Algorithms for Bioinformatics

2018/2019

Basic Processing of Biological Sequences

Pedro G. Ferreira
[dCC] @ Faculty of Sciences University of Porto

Outline

- Central dogma of molecular biology
- Representation of Biological Sequences
- Basic algorithms on Biological Sequences
- Transcription and Reverse Complement
- Translation
- Exercises

Main Question

Given a (long) genomic sequence how can we find the proteins within this sequence?

>NC_000907.1:c191692-190439 Haemophilus influenzae Rd KW20 chromosome, complete genome
ATTTAAAAGAACTTAATGATCAAGTTCATCAAAATCTTATTGGGGTGCCAAATAAACGTACCCTTGAATT
TGCAAAATATTTGCAAAAACGTAATCAACATACCTGGATTCGTTATGTTGTGGTTCCTGGTTATACTGAT
AGCGATCACGATGTGCATTTATTAGGTCAGTTTATTGAAGGTATGACCAATATTGAAAAAGTTGAACTTC
TTCCTTATCATCGATTAGGTGTGCATAAATGGAAAACCCTTGGGTTAGATTATGAGCTTGAAAATGTATT
ACCGCCAACTAAAGAATCCTTAGAACATATTAAAACAATCCTAGAAGGTTATGGACACACTGTAAAATTC
TAGAATAAATGTCAGCTAACATAAGGAGTAAATAATGAAAAAAATTATTTTAACATTATCACTTGGGTTA
CTTACCGCTTGTTCTGCTCAAATCCAAAAGGCTGAACAAAATGATGTGAAGCTGGCACCGCCGACTGATG
TACGAAGCGGATATATACGTTTGGTAAAGAATGTGAATTATTACATCGATAGTGAATCGATCTGGGTGGA
TAACCAAGAGCCACAAATTGTACATTTTGATGCTGTGGTGAATTTAGATAGGGGATTGTATGTTTATCCT
GAGCCTAAACGTTATGCACGTTCTGTTCGTCAGTATAAGATTTTGAATTGTGCAAATTATCATTTAACTC
AAATACGAACTGATTTCTATGATGAATTTTGGGGACAGGGTTTGCGGGCAGCACCTAAAAAGCAAAAGAA
ACATACGTTAAGTTTAACACCTGATACAACGCTTTATAATGCTGCTCAGATTATTTGTGCAAATTATGGT
AAAGCATTTTCAGTTGATAAAAAATAAAAAAATCTGCACCTTAATTAGTTTAAATTTTATTCAATTTTTA
GGGTGCAGAGAGTATTCGATTTTTCTGCAGTTATTGCTATTTTACTGCTGGCACTTTTAAGTCTGGCTCG
TTTGGTTTTTCAATTGGTGCAAAAGTTTTATCTTTATTCGCATCAATAATTTTTTGAGTATCATTTGCTA
ATGCGGTTAAACCCATTTTTTCATAAGCTTCCTGCATCAGAAATAATCCTTCATAAGTTGCTTTAGTATC
AGGATATTGTTTTAACATTCCTACCACACGATTTGCAACTGCTACCCACGCTTTACGTTTTGCATAGAAT
TTTGCAATCTCTAATTCGTGACGAGCCAGTGCATCTTTAATATAAGCCATACGAGCTAAAGCAT

Genetic Information

- The DNA is four nucleic acid units, called nucleotides or base pairs: (A) Adenine, (G) Guanine, (T) Thymine or (C) Cytosine.
  - Adenine bonds only to Thymine nucleotides, A = T.
  - Guanine bonds only to Cytosine, C = G.

