xzhu42 Xuanyi Zhu 670389878

Q1：Sequence Alignment

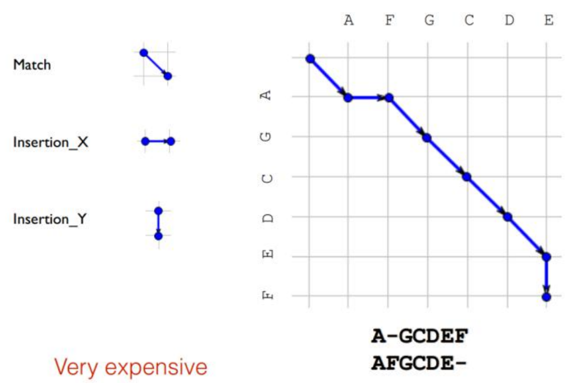
Calculate and show the Dynamic Programming matrix and an optimal alignment for the DNA sequences GCATTGC and GATTAGC, scoring +2 for a match, -1 for a mismatch, and a gap penalty of 2 (i.e., each gap column contributes -2). (If there are more than one optimal alignments, you can report any of them.)

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \text{substitution} \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

scoring +2 for a match,
-1 for a mismatch(substitution),
and a gap penalty(deletion/insertion) of 2 (i.e., each gap column contributes -2)

	*	G	A	T	T	A	G	C
*	0	-2	-4	-6	-8	-10	-12	-14
G	-2	2	0	-2	-4	-6	-8	-10
C	-4	0	1	-1	-3	-5	-7	-6
A	-6	-2	2	0	-2	-1	-3	-5
T	-8	-4	0	4	2	0	-2	-4
T	-10	-6	-2	2	6	4	2	0
G	-12	-8	-4	0	4	5	6	4
C	-14	-10	-6	-2	2	3	4	8



GCATT-GC

G-ATTAGC

Q2： Pairwise Sequence Alignment

Find the optimal pairwise global alignment of the sequences TCTGC**C**TCTGC and ACTGAC**C**ACTGAC with the condition that the C nucleotides shown in bold font must be aligned with each other. What is the optimal alignment score? The scoring parameters are defined as +2 for match, -1 for mismatch and -2 for gap. Show the dynamic programming matrix you used. (And please do not create a 11x13 matrix!)

	*	A	C	T	G	A	C
*	0	-2	-4	-6	-8	-10	-12
T	-2	-1	-3	-2	-4	-6	-8
C	-4	-3	1	-1	-3	-5	-4
T	-6	-5	-1	3	1	-1	-3
G	-8	-7	-3	1	5	3	1
C	-10	-9	-5	-1	3	1	5

TCTG-C C TCTG-C

ACTGAC C ACTGAC

5 + 2 + 5 = 12

Q3:Pattern Matching

Build a keyword tree, with fail edges (as in the Aho-Corasick algorithm) for the following dictionary. Mark with dashed arrows the "fail edges" that do not go back to the root. That is, you do not need to show fail edges going back to the root. You may, but are not required to mark the nodes that correspond to pattern matches.Count and note down the number of fail edges your keyword tree has. (Again, only the fail edges you marked, i.e., fail edges that do not go back to the root.)

