ПРАКТИЧЕСКАЯ РАБОТА №1. БИОЛОГИЧЕСКИЕ ПОСЛЕДОВАТЕЛЬНОСТИ

Оглавление

1. БИОЛОГИЧЕСКИЕ ПОСЛЕДОВАТЕЛЬНОСТИ

Цель работы. Изучить способы решения задач с биологическими последовательностями (БП).

1.1. Описание последовательностей и кодирование белков, ДНК, РНК

Форма, свойства белка зависят от последовательности аминокислотных остатков (amino acids residues). Известны 20 типов аминокислот. Аминокислоты связаны в линейную цепочку полипептидами.

Последовательности представляются в виде строк при одно - символьной кодировке или списков (при трех - символьной кодировке).

```
MYGKIIFVLLLSEIVSISASSTTGVAMHTSTSSSVTKSYISSQTNDTHKRDTYAATPRAH
EVSEISVRTVYPPEEETGERVQLAHHFSEPEITLIIFGVMAGVIGTILLISYGIRRLIKK
SPSDVKPLPSPDTDVPLSSVEIENPETSDQ
```

Рисунок 1. Пример одно - символьной последовательности белка

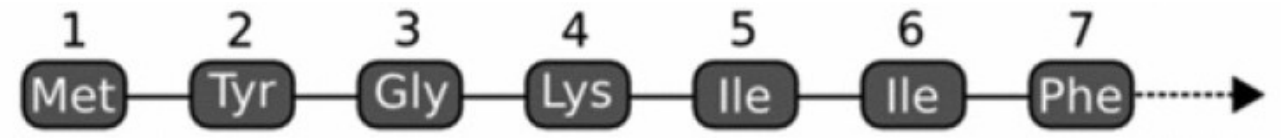

Рисунок 2. Пример трех - символьной последовательности белка

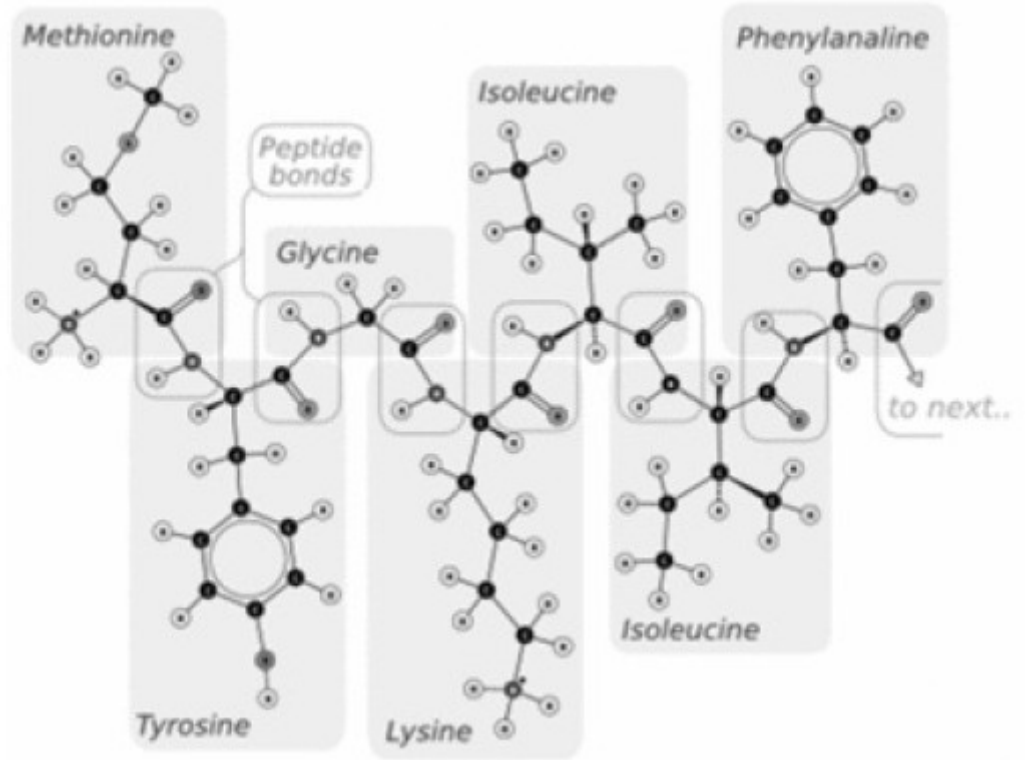

Рисунок 3. Химическая структура, которой соответствует последовательность, изображенная на рисунке 2

Также рассматриваются последовательности нуклеотидов в цепочках ДНК и РНК (DNA and RNA).