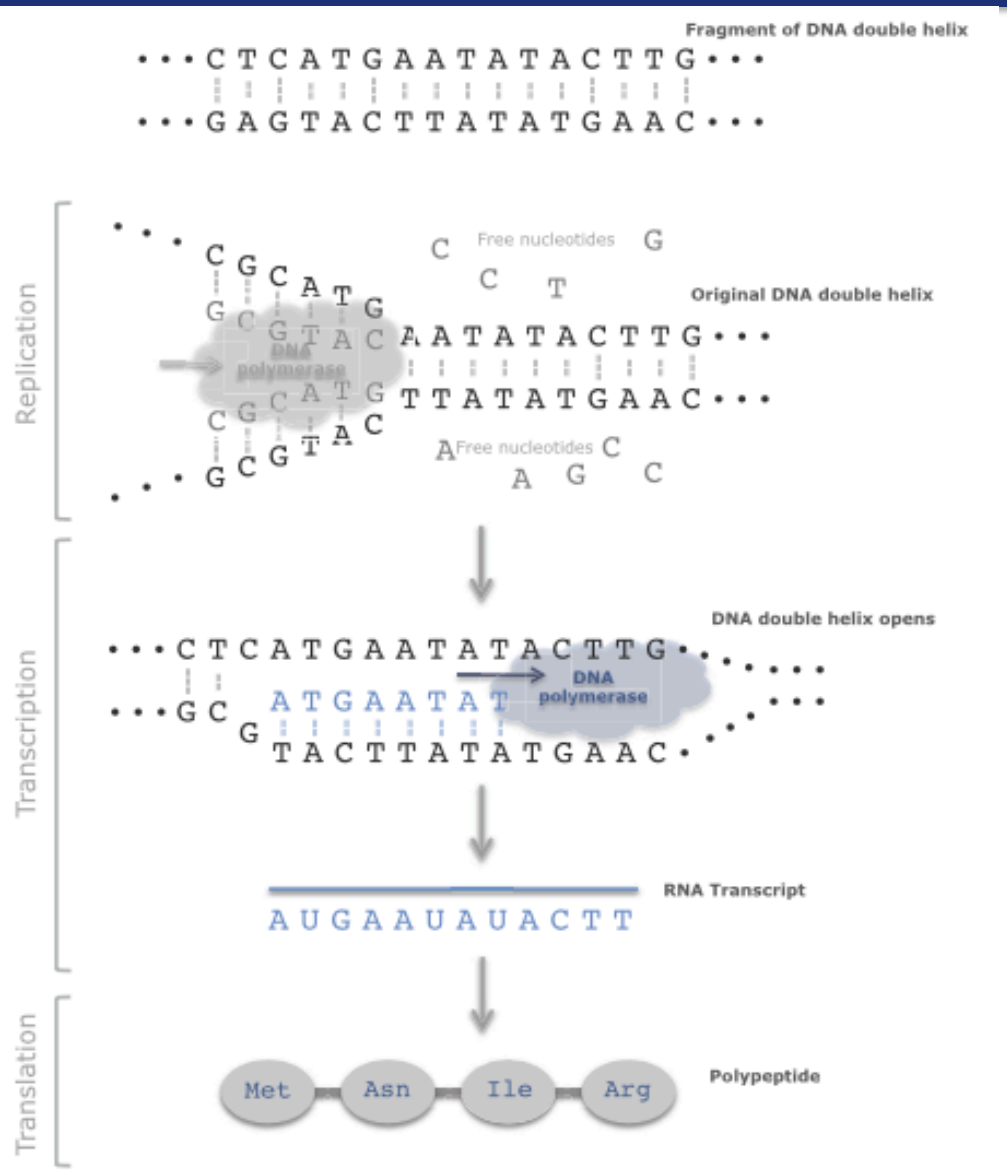
- DNA forms a double-helix structure with two complementary and anti-parallel strands (connected in opposite directions).

- Knowing the sequence of the nucleotides in one of the strands it is possible to obtain the sequence in the opposite strand by taking the complement of its nucleotides.

- RNA is a single strand molecule that in contrast to DNA does not form an helix structure. RNA is also composed of four nucleotides, but instead of Thymine (T) contains Uracil (U).

