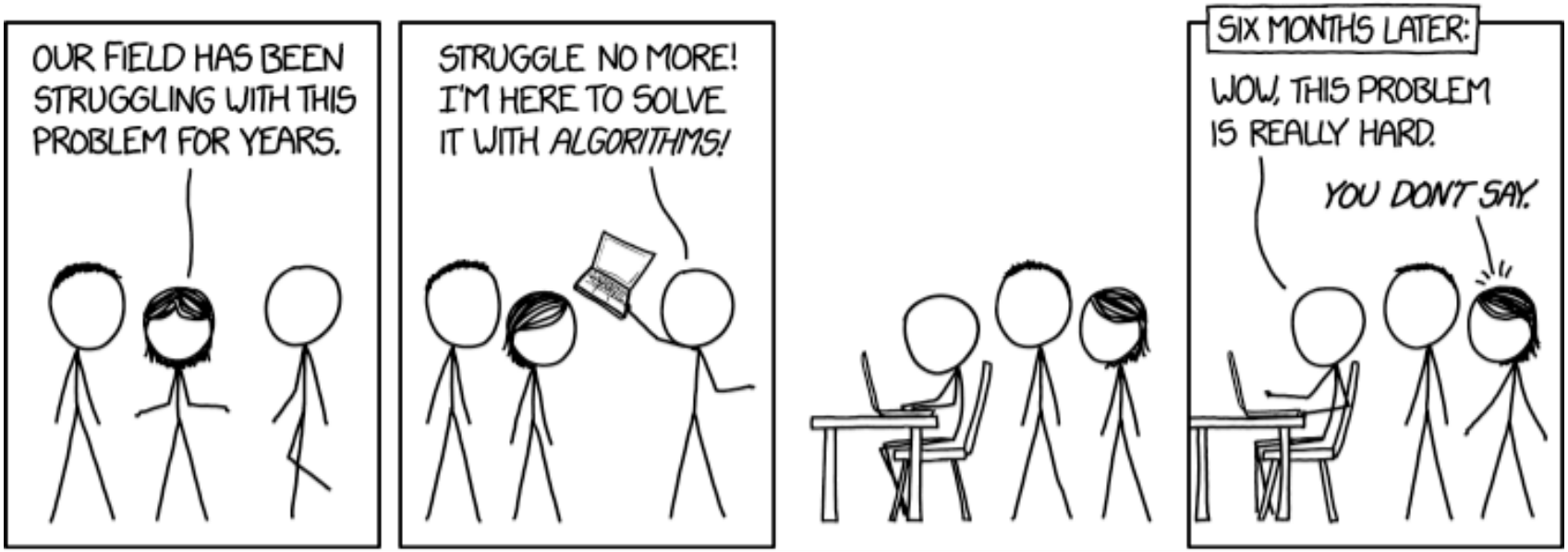
Sequence Alignment

Mark Voorhies

5/4/2017



HTTP://WWW.XKCD.COM/1831/

It is desirable to guard against the possibility of exaggerated ideas that might arise as to the powers of the Analytical Engine. In considering any new subject, there is frequently a tendency, first, to overrate what we find to be already interesting or remarkable; and, secondly, by a sort of natural reaction, to undervalue the true state of the case, when we do discover that our notions have surpassed those that were really tenable.

Ada Lovelace (opening of Note G)
http://www.fourmilab.ch/babbage/sketch.html#NoteG

Loading and re-loading your functions

```
# Use import the first time you load a module
#   (And keep using import until it loads
#     successfully)
import my_module

my_module.my_function(42)

# Once a module has been loaded, use reload to
#   force python to read your new code
reload(my_module)
```

Ways to build a string

```
# Join a list of strings
x = []
for i in s1:
    x.append(f(i))
return "".join(x)

# Work with a string directly
s2 = ""
for i in s1:
    s2 += f(i)
return s2
```

Ways to reverse a string

```
# Prepend to output string
s2 = f(i) + s2
# Use reversed
for i in reversed(s1):
    s2 += f(i)
# Fancy slicing
for i in s1[::-1]:
    s2 += f(i)
# Descending index
for i in xrange(len(s1)-1,-1,-1):
    s2 += f(s1[i])
# Indexing math
L = len(s1)
for i in xrange(L):
    s2 += f(s1[L-i-1])
```

