Биологийн дарааллын үндсэн боловсруулалт(Basic Processing of Biological Sequences)

1. Биологийн дараалал: Тэмдэг тэмдэглэгээ, үндсэн алгоритмууд

1.1 Тэмдэг тэмдэглэгээ

Олон янзын практик зорилгоор биоинформатикийн алгоритм, хэрэгсэлүүдэд генетикийн мэдээлэл бүхий молекулуудыг нуклеотидын "нэг хэмжээст дараалал" хэлбэрээр дүрсэлдэг.

Symbol	Name	Nucleotides represented
A	Adenine	A
C	Cytosine	C
G	Guanine	G
T	Thymine	T
U	Uracil	U
K	Keto	G, T
M	Amino	A, C
R	Purine	A, G
S	Strong	C, G
W	Weak	A, T
Y	Pyrimidine	C, T
B	Not A	C, G, T
D	Not C	A, G, T
H	Not G	A, C, T
V	Not T	A, C, G
N	Any base	A, C, G, T

Нуклеотидын IUPAC тэмдэглэгээ

Symbol	Name
A	Alanine
C	Cysteine
D	Aspartic acid
E	Glutamic acid
F	Phenylalanine
G	Glycine
H	Histidine
I	Isoleucine
L	Lysine
M	Methionine
N	Asparagine
P	Proline
Q	Glutamine
R	Arginine
S	Serine
T	Threonine
V	Valine
T	Tryptophan
Y	Tyrosine

Амин хүчлийн IUPAC тэмдэглэгээ

International Union of Pure and Applied Chemistry-IUPAC

1.2 Basic Algorithms

In [1]:
```python
def validate_dna(dna_seq):
    """ Checks if DNA sequence is valid. Returns True is sequence is valid, or False otherwise. """
    seqm = dna_seq.upper()
    valid = seqm.count("A") + seqm.count("C") + seqm.count("G") + seqm.count("T")
    if valid == len(seqm): return True
    else: return False

validate_dna("atagagagatctcg")
validate_dna("ATAGAXTAGAT")
```

Out[1]:
```
False
```

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

frequency("atagataactcgcatag")
#frequency ("MVVMKKSHHVLHSQSLIK")
```

Out[2]:
```
{'A': 7, 'T': 4, 'G': 3, 'C': 3}
```

In [3]:
```python
# Амин хүчлийн давтамжийг тоолно.
# Давтамжийг эрэмбээр хэвлэнэ.
# lambda функц хэрэглэсэн
seq_aa = input("Protein sequence:")
freq_aa = frequency(seq_aa)
list_f = sorted(freq_aa.items(), key=lambda x: x[1], reverse = True)
for(k,v) in list_f:
    print("Aminoacid :", k, ":", v)
```

```
Protein sequence:MVVMKKSHHVLHSQSLIK
Aminoacid : V : 3
Aminoacid : K : 3
Aminoacid : S : 3
Aminoacid : H : 3
Aminoacid : M : 2
Aminoacid : L : 2
Aminoacid : Q : 1
Aminoacid : I : 1
```

In [4]:
```python
#Дарааллын урттай харьцуулахад 'G' ба 'C' нуклеотидын хувь
def gc_content (dna_seq):
    """ Returns the percentage of G and C nucleotides in a DNA sequence. """
    gc_count = 0
    for s in dna_seq:
        if s in "GCgc": gc_count += 1
    return gc_count / len(dna_seq)