- Proteins are composed of twenty different amino-acids encoded by three nucleotides. An extra symbol is usually added to represent the stop-codon, typically an underscore (_) or the asterisk (*).

Central Dogma



From Bioinformatics Algorithms, Rocha & Ferreira

Representing Biological Sequences

DNA:

Nucleotide Code:	Base:
A	Adenine
C	Cytosine
G	Guanine
T (or U)	Thymine (or Uracil)
R	A or G
Y	C or T
S	G or C
W	A or T
K	G or T
M	A or C
B	C or G or T
D	A or G or T
H	A or C or T
V	A or C or G
N	any base
. or -	gap

Protein:

Amino Acid Code:	Three letter Code:	Amino Acid:
A	Ala	Alanine
B	Asx	Aspartic acid or Asparagine
C	Cys	Cysteine
D	Asp	Aspartic Acid
E	Glu	Glutamic Acid
F	Phe	Phenylalanine
G	Gly	Glycine
H	His	Histidine
I	Ile	Isoleucine
K	Lys	Lysine
L	Leu	Leucine
M	Met	Methionine
N	Asn	Asparagine
P	Pro	Proline
Q	Gln	Glutamine
R	Arg	Arginine
S	Ser	Serine
T	Thr	Threonine
V	Val	Valine
W	Trp	Tryptophan
X	Xaa	Any amino acid
Y	Tyr	Tyrosine
Z	Glx	Glutamine or Glutamic acid

- IUPAC code provide an extended set of symbols that allow ambiguity in the identification of a nucleotide.

- The computational representation of DNA, RNA and Protein sequences consist of strings defined respectively over an alphabet of four distinct symbols {A, C, G, T}, {A, C, G, U} and 20 characters and the stop codon representation (* or _).

Representing Biological Sequences

- Test if a given sequence is a valid DNA sequence.

Representing Biological Sequences

- Test if a given sequence is a valid DNA sequence.

```
1.  def validate_dna (dna_seq):
2.      """ Checks if DNA sequence is valid. Returns True is sequence is valid, or False otherwise. """
3.      seqm = dna_seq.upper()
4.      valid = seqm.count("A") + seqm.count("C") + seqm.count("G") + seqm.count("T")
5.      if valid == len(seqm):
6.          return True
7.      else:
8.          return False
```

Representing Biological Sequences

- Calculate the frequency of the symbols in a given sequence.

Representing Biological Sequences

- Calculate the frequency of the symbols in a given sequence.

```
1.  def frequency (seq):
2.      """ Calculates the frequency of each symbol in the sequence. Returns a dictionary. """
3.      dic = {}
4.      for s in seq.upper():
5.          if s in dic: dic[s] += 1
6.          else: dic[s] = 1
7.      return dic
```

Representing Biological Sequences

- Sorting the dictionary of frequencies (using lambda notation) of the symbols in a protein sequence read from the input.

- The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. They live where they have just been created.

- The general syntax of a lambda function:
  lambda argument_list: expression

```
1.  seq_aa = input("Protein sequence:")
2.  freq_aa = frequency(seq_aa)
3.  # Using lambda notation to sort dictionary with AA frequencies
4.  list_freq = sorted(freq_aa.items(), key = lambda x : x[1], reverse = True)
5.  for (k, v) in list_freq:
6.      print ("AA: ", k, ":", v)
```

Representing Biological Sequences

- Genes are typically found in GC-rich regions of the genome. These means that more than 60% of the sequence is formed by cytosine (C) or guanine (G) bases.