Трехмерной структурой двух - цепочечной ДНК является двойная спираль.

ДНК состоит из цепочки из четырех различных нуклеотидов. Цепь ДНК обычно представлена в виде однобуквенной последовательности, где каждая буква представляет собой тип нуклеотида.

```
ATGATCTCAGGATGTATGGAAAAATAATCTTTGTATTACTATTGTCAGGTAAGTGATTTTATTTCATCTT
GGTTCTGTTATATTGGGTATGAGATCATAGAATAAAATATGAACTACCCTATTTTAGTTCTATCTTATTT
AAATCAATAAATGAGTAGTATTTCCTCTTCCAGTCTGGTGGATGGATTTTACTGGAACTCAGCTACCAAT
GTGGGGGAAATGGCACAAGGGAGCCCAGTATTTATGGCCAAATCCAGTTTTCTAGTATGAGAAGCTTACT
TCAATTCTAAGTCTAGCTAGAATTAAAATAATTTT
```

Рисунок 4. Пример одно - символьной последовательности ДНК

ДНК - G:C, A:T пары оснований.

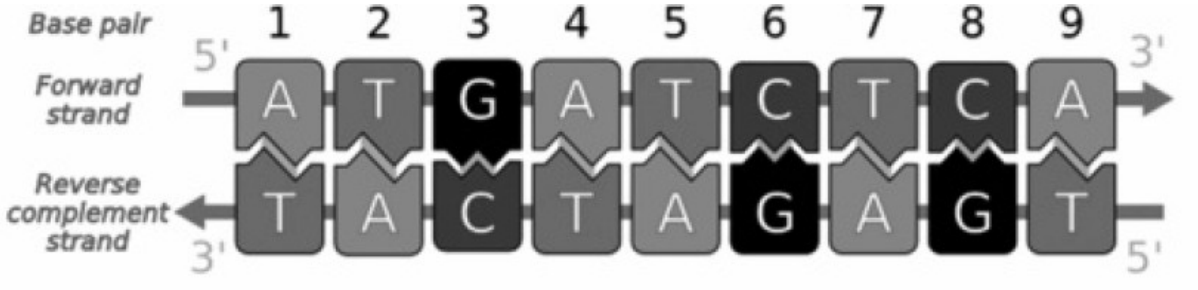

Рисунок 5. Пример двух цепей пар оснований ДНК

РНК является полимером, состоящим, так же, как и в ДНК, из четырех различных типов нуклеотидов.

Исключением является то, что T основание в ДНК заменяется аналогичным U в РНК, которая также составляет пару с A.

Кодирующая последовательность, содержащаяся в РНК молекуле, определяет процесс формирования белка, называемый **трансляцией (translation).**

Эта РНК создается из региона гена в ДНК в процессе, который называется **транскрипцией *(transcription).*** В большинстве случаев первоначальный РНК-транскрипт, копирующий часть одной из нитей (strand) ДНК, дополнительно обрабатывается в процессе, называемом ***сплайсингом (splicing),*** для удаления некодирующих интронов.

Так как существует всего 20 типов аминокислот белков и только четыре типа нуклеотидов РНК, то для кодирования каждой аминокислоты требуется группа из трех оснований, называемая кодоном.

Существует шесть возможных способов группировки нуклеотидов ДНК в кодоны. При формировании последовательности белка используются шесть рамок считывания.

Один ген использует только одну рамку чтения (reading frames), но в разных генах могут использоваться все шесть возможностей.

