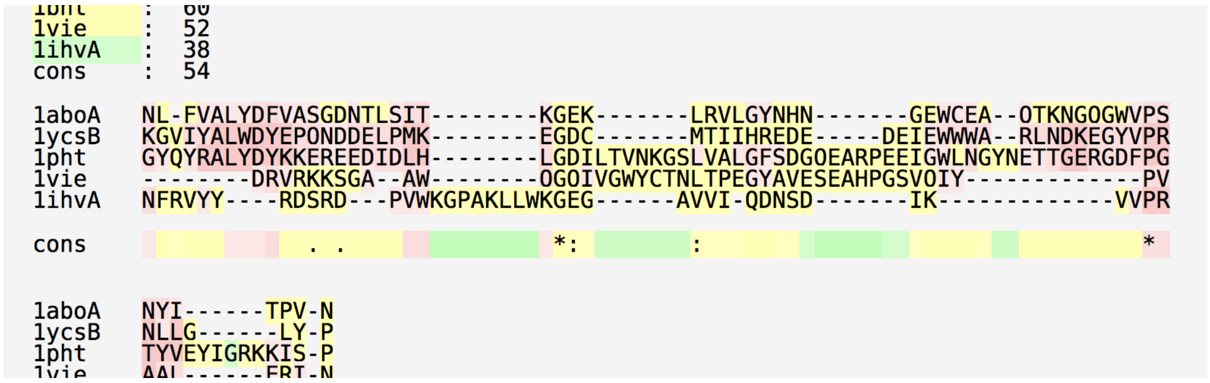


```
1pht     :  60
1vie     :  52
1ihvA    :  38
cons     :  54

1aboA    NL-FVALYDFVASGDNTLSIT--------KGEK-------LRVLGYNHN-------GEWCEA--OTKNGOGWVPS
1ycsB    KGVIYALWDYEPONDDELPMK--------EGDC-------MTIIHREDE-----DEIEWWWA--RLNDKEGYVPR
1pht     GYQYRALYDYKKEREEDIDLH--------LGDILTVNKGSLVALGFSDGOEARPEEIGWLNGYNETTGERGDFPG
1vie     --------DRVRKKSGA--AW--------OGOIVGWYCTNLTPEGYAVESEAHPGSVOIY------------PV
1ihvA    NFRVYY----RDSRD---PVWKGPAKLLWKGEG------AVVI-QDNSD-------IK------------VVPR

cons                  . .                 *:         :                                  *


1aboA    NYI------TPV-N
1ycsB    NLLG------LY-P
1pht     TYVEYIGRKKIS-P
1vie     AAL------ERI-N
```

Assignment 1 - Sequence Alignment

COMP 561 - Computational Biology Methods and Research

LE, Nhat Hung
McGill ID: 260793376
Date: September 25, 2019
Due date: October 2, 2019

Prof. Mathieu Blanchette
Fall 2019

Question 1. (10 points) Needleman-Wunsch algorithm

What is the optimal global alignment for APPLE and HAPE? Show all optimal solutions and the corresponding paths under the match score of +1, mismatch score of -1, and indel penalty cost(L) = -1*L (linear gap penalty).

Solution:

Applying the Needleman-Wunsch algorithm:

```
    .    A    P    P    L    E
  --------------------------
.| 0   -1   -2   -3   -4   -5
 | ↑ ↖    ↖    ↖    ↖    ↖
H|-1   -1 ← -2 ← -3 ← -4 ← -5
 |   ↖
A|-2    0 ← -1 ← -2 ← -3 ← -4
 |      ↑ ↖    ↖
P|-3   -1    1 ←  0 ← -1 ← -2
 |      ↑    ↑ ↖    ↖    ↖
E|-4   -2    0    0 ← -1    0
```

There are 2 paths and therefore 2 optimal alignments:

```
1. -APPLE
   HA-P-E

2. -APPLE
   HAP--E
```

Both have a score of 0.

Question 2. (5 points) Affine gap penalty

Give an example of a pair of short sequences for which the optimal pairwise alignment under the linear gap penalty scheme described in question 1 is different from the optimal pairwise alignment under an affine gap penalty with cost(L) = -2 - 0.5*L. Try to find the shortest sequences possible, and show the optimal alignment under the two scoring schemes.

Solution:

Assume non-empty sequences. Let

$$S_1 = \text{AB}$$
$$S_2 = \text{A}$$

Both scoring schemes yield the same optimal alignment, with scores:

```
S1'             :  A  B
S2'             :  A  -

Scoring scheme 1:  1 -1   =  0
Scoring scheme 2:  1 -2.5 = -1.5
```

Question 3. (45 points) Sequence alignment with arbitrary gap penalty

When aligning two sequences, it is important to choose an alignment scoring scheme (substitution matrix cost and gap penalty function) that best captures the types of evolutionary events that are common in the type of sequences being aligned. For example, if the two sequences being aligned are protein-coding genes, then insertions or deletions of length 3, 6, 9, etc. nucleotides are common, whereas those whose length is not a multiple of three are rare, because they would result in a frameshift, i.e in the complete change of the amino acid encoded by the portion of the gene that follows the indel point. The Needleman-Wunsch dynamic programming algorithm can be modified to identify the optimal alignment under that kind of scoring scheme... but it's a bit too complicated for this assignment. Instead we will consider the following (somewhat artificial) alignment problem, that we call the *multi-gap-free* alignment problem.

We say that an alignment is *multi-gap-free* if it does not contain consecutive gaps in the *same* sequence. For example, the following alignments are multi-gap-free:

```
ACAG
A-T-

AC-A-G
A-T-G-
```

But the following alignments are not multi-gap-free:

```
ACAG
A--T

ACA--
A-ATG
```

a) (5 points) *Multi-gap-free* only exist when the lengths *m* and *n* of the two sequences satisfy a certain constraint. What is that constraint?