- Calculate the GC content (percentage of 'G' and 'C') of the sequence.

```
1.  def gc_content (dna_seq):
2.      """ Returns the percentage of G and C nucleotides in a DNA sequence. """
3.      gc_count = 0
4.      for s in dna_seq:
5.          if s in "GCgc": gc_count += 1
6.      return gc_count / len(dna_seq)
```

Representing Biological Sequences

- Calculate the GC content of the non-overlapping sequences of size *k*.

```
1.  def gc_content_subseq (dna_seq, k=100):
2.      """ Returns GC content of non-overlapping sub-sequences of size k. """
3.       ....
```

Representing Biological Sequences

- Calculate the GC content of the non-overlapping sequences of size *k*.

```
1.  def gc_content_subseq (dna_seq, k=100):
2.      """ Returns GC content of non-overlapping sub-sequences of size k. """
3.      res = []
4.      for i in range(0, len(dna_seq)-k+1, k):
5.          subseq = dna_seq[i:i+k]
6.          gc = gc_content(subseq)
7.          res.append(gc)
8.      return res
```

Transcription

- Transcription is the first step required to produce a protein.
- The new RNA molecule will be created as a complement to one of the strands where the gene is located. The resulting sequence will be similar to the one of the other DNA strand, but with 'U' instead of 'T' nucleotides.
- As a general practice convert sequences to upper case letters.

```
1. def transcription (dna_seq):
2. """ Function that computes the RNA corresponding to the transcription of the DNA sequence provided""
3.     assert validate_dna(dna_seq), "Invalid DNA sequence"
4.     return dna_seq.upper().replace("T","U")
```

Reverse complement

- Given the complementarity of the DNA molecule, it is often necessary to compute the content of one strand given the sequence of the other strand. This is called reverse complement.

Reverse complement

- Given the complementarity of the DNA molecule, it is often necessary to compute the content of one strand given the sequence of the other strand. This is called reverse complement.

```
1.  def reverse_complement (dna_seq):
2.      """ Computes the reverse complement of the inputted DNA sequence. """
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
4.      comp = ""
5.      for c in dna_seq.upper():
6.          if c == 'A':
7.              comp = "T" + comp
8.          elif c == "T":
9.              comp = "A" + comp
10.         elif c == "G":
11.             comp = "C" + comp
12.         elif c== "C":
13.             comp = "G" + comp
14.     return comp
```

Translating a sequence

- The translation of a DNA sequence (possible gene region) occurs with the following steps:
  1. Transcription of DNA into a RNA sequence;
  2. RNA sequence is split into codons (three consecutive and non-overlapping nucleotides);
  3. Through the genetic code table, each codon is mapped into an aminoacids;
  4. The aminoacids are concatenated, keeping their order, forming the protein sequence.

Note: the transcription step is a trivial conversion step. Computationally, one can assume a direct translation from DNA to proteins (keeping the T instead of translating to U).

Translating a sequence


```
1.  def translate_codon (cod):
2.      """Translates a codon into an aminoacid using an internal dictionary with the standard genetic co
    de."""
3.      tc = {"GCT":"A", "GCC":"A", "GCA":"A", "GCG":"A",
4.        "TGT":"C", "TGC":"C",
5.        "GAT":"D", "GAC":"D",
6.        "GAA":"E", "GAG":"E",
7.        "TTT":"F", "TTC":"F",
8.        "GGT":"G", "GGC":"G", "GGA":"G", "GGG":"G",
9.        "CAT":"H", "CAC":"H",
10.       "ATA":"I", "ATT":"I", "ATC":"I",
11.       "AAA":"K", "AAG":"K",
12.       "TTA":"L", "TTG":"L", "CTT":"L", "CTC":"L", "CTA":"L", "CTG":"L",
13.       "ATG":"M", "AAT":"N", "AAC":"N",
14.       "CCT":"P", "CCC":"P", "CCA":"P", "CCG":"P",
15.       "CAA":"Q", "CAG":"Q",
16.       "CGT":"R", "CGC":"R", "CGA":"R", "CGG":"R", "AGA":"R", "AGG":"R",
17.       "TCT":"S", "TCC":"S", "TCA":"S", "TCG":"S", "AGT":"S", "AGC":"S",
18.       "ACT":"T", "ACC":"T", "ACA":"T", "ACG":"T",
19.       "GTT":"V", "GTC":"V", "GTA":"V", "GTG":"V",
20.       "TGG":"W",
21.       "TAT":"Y", "TAC":"Y",
22.       "TAA":"_", "TAG":"_", "TGA":"_"}
23.     if cod in tc: return tc[cod]
24.     else: return None
```