gc_content("atagataactcgcatag")
```

Out[4]:
```
0.35294117647058826
```

In [5]:
```python
# Ихэнх тохиолдолд, ген эсвэл экзон хайхдаа бид дарааллын өөр өөр хэсэг дэх GC-г тоолох хэрэгтэй.
# k урттай хэсгүүд дэд GC-г тоолно.
def gc_content_subseq (dna_seq, k=100):
    """ Returns GC content of non-overlapping sub-sequences of size k. """
    res = []
    for i in range(0, len(dna_seq)-k+1, k):
        subseq = dna_seq[i:i+k]
        gc = gc_content(subseq)
        res.append(gc)
    return res
```

2. Хөрвүүлэлт (Transcription) ба урвуу гүйцээлт(Transcription and Reverse Complement)

ДНХ-ийг РНХ молекул руу хөрвүүлэх: ДНХ-ийн молекулуудад агуулагдах генетик мэдээлэл нь уургийн нийлэгжих бүх процессын үндэс болдог.

In [6]:
```python
# DNA-ийн молекулууд нь урвуу чиглэлд уншигдах хоёр цуваатай.
# Хөрвүүлэлтийг үүсэх үед уг хоёр цуваа салж, шинэ RNA молекул нь нэг цуваaны гүйцээлт болж үүснэ.
# Тиймээс ДНХ-ийн нөгөө цуваатай төстэй (бид үүнийг template цуваа гэдэг), 'T'-ийн оронд 'U' байна.
def transcription (dna_seq):
    """ Function that computes the RNA corresponding to the transcription of the DNA sequence provided. """
    assert validate_dna(dna_seq), "Invalid DNA sequence"
    return dna_seq.upper().replace("T","U")

transcription("atagataactcgcatag")
```

Out[6]:
```
'AUAGAUAACUCGCAUAG'
```

In [7]:
```python
# Ерөнхийдөө биоинформатик өгөгдлийн санд зөвхөн нэг цувааг өгдөг нь тул шаардлагатай бол ДНХ нөгөөг нь тооцоолдог
def reverse_complement (dna_seq):
    """ Computes the reverse complement of the inputted DNA sequence. """
    assert validate_dna(dna_seq), "Invalid DNA sequence"
    comp = ""
    for c in dna_seq.upper():
        if c == 'A':
            comp = "T" + comp
        elif c == "T":
            comp = "A" + comp
        elif c == "G":
            comp = "C" + comp
        elif c== "C":
            comp = "G" + comp
    return comp
print('ATAGATAACTCGCATAG')
print(reverse_complement('ATAGATAACTCGCATAG'))
```

```
ATAGATAACTCGCATAG
CTATGCGAGTTATCTAT
```

3. Хувиргалт(Translation)

Messenger RNA молекулуудад агуулагдах мэдээллийн дагуу амин хүчлүүдийн гинжийн дарааллыг үүсгэн уураг нийлэгжүүлдэг.

In [8]:
```python
# Өмнөх алгоритмаас харахад DNA-ээс mRNA үүсгэх нь хялбар тул DNA-ээс шууд уураг үүсгэе.
# Түлхүүрүүд нь кодонууд (нийт 64), холбогдох утгууд нь амин хүчлүүд.
# Буруу кодоныг аргумент болгон дамжуулсан тохиолдолд функц нь None-г буцаана.
# Зогсох кодонд '_' тэмдгийг ашигласан
def translate_codon(cod):
    """Translates a codon into an aminoacid using an internal dictionary with the standard genetic code."""
    tc = {"GCT":"A", "GCC":"A", "GCA":"A", "GCG":"A",
      "TGT":"C", "TGC":"C",
      "GAT":"D", "GAC":"D",
      "GAA":"E", "GAG":"E",
      "TTT":"F", "TTC":"F",
      "GGT":"G", "GGC":"G", "GGA":"G", "GGG":"G",
      "CAT":"H", "CAC":"H",
      "ATA":"I", "ATT":"I", "ATC":"I",
      "AAA":"K", "AAG":"K",
      "TTA":"L", "TTG":"L", "CTT":"L", "CTC":"L", "CTA":"L", "CTG":"L",
      "ATG":"M", "AAT":"N", "AAC":"N",
      "CCT":"P", "CCC":"P", "CCA":"P", "CCG":"P",
      "CAA":"Q", "CAG":"Q",
      "CGT":"R", "CGC":"R", "CGA":"R", "CGG":"R", "AGA":"R", "AGG":"R",
      "TCT":"S", "TCC":"S", "TCA":"S", "TCG":"S", "AGT":"S", "AGC":"S",
      "ACT":"T", "ACC":"T", "ACA":"T", "ACG":"T",
      "GTT":"V", "GTC":"V", "GTA":"V", "GTG":"V",
      "TGG":"W",
      "TAT":"Y", "TAC":"Y",
      "TAA":"_", "TAG":"_", "TGA":"_"}
    if cod in tc: return tc[cod]
    else: return None
```

In [9]:
```python
# ДНХ-ийн дараалал болон хувиргах анхны байрлалыг оролт болгон авч, холбогдох амин хүчлийн дарааллыг буцаана.
# Дарааллыг кодон болгон хуваах нь хамгийн сүүлийн аргумент нь алхамын утгыг тодорхойлох боломжийг олгодог
# range функцийг ашиглан тохирох индексүүдийг үүсгэх замаар хийгдэнэ.
# Хуваасан кодонуудын хувиргалт нь өмнө боловсруулсан функцийг буцаах утгаар ирнэ

