Sequence Alignment

Mark Voorhies

5/29/2013

Exercise: Scoring an ungapped alignment

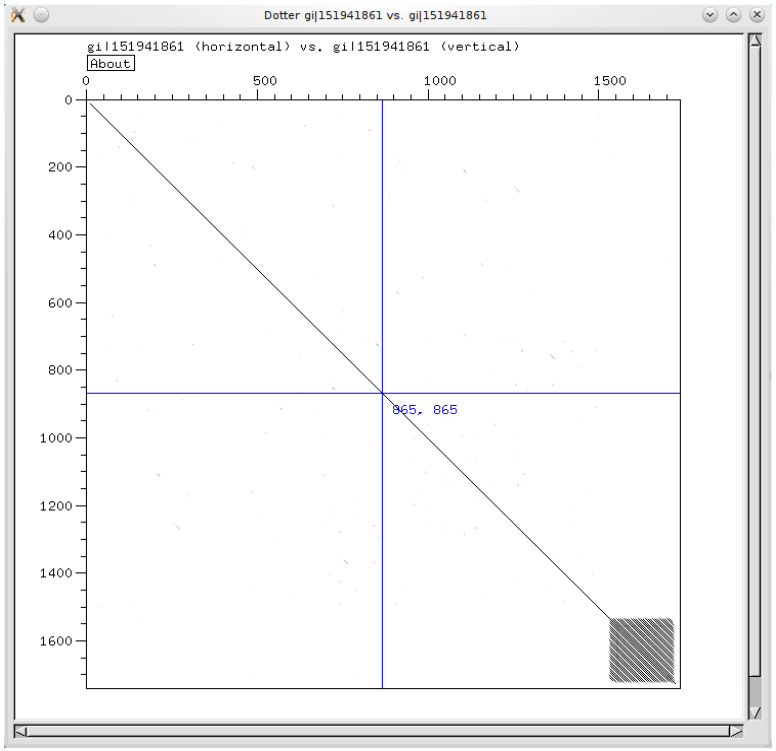
Given two sequences and a scoring matrix, find the offset that yields the best scoring ungapped alignment.

Exercise: Scoring an ungapped alignment

Given two sequences and a scoring matrix, find the offset that yields the best scoring ungapped alignment.

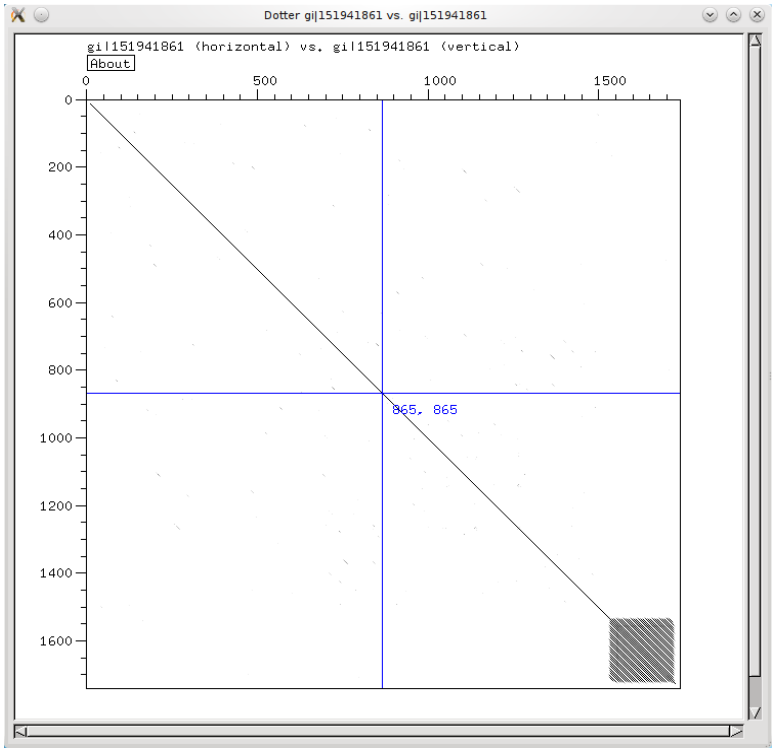
```
def score(S,x,y):
    """Return alignment score for subsequences x and y
    for scoring matrix S (represented as a dict)"""
    assert(len(x) == len(y))
    return sum(S[i][j] for (i,j) in zip(x,y))

def subseqs(x,y,i):
    """Return subsequences of x and y for offset i."""
    if(i > 0):
        y = y[i:]
    elif(i < 0):
        x = x[-i:]
    L = min(len(x),len(y))
    return x[:L],y[:L]

def ungapped(S, x, y):
    """Return best offset, score, and alignment
    between sequences x and y for matrix S."""
    best = None
    best_score = None
    for i in range(-len(x)+1, len(y)):
        (sx,sy) = subseqs(x,y,i)
        s = score(S,sx,sy)
        if(s > best_score):
            best_score = s
            best = i
    return best, best_score, subseqs(x,y,best)
```

Dotplots


Dotter gi|151941861 vs. gi|151941861
gi|151941861 (horizontal) vs. gi|151941861 (vertical)
About
0
500
1000
1500
0
200
400
600
800
1000
1200
1400
1600
865, 865