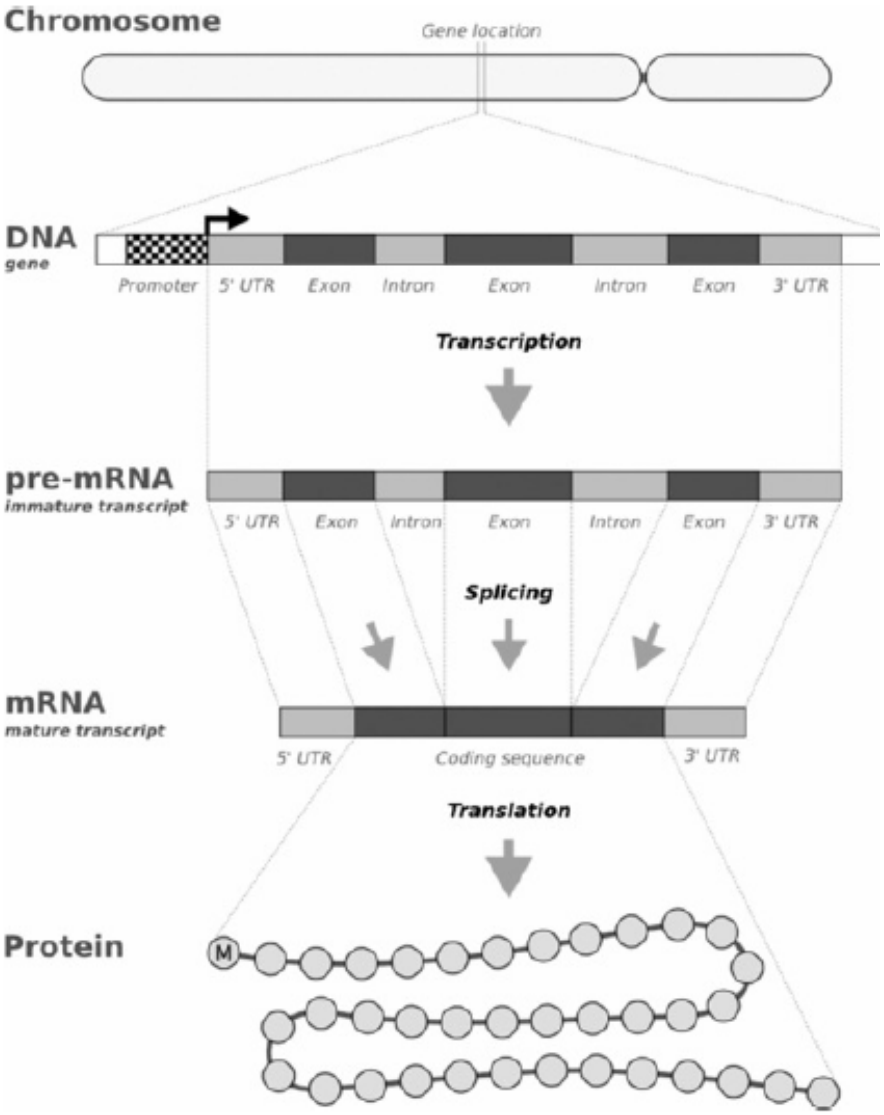

Рисунок 6. Процессы транскрипции и трансляции

Самый распространенный и простой способ для хранения последовательностей белков, ДНК и РНК – это текст. Таким образом, в одно - символьной кодировке текста для ДНК и РНК используется алфавит из четырех букв, представляющих нуклеотиды, и алфавит из 20 букв, которые обозначают аминокислоты белков. В некоторых случаях применяется трех – символьный код.

В Python, последовательность кодов однобуквенных остатков, как правило, будет представлена в виде строк, а трехбуквенные коды - в виде списков, хотя другие варианты, конечно возможны.

1.2. Трансляция

Во-первых, мы определяем словарь (dictionary), который содержит генетический код.

Для каждого кодона трехбуквенный код соответствует аминокислоте, а для стоп - кодона – используется объект None.

```
STANDARD_GENETIC_CODE = {
'UUU':'Phe', 'UUC':'Phe', 'UCU':'Ser', 'UCC':'Ser',
'UAU':'Tyr', 'UAC':'Tyr', 'UGU':'Cys', 'UGC':'Cys',
'UUA':'Leu', 'UCA':'Ser', 'UAA':None, 'UGA':None,
'UUG':'Leu', 'UCG':'Ser', 'UAG':None, 'UGG':'Trp',
'CUU':'Leu', 'CUC':'Leu', 'CCU':'Pro', 'CCC':'Pro',
'CAU':'His', 'CAC':'His', 'CGU':'Arg', 'CGC':'Arg',
'CUA':'Leu', 'CUG':'Leu', 'CCA':'Pro', 'CCG':'Pro',
'CAA':'Gln', 'CAG':'Gln', 'CGA':'Arg', 'CGG':'Arg',
'AUU':'Ile', 'AUC':'Ile', 'ACU':'Thr', 'ACC':'Thr',
'AAU':'Asn', 'AAC':'Asn', 'AGU':'Ser', 'AGC':'Ser',
'AUA':'Ile', 'ACA':'Thr', 'AAA':'Lys', 'AGA':'Arg',
'AUG':'Met', 'ACG':'Thr', 'AAG':'Lys', 'AGG':'Arg',
'GUU':'Val', 'GUC':'Val', 'GCU':'Ala', 'GCC':'Ala',
'GAU':'Asp', 'GAC':'Asp', 'GGU':'Gly', 'GGC':'Gly',
'GUA':'Val', 'GUG':'Val', 'GCA':'Ala', 'GCG':'Ala',
'GAA':'Glu', 'GAG':'Glu', 'GGA':'Gly', 'GGG':'Gly'}
```