Solution:

Without lost of generality, let

$$n \geq m$$

Then, a necessary condition for multi-gap-free alignments is

$$n \geq m \geq \lceil n/2 \rceil$$

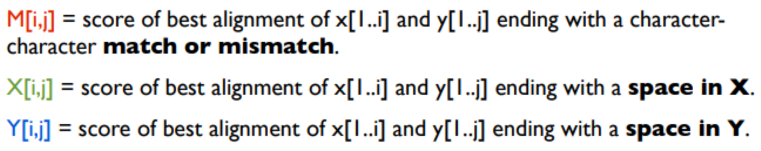
---

b) (20 points) Give the pseudocode of an exact algorithm for optimal pairwise global multi-gap-free alignment. Assume that a substitution cost matrix *M* and a gap penalty cost *b* are given as input, along with the two sequences *S* and *T* to be aligned. Your algorithm should run in time O(*mn*), where *m* and *n* are the lengths of the two sequences.

Hint: Study the algorithm for optimal pairwise alignment that works under the affine gap penalty (see the scan of Durbin et al.'s textbook on MyCourses). This should provide ideas for how to solve this question.

Solution:

The alignment algorithm with affine gap penalty (Durbin et al.) uses **3 matrices**



M[i,j] = score of best alignment of x[1..i] and y[1..j] ending with a character-character **match or mismatch**.

X[i,j] = score of best alignment of x[1..i] and y[1..j] ending with a **space in X**.

Y[i,j] = score of best alignment of x[1..i] and y[1..j] ending with a **space in Y**.

From bioformatics lectures of Carl Kingsford of Carnegie Mellon University

filled according to the following scoring scheme:

$$\underset{\text{match between x and y}}{M[i,j]} = \text{match}(i,j) + \max \begin{cases} M[i-1,j-1] \\ X[i-1,j-1] \\ Y[i-1,j-1] \end{cases}$$

If previous alignment ends in match, this is a new gap

$$\underset{\text{gap in x}}{X[i,j]} = \max \begin{cases} \text{gap\_start} + \text{gap\_extend} + M[i,j-1] \\ \text{gap\_extend} + X[i,j-1] \\ \text{gap\_start} + \text{gap\_extend} + Y[i,j-1] \end{cases}$$

$$\underset{\text{gap in y}}{Y[i,j]} = \max \begin{cases} \text{gap\_start} + \text{gap\_extend} + M[i-1,j] \\ \text{gap\_start} + \text{gap\_extend} + X[i-1,j] \\ \text{gap\_extend} + Y[i-1,j] \end{cases}$$

Carl Kingsford

Our multi-gap-free problem does **not allow an existing gap to extend**. Therefore, we remove from the above lines with *gap_extend* but no *gap_start*.

Adapting to our notation, this gives:

$$M(i,j) = s(T_i, S_j) + max \begin{cases} M(i-1,j-1) \\ S_{gap}(i-1,j-1) \\ T_{gap}(i-1,j-1) \end{cases}$$

$$S_{gap}(i,j) = b + max \begin{cases} M(i-1,j) \\ T_{gap}(i-1,j) \end{cases}$$

$$T_{gap}(i,j) = b + max \begin{cases} M(i,j-1) \\ S_{gap}(i,j-1) \end{cases}$$

S, T sequences
$S_{gap}(i,j)$ score of best alignment of $S[1..j]$ and $T[1..i]$ ending in a gap in S
$T_{gap}(i,j)$ score of best alignment of $S[1..j]$ and $T[1..i]$ ending in a gap in T
s substituion cost matrix (instead of M)
b gap penalty cost

We can show the algorithm runs in **O(mn) time**. With 3 matrices, the number of cells to be filled is

$$3(m+1)(n+1) = O(3mn) = O(mn)$$

The algorithm is as follows:

```
def multi_gap_free(S, T, s, b):
    '''
    :param S: Top sequence (spanning columns)
    :param T: Left sequence (spanning rows)
    :param s: Substitution cost matrix
    :param b: Gap penalty cost
    '''
    m, n = |S|, |T|
    M, S_gap, T_gap = (n+1)x(m+1) matrices

    # Initialize matrices:
    # M[0,j] = "score of best alignment between
    #           0 chars of T and j chars of S
    #           that ends in a match"
    #        = 0          if j == 0
    #        = -infinity if j > 0
    #          because there can't be a match with 0 chars of T and > 0 chars of S,
    #          therefore such alignments don't exist
    #
    # Same reasoning with M[i,0]
    M[0,:] = -inf
    M[:,0] = -inf
    M[0,0] = 0

    # S_gap[0,j] has the same reasoning as T_gap[i,0]
    # S_gap[i,0] has the same reasoning as T_gap[0,j]
    S_gap[0,:] = -inf
    S_gap[:,0] = -inf
    S_gap[1,0] = b

    # T_gap[0,j] = "score of best alignment between
    #                0 chars of T and j chars of S
    #                that ends in a gap in T"
    #             = b if j == 1 else -infinity
    #
    # When j == 1, the alignment looks like:
    #     s
    #     -
    #     which is allowed
    #
    # When j != 1:
    #     ssss
    #     ----
    #     which isn't allowed
    #
    # T_gap[i, 0] = "score of best alignment between
    #                 i chars of T and 0 chars of S
    #                 that ends in a gap in T"
    #              = -infinity
    #                because with 0 chars of S, the gap in T
    #                will align with another gap in S, which isn't allowed
    #
    # This alignment looks like:
    #     ----
    #     ttt-
    T_gap[0,:] = -inf
    T_gap[:,0] = -inf
    T_gap[0,1] = b

    for i = 1:n
        for j = 1:m
            M[i,j] = s(S[j], T[i]) + max(
                M[i-1,j-1],
                S_gap[i-1,j-1],
                T_gap[i-1,j-1]
            )
            S_gap[i,j] = b + max(M[i-1,j], T_gap[i-1,j])
            T_gap[i,j] = b + max(M[i,j-1], S_gap[i,j-1])

    return M, S_gap, T_gap
```

c) (20 points) Implement your algorithm, including the trace-back procedure, in the programming language of your choice. Do not use any external alignment package/library; write your program from scratch. Your algorithm should take four arguments: (1) File containing the two sequences to be aligned (FASTA format). (2) The score for matches; (3) The score for mismatches (we will assume that all mismatches are score identically); (4) The value of gap penalty b. Your program should print out the optimal alignment score, and the optimal alignment itself.

