Приложение 1.

Использование библиотек Python для запуска команды командной строки Linux и работе с фалами.

```
import subprocess
from Bio import SeqIO
```

Запуск программы Muscle.

```
def muscle_msa(name_in, name_out = 'msa.afa'):
    process = subprocess.run(['muscle', '-in', name_in, '-out', name_out],
capture_output=True, text = True)
    return process
```

Из формата txt в FASTA

```
def txt_into_fasta(name_txt, name_fasta):
    f = open(name_txt, 'r')
    string = '>i1 seq;\n' + f.readline()
    f.close()

    f1 = open(name_fasta, 'w')
    f1.write(string)
    f1.close()
```

Из формата FASTA в Stockholm

```
def fasta_into_stockholm(name_in = 'example.fa', name_out = 'example.sto'):
    records = SeqIO.parse(name_in, 'fasta')
    count = SeqIO.write(records, name_out, 'stockholm')
```

Запуск поиска подпоследовательностей исходной строки, наиболее схожих с набором подпоследовательностей.

Принимает*: name_set_msa* — имя файла с подпоследовательностями, *name_seq* — имя файла с исходной последовательностью, *name_out_inf* — имя файла для информации о работе программы, *name_out_posision* — имя файла для информации о найденных позициях, *E*_val — значение E-value, по умолчанию равно 0,01, что означает, что в

среднем на каждые 100 поисковых запросов с различными последовательностями запросов ожидается примерно 1 ложноположительный результат.

Возвращает: *name_out_posision* — имя файла, в который записана информация о найденных позициях

```
def nhmmer_on_stock_msa(name_set_msa = 'msa_sto.sto', name_seq = 'seq.fa',
name_out_inf = 'result_hmm_info.fa', name_out_posision = 'posision.fa', E_val =
0.001):
    command = 'nhmmer -o ' + name_out_inf + ' --aliscoresout ' + name_out_posision +
' --noali --notextw --singlemx --dna --incE ' + str(E_val) + ' ' + name_set_msa + ' '
+ name_seq
    process = subprocess.run(command, shell = True, capture_output = True, text =
True)
    return name_out_posision
```

Получение набора подпоследовательностей из выходного файла работы nHMMER.

Принимает: *name_seq* — имя файла с исходной последовательностью, *name_out_posision* — имя файла для информации о найденных позициях, *N* — количество подпоследовательностей, которые пойдут в новый набор, *left_step* — величина добавленного смещения слева, *right_step* — величина добавленного смещения справа, *name_position_subseq step* — названия файла для записи полученного набора

```
def creating_new_set_from_out_nhmmer(name_seq, name_out_posision, N = 20, left_step =
0, right_step = 0, name_position_subseq = 'position_subseq.fa'):
    #получение исходной последовательности из предоставленного файла
    s = string_from_file(name_seq)

    #чтение файла с найденными позициями подпоследовательностей
    f = open(name_out_posision, 'r')
    all_string = f.readlines()
    f.close()
    len_set = len(all_string)

    new_set = []

    #создание файла с итоговыми подпоследовательностями
    f = open(name_position_subseq, 'w')
    f.write('')
    f.close()
    for i in range(len_set):
        #получение номера начального и конечного символа подпоследовательности
```

```
        string = all_string[i]
        index_sep = string.rfind(':') - 1
        prev_space = string.rfind(' ', 0, index_sep - 1)
        try:
            end = int(string[prev_space + 1:index_sep])
        except:
            print("string:", string)
            print("index_sep:", index_sep)
            print("prev_space:", prev_space)
            print("name_file:",name_out_posision)
            print("i:", i)
            x1 = 1 / 0
        prev_space2 = string.rfind(' ', 0, prev_space - 1)
        begin = int(string[prev_space2 + 1:prev_space])

        #изменение номера начального и конечного символа с учётом входящего смещения
        if i > (N - 1):
            break
        if (begin - left_step) <= 0:
            begin = 0
        else:
            begin = begin - left_step

        if (end + right_step) >= (len(s) - 1):
            end = len(s) - 1
        else:
            end = end + right_step

        #добавление найденной подпоследовательности в получаемый набор new_set
        if end - begin > 0:
            f = open(name_position_subseq, 'a')
            f.write('>' + str(begin) + ' ' + str(end) + ':\n' + str(s[begin:end]) +
'\n')
            f.close()
            new_set.append(s[begin:end])
    return new_set
```

Получение файла с набором подпоследовательностей в формате FASTA из выходного файла работы nHMMER.