Translating a sequence

- Write a function that translates a DNA sequence.
  - Inputs: DNA sequence and initial position in the sequence
  - Output: translated protein sequence

- Note that the translation of a sequence may start in frame 1, 2 or 3.
- Use the previous function to translate codons.
- Use the range function to generate the indices to traverse the sequence:

range(start, stop[, step]) -> range object

Return an object that produces a sequence of integers from start (inclusive) to stop (exclusive) by step. When step is given, it specifies the increment (or decrement).

```
1.  def translate_seq (dna_seq, ini_pos = 0):
2.      """ Translates a DNA sequence into an aminoacid sequence. """
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
4.      ...
```

Translating a sequence


```
1.  def translate_seq (dna_seq, ini_pos = 0):
2.      """ Translates a DNA sequence into an aminoacid sequence. """
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
4.      seqm = dna_seq.upper()
5.      seq_aa = ""
6.      for pos in range(ini_pos,len(seqm)-2,3):
7.          cod = seqm[pos:pos+3]
8.          seq_aa += translate_codon(cod)
9.      return seq_aa
```

Codon usage

- Given the redundancy of the genetic code, multiple codons may code for a single aminoacids.
- Different species tend to use more frequently certain codons in their genes when coding for proteins. This differential coding is typically called codon usage.

- Write a function that calculates the codon usage of an aminoacid:
  - Input: DNA sequence and an aminoacid
  - Output: dictionary with the frequencies of each codon

  Note: keep track of codon frequency (counts) by using a dictionary.

```
1.  def codon_usage(dna_seq, aa):
2.      """Provides the frequency of each codon encoding a given aminoacid, in a DNA sequence ."""
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
4.      seqm = dna_seq.upper()
5.      dic = {}
6.      total = 0
7.      ...
8.      return dic
```

Codon usage


```
1.  def codon_usage(dna_seq, aa):
2.      """Provides the frequency of each codon encoding a given aminoacid, in a DNA sequence ."""
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
4.      seqm = dna_seq.upper()
5.      dic = {}
6.      total = 0
7.      for i in range(0, len(seqm)-2, 3):
8.          cod = seqm[i:i+3]
9.          if translate_codon(cod) == aa:
10.             if cod in dic:
11.                 dic[cod] += 1
12.             else: dic[cod] = 1
13.             total += 1
14.     if total >0:
15.         for k in dic:
16.             dic[k] /= total
17.     return dic
```

Exercises

- Write a function that reads a multi-line sequence from a text file (used the provided example).

- Write a function that writes a sequence to a text file.

- Write a function that reads the standard genetic code from a file to a dictionary (file provided in the class).

Main Question

Given a (long) genomic sequence how can we find the proteins within the sequence?