Use your program to compute the optimal multi-gap-free alignment for the following pairs of sequences, assuming matchScore = 1, mismatchScore = -1, b = -1:
http://www.cs.mcgill.ca/~blanchem/561/hw1_short.fa
http://www.cs.mcgill.ca/~blanchem/561/hw1_medium.fa
http://www.cs.mcgill.ca/~blanchem/561/hw1_long.fa

Report the alignments found, together with their score, and submit your code on MyCourses. Your code will not be marked, but submitting an answer (alignment score) that would not have been produced by your own code would be considered cheating.

Solution:

The scores for all 3 pairs are:

- Short: 74
- Medium: 314
- Long: 2708

An alignment for each pair is shown below, and is also included in the submitted `code.zip` archive:

```
hw1_short.fa:
Score: 74

HUMAN_BRCA1: ATGGATTTATCTGCTCTTCGC-G-TTGAAGAAGTACAAAATGTCATTAATGCTATGCAGAAAATCTTAGAGTGTCCCATCTGTCTGGAGTTGATCAAGGAAC
MOUSE_BRCA1: ATGGATTTATCTGC-CGTC-CAAATTCAAGAAGTACAAAATGTCCTTCATGCTATGCAGAAAATCTTAGAGTGTCCGATCTGTTTGGAACTGATCAAAGAAC

--------------------

hw1_medium.fa:
Score: 314

HUMAN_BRCA1: ATGGATTTATCTGCTCTTCGC-G-TTGAAGAAGTACAAAATGTCATTAATGCTATGCAGAAAATCTTAGAGTGTCCCATCTGTCTGGAGTTGATCAAGGAAC
MOUSE_BRCA1: ATGGATTTATCTGC-CGTC-CAAATTCAAGAAGTACAAAATGTCCTTCATGCTATGCAGAAAATCTTAGAGTGTCCGATCTGTTTGGAACTGATCAAAGAAC

CTGTCTCCACAAAGTGTGACCACATATTTTGCAAATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCTTTATGTAAGAATGATATAACCAAAAGGAG
CTGTTTCCACAAAGTGTGACCACATATTTTGCAAATTTTGTATGCTGAAACTTCTTAACCAGAAGAAAGGGCCTTCACAATGTCCTTTGTGTAAGAATGAGATAACCAAAAGGAG

CCTACAAGAAAGTACGAGATTTAGTCAACTTGTTGAAGAGCTATTGAAAATCATTTG-TGCTTTTCAGCTTGACACAGGTTTGGAG-TATGCAAACAGCTATAATTTTGCAAAAA
CCTACAGGGAAGCACAAGGTTTAGTCAGCTTGCTGAAGAGCTGCTGAGAAT-AATGGCTGCTTTTGAGCTTGACACGGGAATGCAGCT-TACAAATGGTTTTAGTTTTTCAAAAA

AGGA-AAATAACTCTCCTGAACATCTAAAAGATGAAGTTTCTATCATCCAAAGTATGGGCTACAGAAACCGTGCCAAAAGACTTCTACAGAGT-GAACCCGAAAAT-CCTTCCTT
A-GAGAAATAATTCTTGTGAGCGTTTGAATGAGGAGGCGTCGATCATCCAGAGCGTGGGCTACCGGAACCGTGTCAGAAGGCTTCCCCAG-GTCGAACCTGGAAATGCC-ACCTT

GCAGGAAACCAGTCTCA-GTGTCCAACTCTCTAACCTTGGAACT-GTGAGAACTCTG-AG-G-AC
GAAGG-A-C-AGCCT-AGGTGTCCAGCTGTCTAACCTTGGAA-TCGTGAGATCAGTGAAGAAAAA

--------------------

hw1_long.fa:
Score: 2708

HUMAN_BRCA1: ATGGATTTATCTGCTCTTCGC-G-TTGAAGAAGTACAAAATGTCATTAATGCTATGCAGAAAATCTTAGAGTGTCCCATCTGTCTGGAGTTGATCAAGGAAC
MOUSE_BRCA1: ATGGATTTATCTGC-CGTC-CAAATTCAAGAAGTACAAAATGTCCTTCATGCTATGCAGAAAATCTTAGAGTGTCCGATCTGTTTGGAACTGATCAAAGAAC

CTGTCTCCACAAAGTGTGACCACATATTTTGCAAATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCTTTATGTAAGAATGATATAACCAAAAGGAG
CTGTTTCCACAAAGTGTGACCACATATTTTGCAAATTTTGTATGCTGAAACTTCTTAACCAGAAGAAAGGGCCTTCACAATGTCCTTTGTGTAAGAATGAGATAACCAAAAGGAG

CCTACAAGAAAGTACGAGATTTAGTCAACTTGTTGAAGAGCTATTGAAAATCATTTG-TGCTTTTCAGCTTGACACAGGTTTGGAG-TATGCAAACAGCTATAATTTTGCAAAAA