1

Dotplots



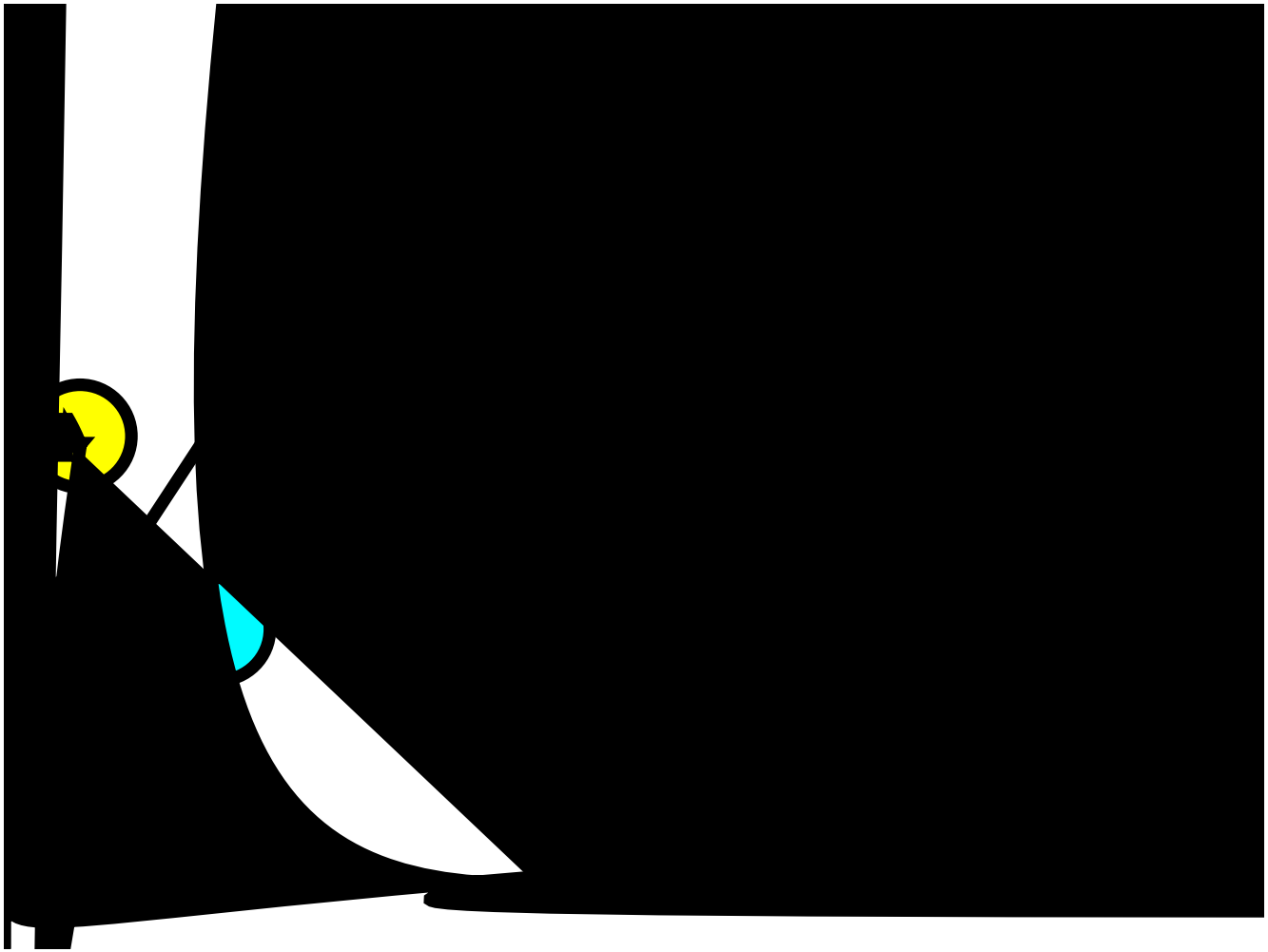
1. Unbiased view of all ungapped alignments of two sequences
2. Noise can be filtered by applying a smoothing window to the diagonals.

Exercise: Scoring a gapped alignment

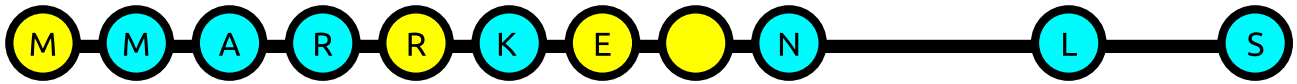
1. Given two equal length gapped sequences (where "-" represents a gap) and a scoring matrix, calculate an alignment score with a -1 penalty for each base aligned to a gap.
2. Write a new scoring function with separate penalties for opening a zero length gap (*e.g.*, G = -11) and extending an open gap by one base (*e.g.*, E = -1).

$$S_{gapped}(x, y) = S(x, y) + \sum_{i}^{gaps} (G + E \quad len(i))$$

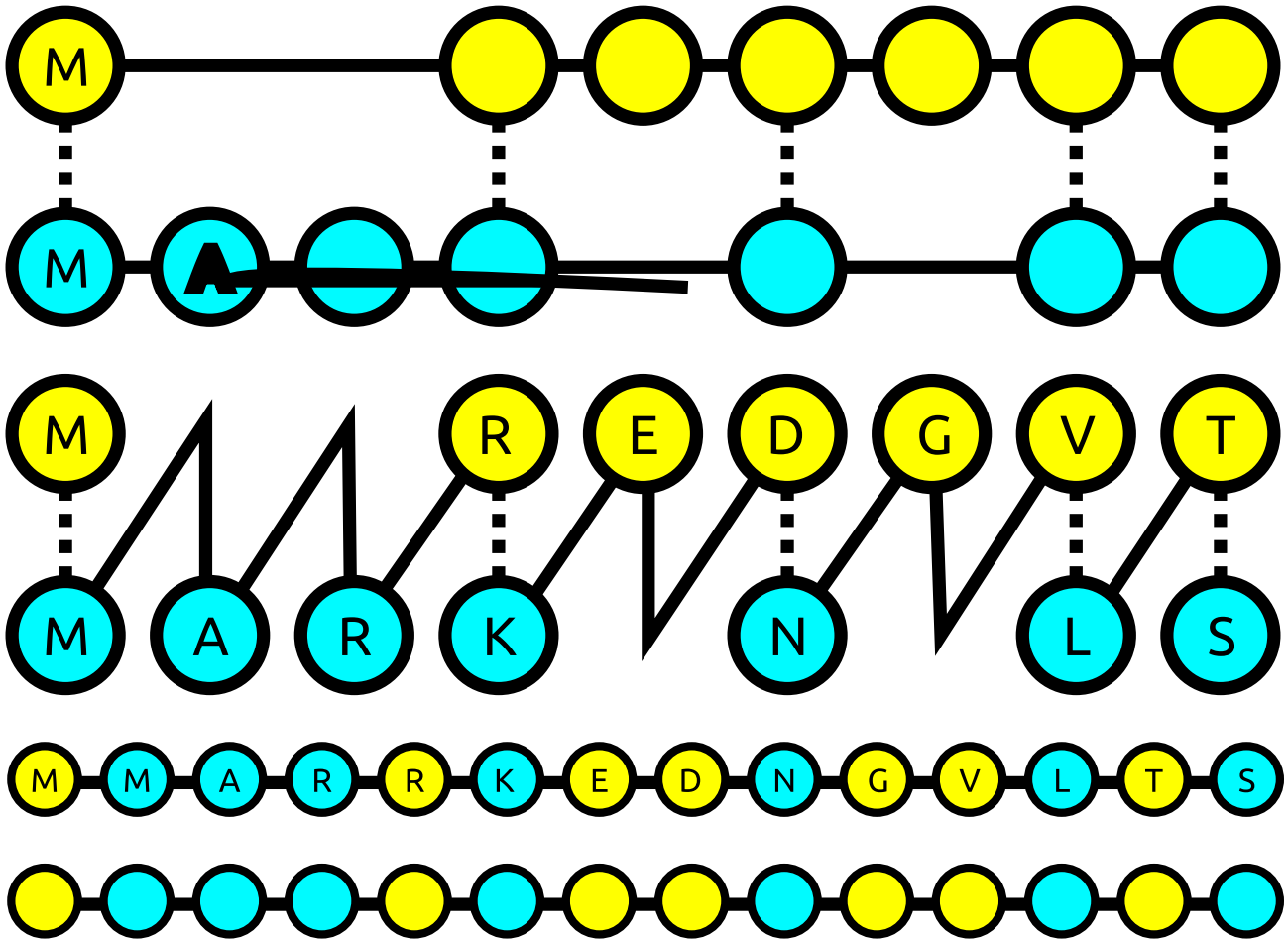
How many ways can we align two sequences?