Далее определим последовательность для целей тестирования программ. В реальных задачах, конечно, возникает необходимость получения различных последовательностей из файлов и баз данных.

```
dnaSeq='ATGGTGCATCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGGGGCA
AGGTG'
```

Учитывая заданную последовательность нуклеотидов, мы берем каждую группу из трех нуклеотидных букв и используем эту группу в качестве ключа для поиска соответствующего кода аминокислот, помня, конечно, что мы должны преобразовать любые T остатки ДНК в U остатки РНК (которые требуются для нашего Словаря генетического кода).

```
rnaSeq = dnaSeq.replace('T','U')
```

Если мы находим код аминокислоты, то добавляем его в список, который представляет собой последовательность белка.

Если мы не можем найти аминокислоту для кодона, то мы достигли стоп - кодона, после чего наша последовательность белка является полной и мы можем немедленно прекратить трансляцию.

Обратите внимание, что мы применяем трех – буквенный код под - последовательности, используя обозначение seq[i:i+3], помня, что это вырезает подстроку с позиции i, до позиции i+3, но не включая ее.

```
def proteinTranslation(seq, geneticCode):
    """ This function translates a nucleic acid sequence into a
    protein sequence, until the end or until it comes across
    a stop codon """
    seq = seq.replace('T','U') # Make sure we have RNA sequence
    proteinSeq = []
    i = 0
  while i+2 < len(seq):
    codon = seq[i:i+3]
```

```
        aminoAcid = geneticCode[codon]
        if aminoAcid is None: # Found stop codon
                break


        proteinSeq.append(aminoAcid)
        i += 3

return proteinSeq

protein3LetterSeq = proteinTranslation(dnaSeq, STANDARD_GENETIC_CODE)
print protein3LetterSeq
```

1.3. Оценка молекулярной массы

Разработаем сценарий оценивает массу ДНК, РНК или молекулы белка в единицах – дальтон (dalton).

Определим функцию, которая в качестве первого аргумента принимает последовательность, а второй – тип молекулы (последовательность белка, последовательность ДНК или РНК).

Внутри функции мы определим словарь, который хранит средние значения молекулярных масс различных видов остатков. Этот словарь содержит три внутренних subdictionaries, по одному для каждого из различных типов молекул. Мы обращаемся к правильным внутренним словарям с использованием molType в качестве ключа.

Начальное значение переменной суммарной массы равняется 18.02 (с учетом OH и H).

```
def estimateMolMass(seq, molType='protein'):
        """Calculate the molecular weight of a biological sequence assuming
          normal isotopic ratios and protonation/modification states
        """

        residueMasses = {"DNA":
                        {"G":329.21,"C":289.18,"A":323.21,"T":304.19,},
                        "RNA":
                        {"G":345.21,"C":305.18,"A":329.21,"U":302.16,},
                        "protein":
                        {"A": 71.07,"R":156.18,"N":114.08, "D":115.08,
                         "C":103.10,"Q":128.13, "E":129.11,"G": 57.05,
                         "H":137.14, "I":113.15,"L":113.15,"K":128.17,
                         "M":131.19,"F":147.17,"P": 97.11, "S": 87.07,
                         "T":101.10,"W":186.20, "Y":163.17,"V": 99.13}}

        massDict = residueMasses[molType]

        # Begin with mass of extra end atoms H + OH
        molMass = 18.02

        for letter in seq:
          molMass += massDict.get(letter, 0.0)
```

```
  return molMass
```