Принимает: *name_seq* — имя файла с исходной последовательностью, *name_out_posision* — имя файла для информации о найденных позициях В FASTA, *N* — количество подпоследовательностей, которые пойдут в новый набор, *left_step* — величина добавленного смещения слева, *right_step* — величина добавленного смещения справа.

```
def creating_fasta_new_nhmmer_set(name_seq, name_out_posision, N = 20, name_fasta_set
= 'example.fa', left_step = 0, right_step = 0):
    #получение набора подпоследовательностей
```

```
    set_of_subseq = creating_new_set_from_out_nhmmer(name_seq, name_out_posision, N,
left_step = left_step, right_step = right_step)

    #запись в файл набора подпоследовательностей
    string_set_of_subset = ''
    for i in range(len(set_of_subseq)):
        string_set_of_subset += '>i' + str(i) + ' iteration;\n' +
str(set_of_subseq[i]) + '\n'
    file = open(name_fasta_set,'w')
    file.write(string_set_of_subset)
    file.close()
```

Получение результирующего файла с набором подпоследовательностей в формате txt из выходного файла работы nHMMER с заданным диапазоном длин и возвращающих их количество.

Принимает: *name_seq* — имя файла с исходной последовательностью, *name_out_posision* — имя файла для информации о найденных позициях В FASTA, *N* — максимальное количество подпоследовательностей, которые пойдут в новый набор, *left_step* — величина добавленного смещения слева, *right_step* — величина добавленного смещения справа, *min_len_subseq* — минимальная удовлетворяющая величина, *max_len_subseq* — максимальная удовлетворяющая величина.

Возвращает: *number_subseq* — число удовлетворяющих подпоследовательностей.

```
def creating_txt_res_nhmmer_set(name_seq, name_out_posision, min_len_subseq = 300,
max_len_subseq = 500, N = 10 ** 6, name_res_txt = 'example.fa', left_step = 0,
right_step = 0):
    #получение набора подпоследовательностей
    set_of_subseq = creating_new_set_from_out_nhmmer(name_seq, name_out_posision, N,
left_step = left_step, right_step = right_step)

    #запись в файл набора подпоследовательностей
    string_set_of_subset = ''
    number_subseq = 0
    for i in range(len(set_of_subseq)):
        if (len(set_of_subseq[i]) >= min_len_subseq) and (len(set_of_subseq[i]) <=
max_len_subseq):
            string_set_of_subset +=  str(i) + ': ' + str(len(str(set_of_subseq[i])))
+ '\n' + str(set_of_subseq[i]) + '\n'
            number_subseq += 1
    file = open(name_res_txt,'w')
    file.write(string_set_of_subset)
    file.close()
    return number_subseq
```

Реализация работы с аргументами командной строки.

Параметр dest определяет название переменной, куда будет положено значение, required определяет необходимость наличия данного флага, help – информация о параметрах, type – тип переменной, default – значение по умолчанию, при отсутствии флага.

```
parser = argparse.ArgumentParser(description="Ping script")

    parser.add_argument('-f', '--filename', dest='s_name', required=True, help='Name
of file with sequence')
    parser.add_argument('--ml', '--mean_len', dest='mean_len', default=400, type=int,
required=False, help='Mean len of repeat sequence')
    parser.add_argument('--ns', '--number_subseq', dest='number_subseq', default=100,
type=int, required=False, help='Number of different search')
    parser.add_argument('-N', '--number_diff_subseq', dest='N', default=3, type=int,
required=False, help='Number of different initial subsequences')
    parser.add_argument('--rp', '--repeat_subseq', dest='repeat_subseq', default=6,
type=int, required=False, help='Number of repeat initial subsequences')
    parser.add_argument('-l', '--len_subseq', dest='len_subseq', default=410,
type=int, required=False,  help='Len initial subsequence')
    parser.add_argument('-E', '--E_value', dest='E', default=0.01, type=float,
required=False,  help='E-value')
    parser.add_argument('--ni', '--number_iteration', dest='number_iteration',
default=10, type=int, required=False,  help='Number of iteration in one proccess')
    parser.add_argument('--N_search', dest='N_search', default=40, type=int,
required=False,  help='Number of selected sequences')
    parser.add_argument('--increase_len', dest='increase_len', default=True,
type=bool, required=False,  help='Increase len, bool')
    parser.add_argument('--nf', '--name_folder', dest='name_folder_root',
default='SDR', type=str, required=False,  help='Name folder for files')
    parser.add_argument('-o', '--name_file_out', dest='name_file_out', default='-',
type=str, required=False,  help='Name file for result')

    args = parser.parse_args()
```