>NC_000907.1:c191692-190439 Haemophilus influenzae Rd KW20 chromosome, complete genome
ATTTAAAAGAACTTAATGATCAAGTTCATCAAAATCTTATTGGGGTGCCAAATAAACGTACCCTTGAATT
TGCAAAATATTTGCAAAAACGTAATCAACATACCTGGATTCGTTATGTTGTGGTTCCTGGTTATACTGAT
AGCGATCACGATGTGCATTTATTAGGTCAGTTTATTGAAGGTATGACCAATATTGAAAAAGTTGAACTTC
TTCCTTATCATCGATTAGGTGTGCATAAATGGAAAACCCTTGGGTTAGATTATGAGCTTGAAAATGTATT
ACCGCCAACTAAAGAATCCTTAGAACATATTAAAACAATCCTAGAAGGTTATGGACACACTGTAAAATTC
TAGAATAAATGTCAGCTAACATAAGGAGTAAATAATGAAAAAAATTATTTTAACATTATCACTTGGGTTA
CTTACCGCTTGTTCTGCTCAAATCCAAAAGGCTGAACAAAATGATGTGAAGCTGGCACCGCCGACTGATG
TACGAAGCGGATATATACGTTTGGTAAAGAATGTGAATTATTACATCGATAGTGAATCGATCTGGGTGGA
TAACCAAGAGCCACAAATTGTACATTTTGATGCTGTGGTGAATTTAGATAGGGGATTGTATGTTTATCCT
GAGCCTAAACGTTATGCACGTTCTGTTCGTCAGTATAAGATTTTGAATTGTGCAAATTATCATTTAACTC
AAATACGAACTGATTTCTATGATGAATTTTGGGGACAGGGTTTGCGGGCAGCACCTAAAAAGCAAAAGAA
ACATACGTTAAGTTTAACACCTGATACAACGCTTTATAATGCTGCTCAGATTATTTGTGCAAATTATGGT
AAAGCATTTTCAGTTGATAAAAAATAAAAAAATCTGCACCTTAATTAGTTTAAATTTTATTCAATTTTTA
GGGTGCAGAGAGTATTCGATTTTTCTGCAGTTATTGCTATTTTACTGCTGGCACTTTTAAGTCTGGCTCG
TTTGGTTTTTCAATTGGTGCAAAAGTTTTATCTTTATTCGCATCAATAATTTTTTGAGTATCATTTGCTA
ATGCGGTTAAACCCATTTTTTCATAAGCTTCCTGCATCAGAAATAATCCTTCATAAGTTGCTTTAGTATC
AGGATATTGTTTTAACATTCCTACCACACGATTTGCAACTGCTACCCACGCTTTACGTTTTGCATAGAAT
TTTGCAATCTCTAATTCGTGACGAGCCAGTGCATCTTTAATATAAGCCATACGAGCTAAAGCAT

Main Question

Given a (long) genomic sequence how can we find the proteins?

http://www.bioinformatics.org/sms2/orf_find.html; ORF begin with ATG; search direct strand; >= 30 codons long; Standard genetic code;

ORF Finder results

Results for 1254 residue sequence "NC_000907.1:c191692-190439 Haemophilus influenzae Rd KW20 chromosome, complete genome" starting "ATTTAAAAGA"