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

print(translate_seq('ATAGATAACTCGCATAG'))
```

```
IDNSH
```

In [10]:
```python
# Генетик код нь олон сонголттой тул давтагдсан утгууд байна,
#    Жнь: нэг амин хүчлийг өөр өөр кодоноор кодлодог.
# Үүнийг кодоны хэрэглээ (codon usage) гэдэг бөгөөд янз бүрийн генүүдийн хувьд сонирхолтой статистикийг өгдөг.
# Функцийн үр дүнд dictionary өгөгдөл буцна
#    Түлхүүр нь кодон
#    Утга нь кодон бүрийн хэрэглээний хувь.
def codon_usage(dna_seq, aa):
    """Provides the frequency of each codon encoding a given aminoacid, in a DNA sequence ."""
    assert validate_dna(dna_seq), "Invalid DNA sequence"
    seqm = dna_seq.upper()
    dic = {}
    total = 0
    for i in range(0, len(seqm)-2, 3):
        cod = seqm[i:i+3]
        if translate_codon(cod) == aa:
            if cod in dic:
                dic[cod] += 1
            else: dic[cod] = 1
            total += 1
    if total >0:
        for k in dic:
            dic[k] /= total
    return dic
```

4. Таамаг ген (Putative Genes) илрүүлэх

Ерөнхий тохиолдолд DNA дарааллын (жнь, genome sequencing project) кодлох бүсүүд хаана байгаа нь урьдчилан мэдэгддэггүй.

- Уургийн хувиргалт нь үргэлж **Methionine ("M")** амин хүчлийн кодоноор (эхлэх кодон - "ATG") эхэлдэг.
- Энэ амин хүчил нь уургийн эхлэлээс гадна бусад байрлалд ч тохиолдож болно.
- Мөн зогсох кодон олдвол хувиргалты процесс дуусдаг байх ёстой.
- Уг сэжигтэй хэсгүүдийг хайхдаа эхлээд DNA (эсвэл RNA) дараалал дээрх унших хүрээг (reading frames) тооцоолох юм.
- Унших хүрээ нь ДНХ-ийн дарааллыг дараалсан давхцаагүй гурвалсан (боломжтой кодон) болгон хуваах арга юм.

In [11]:
```python
# Өгөгдсөн дараалал нь эхний, хоёр, гурав дахь байрлалаас эхлэн унших боломжтой гурван тохиолдолтой.
# Энэ гурваас гадна урвуу гүйцээлтийнх нь гурван хүрээг тооцоолох хэрэгтэй.
# Ингэснээр геномын хэсэг дээрх уургийн байж болох бүх кодыг хайх боломжийг олгодог.
def reading_frames (dna_seq):
    """Computes the six reading frames of a DNA sequence (including the reverse complement."""
    assert validate_dna(dna_seq), "Invalid DNA sequence"
    res = []
    res.append(translate_seq(dna_seq,0))
    res.append(translate_seq(dna_seq,1))
    res.append(translate_seq(dna_seq,2))
    rc = reverse_complement(dna_seq)
    res.append(translate_seq(rc,0))
    res.append(translate_seq(rc,1))
    res.append(translate_seq(rc,2))
    return res