Определим массы молекул белка, ДНК и РНК:

```
proteinSeq = 'IRTNGTHMQPLLKLMKFQKFLLELFTLQKRKPEKGYNLPIISLNQ'
proteinMass = estimateMolMass(proteinSeq)
print (proteinMass)

dnaMass = estimateMolMass(dnaSeq, molType='DNA')
print (dnaMass)

rnaSeq = dnaSeq.replace('T','U') # Make sure we have RNA sequence
rnaMass = estimateMolMass(rnaSeq, molType='RNA')
print (rnaMass)
```

1.4. Поиск частей последовательностей (мотивов - motifs)

Скрипт предназначен для поиска суб - последовательности в пределах большей последовательности. Небольшая суб - последовательность называется - мотивом (motif). Мотивы часто имеют важные биологические функции, например, указывают на начало генов или где кодирующие области генов сращиваютсяя (spliced) (удаляются интроны) и др.

Пример того, как найти фиксированную суб - последовательность в пределах большей последовательности:

```
seq = 'AGCTCGCTCGCTGCGTATAAAATCGCATCGCGCGCAGC'
position1 = seq.find('TATAAA')
position2 = seq.find('GAGGAG')
```

Как правило, есть ряд похожих суб - последовательностей, которые выполняют ту же роль. Соответственно, далее для определения мотива в последовательности используется профиль остатков (residue profile).

Профиль остатков описывает для каждой позиции в мотиве - какой диапазон остатков найден и как часто один вид остатка присутствует по сравнению с другими.

Скрипт с использованием профайла рассматривается на примере поиска области последовательности ДНК, которая называется «ТАТА-блок». Эта последовательность используется при определении начала гена. Следует отметить, что система «ТАТА» используется не для всех генов.

Во-первых, мы определяем профиль, который кодирует предпочитаемые символы в последовательности для каждой позиции в секции ДНК.

```
profile = {
'A':[ 61, 16,352, 3,354,268,360,222,155, 56, 83, 82, 82, 68, 77],
'C':[145, 46, 0, 10, 0, 0, 3, 2, 44,135,147,127,118,107,101],
'G':[152, 18, 2, 2, 5, 0, 10, 44,157,150,128,128,128,139,140],
'T':[ 31,309, 35,374, 30,121, 6,121, 33, 48, 31, 52, 61, 75, 71]}
```

Затем мы определяем функцию, которая в качестве аргументов использует последовательность ДНК и профиль, чтобы определить секцию, которая больше соответствует профилю.

```
def matchDnaProfile(seq, profile):
    """ Find the best matching position and score when comparing a DNA
```

```
    sequence with a DNA sequence profile"""

  bestScore = 0
  bestPosition = None # Just to start with

  width = len(profile['A'])

  for i in range(len(seq)-width):
    score = 0

    for j in range(width):
      letter = seq[i+j]
      score += profile[letter][j]

    if score > bestScore:
      bestScore = score
      bestPosition = i

 return bestScore, bestPosition
```

Определим позицию мотива:

```
score, position = matchDnaProfile(dnaSeq, profile)
print(score, position, dnaSeq[position:position+15])
```

1.5. Оценка доли GC (GC content)

Содержание GC - это процент пар G: C от общего числа пар оснований (по сравнению с A: T). Измерение содержания GC в ДНК имеет биологически значение, поскольку в регионах хромосомы, которые богаты G и C, возможно наличие кодирующих последовательностей генов.

```
def calcGcContent(seq, winSize=10):

  gcValues = []

  for i in range(len(seq)-winSize):

    subSeq = seq[i:i+winSize]

    numGc = subSeq.count('G') + subSeq.count('C')

    value = numGc/float(winSize)

    gcValues.append(value)

 return gcValues
```

Для визуализации результатов используется библиотека Matplotlib.

```
from matplotlib import pyplot
dnaSeq='ATGGTGCATCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGGGGCAA
GGTG'
```

```
    gcResults = calcGcContent(dnaSeq)
    print (gcResults)
    pyplot.plot(gcResults)
pyplot.show()
```

1.6. Измерение повторяемости и сложности последовательности (sequence complexity)

Часто бывает полезно оценить, как повторяются сегменты в последовательности ДНК или белка. Часто повторяющиеся регионы последовательности ДНК связаны с не - кодированием (non-coding), и, особенно, «мусорной»(‘junk’) ДНК. Следовательно, если мы ищем гены и их регуляторные области, то необходимо найти суб - последовательность, которая не повторяется.