CCTACAGGGAAGCACAAGGTTTAGTCAGCTTGCTGAAGAGCTGCTGAGAAT-AATGGCTGCTTTTGAGCTTGACACGGGAATGCAGCT-TACAAATGGTTTTAGTTTTTCAAAAA

AG-GAAAATAACTCTCCTGAACATCTAAAAGATGAAGTTTCTATCATCCAAAGTATGGGCTACAGAAACCGTGCCAAAAGACTTCTACAGAGT-GAACCCGAAAAT-CCTTCCTT
AGAG-AAATAATTCTTGTGAGCGTTTGAATGAGGAGGCGTCGATCATCCAGAGCGTGGGCTACCGGAACCGTGTCAGAAGGCTTCCCCAG-GTCGAACCTGGAAATGCC-ACCTT

GCAGGAAACCAGTCTCA-GTGTCCAACTCTCTAACCTTGGAA-CTGTGAGAACTCTGAGGACAAAGCA-GCGGATACAACCTCAAAAGACGTCTGTCTACATTGAATTGGGA-TC
GAAGG-A-C-AGCCT-AGGTGTCCAGCTGTCTAACCTTGGAATC-GTGAGATCAGTGAAGA-AAAACAGGCAGACCCAACCTCGAAAGAAATCTGTCTACATTGAACT-AGACTC

TGATTCTTCTGAAGATACCGTTAA-TAAGGCAACTTATTGCAGTGTGGGAGATCAAGAATTGTTACAAATCACCCCTCAAGGAACCAGG-GATGAAATC-AGTTTGGATTCTGCA
TGATTCTTCTGAAGAGACAG-TAACTAAGCCAGGTGATTGCAGTGTGAGAGACCAGGAATTGTTACAGACCGCCCCTCAA-GAAGCTGGAGATGAAGGCAAG-CTGCACTCTGCA

AAAAAGGCTGCTTGTGAATTTTCTGAGACGGATGTAACAAATACTGAACATCATCAACCCAGTAATAATGATTTGAACACC-ACTGAGAAGCGTGCAGCTGAGAGGCATCCAGAA
GAAGAGGCTGCTTGTGAGTTTTCTGA-G-GG-CATAAGAAACATTGAACATCATCAATGCAGT-G-A-TGATTTAAAC-CCTACTGAGAATCATGCAACTGAAAGGCATCCAGAA

AAGTATCAGGGTAGTTCTGTTTCAAACT-TGCATGTGGAGCCATGTGGCACAAATACTCATGCCAGCTCATTACAGCATGAGAACAGCAGTTTATTACTCACTAAAGACAGAATG
AAATGTCAGAGTATTTCTATTTCAAA-TGTGTGTGTGGAGCCATGTGGCACAGATGCTCATGCCAGCTCATTACAGCCTGAGACCAGCAGTTTATTGCTCATTGAAGACAGAATG

AATGTAGAAAAGGCTGAATTCTGTAATAAAAGCAAACAGCCTGGCTTAGCAAG-GAGCCAACATAACAGATGGGCTGGAAGT-AAGGAAACATGTAATGATAGGC-GGACTCCCA
AATGCAGAAAAGGCTGAATTCTGTAATAAAAGCAAACAGCCTGGCATAGC-AGTGAGCCAGCAGAGCAGATGGGCTGCAAGTAAAGG-AACATGTAACGACAGGCAGG-TTCCCA

GCACAGAAAAAAAGGTAGATCTGAATGCTGA-TCCCCTGT-GTGAGAGAAAAGAA-TGGAATAAGCAG-AAACTGCCATGCTCAGAGAATCCTAGAGATACTGA-AGATGTTCCT
GCACTGGGGAAAAGGTAGGTCCAAACGCTGACT-CCCT-TAGTGATAG-AGAGAAGTGGACTCACCCGCAAAGT-CTGTGCCCTGAGAATTCTGGAGCTAC-CACCGATGTTCCT

TGGATAACACTAAATAGCAGCATTCAGAAAGTTAATGAGTGGTTTTCCAGAAGTGATGAACTGTTAGGTTCTGA-TGACTCA-CATGATGGGGAGTC-TGAATCAAATGCCAAAG
TGGATAACACTAAATAGCAGCGTTCAGAAAGTTAATGAGTGGTTTTCCAGAACTGGTGAAATGTTAACTTCTGACAG-CGCATC-TG-CCAGGAGGCACGAGTCAAATGCTGAAG

TAGCTGATGTATTGGACG-TTCTAAATGAGGTAGATGAATATTCTGGTTCTTC-AGAGAAAATAGACTTACTGGCCAGTGATCCTCATGAGGCTTTAATATGTAAAAGTGAAAGA
CAGCTGTTGTGTTGGAAGTTTC-AAACGAAGTGGATGGGGGTTTTAGTTCTTCAAG-GAAAACAGACTTAGTAACCCCCGACCCCCATCATACTTTAATGTGTAAAAGTGGAAGA

G-TTCACTCCAAATCAGTAGA-GAGTAATAT-TGAAGACAAAATATTTGGGAAAACCTATCGGA-AGAAGGCAAGCCTCCC-CAACTTAAGCCATGTAACTGAAAATCTAATTAT
GACT-TCTCCAAACCAGTAGAGGA-TAATATCAG-TGATAAAATATTTGGGAAATCCTATCAGAGA-AAGGGAAGCCGCCCTCACCTGAA-CCATGTGACTG-AAAT-T-A-TA-

AGGAGCATTTGTTACTGAGCCACAGATAATACAAGAGC-GTCCCCTCACAAATAAATTAAAGCGTAAAAGGAGACCTACATCAGGCCTTCATCCTGAGGATTTTATCAAGAAAGC
GGCA-CATTTATTACAGAACCACAGATAACACAAGAGCAG-CCCTTCACAAATAAATTAAAACGT-A-A-GAGAAGTACATC-C-C-TTCAACCTGAGGACTTCATCAAGAAAGC

AGATTTGGCA-GT-T-CAAAAGACTCCTGA-AATGATAAATCAGGGAACTAACCAAACGGAG-CAGAATG-GTCAAG-TGATGAATATTACTAAT-AGTGGTCATGAGAATAAAA