reading_frames('ATAGATAACTCGCATAG')
```

Out[11]:
```
['IDNSH', '_ITRI', 'R_LA_', 'LCELS', 'YASYL', 'MRVIY']
```

4.1 Уншилтын нээлттэй хүрээ хайх

Open Reading Frames-ORF хайх: Унших хүрээнүүдийг тооцоолсны дараагийн алхам бол эдгээр хүрээгээр кодлох боломжтой уураг олох явдал юм.

In [12]:
```python
# Амин хүчлийн дарааллаас бүх боломжит уургийг гаргаж авна.
# Энэ функц нь дарааллаар явагддаг бөгөөд
# 'M' илэрсэн үед таамагласан уургийг current_prot жагсаалтад үүсгэж эхлэнэ
# Зогсох тэмдэг олдвол энэ жагсаалтын бүх уураг гаралтын хувьсагчид нэмнэ
def all_proteins_rf (aa_seq):
    """Computes all posible proteins in an aminoacid sequence."""
    aa_seq = aa_seq.upper()
    current_prot = []
    proteins = []
    for aa in aa_seq:
        if aa == "_":
            if current_prot:
                for p in current_prot:
                    proteins.append(p)
                current_prot = []
        else:
            if aa == "M":
                current_prot.append("")
            for i in range(len(current_prot)):
                current_prot[i] += aa
    return proteins
```

4.2 Бүх боломжит уургийг тооцоолох

- Эхлээд унших хүрээнүүдийн хөрвүүлэлтийг тооцоолно
- Дараа нь зургаан амин хүчлийн дараалал дээр давталт хийнэ
- Өмнөх функцийг ашиглан бүх боломжит уургийг олж, үр дүнгийн жагсаалтад цуглуулна.
- Уг функцийн буцаах жагсаалт нь бодит тохиолдолд олон тооны уураг агуулж болзошгүй
- Жагсаалтыг таамагласан уургийн хэмжээгээр эрэмбэлж, хамгийн бага хэмжээтэйгээс нь эхлэх хэрэгтэй.
- Жижиг таамаг уургууд нь санамсаргүй байдлаар олдож болох ч хэдэн арван амин хүчил агуулсан уураг ингэж тохиолдох магадлал багатай юм.
- Зогсоох кодон үүсэх магадлал 3/64 буюу 5% орчим байдаг тул ойролцоогоор 20 амин хүчил тутамд зогсолт кодон үүсэх төлөвтэй байдаг

In [13]:
```python
def all_orfs (dna_seq):
    """Computes all possible proteins for all open reading frames."""
    assert validate_dna(dna_seq), "Invalid DNA sequence"
    rfs = reading_frames (dna_seq)
    res = []
    for rf in rfs:
        prots = all_proteins_rf(rf)
        for p in prots: res.append(p)
    return res
```

In [14]:
```python
def all_orfs_ord (dna_seq, minsize = 0):
    """Computes all possible proteins for all open reading frames. Returns ordered list of proteins with minimum size."""
    assert validate_dna(dna_seq), "Invalid DNA sequence"
    rfs = reading_frames (dna_seq)
    res = []
    for rf in rfs:
        prots = all_proteins_rf(rf)
        for p in prots:
            if len(p) > minsize: insert_prot_ord(p, res)
    return res

def insert_prot_ord (prot, list_prots):
    i = 0
    while i < len(list_prots) and len(prot) < len(list_prots[i]):
        i += 1
    list_prots.insert(i, prot)
```

4.3 Putting It All Together

In [15]:
```python
seq = input("Insert DNA sequence:")
if validate_dna (seq):
    print ("Valid sequence")
    print ("Transcription: ", transcription (seq))
    print("Reverse complement:", reverse_complement(seq))
    print("GC content (global):", gc_content(seq))
    print("Direct translation:" , translate_seq(seq))
    print("All proteins in ORFs (decreasing size): ", all_orfs_ord(seq))
else: print("DNA sequence is not valid")
```