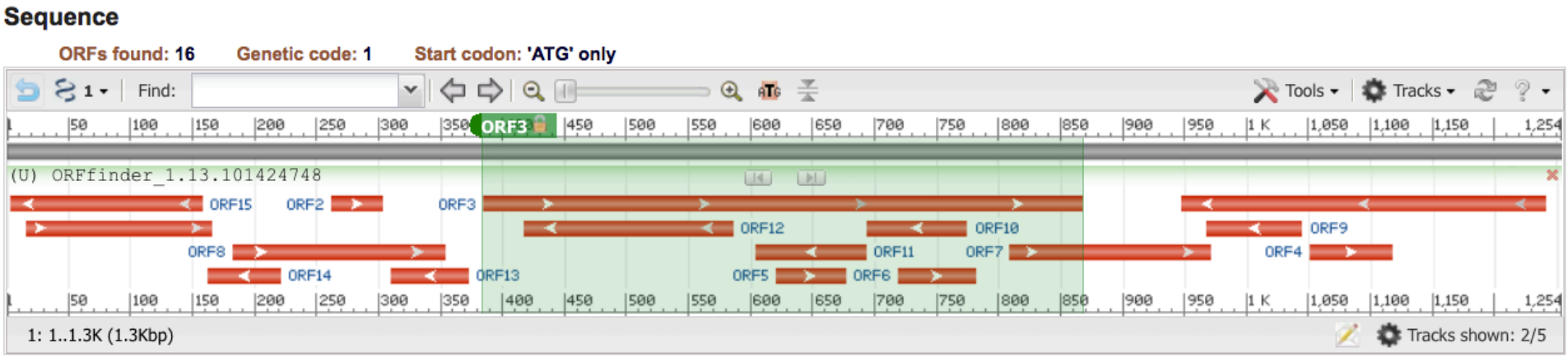
```
>ORF number 1 in reading frame 1 on the direct strand extends from base 16 to base 165.
ATGATCAAGTTCATCAAAATCTTATTGGGGTGCCAAATAAACGTACCCTTGAATTTGCAA
AATATTTGCAAAAACGTAATCAACATACCTGGATTCGTTATGTTGTGGTTCCTGGTTATA
CTGATAGCGATCACGATGTGCATTTATTAG

>Translation of ORF number 1 in reading frame 1 on the direct strand.
MIKFIKILLGCQINVPLNLQNICKNVINIPGFVMLWFLVILIAITMCIY*

>ORF number 2 in reading frame 1 on the direct strand extends from base 385 to base 867.
ATGAAAAAAATTATTTTAACATTATCACTTGGGTTACTTACCGCTTGTTCTGCTCAAATC
CAAAAGGCTGAACAAAATGATGTGAAGCTGGCACCGCCGACTGATGTACGAAGCGGATAT
ATACGTTTGGTAAAGAATGTGAATTATTACATCGATAGTGAATCGATCTGGGTGGATAAC
CAAGAGCCACAAATTGTACATTTTGATGCTGTGGTGAATTTAGATAGGGGATTGTATGTT
TATCCTGAGCCTAAACGTTATGCACGTTCTGTTCGTCAGTATAAGATTTTGAATTGTGCA
AATTATCATTTAACTCAAATACGAACTGATTTCTATGATGAATTTTGGGGACAGGGTTTG
CGGGCAGCACCTAAAAAGCAAAAGAAACATACGTTAAGTTTAACACCTGATACAACGCTT
TATAATGCTGCTCAGATTATTTGTGCAAATTATGGTAAAGCATTTTCAGTTGATAAAAAA
TAA

>Translation of ORF number 2 in reading frame 1 on the direct strand.
MKKIILTLSLGLLTACSAQIQKAEQNDVKLAPPTDVRSGYIRLVKNVNYYIDSESIWVDN
QEPQIVHFDAVVNLDRGLYVYPEPKRYARSVRQYKILNCANYHLTQIRTDFYDEFWGQGL
RAAPKKQKKHTLSLTPDTTLYNAAQIICANYGKAFSVDKK*
```

- There are 8 ORFs in the two strands and the 3 frames.

Main Question

Given a (long) genomic sequence how can we find the proteins the constitute the cell?

- NCBI ORF Finder



Open Reading Frames

- The translation of a protein sequence occurs for the coding region of the gene. This region start with a *start codon* (ATG) and stops when one of the *stop codons* is found.

- Often, when we are given a DNA sequence containing a gene we do not know in advance where the coding regions starts (the sequence may have multiple ATG codons).

- A *reading frame* is a way of dividing a DNA (or RNA) sequence into a set of consecutive non-overlapping triplets or codons. Recall that a sequence may have 6 reading frames: 3 in one strand: +1, +2, +3 starting at position 1, 2 and 3 respectively of the sequence (in python strings at index 0, 1 and 2).

- An *open reading frame* is a reading frame with the potential to be translated into protein. It should not be too short and contain a minimum number of codons.

Reading Frames

- Given a DNA sequence calculate all the possible reading frames using previously defined functions.

```
1.  def reading_frames (dna_seq):
2.      """Computes the six reading frames of a DNA sequence (including the reverse complement.""
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
4.      res = []
5.      res.append(translate_seq(dna_seq,0))
6.      res.append(translate_seq(dna_seq,1))
7.      res.append(translate_seq(dna_seq,2))
8.      rc = reverse_complement(dna_seq)
9.      res.append(translate_seq(rc,0))
10.     res.append(translate_seq(rc,1))
11.     res.append(translate_seq(rc,2))
12.     return res
```

```
DNA:
ATGAAATTATGAATGAGCCTCAGCTGAAGCATCGCGCATCAGACTACGCTCAGACTCAGACTCAGCATTATAGTGAAT
GTTAATAAATAAAATAA

Reading frames:
MKL_MSLS_SIAHQTTLRLRLSIIVNVNK_N
_NYE_ASAEASRIRLRSDSDSAL__MLINKI
EIMNEPQLKHRASDYAQTQTQHYSEC__IK_
LFYLLTFTIMLSLSLSVV_CAMLQLRLIHNF
YFIY_HSL_C_V_V_A_SDARCFS_GSFIIS
ILFINIHYNAESESERSLMRDASAEAHS_FH
```