AGATTCAGCAGGTGTTCAAAGGACTCCTGACAA-CATAAATCAGGGAACTGACCTAATGGAGCCA-AATGAG-CAAGCAG-TGAGTACTACCAGTAACT-GTCAGGAGAACAAAA

CAAAAGGT-G-ATTCTATTCAGAATGAGAAAAATC-CTAACCCAA-TAGAATCACTCGA-AAAAGAATCTGCTTTCAAAAC-GAAAGCTGAACCTATAAGCAGCAGTATAAGCAA
TAGCAGGTAGTAATCT-C-CAGAAAGAG-AAAAGCGCTCATCCAACT-GAATCA-TTGAGAAAGGAACCTGCTTCCACAGCAG-GAGCCAAATCTATAAGCAACAGTGTAAGTGA

TATGGAACTCGAATTAAATATCCACAATTCAAAAGCACCTAA-AAAGAATAGGCTGAGGAGGAAGTCTTCTACCAGGCATATTCATGCGCTTGAACTAGTAGTCAGTAGAAATCT
TTTGGAGGTAGAATTAAACGTCCACAGTTCAAAAGCACCTAAGAAA-AATAGGCTGAGGAGGAAGTCTTCTATCAGGTGTGCTCTTCCACTTGAAC-C-AA-TCAGTAGAAATCC

AAGCCCACCTAATTGTACTGA-ATTGCAAATTGATAGTTGTTCTAGCAGTGAAGAGATAAAGAAAAAAAAGTACAACCAA-ATGCCAGTCAGGCACAGCAGA-AACCTACAACTC
```

```
AAGCCCACCTACTTGTGCTGAGCTT-CAAATCGATAGTTGTGGTAGCAGTGAAGAAACAAAGAAAAACCATTCCAACCAACA-GCCAGCCGGGCACCTTAGAGAGCCT-CAACTC

ATGGAAGGTAAAGAACCTGCAACTGGA-GCCAAGAAGAGTAACAAGCCAAATGAACAGACAAGTAAAAGACATGACAGCGATACTTTCCCAG-AGCTGAAGTTAACAAATGCACC
ATCGAAGACACTGAACCTGCAGC-GGATGCCAAG-A-AG-AACGAGCCAAATGAACACATAAGGAAGAGACGTGCCAGCGATGCTTTCCCAGAAG-AGAAATTAATGAACAAAGC

TGGTTCT-TTTACTAAG-TGTTCAAATACC-AGTGAA-CTTAAAGAATTTGTCAATCCTAGCCTTCCAAGAGAAGAAAAAGAAGAGAAACTAGAAACA-GTTAAAGTGTCTAATA
TGGTT-TATTAACT-AGCTGTTCAAGT-CCTAGAAAATC-TCAAGGGCCTGTCAATCCCAGCCCT-C-AGAGAACAGGAA-CAGAGCAACTTGAAACACGCCAAA-TGTCTGACA

ATGCTGAAGACCCCAAAGATCTCATGTTAAGTGGAGAAAG-GGTTTTGC-A-A-A-CTGAAAGATCTGTA-GAGAGTAGCAGTATTTCATTGGTACCTGGTACTGATTATGGCAC
GTGCCAAAGAACTCGGGGATCGGGTCCTAGGAGGAG-A-GCCCAGTGGCAAAACCACTGACCGATCTG-AGGAGAGCACCAGCGTATCCTTGGTATCTGACACTGACTACGACAC

TCAGGAA-AGTATCTC-GTTACTGGAAG-TTAGCACTCT-AGGGA-AGGCAAAAACA-GAACCAAATAAATGTGTGAGTCAGTGTGCAGCATTTGAAAACCCCAAGGGACTAATT
TCA-GAACAGTGTCTCAG-TCCTGGACGCTCA-CACTGTCA-G-ATATGCAAGAACAGGATCC-GCTCAGTGTATGACTCAGTTTGTAGCAAGCGAAAACCCCAAGGAACTCGTC

CATGGTTGTTCCAA-AGATAATAGAAATGACACAGAAGG-CTTTAAGTATCCATTGGGACATGAAG-TTAACCACAGTCGGGAAACAAGCATAGAAATGGAAGAAAGTGAACTTG
CATGG-C-T-CTAACA-ATGCTGGGAGTGGCACAGAGGGTC-TCAAGCCCCCCTTGAGACACG-CGCTTAACCTCAGTCAGGAGA-AA-G-TAGAAATGGAAGACAGTGAACTTG

ATGCTCAGTATTTGCAGAATACA-TTCAAGGTTTCAAAGCGCCAGTCATTTGCTCCGTTTTCAAATCC-AGGAAATGCAGAAGAGGAATGTGCAACATTCTCTGCCCACTCTGGG
ATACTCAGTATTTGCAGAATACATTTCAA-GTTTCAAAGCGTCAGTCATTTGCTTTATTTTCAAAACCTA-GAAGT-C-C-CCA-AAA-G-G-A-C-T-G-T-GCTCACTCTGTG

TCCTTAAAGAAACAAAGTCCAAAAGTCACTTTTGAATGTGAACAAAAGGAAGAAAATCAAGGA-AAGAATGAGTCT-AATATCAAG-C-CTGTAC-AGACAGTTAATATCACTGC
CCCTCAAAGGAACTGAGTCCAAAGGTGACAGCTAAAGGT-AA-ACAA-AAAGAACGTCAGGGACAGGAA-GAATTTGAA-ATC-AGTCAC-GTACAAG-CAGTTGCGGCCACAGT

AGGCTTTCCTGTG-GTTGGTC-AGAAAGATAAGCCAGTTGATAATGCCAAATGTAGTATCAAAGGAGGCTCTAGGTTTTGTCTATCATCTCAGTT-CAGAGGCAACGA-AACTGG
GGGCTTACCTGTGCCCT-GTCAAG-AAGGTAAGCTAGCTGCTGAT-AC-AATGT-GT-G-ATA-GAGGTTGTAGGCTTTGTCCATCATCTCA-TTACAGAAGCGGGGAGAA-TGG

ACTCATTACTCCA-A-ATAAA-CATGGACTTTTACAAAACCCATATCGTATACCACCACTTTTTCCCATCAAGTCATTTGTTAAAACT-AAATGTAAGAAAAATCTG-CTAGA-G