```
Insert DNA sequence:ATGTGATAAGAA
Valid sequence
Transcription:  AUGUGAUAAGAA
Reverse complement: TTCTTATCACAT
GC content (global): 0.25
Direct translation: M__E
All proteins in ORFs (decreasing size):  ['M']
```

5. Биологийн дарааллын класс

Илүү зохион байгуулалттай,цаашид хэрэглэх боломжтой КЛАСС бүтэц Объект хандалгат программчлал

In [16]:
```python
class MySeq:
    """ Class for biological sequences. """

    def __init__ (self, seq, seq_type = "DNA"):
        self.seq = seq.upper()
        self.seq_type = seq_type

    def __len__(self):
        return len(self.seq)

    def __getitem__(self, n):
        return self.seq[n]

    def __getslice__(self, i, j):
        return self.seq[i:j]

    def __str__(self):
        return self.seq

    def get_seq_biotype (self):
        return self.seq_type

    def show_info_seq (self):
        print ("Sequence: " + self.seq + " biotype: " + self.seq_type)

    def alphabet (self):
        if (self.seq_type=="DNA"): return "ACGT"
        elif (self.seq_type=="RNA"): return "ACGU"
        elif (self.seq_type=="PROTEIN"): return "ACDEFGHIKLMNPQRSTVWY"
        else: return None

    def validate (self):
        alp = self.alphabet()
        res = True
        i = 0
        while i < len(self.seq) and res:
            if self.seq[i] not in alp: res = False
            else: i += 1
        return res

    def transcription (self):
        if (self.seq_type == "DNA"):
            return MySeq(self.seq.replace("T","U"), "RNA")
        else:
            return None

    def reverse_comp (self):
        if (self.seq_type != "DNA"): return None
        comp = ""
        for c in self.seq:
            if (c == 'A'): comp = "T" + comp
            elif (c == "T"): comp = "A" + comp
            elif (c == "G"): comp = "C" + comp
            elif (c== "C"): comp = "G" + comp
        return MySeq(comp, "DNA")

    def translate (self, iniPos= 0):
        if (self.seq_type != "DNA"): return None
        seq_aa = ""
        for pos in range(iniPos,len(self.seq)-2,3):
            cod = self.seq[pos:pos+3]
            seq_aa += translate_codon(cod)
        return MySeq(seq_aa, "PROTEIN")

## main

s1 = MySeq("ATGTGATAAGAATAGAATGCTGAATAAATAGAATGACAT")
s2 = MySeq("MKVVLSVQERSVVSLL", "PROTEIN")
print(s1.validate(),s2.validate())
print(s1)
s3 = s1.transcription()
s3.show_info_seq()
s4 = s1.reverse_comp().translate()
s4.show_info_seq()
```

