proteins. Therefore, by translating the RNA strand, we can obtain the amino acid sequence of the protein being produced.

The first 5 codons and their mappings are shown here:

codon	amino acid	3-letter abbreviation	1-letter abbreviation
AAA	Lysine	Lys	K
AAC	Asparagine	Asn	N
AAG	Lysine	Lys	K
AAU	Asparagine	Asn	N
ACA	Threonine	Thr	T
...	...	...	...

The full list of 64 codons and their respective amino acids is found in `codon_mapping.csv`.

Define the function `get_mapping`, that takes in a CSV filename. The associated file will contain a header row and 4 columns of data. The function should return a dictionary with the first column as the dictionary-keys, and the last column as the corresponding values. Your function should work for any arbitary CSV file of the same format. You may wish to use the `read_csv` function provided to you.

```python
def read_csv(csvfilename): # provided in template file
    rows = ()
    with open(csvfilename) as csvfile:
        file_reader = csv.reader(csvfile)
        for row in file_reader:
            rows += (tuple(row), )
    return rows

def get_mapping(csvfilename):
    """Your code here"""

codon2amino = get_mapping("codon_mapping.csv")

codon2amino["ACA"] # 'T'
codon2amino["AUU"] # 'I'
codon2amino["CUC"] # 'L'
codon2amino["ACU"] # 'T'
codon2amino["UAG"] # '_'
codon2amino["UGA"] # '_'
```

Task 4: RNA to Protein translation (6 marks)

An RNA strand is a template to create a chain of amino acids that form a polymer chain, the protein. During translation, the RNA strand is read in non-overlapping groups of 3 bases. Each group of 3 bases corresponds to a codon, and each codon maps to a particular amino acid.

There are FOUR special triplets to take note of: the start codon "AUG", and the stop codons "UAA", "UAG" and "UGA". A protein begins translation from the first start codon encountered (Figure 4) and only stops when the first stop codon is reached (Figure 5), in the 5' to 3' direction.

Define the function translate, that takes in an uppercase string representing a RNA strand in the 5' to 3' direction, and returns a new protein (represented as the string of 1-letter amino acid abbreviations) produced by the translation process. All valid proteins start with "M" and end with "_". If no valid protein exists, then return None.

```
def translate(rna_strand):
    codon2amino = get_mapping("codon_mapping.csv")
    """Your code here"""

translate("AUGUAA")            # 'M_'
translate("AGAGAUGCCCUGAGGG") # 'MP_'

protein = translate(rna)
protein # 'MANLTNFHLK...YNHTHLTTY_'
```

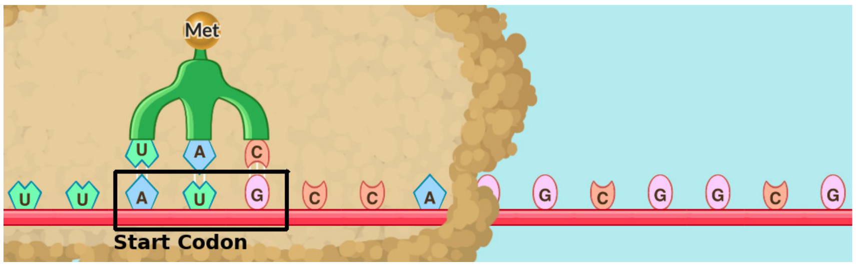


Figure 4: Translation begins at the first "AUG" start codon encountered.

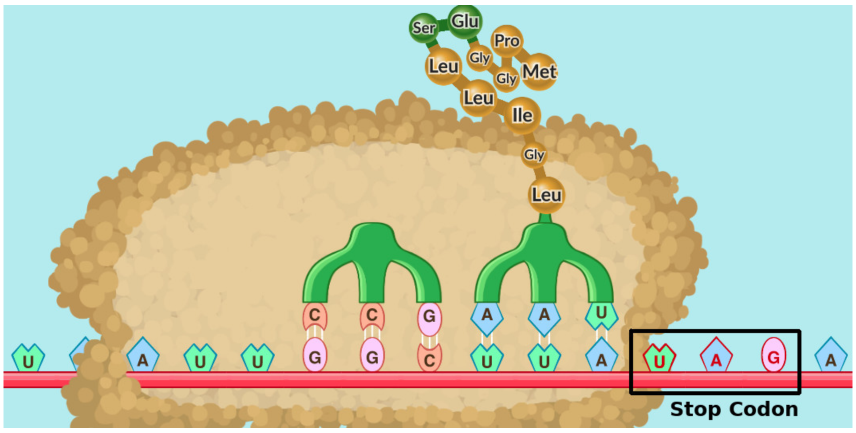


Figure 5: Translation ends at the first stop codon ("UAA", "UAG" or "UGA") reached.

To see the entire process from DNA to protein, refer to this link by **Concord Consortium**: `lab.concord.org/embeddable.html#interactives/sam/DNA-to-proteins/1-dna-to-protein.json`

Task 5: Dictionaries vs Lists (3 marks)

The current implementation of `get_mapping` returns a **dictionary**. We obtain the amino acid via accessing the dictionary value associated with the codon key in square brackets.

```
# Using Dictionary implementation in Task 3
codon2amino = get_mapping("codon_mapping.csv")

codon2amino["GAC"] # 'D'
codon2amino["AUA"] # 'I'
codon2amino["UGU"] # 'C'
codon2amino["ACU"] # 'T'
```

We could alternatively implement `get_mapping` to return a function, that internally references a **list** of codon—amino acid pairs. A possible implementation is shown below and the usage is similar except that we use parentheses to call the function instead.

```
def get_mapping(csvfilename): # Alternative implementation
    data = read_csv(csvfilename)[1:]
    pairs = list(map(lambda row: [row[0], row[-1]], data))
    def lookup(codon1):
        for codon2, amino in pairs:
            if codon1 == codon2:
                return amino
    return lookup

# Using List implementation
codon2amino = get_mapping("codon_mapping.csv")

codon2amino("CUC") # 'L'
codon2amino("AUA") # 'I'
codon2amino("UCU") # 'S'
codon2amino("ACA") # 'T'
```

Note: For the following space and time complexity analysis questions, give your answer in terms of the number of rows of the input csv file, **N**.

Space Complexity Analysis (1 mark)

State the **space complexity** of the **dictionary** version of **`get_mapping`**.

State the **space complexity** of the **list** version of **`get_mapping`**.

Time Complexity Analysis (1 mark)

State the **time complexity** to look up a codon using the **dictionary** version of **`codon2amino`**.

State the **time complexity** to look up a codon using the **list** version of **`codon2amino`**.

Conclusion (1 mark)

Using the example of translating an RNA strand of length **L** to protein, justify—by comparing space and time complexities—which implementation is better.