ACTCA-G-CGCCACAGGTAAATCA-GGAATTTCACAAAACTCACATTTTAAACAATCAGTTTCTCCCATCAGGTCATCTATAAAAACTGACA-ATAGGAAACCTCTGAC-AGAGG

GAAAACTTTGAG-GAACATTCAAT-GTCACCTGA-AAGAGAAATGGGAAATGAGAACA-T-TCCA-AGTACAGTG-AGCACAATTAGCC-GTAATAACATTAGAGAAAATGTTTT
GACGA-TTTGAGAG-ACATAC-ATCATCAACTGAGATG-GCGGTGGGAAATGAGAACATTCTTCAGAGTACAGTGCA-CACAGTTAGCCTG-AATAACA-GAG-G-AAATGCTTG

TAAAGAAGCCAGCTCAAGCAATATTAATGAAGTAGGTTCCAGTACTAATGAAGTGGGCTCCAGTATTAATGAAATAGGTTCCAGTGATGAAAACATTCAAGCAGAACTAGGTAGA
TCAAGAAGCCGGCTCGGGCAGTATTCATGAAGTATGTT-C-C-A-C-T-G-G-T-GACTCC-TT-CCCAGGACA-A-C-T-AGGT-A-G-AAACA-G-A-G-GG-CC-T-A-AG-

AACAGAGGGCCAAAATTGAATGCTATGC-TTAGATTAGGGGTTTTGCAACCTGAG-GTCTATAAACAAAGTCTTCCTGGAAGTAATTGTAAGCATCCTGAAATAAAAAAGCAAGA
G-T-GA-A-C-A-C-T-G-T-GC-CTCCATTAGA-T-A-G-T-ATGCAGCCTG-GTGTCTGTCAGCAAAGTGTTCCTGTAAGT-G-A-TAAGTATCTTGAAATAAAAAAGC-AG-

ATATGAAGAAGTAGTTCAGACTGTTAATACAGATTTCTCTCCATATCTGATT-TCAGA-TAACTTAGAACAGCCTATGGGAAGTAGTCATGCATCTCAGGTTTGTTCTGAGACAC
G-A-G-GG-TG-AG-GCTGTCTG-T-G-C-AGACTTCTCTCCATGTCT-ATTCTCAGACCATCTT-GAGCAATCTAT-G-A-GTGGTAAGGTTTTTCAGGTTTGCTCTGAGACAC

CTGATGACCTGTTAGATGATGGTGAAATAAAGGAAGATACTAGTTTTGCTGAAAATGACATTAA-GGAAAGTTCTGCTGTTTTTAGCAAAAGCGTCCAGA-AAGGAGAG-CTTAG
CTGATGACCTGCTGGATGATGTTGAAATACAGGGACATACTAGCTTTGGTGAAGGTGACA-TAATGGAGAGATCTGCTGTCTTTAACGGAAGCATCCTGAGAAGG-GAGTC-CAG

CAGGAGTCCTAGCCCTTTCACCCATACA-C-ATTTGGCTCAGGGT-TACCGA-AGAGGGGCCAAGAAATTAGAGTCCTCAGAAGAGAACTTA-TCTAGTGAGGATGAAGAGCTTC
TAGGAGCCCTAGTCCTGTAACCCATGCATCGA-A-GTCTCAGAGTCT-CC-ACAGAGCGTCTAGGAAATTAGAATCGTCAGAAGAGAGC-GACTCCACTGAGGATGAAGATCTTC

CCTGCTTCCAACACTTGTTATTTG-GTA-AAGTAAACAATATACCTTCTCAG-TCTACTAGGCATAGCA-C-C-GTTGCT-AC-CGAGTGTCTGTCTA-AGAACA-CAGAGGAGA
CCTGCTTCCAACAC-T-T-A-CTGAGCAGAA-TAAGCAACACACC-T-G-AGCT-TACCA-G-AT-GCAGCAGTGCTG-TGACAC-AGCGTATG-CCAGAGAA-AGCGGAGGGGA

ATTTATTATCATTGAAGAATAGCTTAAATGACTGCAGTAACCAGGTAAT-ATTGGCAAAGGCATCTCAGGAACATCACCTTAGTGAGGAAACAAAATGTTCT-GCTAGCTTGTTT
CCCAAGCACCATGGAAGGGTAGCAGCAGTGACTGCAATAATGAGGTGATCA-TGATAGAGGCATCTCAGGAGCATCAGTTTAGTGAGGATCCAAGATGCTCTGGC-AGCATGTTC

TCTTCACAGTGCAGTGAATTG-G-AAGACTTGACTGCAAATACAAACACCCAGGA-TCC-TTTCTTGATTGGTTCTTCCAAACAAATGAGGCATCAGTCT-GAAAGCCAGGGAG-
TCTTCACAGCACAGTG-C-TGCCCAAGGGTCAACTGCAAATGCAAACTCCCAGGATTCCAATT-TT-ATTCCACCTTCCAAACAGAGGAGTCACCAGTGTGGGAA-TGAGGAAGC

TTGGTCTGAGTGACAAGGAATTGGTTTCAGATGATGAAGAAA-GAGGAACGGGCTTGGAAGAAAATAATCAAGAAGAGCAA-AGCAT-GGATTCAAACTTAGGTGAAGCA-GCAT
TT-TCCTAAGTGACAAGGAATTGATTTCAGATAACGAGGAAATG-GCAACTTGCCTAGAAGAGGATAAT-GACCA-AG-AAGAGGATAGTA-T-AATCCCAGATTCAG-AGGCAT

CTGGGTGTGAGAGTGAAACAAGCGTCTCTGAAGACTGCTCAGGGCTATCCTCTCAGAGTGACATTTTAACCACTCAGCAGAGGGATACCATGCAA-CATAACCTGATAAAGCTCC
CCGGATACGAGAGTGAAAC-A-A-AC-CT-T-T-CTG-A-A-GACT-G-CTCGCAGAGTGATATTTTAACCACTCAGCAGCGGGCGACCATG-AAGTATAACCTGATAAAGCTGC

AGCAGGAAATGGCTGAACTAGAAGCTGTGTTAGAACAGCATGGGAGCCAGCCTTCT-AACAGCTACCCTTCCATCATAAG-TGACTCTTCTGCCCTTGAGGACCTGCGAAATCCA