Запуск основного алгоритма поиска при его распараллеливании и аналогичной процедуры увеличения длин.

```
    #запуск таймера времни
    time_start = time.time()

    #создание общей папки
    name_folder_root = args.name_folder_root

    if not os.path.isdir(name_folder_root):
        os.mkdir(name_folder_root)
```

```
    #создание общей папки, завязанной на время запуска
    name_folder = os.path.join(name_folder_root, str(int(time_start)))

    if not os.path.isdir(name_folder):
        os.mkdir(name_folder)

    #функция, необходимая для запуска основной программы по средствам Pool.map,
принимающей функцию с 1 аргументом
    #запуск основной программы с переданными начальными аргументами
    def iteration_run(x):
        return mult_iteration(x, N = args.N, repeat_set = args.repeat_subseq,
len_subseq = args.len_subseq,
                              E_val = args.E, number_set_of_subseq =
args.number_subseq,
                              s_name_txt_orig = args.s_name,
                              name_folder_root = name_folder, N1 = args.N_search,
number_of_iteration = args.number_iteration)

    #функция, необходимая для запуска функции увеличения длин по средствам Pool.map,
принимающей функцию с 1 аргументом
    def iteration_run_increase(x):
        return increasing_best(x, mean_predict_len_subseq = args.mean_len, working =
args.increase_len)

    #создание процессов по количеству доступных ядер и запуск основной программы
    with Pool(processes=cpu_count()) as pool:
        values = [j + 1 for j in range(args.number_subseq)]
        #results_folder - названия папок, созданных для отдельных процессов
        results_folder = pool.map(iteration_run, values)

    #отсчет времени работы основной программ поиска
    time_finish_search = time.time()

    #создание массива с названиями папок без повторений
    results_folder_set = set(results_folder)
    results_folder_diff = []
    for res_folder in results_folder_set:
        results_folder_diff.append(res_folder)

    #запуск по процессам функций увеличения длин подпоследовательностей
    with Pool(processes=cpu_count()) as pool2:
        results_N = pool2.map(iteration_run_increase, results_folder_diff)

    #получение максимального количества найденных подпоследовательностей и
определение процесса, в котором он найден
    max_N = max(results_N)
    max_N_folder = results_folder_diff[results_N.index(max_N)]

    #получение времени работы алгоритма поиска и алгоритма увеличения длин
    time_finish = time.time()
    time_res_seach = time_finish_search - time_start
    time_res_increase = time_finish - time_finish_search

    #print('N:', *results_N)
```

```
    #запись в файл или в основной поток вывода результата о времени, количества
найденных подпоследовательностей и названия папки, где лежит ответ
    s = 'N = {}, max_N_folder = {}, time_search = {}, time_increase =
{}'.format(max_N, max_N_folder, round(time_res_seach / 60, 2),
 round(time_res_increase / 60, 2))
    if args.name_file_out == '-':
        print(s)
    else:
        f = open(args.name_file_out)
        f.write(s)
        f.close()
```