```
True True
ATGTGATAAGAATAGAATGCTGAATAAATAGAATGACAT
Sequence: AUGUGAUAAGAAUAGAAUGCUGAAUAAAUAGAAUGACAU biotype: RNA
Sequence: MSFYLFSILFLSH biotype: PROTEIN
```

6. BioPython ашиглан дараалал боловсруулах

biopython

6.1 Дарааллын аннотоц объектууд

Bio.SeqRecord нь дараалал ба тэдгээрийн аннотацын үндсэн контейнер класс.

SeqRecord

- .seq – дараалал өөрөө буюу Seq классын объект
- .id – дарааллын дугаар
- .name – дарааллын нэр
- .description – дарааллын тайлбар
- .annotations – бүхэл дарааллын глобал тайлбарууд, dictionary төрөлтэй,
- Түлхүүр: аннотацын төрлүүд болон бүтэцлэгдээгүй талбарууд
- Утга: тухайн дарааллын талбаруудын утгууд
- .features – бүтэцлэгдсэн үзүүлэлтүүд, дараалалд эсвэл хэсгүүдэд нь хамаарах SeqFeature объектын жагсаалт;
- .letter_annotations – дараалал дахь үсэг (байрлал) бүрийн боломжит аннотацууд;
- .dbxrefs – өгөгдлийн сангийн лавлах мэдээлэл

SeqFeature

- .location – энэ аннотацад хамаарах дарааллын бүс, FeatureLocation классын объект;
- .type – тухайн үзүүлэлтийн төрлийн төлөв, тэмдэгт мөр;
- .qualifiers – тухайн үзүүлэлтийн талаар нэмэлт мэдээлэл, талбар-утга dictionary төрөлтэй.

7. Exercises

In [17]:
```python
DNA_Codons = {
    # 'M' - START, '_' - STOP
    "GCT": "A", "GCC": "A", "GCA": "A", "GCG": "A",
    "TGT": "C", "TGC": "C",
    "GAT": "D", "GAC": "D",
    "GAA": "E", "GAG": "E",
    "TTT": "F", "TTC": "F",
    "GGT": "G", "GGC": "G", "GGA": "G", "GGG": "G",
    "CAT": "H", "CAC": "H",
    "ATA": "I", "ATT": "I", "ATC": "I",
    "AAA": "K", "AAG": "K",
    "TTA": "L", "TTG": "L", "CTT": "L", "CTC": "L", "CTA": "L", "CTG": "L",
    "ATG": "M",
    "AAT": "N", "AAC": "N",
    "CCT": "P", "CCC": "P", "CCA": "P", "CCG": "P",
    "CAA": "Q", "CAG": "Q",
    "CGT": "R", "CGC": "R", "CGA": "R", "CGG": "R", "AGA": "R", "AGG": "R",
    "TCT": "S", "TCC": "S", "TCA": "S", "TCG": "S", "AGT": "S", "AGC": "S",
    "ACT": "T", "ACC": "T", "ACA": "T", "ACG": "T",
    "GTT": "V", "GTC": "V", "GTA": "V", "GTG": "V",
    "TGG": "W",
    "TAT": "Y", "TAC": "Y",
    "TAA": "_", "TAG": "_", "TGA": "_"
}
```

In [18]:
```python
class Bio_seq:
  def __init__(self, seq, seq_type = 'DNA'):
    self.seq = seq.upper()
    self.seq_type = seq_type
    self.valid = self.validate()

  # DNA дээр суурилсан эсэх ийг шалгана
  def validate(self):
    validDna = self.seq.count("A") + self.seq.count("C") + self.seq.count("G") + self.seq.count("T")
    if(validDna == len(self.seq)):
      return True
    else:
      return False

  # ДНХ-ийн дарааллыг уншиж, том үсэг болгон хувиргаж, хэдэн нуклеотид нь пурин, пиримидин болохыг тоолдог функц
  def purines_pyrimidines(self):
    purines = 0
    pyrim = 0
    for i in self.seq:
      if(i == 'A' or i == 'G'):
        purines += 1
      elif(i == 'U' or i == 'T' or i == 'C'):
        pyrim += 1
    return "Purines: " + str(purines) + " Pyrim: " + str(pyrim)

  # Кодонуудыг гаргаж авна.
  def codons(self):
    codon_lst = []
    for i in range(0, len(self.seq), 3):
      codon_lst.append(self.seq[i:i+3])
    return codon_lst

  # Дараалийг палиндром эсэхийг шалгана.
  def isPalindrome(self):
    revSeq = self.seq[::-1]
    if(self.seq == revSeq):
      return "Палиндром мөн"
    else: return "Палиндром биш"

  # Дарааллын C,G нуклеотидын хувь
  def CG_content(self):
    return round((self.seq.count('C') + self.seq.count('G')) / len(self.seq) * 100)

  # Өгөгдсөн амин хүчлийн дарааллууд нь уураг үүсгэж байгаа эсэхийг шалгадаг функц
  # Эхний давхар давталтаар 3 аар тасдаад 1 алхам хойшлоод 3 аар аваад явна
  # 2 дохь давталтаар кодонуудыг байдаг эсэхийг шалгаж тоолно
  # хэрвээ байдаг бол тоог нь үгүй бол -1 буцаана
  def checkProtein(self):
    seq = []
    count = 0
    for i in range(3):
      for j in range(i, len(self.seq), 3):
        if(len(self.seq[j:j + 3]) == 3):
          seq.append(self.seq[j:j + 3])

    for i in seq:
      if(i in DNA_Codons):
        count += 1
    print("Total codons", seq)
    if(count > 0):
      return count
    else: return -1

  # Олдсон дараалал нь уураг мөн үү биш үү гэдгийг логик оператороор тодорхойлдог функц
  def boolProtein(self):
    if(self.checkProtein != -1):
      return True
    else: return False

