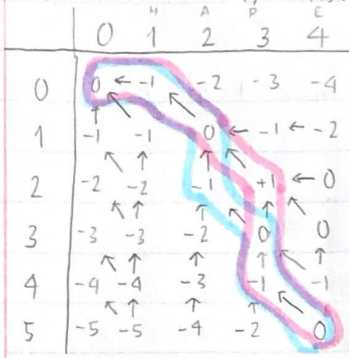
COMP 561 Ass 1:

1) match = +1, mismatch = -1, indel = -1 * L



opt sol #1 (pink)

- A P P L E
H A P - - E

} cost = 0

opt sol #2 (blue)

- A P P L E
H A - P - E

} cost = 0

these are the optimal global alignments.

2) let S = C, T = CCC

using linear gap penalty (with indel = -1 * L):

3 alignments to consider

① S' - C -
T' C C C

score = -1 + 1 - 1 = -1

② S' C - -
T' C C C

score = +1 - 1(2) = -1

③ S' - - C
T' C C C

score = -1(2) + 1 = -1

so all 3 alignments are optimal

using affine gap penalty (with indel = -2 - 0.5 L)

considering same 3 alignments

① S' - C -
T' C C C

score = -2 - 0.5(1) + 1 - 2 - 0.5(1)
= -4

② S' C - -
T' C C C

score = 1 - 2 - 0.5(2)
= -2

③ S' - - C
T' C C C

score = -2 - 0.5(2) + 1
= -2

so opt global alignments are only 2 & 3 →
different from result using linear gap penalty (1 was also opt).

3) a) you can only have multi-gap if it is possible to have 2+ consecutive gaps

ie CANT have if S = A, T = AA (can have max 1 gap)

so constraint is $|m-n| \leq \min(m,n) + 1$

b) Algo: // assumes input M, b, S, T (b < 0)

Initializing:

```
for i = 1 → m:
    X_{i,0} = b * i
    P_{i,0} = ↑          // let P = pointer matrix
for j = 1 → n:
    X_{0,j} = j * b
    P_{0,j} = ←
P_{0,0} = 0              // know when to stop
```

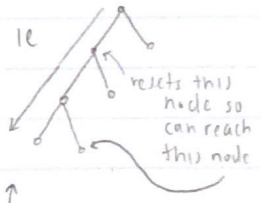
calculation:

```
for i = 1 → m:
    for j = 1 → n
        maxval = max(X_{i,j} + M(S_i, T_j), X_{i-1,j} + b, X_{i,j-1} + b)
        if maxval = X_{i,j} + M(S_i, T_i):
            P_{i,j} = ↖
        if maxval = X_{i-1,j} + b:
            if X_{i-1,j-1} doesn't contain ↑:    // prevents having ↑↑ as only option
                P_{i,j} += ↑
        if maxval = X_{i,j-1} + b:               // prevents having ←← as only option
            if X_{i,j-1} doesn't have ←
                P_{i,j} += ←
        if P_{i,j} is empty:
            maxval = X_{i,j} + M(S_i, T_j)
            P_{i,j} = ↖
        X_{i,j} = maxval
```

3b) continued:

```
S' = '', T' = '', p_copy = p.copy()
num = 0, iters = 1, prev = [], branch = []
while iters > 0:        // continue until iters = 0 OR p(m,n) = ''
    if num != 0:
        S'.append('')
        T'.append('')
    i = m
    j = n
    lastS = ''
    lastT = ''
    others = ''
    while (i > 0 or j > 0):
        if (len(P_i,j) = 0):
            if not others:    // options left in previous step &
                              current step is empty
                S'[num] = '', T'[num] = ''  //empty S' & T'
                iters += 1
                num -= 1
            else:             // no options left
                S'[num] = ''
                T'[num] = ''
                break
        if (len(P_i,j) > 1)   // multiple options (ie. ↑ or ←)
            iters += len(P_i,j) - 1
            branch.append((i,j))
            others = P_i,j [:-1]
            move = P_i,j [-1]
        else:                 //only one option
            move = P_i,j
            others = move
```

```
if move = ↖
    S'[num] = S[i-1] + S'[num]
    T'[num] = T'[j-1] + T'[num]
    i -= 1
    j -= 1
    lastS = S[i-1]
    lastT = T[j-1]
elif move = ←
    if lastS != '-':
        S'[num] = "-" + S'[num]
        T'[num] = T[j-1] + T'[num]
        lastS = '-', lastT = T[j-1]
        j -= 1
    else:
        i=0, j=0
        S'[num] = '', T'[num] = ''
elif move = ↑
    if lastT != '-'
        S'[num] = S[i-1] + S'[num]
        T'[num] = '-' + T'[num]
        lastS = S[i-1], lastT = "-"
        i = -1
    else:
        i=0, j=0
        S'[num] = '', T'[num] = ''
iters -= 1
numt = 1
if branch:
    if prev:
        if prev[-1] < branch[-1] then    P[prev[-1] coords] = P_copy[prev[-1]],
                                         prev = prev[:-1]
        P[branch[-1] coords] = P[branch[-1] coords][:-1] //removes last operation
```