Реализация основного алгоритма поиска повторов

```
def mult_iteration(j, N = 3, repeat_set = 6, len_subseq = 410, E_val = 1,
number_set_of_subseq = 8,
                   s_name_txt_orig = 'iter4/example_new.txt', name_folder_root =
'SDR1/1/', N1 = 40, number_of_iteration = 10):
    #j - номер начального набора
    #N - количество различных подпоследовательностей в начальном наборе
    #repeat_set - количество повторов подпоследовательностей в начальном наборе
    #len_subseq - длина подпоследовательностей в начальном наборе
    #E-val - E-value
    #number_set_of_subseq - количество начальных наборов
    #s_name_txt_orig - название файла с исходной последовательностью
    #name_folder_root - название папки, где сохранять результат
    #N1 - количество подпоследовательностей, которые нужно брать на шаге улучшения
    #number_of_iteration - количество итераций улучшения

    #создание папок под процесс
    name_process = os.getpid()

    name_folder = os.path.join(name_folder_root, str(name_process))

    if not os.path.isdir(name_folder):
        os.mkdir(name_folder)

    #создание исходной подпоследовательности в папке
    s_local_txt = 's.txt'
    s_name_txt = os.path.join(name_folder, s_local_txt)
    if not os.path.exists(s_name_txt):
        shutil.copy(s_name_txt_orig, s_name_txt)

    #создание исходной подпоследовательности в папке в формате FASTA
    s_local_fa = 's.fa'
    s_name_fasta = os.path.join(name_folder, s_local_fa)
    if not os.path.exists(s_name_fasta):
        sq.txt_into_fasta(s_name_txt, s_name_fasta)

    #создание файла для записи лучшего по процессу результата
    best_res_txt_local = 'best_result.txt'
```

```
    best_res_txt = os.path.join(name_folder, best_res_txt_local)
    if not os.path.exists(best_res_txt):
        f = open(best_res_txt, 'w')
        f.write('')
        f.close()

    #создание файла для записи позиций лучшего по процессу результата
    best_nhmm_posision_local = 'best_posision.fa'
    best_nhmm_posision = os.path.join(name_folder, best_nhmm_posision_local)
    if not os.path.exists(best_nhmm_posision):
        f = open(best_nhmm_posision, 'w')
        f.write('')
        f.close()

    #создание файла для записи текущего набора подпоследовательностей
    name_set_subseq = os.path.join(name_folder, 'set_subseq.fa')
    #создание файла для записи выравненного текущего набора подпоследовательностей
    name_set_subseq_msa = os.path.join(name_folder, 'set_subseq_msa.fa')
    #создание файла для записи текущего набора подпоследовательностей в формате
Stockholm
    name_set_subseq_msa_stockh = os.path.join(name_folder, 'set_subseq_msa.sto')
    #создание файла для записи текущей информации о работе nHMMER
    name_result_nhmmer_info = os.path.join(name_folder, 'result_nhmmer_info.fa')
    #создание файла для записи текущей информации о позициях, найденных nHMMER
    name_result_nhmmer_posision = os.path.join(name_folder,
'result_nhmmer_posision.fa')
    #создание файла для записи текущих найденных подпоследовательностей в формате txt
    name_set_res = os.path.join(name_folder, 'resulst.txt')

    #создание начального набора в формате FASTA
    sq.creating_fasta_set(s_name_txt, N, len_subseq, repeat_set, name_set_subseq)

    #запись количества лучших по потоку и текущих найденных наборов
    number_of_subseq_i = 0
    number_of_subseq_i_best = 0

    #запуск итерационной процедуры улучшения
    for i in range(number_of_iteration):
        #получение множественного выравнивания набора
        sq.muscle_msa(name_set_subseq, name_set_subseq_msa)
        #перевод его в формат Stockholm
        sq.fasta_into_stockholm(name_set_subseq_msa, name_set_subseq_msa_stockh)

        #запуск поиска наиболее схожих с текущем набором подпоследовательностей
исходной строки
        sq.nhmmer_on_stock_msa(name_set_subseq_msa_stockh, s_name_fasta,
name_result_nhmmer_info, name_result_nhmmer_posision, 20)
        #создание нового набора в формате FASTA
        sq.creating_fasta_new_nhmmer_set(s_name_txt, name_result_nhmmer_posision, N1,
name_set_subseq)
        #получение количества найденных подпоследовательностей
        number_of_subseq_i = sq.creating_txt_res_nhmmer_set(s_name_txt,
name_result_nhmmer_posision, min_len_subseq = 0, max_len_subseq = 1000, N =  10 ** 6,
name_res_txt = name_set_res)
```

```
        #критерий прекращения итерации улучшения
        if number_of_subseq_i <= N:
            break

    #получение лучшего числа найденных по потоку подпоследовательностей
    number_of_subseq_i_best = sq.creating_txt_res_nhmmer_set(s_name_txt,
best_nhmm_posision, min_len_subseq = 0, max_len_subseq = 1000, N =  10 ** 6,
name_res_txt = best_res_txt)

    #измениние лучшего по потоку значения в случае большем найденном количестве
подпоследовательностей
    if number_of_subseq_i > number_of_subseq_i_best:
        #изменение лучшего набора
        f = open(name_set_res, 'r')
        l = f.read()
        f.close()

        f = open(best_res_txt, 'w')
        f.write(l)
        f.close()

        #изменение лучших позиций
        f = open(name_result_nhmmer_posision, 'r')
        l = f.read()
        f.close()

        f = open(best_nhmm_posision, 'w')
        f.write(l)
        f.close()

    #возвращение названия папки по потоку
    return name_folder
```