Для белка - повторяющиеся аминокислотные последовательности связаны с частями, которые не структурированы.

Для этого примера мы будем измерять повторяемость вдоль последовательности, считая, как много различных видов остатка присутствуют в заданном окне поиска. Если в данной суб - последовательности используется только ограниченное количество типов остатков, а не разнообразной их смеси, то мы делаем вывод, что последовательность является повторяющейся или имеет малую сложность последовательности.

Для более формального математического определения следует обратиться к теории информации и использовать меру информационной энтропии Шеннона.

```
def calcRelativeEntropy(seq, resCodes):
    """Calculate a relative entropy value for the residues in a
      sequence compared to a null hypothesis where each residue
      appears in a randomly and unbiased.
    """

    from math import log

    N = float(len(seq))

    base = 1.0/len(resCodes)

    prop = {}
    for r in resCodes:
      prop[r] = 0

    for r in seq:
      prop[r] += 1
```

```
        for r in resCodes:
          prop[r] /= N

        DKL = 0
        for r in resCodes:
          if prop[r] != 0.0:
            d = prop[r]* log(prop[r]/base, 2.0)
            DKL += d

  return DKL


def relativeEntropySearch(seq, winSize, isProtein=False):
        """Scan a sequence for repetitiveness by calculating relative
          information entropy.
        """

        lenSeq = len(seq)
        scores = [0.0] * lenSeq

        extraSeq = seq[:winSize]
        seq += extraSeq

        if isProtein:
          resCodes = 'ACDEFGHIKLMNPQRSTVWY'
        else:
          resCodes = 'GCAT'

        for i in range(lenSeq):
          subSeq = seq[i:i+winSize]
          scores[i] = calcRelativeEntropy(subSeq, resCodes)

  return scores
```

Оценка энтропии для последовательности ДНК:

```
from matplotlib import pyplot
dnaScores = relativeEntropySearch(dnaSeq, 6)
proteinScores = relativeEntropySearch(proteinSeq, 10, isProtein=True)
pyplot.plot(dnaScores)
pyplot.plot(proteinScores)
pyplot.show()
```

1.7. Извлечение последовательностей из файлов в формате FASTA

Чтобы прочитать файл в FASTA-формате с помощью BioPython мы используем модуль SeqIO.

```
from Bio import SeqIO
fileObj = open("demoSequences.fasta", "rU")

for protein in SeqIO.parse(fileObj, 'fasta'):
        print(protein.id)
        print(protein.seq)

fileObj.close()
```

Для записи одно – символьной последовательности используется скрипт:

```
from Bio.SeqRecord import SeqRecord
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC

fileObj = open("output.fasta", "w")
proteinSeq='MFADRWLFSTNHKDIGTLYLLFGAWAGVLGTALSLLIRAELGQPGNL
LGNDHIYNVIVTAHAFVMIFFMVMPIMIGGFGNWLVPLMIGAPDMAFPRMNNMSF
WLLPPSLLLLLASAMVEAGAGTGWTVYPPLAGNYSHPGASVDLTIFSLHLAGVSSIL
GAINFITTIINMKPPAMTQYQTPLFVWSVLITAVLLLLSLPVLAAGITMLLTDRNLNTT
FFDPAGGGDPILYQHLFWFFGHPEVYILILPGFGMISHIVTYYSGKKEPFGYMGMV
WAMMSIGFLGFIVWAHHMFTVGMDVDTRAYFTSATMIIAIPTGVKVFSWLATLHGS
NMKWSAAVLWALGFIFLFTVGGLTGIVLANSSLDIVLHDTYYVVAHFHYVLSMGAV
FAIMGGFIHWFPLFSGYTLDQTYAKIHFTIMFIGVNLTFFPQHFLGLSGMPRRYSDY
PDAYTTWNILSSVGSFISLTAVMLMIFMIWEAFASKRKVLMVEEPSMNLEWLYGCPP
PYHTFEEPVYMKS'
seqObj = Seq(proteinSeq, IUPAC.protein)
proteinObj = SeqRecord(seqObj, id="TEST")

SeqIO.write([proteinObj,], fileObj, 'fasta')

fileObj.close()
```

1.8Содержание отчета

В отчет включаются программы и алгоритмы, описанные во всех разделах темы и результаты их выполнения на тестовых данных по каждому выполненному пункту.