SMEAR

EARN

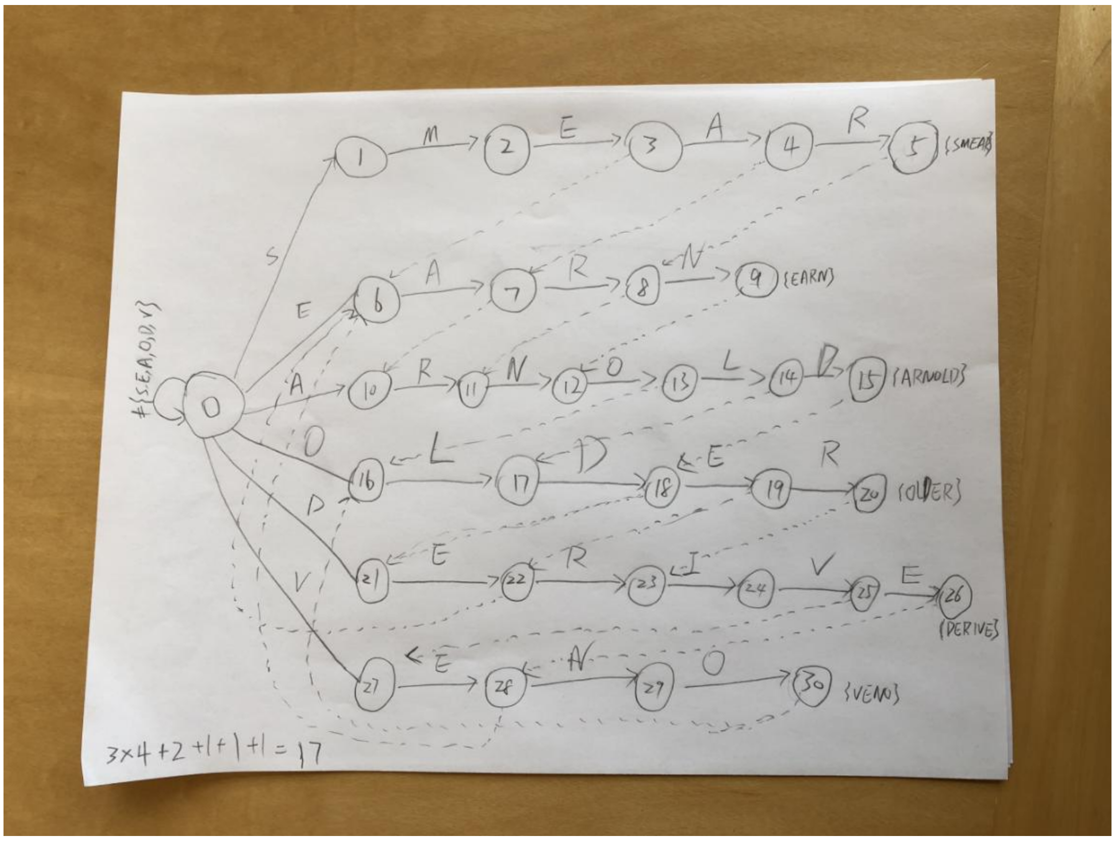
ARNOLD

OLDER

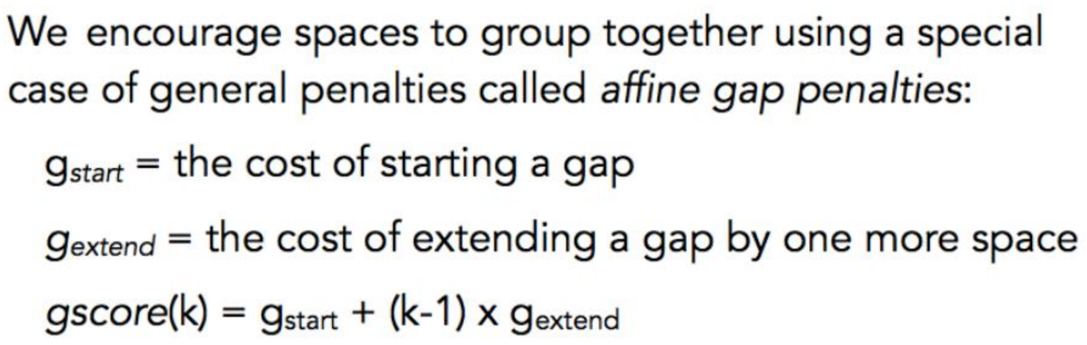
DERIVE

VENO

the number of fail edges your keyword tree has : 17



Q4:Sequence alignment
Consider the affine gap penalty:
gap_start = the cost of starting a gap
gap_extend = the cost of extending a gap by one more space


We encourage spaces to group together using a special case of general penalties called *affine gap penalties*:

g_{start} = the cost of starting a gap

g_{extend} = the cost of extending a gap by one more space

$gscore(k) = g_{start} + (k-1) \times g_{extend}$

1. Write down a dynamic programming algorithm for global pairwise alignment
with the affine gap penalty. You can assume that the match score and mismatch
score are also given.
2. Provide the complexity of your algorithm. Is it optimal?

Assumption:
The size of 1st sequence X is m
The size of 2nd sequence Y is n
(match + mismatch/substitution)
M[i,j]= store the best alignment score of x[1..i] and y[1..j] ending with a match or mismatch pair.
(gap)
X[i,j]= store the best alignment score of x[1..i] and y[1..j] ending with a gap in X.
Y[i,j]= store the best alignment score of x[1..i] and y[1..j] ending with a gap in Y.

Base Case:
M[i, 0] = M[0, i] = Integer.MIN_VALUE;
X[0, i] = gap_start + i*gap_extend;
X[i, 0] = Integer.MIN_VALUE;
Y[0, i] = Integer.MIN_VALUE;
Y[i, 0] = gap_start + i*gap_extend;

Induction Rule:

```
M [i, j] = Math.max(M[i - 1, j - 1], X[i - 1, j - 1], Y [i - 1, j - 1])    +    match(i, j)

X[i, j] = Math.max([gap_start + gap_extend] + M[i, j -1],
                   [gap_extend] + X[i, j-1],
                   [gap_start + gap_extend] + Y[i, j - 1])

Y [i, j] = Math.max([gap_start + gap_extend] + M[i - 1, j],
                    [gap_start + gap_extend] + X[i- 1, j],
                    [gap_extend] + Y[i - 1, j])
```

TIME COMPLEXITY:
Since we maintain 3 m*n matrices, there are totally 3mn cells we need to fill in, each cell takes O(1) time to fill in, thus total runtime is O(mn)