Некоторые используемые функции в функции поиска:

- Создание начального набора

Запись в файл

```
def creating_fasta_set(name_S, N, len_subsequence, repeat_set = 1, name_fasta_set =
'example.fa'):
    #получение исходной последовательности
    S = string_from_file(name_S)

    #получение начального набора
    set_of_subseq = string_split(S, N, len_subsequence) * repeat_set

    #запись в файл в формате FASTA
    string_set_of_subset = ''
    for i in range(len(set_of_subseq)):
```

```
        string_set_of_subset +=  '>i' + str(i) + ' iteration;\n'
+str(set_of_subseq[i]) + '\n'
    file = open(name_fasta_set,'w')
    file.write(string_set_of_subset)
    file.close()
```

Создание массива подпоследовательностей

```
def string_split(S, N, len_subsequence):
    #получение номеров начал подпоследовательностей
    beginnings = number_split(len(S), N, len_subsequence)

    #создание массива подпоследовательностей
    set_of_subseq = [S[i:i + len_subsequence] for i in beginnings]
    return set_of_subseq
```

Создание начальных положений подпоследовательностей

```
def number_split(len_s, N, len_subsequence, dist = 0):
    #dist - минимальное расстояние между подпоследовательностями

    #создание массива для записи позиций начал
    beginnings = [0] * N

    for i in range(N):
        #генерация случайного числа таким образом, чтобы подпоследовательность влезла
в последовательность
        begin_i = random.randint(0, len_s - len_subsequence - 1)
        j = 0
        while j < i:
            #добавляем подпоследовательнсть в случае, когда она не пересекается ни с
одной из предыдущих
            if abs(beginnings[j] - begin_i) < (len_subsequence + dist):
                j = 0
                begin_i = random.randint(0, len_s - len_subsequence - 1)
            else:
                j += 1
        beginnings[i] = begin_i
    #возвращаем массив начальных позиций
    return sorted(beginnings)
```

- Создание нового набора по полученному из nHMMER файла с позициями в формате FASTA

Запись в файл

```
def creating_fasta_new_nhmmer_set(name_seq, name_out_posision, N = 20, name_fasta_set
= 'example.fa', left_step = 0, right_step = 0):
    #функция создание нового набора подпоследовательностей в формате FASTA
    #создается по файлу, созданному nHMMER
    #left_step и right_step - возможные расширения подпоследовательностей

    #создание массива подпоследовательностей
    set_of_subseq = creating_new_set_from_out_nhmmer(name_seq, name_out_posision, N,
left_step = left_step, right_step = right_step)

    #запись в файл
    string_set_of_subset = ''
    for i in range(len(set_of_subseq)):
        string_set_of_subset += '>i' + str(i) + ' iteration;\n' +
str(set_of_subseq[i]) + '\n'
    file = open(name_fasta_set,'w')
    file.write(string_set_of_subset)
    file.close()
```

Получение массива подпоследовательностей см. выше.