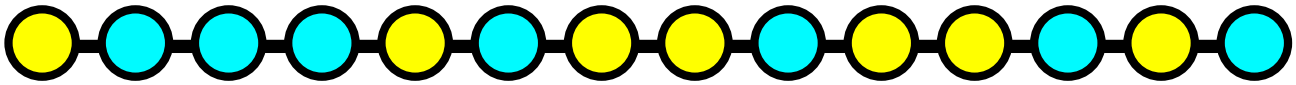
How many ways can we align two sequences?



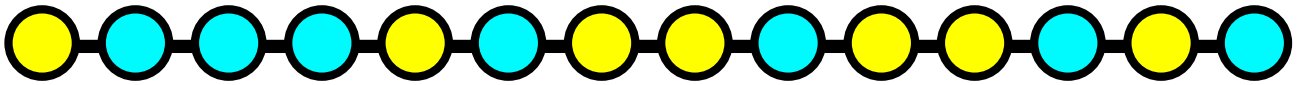
How many ways can we align two sequences?



How many ways can we align two sequences?



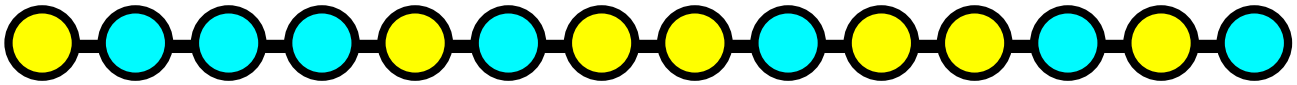
How many ways can we align two sequences?



Binomial formula:

$$\binom{k}{r} = \frac{k!}{(k \quad r)!r!}$$

How many ways can we align two sequences?

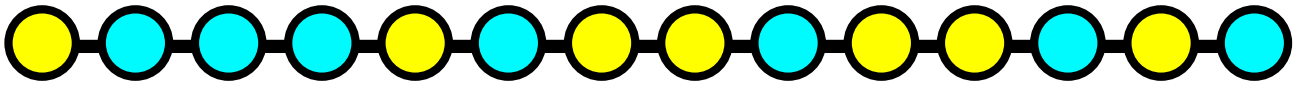


Binomial formula:

$$\binom{k}{r} = \frac{k!}{(k \quad r)!r!}$$

$$\binom{2n}{n} = \frac{(2n)!}{n!n!}$$

How many ways can we align two sequences?



Binomial formula:

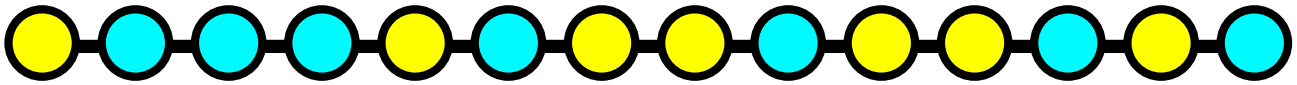
$$\binom{k}{r} = \frac{k!}{(k \quad r)!r!}$$

$$\binom{2n}{n} = \frac{(2n)!}{n!n!}$$

Stirling's approximation:

$$x! \qquad \sqrt{2\pi}\left(x^{x+\frac{1}{2}}\right)e^{-x}$$

How many ways can we align two sequences?



Binomial formula:

$$\binom{k}{r} = \frac{k!}{(k \quad r)!r!}$$

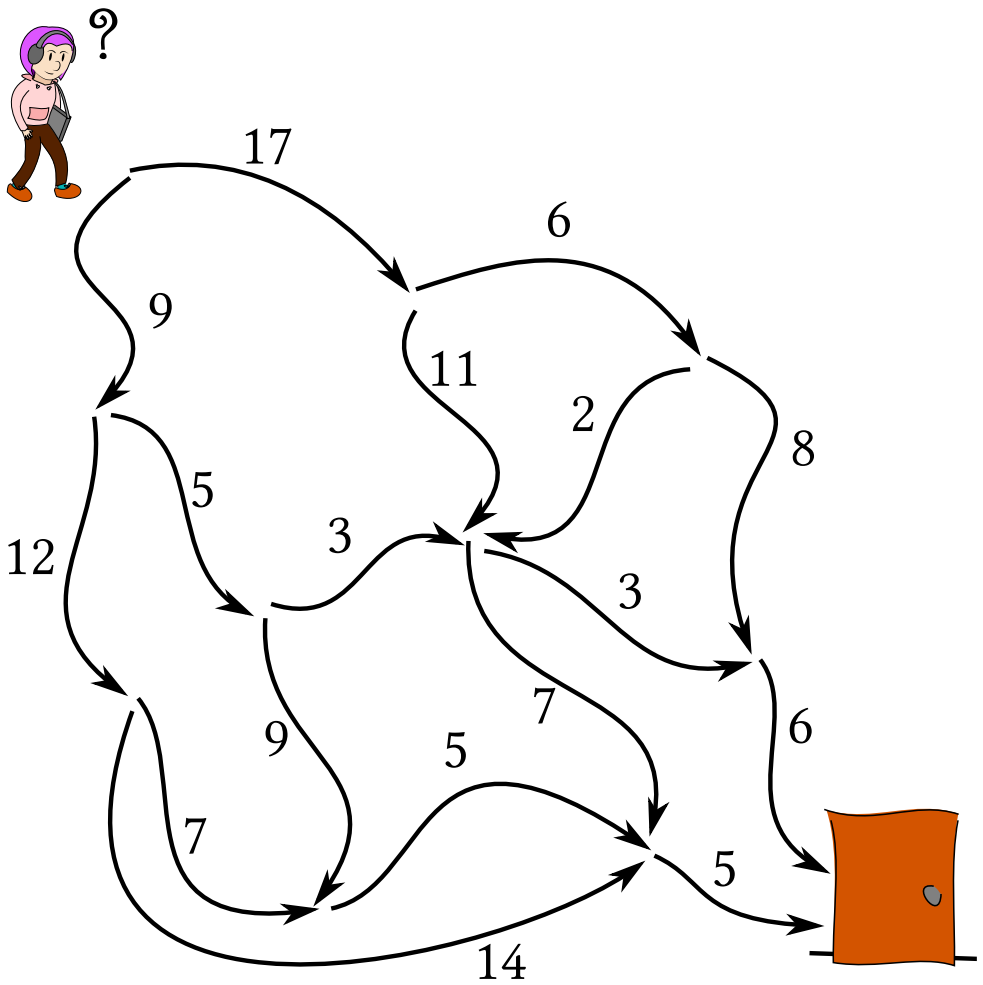
$$\binom{2n}{n} = \frac{(2n)!}{n!n!}$$

Stirling's approximation:

$$x! \quad \sqrt{2\pi}\left(x^{x+\frac{1}{2}}\right)e^{-x}$$

$$\binom{2n}{n} \quad \frac{2^{2n}}{\sqrt{\pi n}}$$

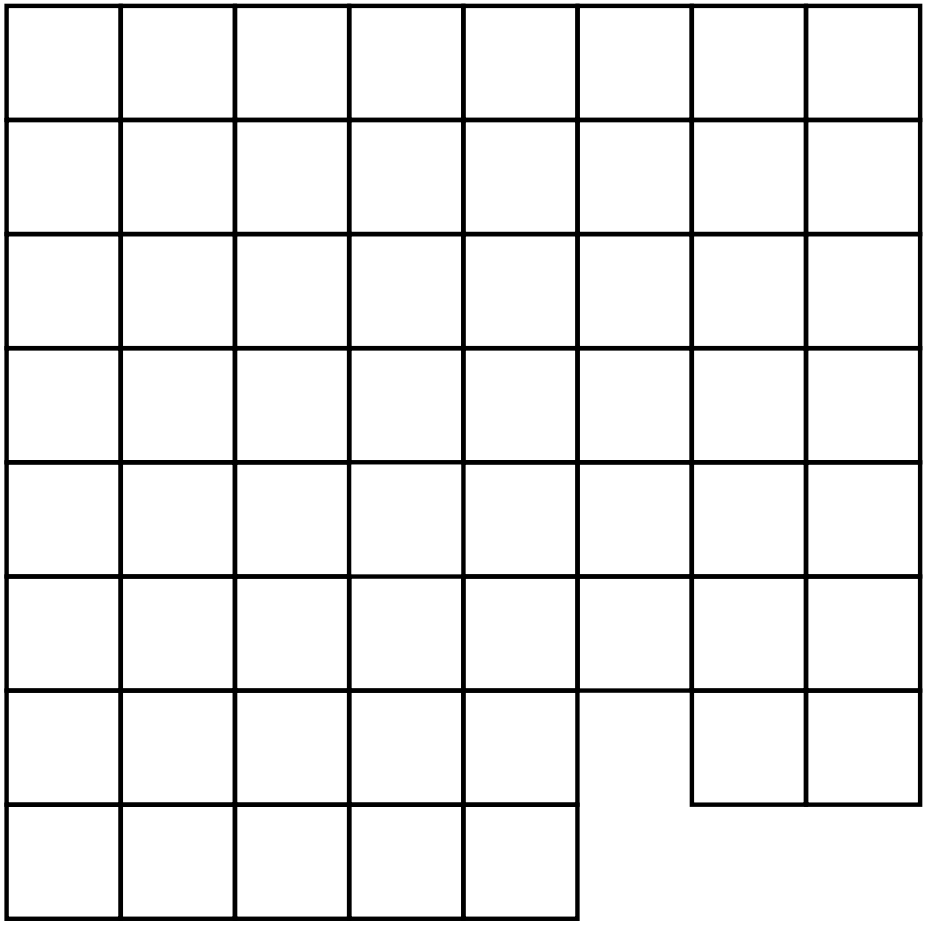
Dynamic Programming



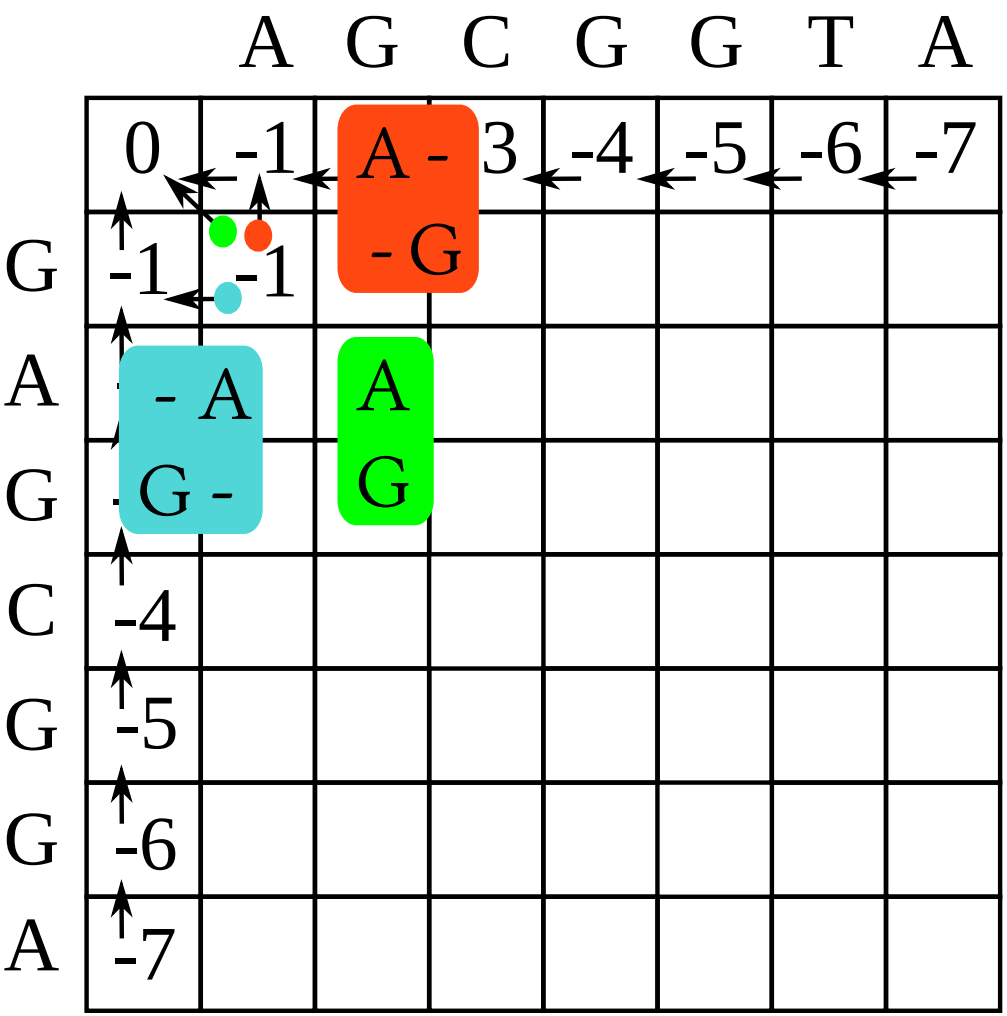
Needleman-Wunsch

		A	G	C	G	G	T	A