  # Дарааллыг амин хүчлүүдийг dictionary бүтэцээр буцаана
  def mappingDict(self):
    codon_lst = []
    dct = {}

    for i in range(0, len(seq), 3):
      codon_lst.append(seq[i:i+3])

    for s in codon_lst:
      if(s in dct):
        dct[s] += 1
      else: dct[s] = 1

    return dct

  # ДНХ дарааллыг бүрдүүлж буй амин хүчлүүдийн дэд дараалуудыг олдог функц.
  # exam: ACG: T, GTG: V, CTA: L, ACG: T, GTG: V, CTA: L = Thr, Val, Leu, Thr, Val, Leu
  def allProteins(self):
    aa = []
    seq = self.codons()
    for s in seq:
      aa.append(DNA_Codons[s])
    return aa

  # к хэмжээтэй дэд дараалал өгөгдсөн дараалалд оршиж байна уу эсэхийг шалгана.
  # Хэрвээ оршиж байвал түлхүүр нь тухайн дэд дараалал ба утга нь хэдэн удаа дэд дараалал давтагдсан тоо болох dictionary буцаана.
  def k_freq(self, seq,k = 4):
    codon_lst = []
    dct = {}

    for i in range(0, len(seq), k):
      codon_lst.append(seq[i:i+k])

    for s in codon_lst:
      if(s in dct):
        dct[s] += 1
      else: dct[s] = 1

    return dct

seq = "ACGGTGCTAACGGTGCTA"
sqq = "ACGGACGGACGGAAAAACGGACGGACGGAAAA"
dna = Bio_seq(seq)
print(dna.purines_pyrimidines())
print(dna.isPalindrome())
print("CG: " + str(dna.CG_content()) + "%")
print("Уураг үүсгэж байгаа тоо " + str(dna.checkProtein()))
print("Уураг мөн үү! " + str(dna.boolProtein()))
print("dictionary бүтцээр ", dna.mappingDict())
print("Reading Frame", dna.allProteins())
print("К хэмжээгээр амин хүчил гаргах ", dna.k_freq(sqq, 4))
```

```
Purines: 10 Pyrim: 8
Палиндром биш
CG: 56%
Total codons ['ACG', 'GTG', 'CTA', 'ACG', 'GTG', 'CTA', 'CGG', 'TGC', 'TAA', 'CGG', 'TGC', 'GGT', 'GCT', 'AAC', 'GGT', 'GCT']
Уураг үүсгэж байгаа тоо 16
Уураг мөн үү! True
dictionary бүтцээр  {'ACG': 2, 'GTG': 2, 'CTA': 2}
Reading Frame ['T', 'V', 'L', 'T', 'V', 'L']
К хэмжээгээр амин хүчил гаргах  {'ACGG': 6, 'AAAA': 2}
```

1. Write a program that reads a DNA sequence, converts it to capital letters, and counts how many nucleotides are purines and pyrimidines.

2. Write a program that reads a DNA sequence and checks if it is equal to its reverse com- plement.

3. Write and test a function that, given a DNA sequence, returns the total number of "CG" duplets contained in it.

4. Write and test a Python function that, given a DNA sequence, returns the size of the first protein that can be encoded by that sequence (in any of the three reading frames). The function should return −1 if no protein is found.

5. Write and test a function that given an aminoacid sequence returns a logic value indicat-ing if the sequence can be a protein or not.

6. Write and test a Python function that, given a DNA sequence, creates a map (dictionary) with the frequencies of the aminoacids it encodes (assuming the translation is initiated in the first position of the DNA sequence). Stop codons should be ignored.

7. Write a program that reads an aminoacid sequence and a DNA sequence and prints the list of all sub-sequences of the DNA sequence that encode the given protein sequence.

8. Write a function that, given a sequence as an argument, allows to detect if there are re-peated sub-sequences of size k (the second argument of the function). The result should be a dictionary where keys are sub-sequences and values are the number of times they oc-cur (at least 2). Use the function in a program that reads the sequence and k and prints the result by decreasing frequency.