// this rejects part of branch

```
        prev.append(branch[-1])
        branch = branch[:-1]
    if p_min = 'h': break

// print alignments & scores
for k = 0 → len(s'):
    print s'[k]
    print T'[k]
print("SCORE", X_min).
```

3c) see Q3c.py for code, Q3_results.txt for results

* https://www.ics.uci.edu/~goodrich/teach/cs260P/notes/LCS.pdf

4) let $X_{i,j}$ be the length of the longest common subsequence of $S = S_1 \dots S_i$ and $T = T_1 \dots T_j$

- we know that smith-waterman (S-W) algorithm computes

$$X_{i,j} = \begin{cases} 0 \\ X_{i-1,j-1} + M(S_i, T_j) \\ X_{i-1,j} + C \\ X_{i,j-1} + C \end{cases}$$

- If want longest common subsequence, want ↖ if is a match, don't allow ↖ if is a mismatch (meaning only consider ← or ↑).

CHANGES:

① set $C = 0$ → subsequence allows for gaps, so don't want to penalize → also, LCS algo from source says, if $S_i \neq T_j$, $X_{i,j} = \max(X_{i-1,j}, X_{i,j-1})$ *

② set $M(S_i, T_j) = \begin{cases} 1 \text{ if } S_i = T_j \\ -\min(\text{len}(S), \text{len}(T)) \end{cases}$

↳ is because, if mismatch, don't want to consider ↖, so want

$X_{i-1,j-1} + M(S_i, T_j) \leq 0$

so $M(S_i, T_j) \leq -(X_{i-1,j-1})$

↳ worst case = only matches before

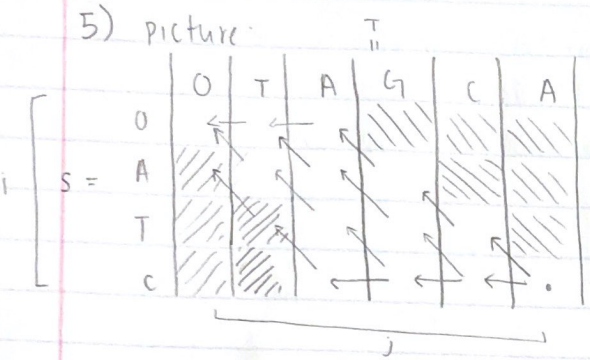
		G	A	A	A
	0	0	0	0	0
A	0	1	←1	←1	←1
T	0	↑1	X		

↳ DONT want: $1 - 2 = -1 < 0$

So, run S-W algo, with above changes.

- Then, during traceback, start NOT at $X_{i,j}$ but rather at $\max(X_{i,j})$. Additionally, count S_i or T_j in X if ↖ (don't count if ← or ↑ because is nucleotide only present in one of the sequences).

5) picture:



deletion from S NOT allowed → so "↑" not allowed

//// = can't go because would require ↑

let $S = S_1 \dots S_m$, $T = T_1 \dots T_n$, with $m \leq n$, $k = m - n$,

b = linear gap penalty ($b < 0$)

algo: for $j = 0 \rightarrow k$:

$X_{0,j} = b * j$

// DON'T need to initialize $X_{i,0}$ b/c can never reach

for $i = 1 \rightarrow m$: // because can reach all rows

for $j = 1 \rightarrow k+1$:

$$X_{i,j} = \max \begin{cases} X_{i-1,j-1} + M(S_i, T_j) \\ X_{i,j-1} + b \end{cases}$$

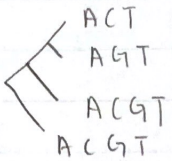
// give matrix "←" or "↑"

// do same traceback as in N-W algo

↳ this is $O(m * k)$

6) given $c = -1$, mismatch $= -1$, match $= 0$

assume



algo gives:

```
AC-T        or      A-CT
AG-T                A-GT
ACGT                ACGT
ACGT                ACGT
```

score = (-1) + (-1) + (-1) + (-1-1) + (-1-1) = -7

score = (-1) + (-1-1) + (-1-1) + (-1) + (-1) = -7

optimal solution is:

```
AC-T
A-GT
ACGT
ACGT
```

score = (-1-1) + (-1) + (-1) + (-1) + (-1) = -6

↳ score of $-6 > -7$ → so algo DOESN'T give optimal result.

7) S = AAAAC AAAAC AAAAC AAAAC

T = AAAAA AAAAA AAAAA AAAAA

↳ never get 5-mers that match, so fail to align