Операции по увеличению длин подпоследовательностей

```
def increasing_best(name_folder, mean_predict_len_subseq = 400, working = True):
    #применяет операцию увеличения длин до определенного среднего для лучшего по
потоку значения

    #получение файла с исходной последовательностью
    s_local_txt = 's.txt'
    s_name_txt = os.path.join(name_folder, s_local_txt)
    if not os.path.exists(s_name_txt):
        f = open(s_name_txt, 'w')
        f.write('')
        f.close()

    #получение файла с лучшими позициями по потоку
    best_nhmm_posision_local = 'best_posision.fa'
    best_nhmm_posision = os.path.join(name_folder, best_nhmm_posision_local)
    if not os.path.exists(best_nhmm_posision):
        f = open(best_nhmm_posision, 'w')
        f.write('')
        f.close()

    #получение файла с лучшим результатом по потоку
    best_res_txt_local = 'best_result.txt'
    best_res_txt = os.path.join(name_folder, best_res_txt_local)
    if not os.path.exists(best_res_txt):
```

```
        f = open(best_res_txt, 'w')
        f.write('')
        f.close()

    #создание файля для новых лучший позиций
    best_nhmm_posision_local_new = 'best_posision_new.fa'
    best_nhmm_posision_new = os.path.join(name_folder, best_nhmm_posision_local_new)
    if not os.path.exists(best_nhmm_posision_new):
        f = open(best_nhmm_posision_new, 'w')
        f.write('')
        f.close()

    #получение лучшего количества найденных повторов по потоку
    number_of_subseq =  sq.creating_txt_res_nhmmer_set(s_name_txt,
best_nhmm_posision, min_len_subseq = 0,
                                                       max_len_subseq = 1000, N =  10
** 6, name_res_txt = os.path.join(name_folder, 'best_without_increase_result.txt'))
    #запуск итерации улучшения
    if number_of_subseq >= 6 and working:
        number_of_subseq = sq.increase(s_name_txt, best_nhmm_posision,
                name_result_nhmmer_posision_new=best_nhmm_posision_new,
                name_folder=name_folder,
                best_name_res_txt = os.path.join(name_folder,
'best_with_increase_result.txt'),
                mean_predict=mean_predict_len_subseq, N=70, diff_variants=5)

    #возвращает новое улучшенное количество значений
    return number_of_subseq
```

Процедура улучшения

```
def increase(s_name_txt, name_result_nhmmer_posision,
             name_result_nhmmer_posision_new = 'increase2/name_result_pos.fa',
             name_folder = 'increase/',
             best_name_res_txt = 'best_result_posision_increase.fa',
             mean_predict = 400, N = 150, diff_variants = 5):

    #производит процедуру увеличения длин по файлу с позициями
    #mean_predict - средняя ожидаемая длина
    #N - количество подпоследовательностей для шага итерации
    #diff_variants - количество вариантов движений окна с позициями

    #создание папки для вспомогательных файлов
    name_folder = os.path.join(name_folder, 'increase')
    if not os.path.isdir(name_folder):
        os.mkdir(name_folder)

    #получение файла с подпоследовательностями
    creating_txt_res_nhmmer_set(s_name_txt, name_result_nhmmer_posision,
min_len_subseq = 0, max_len_subseq = 1000, N =  10 ** 6, name_res_txt =
os.path.join(name_folder, 'first_subseq.txt'))
    f = open(os.path.join(name_folder, 'first_subseq.txt'), 'r')
    set_subseq = f.read()
```

```
    f.close()

    #получение максимальной и минимальной длины подпоследовательности в наборе
    min_x = 10000
    max_x = 0
    for i in range(1, min(len(set_subseq), N), 2):
        if min_x > len(set_subseq[i]):
            min_x = len(set_subseq[i])
        if max_x < len(set_subseq[i]):
            max_x = len(set_subseq[i])

    #получение общего количества символов для увеличения
    diff = mean_predict - max_x

    #создание массива для результирующих длин в зависимости от положения окна
    res_number_all = []
    max_res_number = 0
    max_res_number_index = 0


    for i, left_diff in enumerate([j for j in range(0, diff + 1, diff //
diff_variants)]):
        #запуск увеличения по методу 2 для разных позиций окна
        name_folder_increase_i = os.path.join(name_folder, 'increase_' + str(i))
        if not os.path.isdir(name_folder_increase_i):
            os.mkdir(name_folder_increase_i)
        name_result_nhmmer_posision_new_i = os.path.join(name_folder, str(i) +
'_result_pos.fa')
        res_number = increase_len_subseq_msa(s_name_txt,
 name_result_nhmmer_posision,
                                             name_result_nhmmer_posision_new =
name_result_nhmmer_posision_new_i,
                                             left_step = left_diff, right_step = diff
- left_diff,
                                             name_folder = name_folder_increase_i,
                                             min_len_subseq = 20, max_len_subseq =
1000, N = N)
        res_number_all.append(res_number)
        #поиск лучшего положения окна
        if res_number >= max_res_number:
            max_res_number_index = i
            max_res_number = res_number
    #print(res_number_all)

    f = open(os.path.join(name_folder, str(max_res_number_index) + '_result_pos.fa'),
'r')
    res = f.read()
    f.close()

    #запись новых лучший позиций
    f = open(name_result_nhmmer_posision_new, 'w')
    f.write(res)
    f.close()

    #запись нового лучшего набора подпоследовательностей
```

```
    number_of_subseq = creating_txt_res_nhmmer_set(s_name_txt,
name_result_nhmmer_posision_new, min_len_subseq = 0, max_len_subseq = 1000, N =  10
** 6, name_res_txt = best_name_res_txt)

    #возвращает лучшее количество найденных повторов
    return number_of_subseq
```

