NSCI0007_practice_exam_answers

November 30, 2021

1 NSCI0007 Practice Exam 1

1.1 Specimen Answers and Mark Scheme

- The specimen code below demonstrates one way to correctly answer the questions.
- Full marks will be awarded if the candidate has implemented another suitable method and the code behaves as specified in the question.
- If the candidate's code produces an error, or does not behave as specified in the question, partial credit will be awarded as described in the mark scheme.
- Where a candidate has used a different method to below, partial credit will be awarded in an analogous way.

1.2 Question 1 [7]

[1]:
```
def overlap(x, y):
    n = min(len(x), len(y))
    for i in range(n, 0, -1):
        if x[-i:] == y[:i]:
            return i
    return 0

n1 = overlap("XXXABC", "ABCYYY")
n2 = overlap("ABCYYY", "XXXABC")
n3 = overlap("XXXABC", "ABC")
print(n1, n2, n3)

# [2] find minimum of length of two strings
# [1] appropriate looping construct
# [2] if statement with correct string indexing
# [2] tests pass and function behaves as specified
```

```
3 0 3
```

1.3 Question 2 [5]

[2]:
```
def merge(x, y):
    i = overlap(x, y)
    return x + y[i:]

s1 = merge("XXXABC", "ABCYYY")
s2 = merge("ABCYYY", "XXXABC")
s3 = merge("XXXABC", "ABC")
print(s1, s2, s3)

# [1] call overlap function
# [2] calculate merged string
# [2] tests pass and function behaves as specified
```

```
XXXABCYYY ABCYYYXXXABC XXXABC
```

1.4 Question 3 [10]

[3]:
```
def longest_overlap(sequences):
    max_overlap = 0
    max_i = 0
    max_j = 0
    for i in range(len(sequences)):
        for j in range(len(sequences)):
            if i != j:
                d = overlap(sequences[i], sequences[j])
                if d > max_overlap:
                    max_overlap = d
                    max_i = i
                    max_j = j

    return [max_i, max_j, max_overlap]

i, j, k = longest_overlap(["XXXABC", "ABCYYY", "BC"])
print(i, j, k)

# [1] declare max variables
# [2] two nested for loops
# [1] test for i=j
# [1] call overlap function
# [1] check for maximum
# [1] update max values
# [1] return list of values
# [2] tests pass and function behaves as specified
```

```
0 1 3
```

1.5 Question 4 [10]

[4]:
```
def identify_strand(sequences, n):
    i, j, d = longest_overlap(sequences)
    while d >= n:
        z = merge(sequences[i], sequences[j])
        del sequences[max(i, j)]
        del sequences[min(i, j)]
        sequences.append(z)
        i, j, d = longest_overlap(sequences)
    return sequences

# [2] suitable looping construct with correct condition for termination
# [1] call merge function
# [3] remove two items in correct order
# [1] append merged string to list
# [1] call longest_overlap function
# [2] tests pass and function behaves as specified

sequences = ['tgaaaattcctttctattttaggccc', 'tgaaaattcctttctattttaggcccatgcaat',␣
↪'ggcattagggcggttaa', 'atgcaatggcattagggcggttaa', 'ggttaa',␣
↪'tgaaaattcctttctattt', 'taggcccatgcaatggcattagggc']
identify_strand(sequences, 4)
```

[4]:
```
['tgaaaattcctttctattttaggcccatgcaatggcattagggcggttaa']
```

1.6 Question 5 [8]

[5]:
```
sequence_list = []
with open("dna_fragments/strand_100.fasta") as f:
    for line in f:
        if line[0] != ">":
            sequence_list.append(line.strip())

s = identify_strand(sequence_list, 4)
print(s)
```

```
['CCCAGGGAGACCACTGACCCATCAACCTGTACGGGAACCTTCTGTATCGTTCTCGGACGGAGAGATAACTACAGTGCC
GCTTACAGCCCCTCTGTCGTCG']
```

[6]:
```
sequence_list = []
with open("dna_fragments/strand_200.fasta") as f:
    for line in f:
        if line[0] != ">":
            sequence_list.append(line.strip())

s = identify_strand(sequence_list, 4)
```

```
print(s)
print(s[-1]) # longest string is last one in list
```

```
['GTGTAGGTTCTGACCGATTCGTGC', 'CCGACGTCTGTAATGTAGCCTCATTGTGATTCCACCCTATTGAGGCATTG
ACTGATGCGGGAAGAGATCTGAAATGAACTGGTCTATGCGACAGAAACTGTGCAGCTACCTAATCTCCTTAGTGTAGGTT
CTGACCGATTCGTGCTTCGTTGAGAACTCACAATTTAACAACAGAGGACATAAGCCCTACGCCCATGATC']
CCGACGTCTGTAATGTAGCCTCATTGTGATTCCACCCTATTGAGGCATTGACTGATGCGGGAAGAGATCTGAAATGAACT
GGTCTATGCGACAGAAACTGTGCAGCTACCTAATCTCCTTAGTGTAGGTTCTGACCGATTCGTGCTTCGTTGAGAACTCA
CAATTTAACAACAGAGGACATAAGCCCTACGCCCATGATC
```

[7]:
```
sequence_list = []
with open("dna_fragments/strand_500.fasta") as f:
    for line in f:
        if line[0] != ">":
            sequence_list.append(line.strip())

s = identify_strand(sequence_list, 4)
print(s[1]) #longest string is last one in list
```

```
AATCTTTTTCACTGACAGTCATATTGGGGTGCTCCTAAGCTTTTCCACTTGGCTGGGTCTGCTAGGCCTCCGTGCCCGGA
GTTTCGGCGCTGTGCTGCCGAGAGCCGGCCATTGTCATTGGGGCCTCACTTGAGGATACCCCGACCTATTTTGTCGGGAC
CACTCGGGGTAGTCGTTGGGCTTATGCACCGTAAAGTCCTCCGCCGGCCTCCCCGCTACAGAAGATGATAAGCTCCGGCA
AGCAATTATGAACAACGCAAGGATCGGCGATATAAACAGAGAAACGGCTGATTACACTTGTTCGTGTGGTATCGCTAAAT
AGCCTCGCGGAGCCTTATGCCATACTCGTCCGCGGAGCACTCTGGTAACGCTTATGGTCCATAGGACATTCATCGCTTCC
GGGTATGCGCTCTATTTGACGATCCTTTGGCGCACAGATGCTGGCCACGAGCTAAATTAGAGCGACTGCACAACTGTAAG
GTCCGTCACGCAGACGACGG
```

[8]:
```
# [1] correctly open file
# [1] loop over lines
# [2] form list of strands ommiting lines starting '>'
# [1] call identify_strand
# [1] identify longest one (OK to do this by eye but must be commented or␣
 ↪otherwise identified)
# [2] repeat for the other two files (could be loop or repeated code)
```