AGCAGGAAATGGCTCACCTGGAAGCTGTGCTGGAGCAGCGTGGGAACCAGCCTTCTGGCCA-CTCCCCTTCCCTCCT-AGCGGACCCTTGTGCCCTGGAAGACCTGCCAGATCTG

GAA-CAAAGCACATCAGAAAAAGCAGTATTAACTTCACAGAAAAGTAGTGA-ATACCCTATAAGCCAGAATCCAGAAG-GC-CTTTCTGCTGACAAGTTTGAGGT-G-TC-TGC-
GAACCAAA-CATGTCAGGAGCAGCAATTTTAACTTCAAAGAACATTAATGAGA-ATCCTGTAAGCCAAAAT-TTGAAGAGCGC-TTGTGATGACAAATTCCAACTACAACAT-CT

AGATAGTTCTACCAGTAAAAATAAAGAACCA-GGAGTGGAAAGGTCATCCCCTTCTAAATGC-CCATCATTA-G-ATGATAGGTGG-TACATGCACA-G-TTGCTCT-GGGAGTC
GGAGGGTCCCACCAGTGGAGATGACGAGTCAGGGA-TGGGAAGGCCTTCCCCTTTTAAAT-CTCCGTTGGCAGGCA-G-TAGG-GGCT-C-TGCACATGGCTGCTCTAGGCA-TC

TTCAGAATAGAAACTACCCATCTCAAGAGGAGCTCATTAAGGTTGTTGATGTGGAGGAGCAACAGCTGGAAGAGTCTGGGCCACACGATTTGAC-GGAAACA-TCTTACTTGCCA
TTCAAAAGAGAAACTCCCCCTCTCAGGAGGAGCTCCTCCA-G-C-CTG-C-TGGATCAG-A-G-GC-G-T-CA-TCTGAGCCACACAATTCAACAGG-G-CAGTCTTGCCTGCCA

AGGCAAGATCTAGAGGGAACCCCTTACCTGGAATCTGGAATCAGCCTCTTCTCT-G-A-TGACCCTGAATCTGA-TCCTTCTGAAGACAGAGCCCCAGAGTCAGCTCGTGTTGGC
AGGCGAGAGCTAGAAGGAACCCCATACCTGGGATCTGGAATCAGCCTTTTCTCTAGTAGAGACCCCGAATCTGAGTCC-CC-T-A-A-AGAG-C-CAG-CCCA-C-A-T-T-GGC

AACATACCATCTTCAACCTCTGCATTGAAAGTTCCCCAA-TTGAAAGTTGCAGAATCTGCCCAGAGTCCAGCTGCTGCTCATACTACTGATACTGCTGGGTATAATGCAATGGAA
ACCACACCAGCTTCAACCTCTGCACTGAAAATACCCCAAGGT-CAAGTTG-C-T-T-T-C-CGGAGTGCAGCTGCTGCT-G-G-TGCTGATA-A-A-G-C-A-G-TGGTA-GG-A

GAAAGTGTGAGCAGGGA-GAAGCCAGAATTGACAGCTTCA-ACAGAAAGGGTC-AACAAAAGA-ATGTCCATGGTGGTGTCTGGCCTGA-CCCCAGAAGAATTTATG-CTCGTGT
-A-T-TGTGAGCA-AGATAAAGCCGGAATTGACATCTTCAGA-AGAAAGAG-CGGA-TAGAGACATATCCATGGTGGTGTCAGGCTTGACCCCCA-AAGAAGTAATGAC-CGTGC

ACAAGTTTGC-CAGAAAACACCACATCACTTTAACTAATC-TAATTACTGAAGAGACTACTCATGTTGTTATGAAAACAGATGCTGAGTTTGTGTGTGAACGGACACTGAAATAT
AAAAGTTTGCTGA-AAAATACCGCCTCACTTTAACTGA-CGCAATTACTGAGGAGACTACACATGTAATTATAAAAACAGATGCGGAGTTTGTGTGTGAGCGGACACTGAAATAT

TTTCTAGGAATTGCGGGAGGAAAATGGGTAGTTAGCTATTTC-TGGGTGACCCAGTCTATTAAAGAAAGAAAAATGCTGAATGAGCATGATTTTGAAGTCAGAGGAGATGTGGTC
```

```
TTTCTGGGCATTGCAGGAGGAAAGTGGATAGTTAGCTA-TTCATGGGTGGTCCGGTCTATCCAAGAAAGAAGACTTCTGAATGTGCATGAATTTGAAGTCAAAGGAGATGTTGTG

AATGGAAGAAACCACCAAGGTCCAAAGCGAGCAAGAGAATCCCAGGACAGAAAGATCTTCAGGGGGCTAGAAATCTGTTGCTATGGGCCCTTCACCAACATGCCCACAGATCAAC
ACTGGAAGAAATCACCAAGGTCCAAGGCGATCCAGAGAATCCC-GG-GA-AAAGCTCTTCAAGGGCCTACAGGTCTATTGTTGTGAGCCCTTCACCAACATGCCCAAAGATGAGC

TGGA-ATGGATGGTACAGCTGTGTGGTGCTTCTGTGGTGAAGGAGCTTTCATCATTCACCCTTGGCACAGGTG-TCCACCCAATTGTGGTTGTGCAGCC-AGATGCCTGGACAGA
TGGAGA-GGATGCTGCAGCTGTGTGGGGCTTCCGTGGTGAAGGAGCTTCCATCGCTCACCCATGACACAGGTGCT-CATCTAGTTGTGATCGTGCAGCCAAG-CGCCTGGACAGA

GGACAATGGCT-TCCATGCA-ATTGGGCAGATGTGTGAGGCACCTGTGGTGACCCGAGAGTGGGTGTTGGACAGTGTA-GCACTCTACCAGTGCCAGGAGCT-G-G-AC-A-C-C
AGACAGCAACTGCCCA-G-ATATTGGGCAGCTGTGCAAGGCACGTCTTGTGATGTGGGACTGGGTGTTGGACAGTCTATCCA-GCTACCGGTGTCGGGATCTGGATGCCTACCTG

-TAC-C-TGAT-A-C-C-C-CAG-A-TCCCCCAC-AG-C-C-ACT-ACTGA
GTACAGAATATCACCTGTGACAGTAGTGAGCCACAAGACTCCAATGATTAA

--------------------
```