Процедура улучшения по методу 2

```
def increase_len_subseq_msa(s_name_txt, name_result_nhmmer_posision,
                            name_result_nhmmer_posision_new =
'increase2/name_result_pos.fa',
                            name_folder = 'increase/',
                            left_step = 0, right_step = 0,
                             min_len_subseq = 20, max_len_subseq = 1000, N = 150):
    #создание локальных файлов
    name_set_subseq0 = os.path.join(name_folder, 'name_old_set_fasta.fa')
    name_set_subseq0_msa = os.path.join(name_folder, 'name_old_set_fasta_msa.fa')
    name_set_subseq0_msa_stockh = os.path.join(name_folder,
'name_old_set_fasta_msa.sto')
    s_name_fasta = os.path.join(name_folder,'s_name.fa')
    name_result_nhmmer_info = os.path.join(name_folder, 'name_result_nhmmer_info.fa')
    name_position_subseq = os.path.join(name_folder, 'pos_sub2.fa')
    name_position_subseq_msa = os.path.join(name_folder, 'pos_sub2_msa.fa')

    txt_into_fasta(s_name_txt, s_name_fasta)

    set_subseq = creating_new_set_from_out_nhmmer(s_name_txt,
name_result_nhmmer_posision, N = N, left_step = 0, right_step = 0,
name_position_subseq = name_position_subseq)

    #создание выравнивания по позициям
    muscle_msa(name_position_subseq, name_position_subseq_msa)
    f = open(name_position_subseq_msa, 'r')
    msa_pos = f.read()
    f.close()

    s = string_from_file(s_name_txt)

    #создание нового массива подпоследовательностей с заданным расширением
    set_of_subseq = []
    i = 0
    while i != -1:
        g = msa_pos.find(':', i + 1)
        begin, end = map(int, msa_pos[i + 1:g].split())

        i = msa_pos.find('>', i + 1)
```

```
        seq = msa_pos[g + 2:i - 1]
        seq = seq.replace('\n', '')

        j_left = 0
        k = seq[0]
        while k == '-':
            j_left += 1
            k = seq[j_left]
        begin -= j_left

        j_right = -1
        k = seq[-1]
        while k == '-':
            j_right -= 1
            k = seq[j_right]
        end -= j_right + 1

        #добавление смещения слева и справа
        if (begin - left_step) <= 0:
            begin = 0
        else:
            begin = begin - left_step

        if (end + right_step) >= (len(s) - 1):
            end = len(s) - 1
        else:
            end = end + right_step
        set_of_subseq.append(s[begin:end])

    #запись в новый набормв формате FASTA
    string_set_of_subset = ''
    for i in range(len(set_of_subseq)):
        string_set_of_subset +=  '>i' + str(i) + ' iteration;\n' +
str(set_of_subseq[i]) + '\n'

    file = open(name_set_subseq0,'w')
    file.write(string_set_of_subset)
    file.close()

    #совершение одной итерации улучшения по новому набору
    muscle_msa(name_set_subseq0, name_set_subseq0_msa)
    fasta_into_stockholm(name_set_subseq0_msa, name_set_subseq0_msa_stockh)
    nhmmer_on_stock_msa(name_set_subseq0_msa_stockh, s_name_fasta,
name_result_nhmmer_info, name_result_nhmmer_posision_new, N)

    name_res_txt = os.path.join(name_folder, 'second_subseq.txt')
```

```
    #возращает количество найденных последовательностей
    return creating_txt_res_nhmmer_set(s_name_txt, name_result_nhmmer_posision_new,
min_len_subseq = 0, max_len_subseq = 1000, N =  10 ** 6, name_res_txt = name_res_txt)
```