Protein Reading Frames

- For an aminoacid sequence translated from the DNA sequence, find all possible open reading frames. The putative protein sequence should start with the start codon and finish in the stop codon. Note that multiple start codons may appear before the stop codon. Add the resulting "proteins" to a list.

```
1.  def all_proteins_rf (aa_seq):
2.      """Computes all posible proteins in an aminoacid sequence."""
3.      ...
```

Protein Reading Frames

- For an aminoacid sequence translated from the DNA sequence, find all possible open reading frames. The putative protein sequence should start with the start codon and finish in the stop codon. Note that multiple start codons may appear before the stop codon. Add the resulting "proteins" to a list.

```
1.  def all_proteins_rf (aa_seq):
2.      """Computes all posible proteins in an aminoacid sequence."""
3.      aa_seq = aa_seq.upper()
4.      current_prot = []
5.      proteins = []
6.      for aa in aa_seq:
7.          if aa == "_":
8.              if current_prot:
9.                  for p in current_prot:
10.                     proteins.append(p)
11.                 current_prot = []
12.         else:
13.             if aa == "M":
14.                 current_prot.append("")
15.             for i in range(len(current_prot)):
16.                 current_prot[i] += aa
17.     return proteins
```

All Protein Reading Frames

- Compute all putative proteins by applying the previous function to all reading frames of the sequence.

```
1.  def all_orfs (dna_seq):
2.      """Computes all possible proteins for all open reading frames."""
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
4.      rfs = reading_frames (dna_seq)
5.      res = []
6.      for rf in rfs:
7.          prots = all_proteins_rf(rf)
8.          for p in prots: res.append(p)
9.      return res
```

All Protein Reading Frames

- Protein sequences of smaller length are unlikely. Rewrite the previous function so that only putative protein sequences with a minimum length are kept and also sort them in increasing order of their length.

```
1.  def all_orfs_ord (dna_seq, minsize = 0):
2.      """Computes all possible proteins for all open reading frames. Returns ordered list of proteins w
    ith minimum size."""
3.      assert validate_dna(dna_seq), "Invalid DNA sequence"
```

All Protein Reading Frames

- Protein sequences of smaller length are unlikely. Rewrite the previous function so that only putative protein sequences with a minimum length are kept and also sort them in increasing order of their length.

```
1.  def all_orfs_ord (dna_seq, minsize = 0):
2.      """Computes all possible proteins for all open reading frames.
3.  eturns ordered list of proteins with minimum size."""
4.      assert validate_dna(dna_seq), "Invalid DNA sequence"
5.      rfs = reading_frames (dna_seq)
6.      res = []
7.      for rf in rfs:
8.          prots = all_proteins_rf(rf)
9.          for p in prots:
10.             if len(p) > minsize: insert_prot_ord(p, res)
11.     return res
12.
13. def insert_prot_ord (prot, list_prots):
14.     i = 0
15.     while i < len(list_prots) and len(prot) < len(list_prots[i]):
16.         i += 1
17.     list_prots.insert(i, prot)
```

Exercises

- Write a test function that reads from the input a long DNA sequence and performs the following steps on the sequence:
  1. Validates;
  2. Translates;
  3. Obtains the reverse complement;
  4. Calculates the GC-content;
  5. Performs the direct translation;
  6. Writes to a file all the putative protein sequences in increasing order of their length.