G								
A								
G								
C								
G								
G								
A								

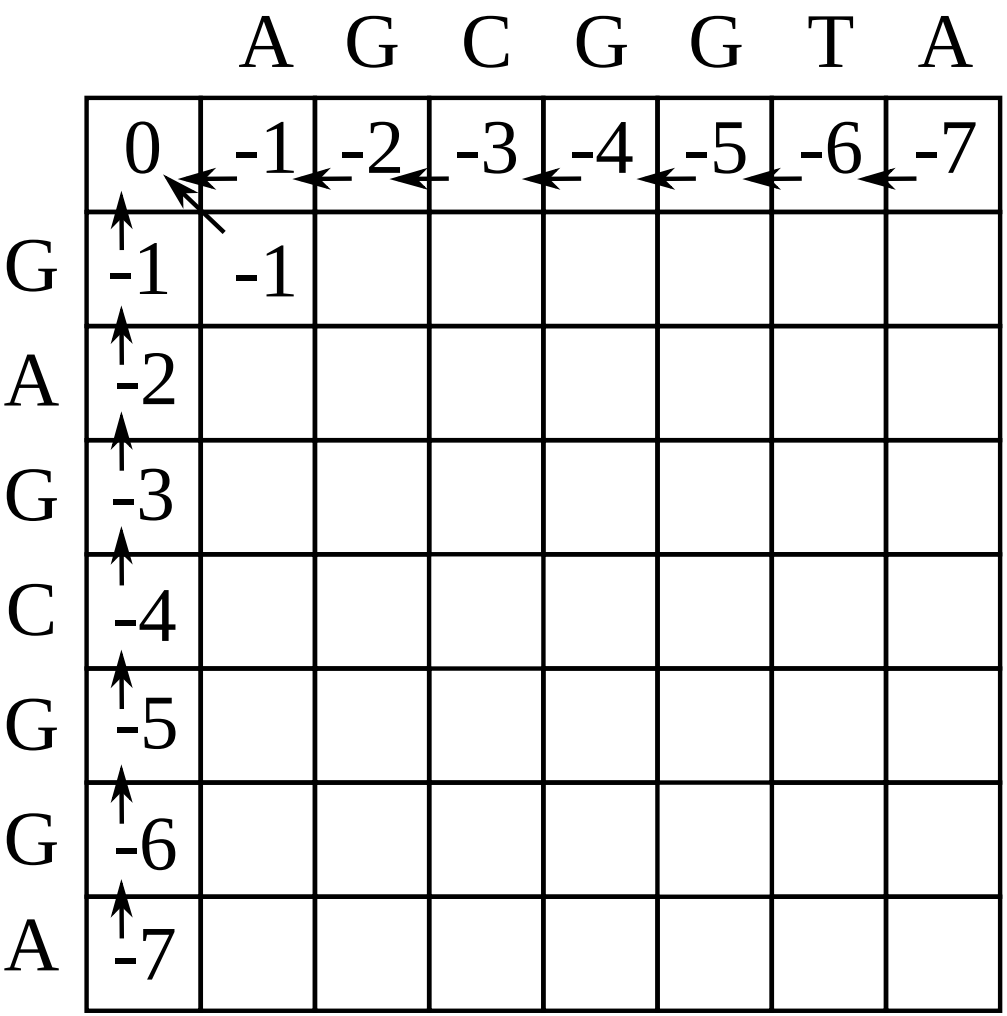
Needleman-Wunsch



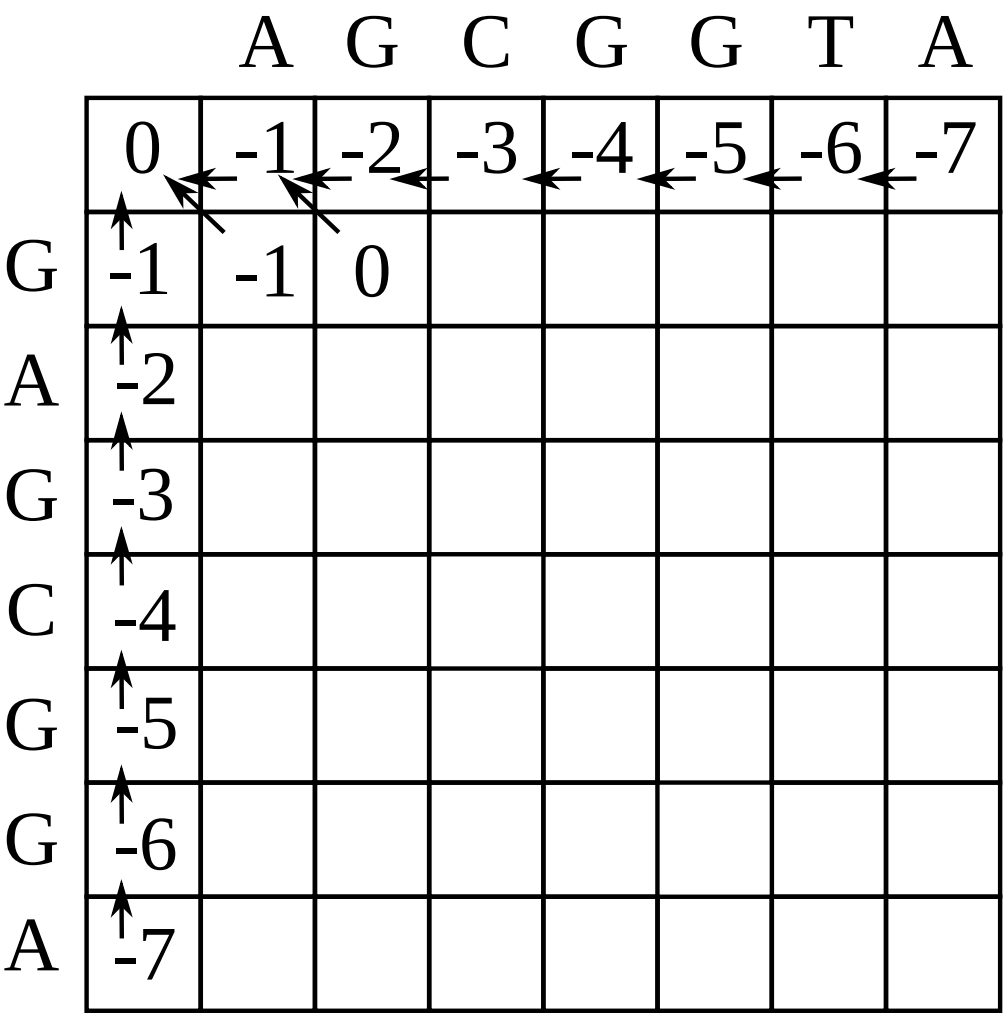
Needleman-Wunsch



Needleman-Wunsch



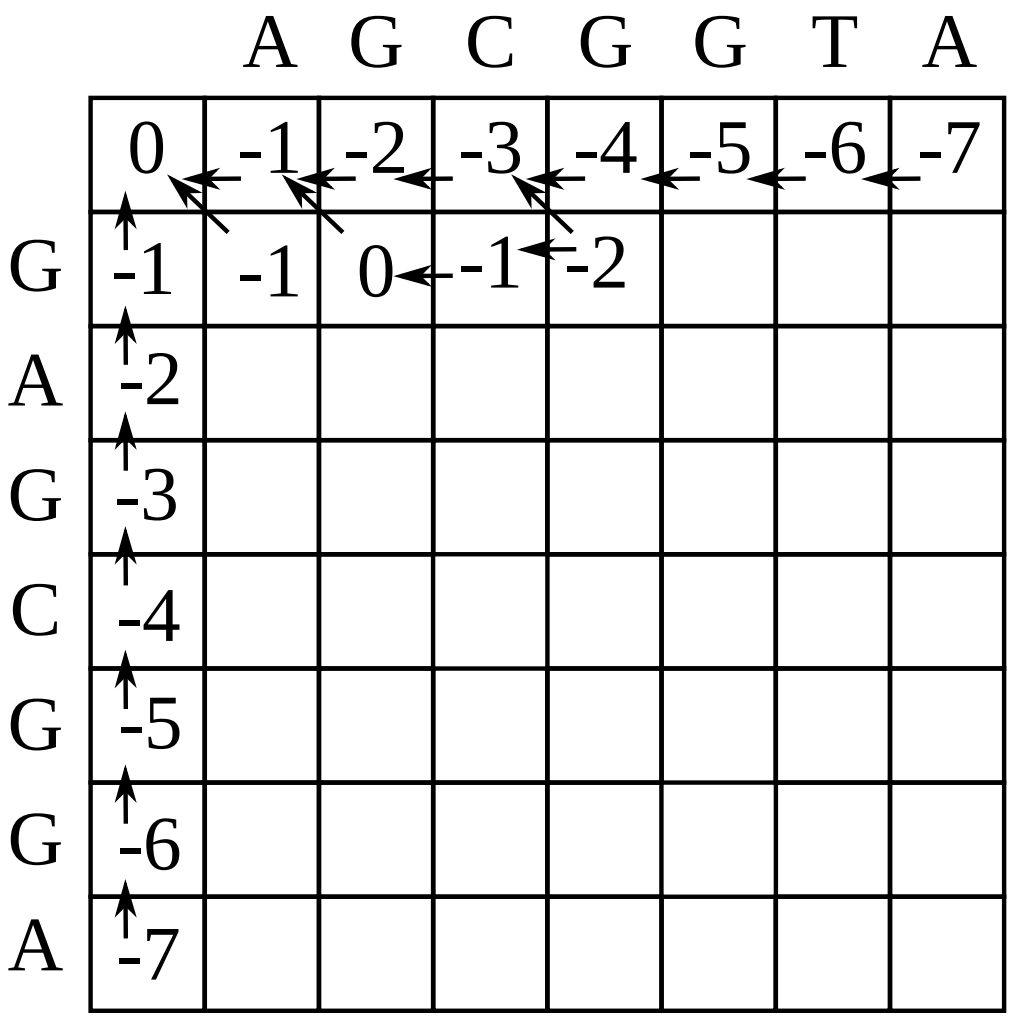