Question 4. (5 points) Longest common subsequence problem

Suppose that you have a very fast machine to execute the Smith-Waterman algorithm with any user-specific substitution cost matrix and linear gap penalty (such special-purpose machines exist and were fashionable in the 90's). How would you use it to solve the longest common subsequence: Given two DNA sequences S and T, find the longest sequence X that is a subsequence of both S and T.

Note: Subsequences and substrings are two different things. A subsequence is made of characters that occur in the right order in the input spring, but not necessarily consecutively. A substring is a set of consecutive characters of the input string. So every substring is a subsequence, but not vice-versa. For example, BOICS is a subsequence of BIOINFORMATICS, but it is not a substring. IRO is neither a subsequence nor a substring of BIOINFORMATICS.

Solution:

The Smith-Waterman scoring scheme is

$$F(i,j) = \max \begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases}$$

From Durbin et al.

whereas LCS's is

$$F(i,j) = \begin{cases} F(i-1,j-1)+1 & \text{if } x_i = y_i \\ max(F(i-1,j), F(i,j-1)) & \text{if } x_i \neq y_i \end{cases}$$

To obtain a scheme equivalent to the above, we set match score 1, mismatch score 0 and indel penalty cost 0.

Then, performing traceback will yield an alignment **containing** the longest common subsequence.

Finally, eliminating pairs with gaps and mismatches from the final alignment yields the longest common subsequence.

Question 5. (15 points) NoDeletion alignment problem

Consider the NoDeletion global alignment problem, which consists of optimally aligning sequences S and T under a linear gap penalty for insertions but where deletions from S to T are not allowed. Let m and n be the lengths of S and T respectively, and let k = n − m (of course, the problem only makes sense if m ⩽ n). Give an algorithm to solve the problem in time O(m*k).

Solution:

In the dynamic programming table, let S be the top sequence (its nucleotides occupy columns) and T be the left sequence (its nucleotides occcupy rows).

No deletions from S to T ⇒ there's **no gaps** in T in the final alignment ⇒ paths in the traceback algorithm **can't go left, only up and to the top-left** ⇒ only a subset of table cells needs to be filled.

We can show that the number of cells that need to be filled is O(mk).

When m = n:

The traceback path can't go up, because it would eventually need to go left **(which isn't allowed here)** to reach the table's top-left cell.

Therefore, only the diagonal needs to be filled:

```
   . S1 S2 S3 S4
 . x
T1    x
T2       x
T3          x
T4             x
```

When m < n:

The traceback path can go up a **limited number of times**. Still, only a subset of cells along the diagonal of the table needs to be filled:

```
   . S1 S2 S3 S4
 . x
T1 x  x
T2    x  x
T3       x  x
T4          x  x
T5             x

Or:

   . S1 S2 S3 S4
 . x
T1 x  x
T2 x  x  x
T3 x  x  x  x
T4    x  x  x  x
T5       x  x  x
T6          x  x
T7             x
```

We can count the filled cells with a formula:

$$(m+1)(n-m+1) = (m+1)(k+1) = O(mk)$$

Therefore the NoDeleletion alignment algorithm has time O(mk).

Since traceback paths can't go left, we modify the scoring scheme accordingly:

$$F(i,j) = max \begin{cases} F(i-1,j-1) + s(x_i, y_i) \\ F(i-1,j) + d \end{cases}$$

F dynamic programming table,
s substitution cost matrix,
d gap penalty

The algorithm is as follows

```
k = m - n
for i = 0:n
    for j = max(i-k, 0):i
        F(i, j) = max(
            F(i-1, j-1) + s(xi, yi),
            F(i-1, j)   + d
        )
```

Question 6. (10 points) Progressive multiple sequence alignment

The progressive alignment algorithm we have seen in class to solve the multiple sequence alignment is not guaranteed to produce an optimal solution. Assume a sum-of-pairs scoring scheme with a linear gap penalty of b = -1 and the cost of mismatches is -1.

Give a simple example of specific set of short DNA strings where the algorithm fails to produce an optimal result. Give the alignment and score produced by the algorithm, as well as the optimal alignment and its score.

Solution:

Let

$$S_1 = ACT$$
$$S_2 = AG$$
$$S_3 = GT$$

We will guess the following phylogenetic tree and apply the MSA algorithm

```
    ACT | S1,2 = ACT | S(1,2),3 = ACT
   /    |        A-G |           A-G
  /\    |            |           G-T
 /   AG |            |
 \      |            |
  \     |            |
   \    |            |
    GT  |            |
```

Assume a match score of 1. Then, the above algorithm scores

$$msa\_score = score(S_1, S_2) + score(S_1, S_3) + score(S_2, S_3) = (1 - 1 - 1) + (-1 - 1 + 1) + (-1 - 1) = -1 -$$

However, the optimal alignment is

```
ACT
AG-
-GT
```

with a score of

$$opt\_score = (1 - 1 - 1) + (-1 - 1 + 1) + (-1 + 1 - 1) = -3$$

In conclusion, the main weakness of MSA is the intermediate freezing of sequence sets during its runtime, which can lead to eventual non-optimal alignments.

Question 7. (10 points) Blast algorithm

Give an example of the two most similar DNA sequences of length 20 that Blast using word length w = 5 will fail to align.

Solution:

Let query q and database D be:

$$q = AAAAAAAAAAAAAAAAAAAA$$
$$D = AAAAGAAAAGAAAAGAAAAG$$

Then, with word size w = 5, no w-mer in q will ever find a match in D. Therefore, the algorithm would not even attempt an extension.

BONUS QUESTION. (20 points) Linear space pairwise alignment

The standard definition of the Needleman-Wunsch algorithm requires O(m*n) space to align sequences of length m and n respectively. In many cases, this is prohibitive. Of course, one can compute the score of the optimal alignment in linear space trivially by only keeping in memory the last two rows of the dynamic programming table. Let’s call this the ForgetThePast-NW algorithm. However, this prevents us from recovering the alignment that achieves that score, since trace-back is impossible.

Describe a O(m+n) space algorithm to compute both the score and the alignment itself. Hint: Think Divide-and-Conquer! Find out where the path corresponding to the optimal alignment will cross the row m/2, using two calls to a modified ForgetThePast-NW. Then, recurse...