Needleman-Wunsch



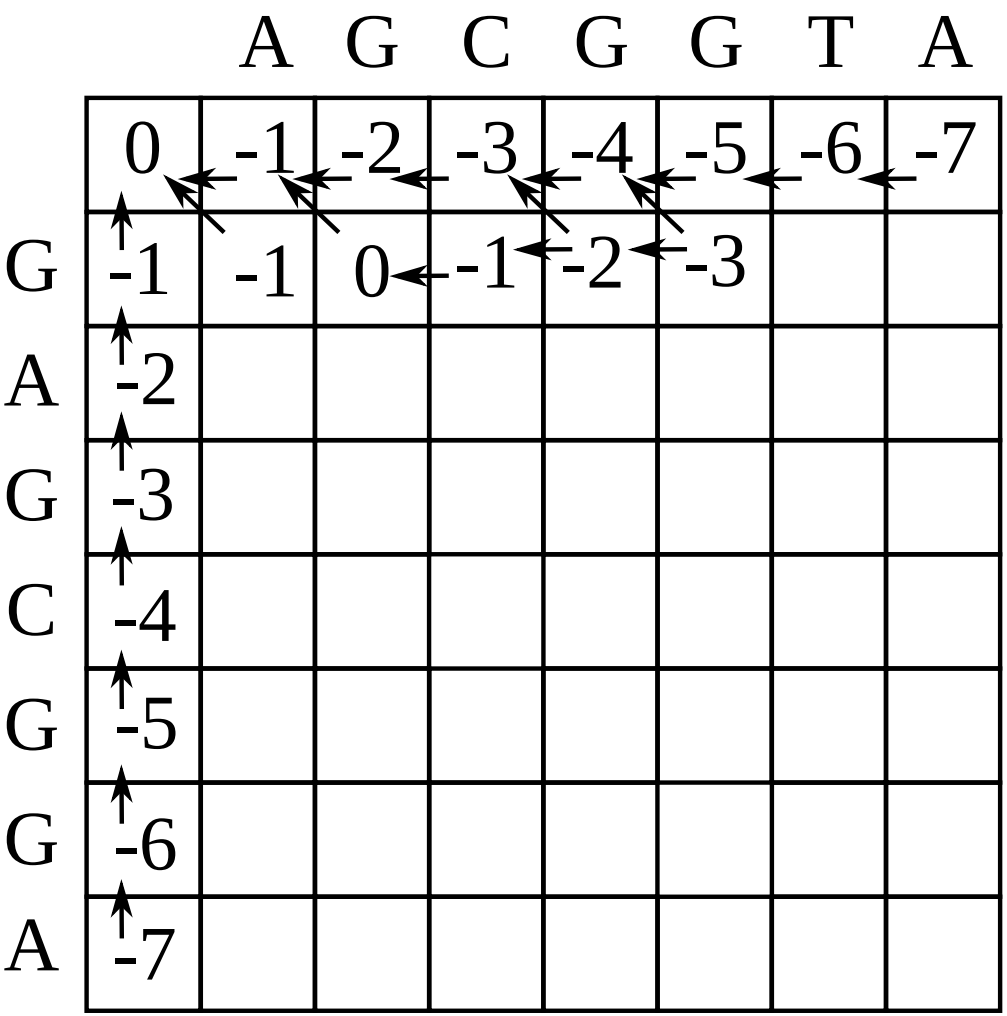
Needleman-Wunsch

		A	G	C	G	G	T	A
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	-1	0	-1				
A	-2							
G	-3							
C	-4							
G	-5							
G	-6							
A	-7							

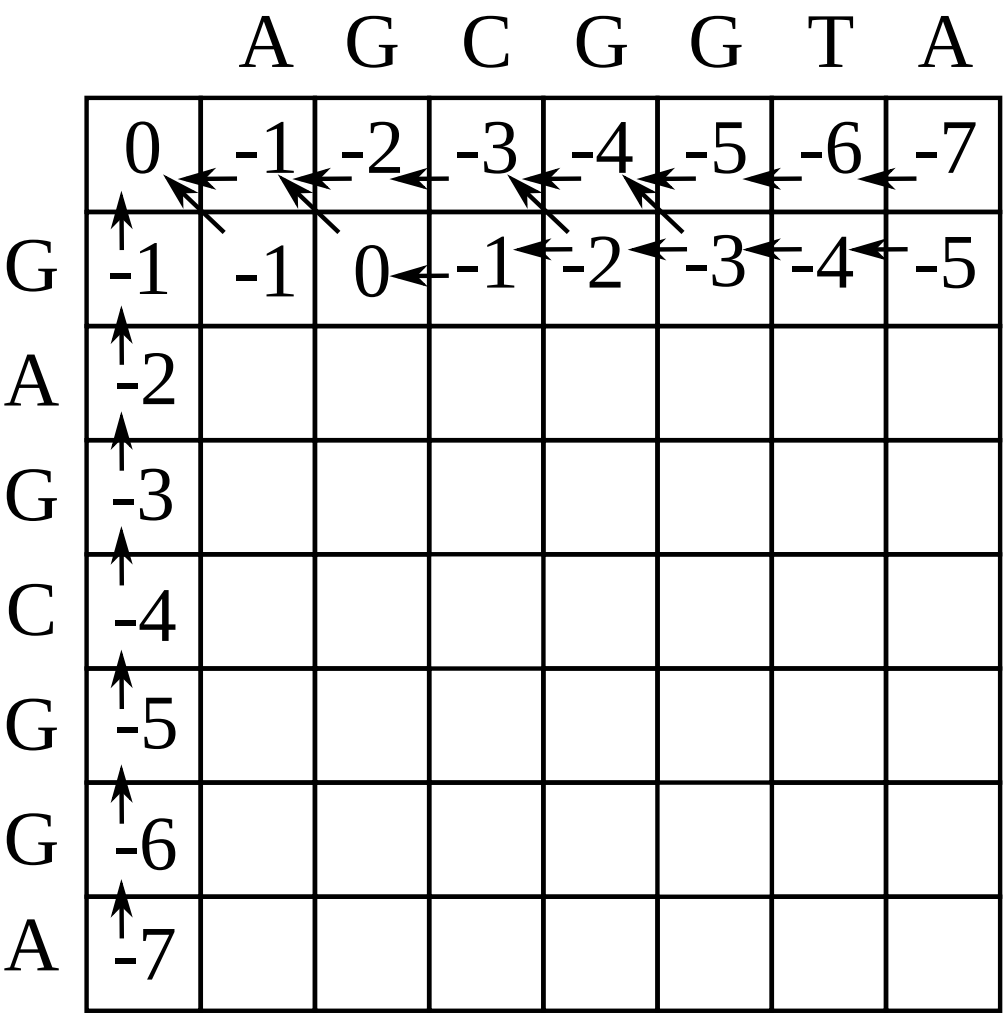
Needleman-Wunsch



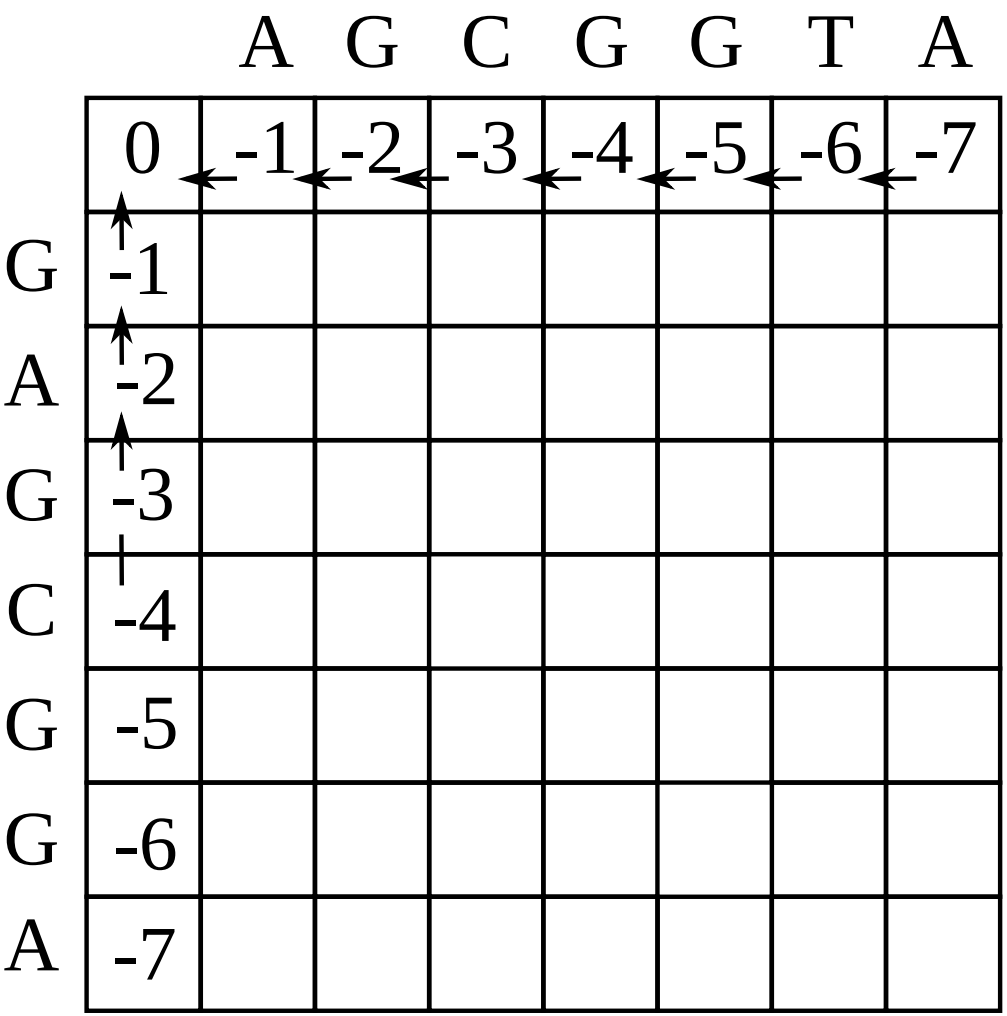
Needleman-Wunsch



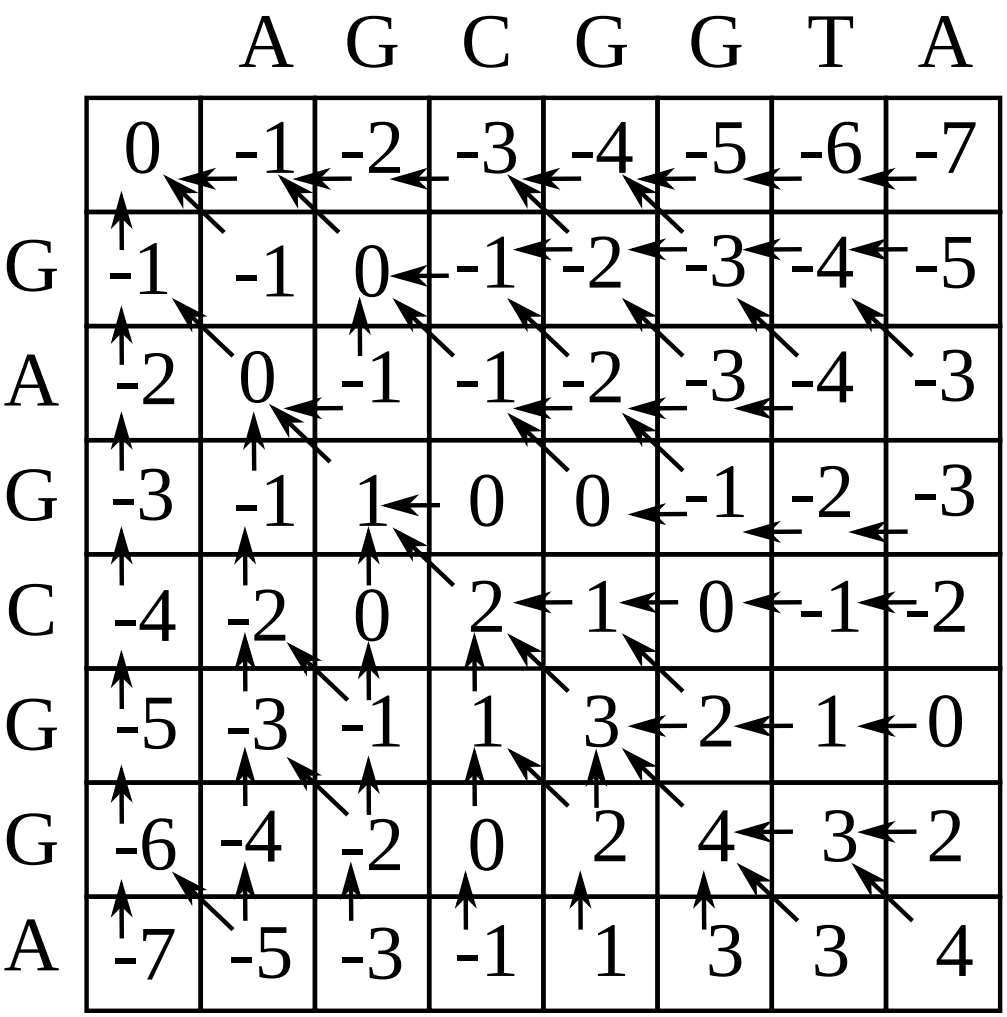
Needleman-Wunsch



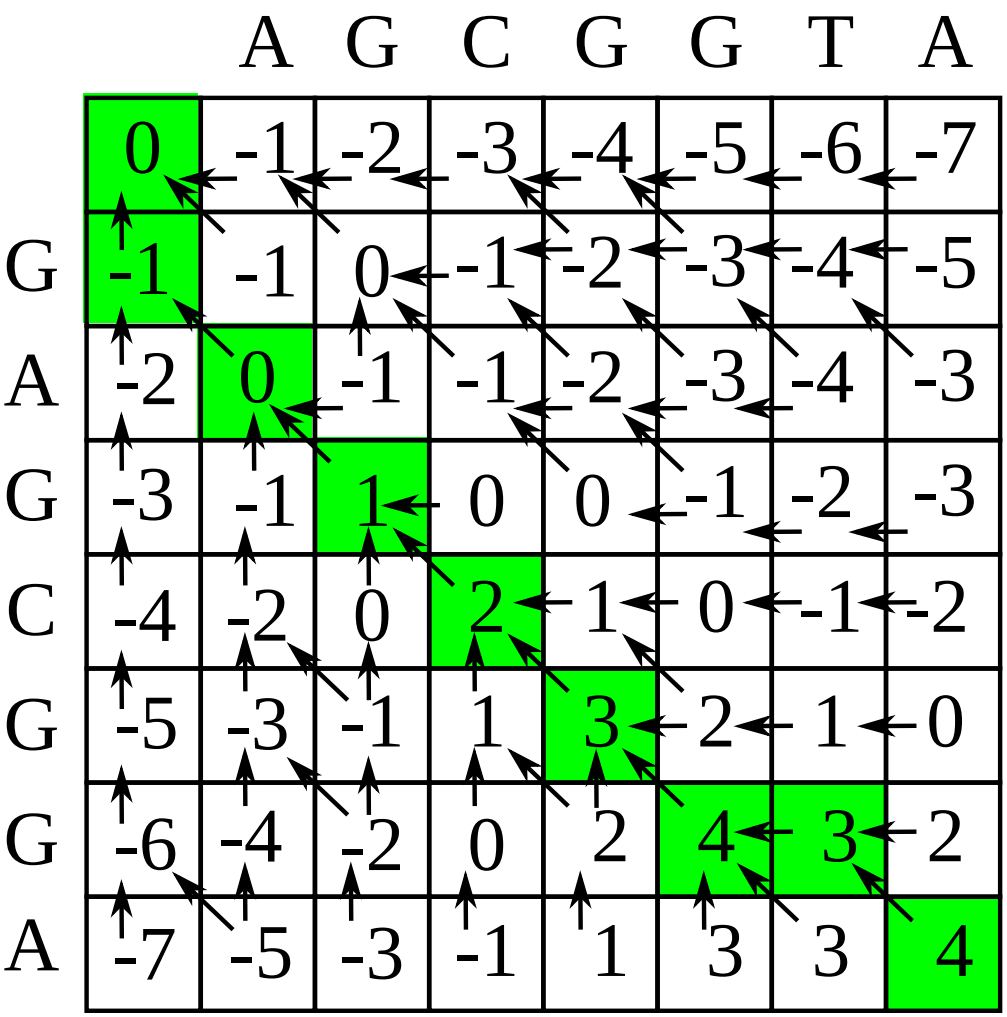
Needleman-Wunsch



Needleman-Wunsch



Needleman-Wunsch



Homework

Implement Needleman-Wunsch global alignment with zero gap opening penalties. Try attacking the problem in this order:

1. Initialize and fill in a dynamic programming matrix by hand (*e.g.*, try reproducing the example from my slides on paper).
2. Write a function to create the dynamic programming matrix and initialize the first row and column.
3. Write a function to fill in the rest of the matrix
4. Rewrite the *initialize* and *fill* steps to store pointers to the best sub-solution for each cell.
5. Write a *backtrace* function to read the optimal alignment from the filled in matrix.

If that isn't enough to keep you occupied, read the dynamic programming references from the class website. Try to articulate in your own words the logic for the speed-ups and trade-offs in the Myers and Miller approach.