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

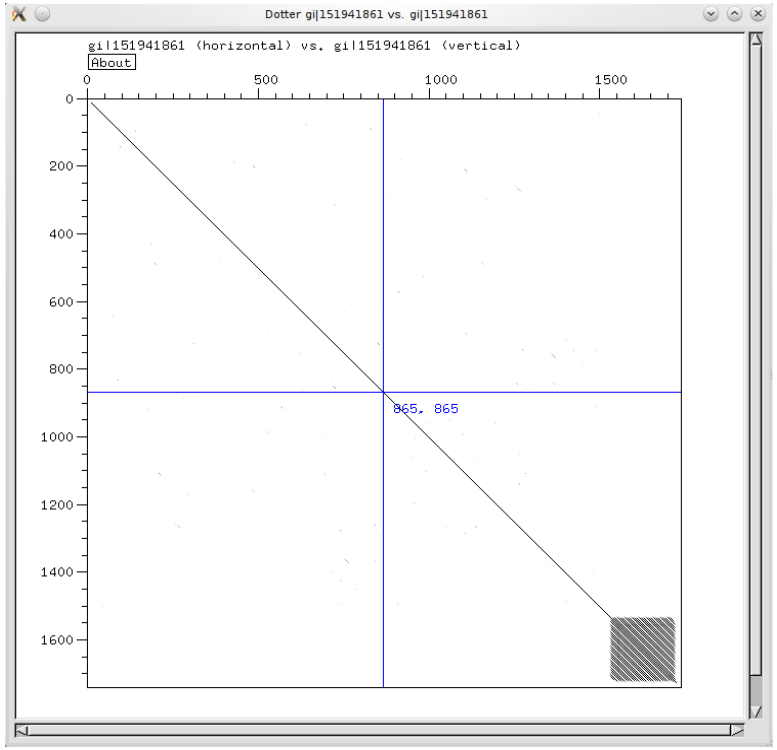
Given two sequences and a scoring matrix, find the offset that yields the best scoring ungapped alignment.

```
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 scoring matrix S (represented as a dict)."""
    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



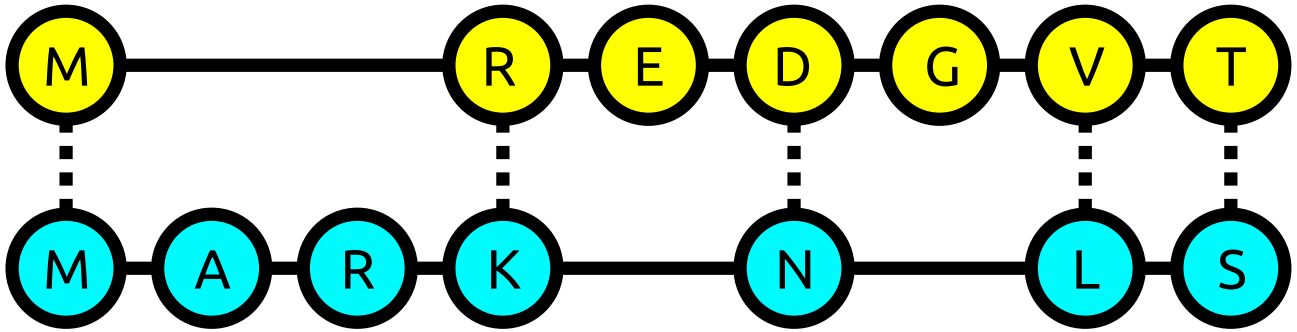
1. Unbiased view of all ungapped alignments of two sequences
2. Noise can be filtered by applying a smoothing window to the diagonals.

Exercise: Scoring a gapped alignment

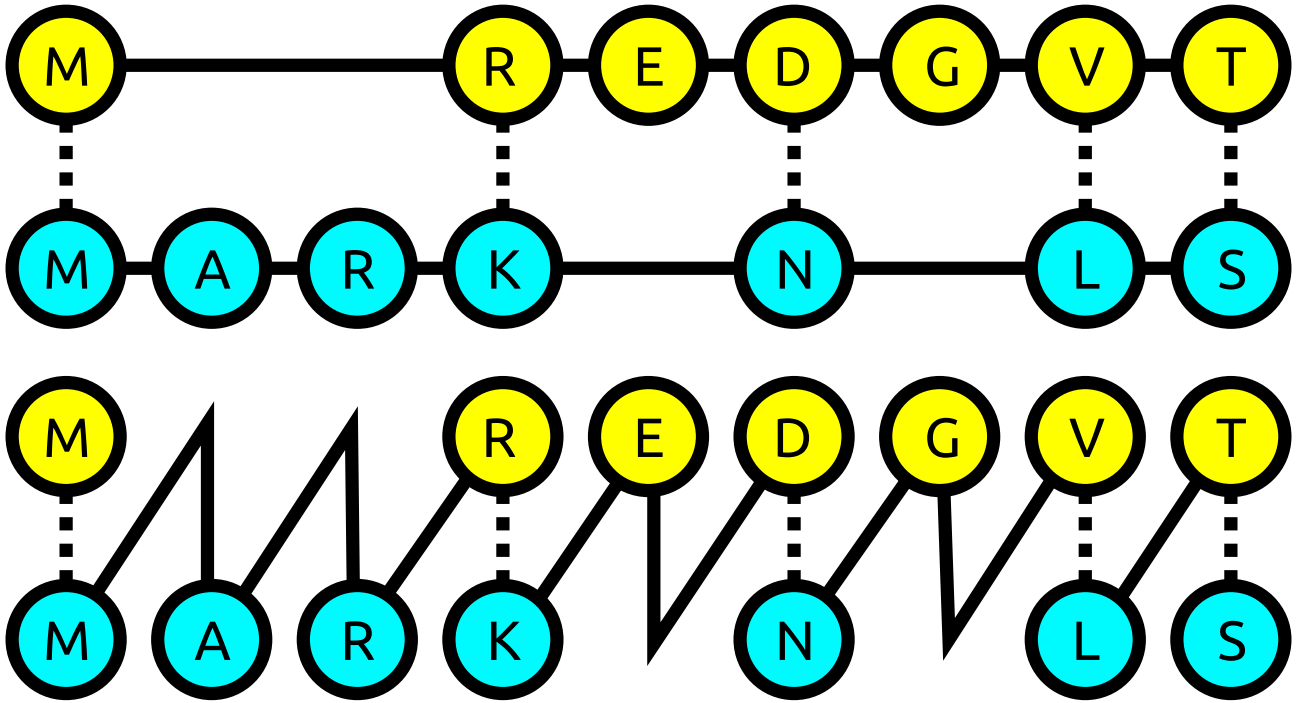
1. Given two equal length gapped sequences (where "-" represents a gap) and a scoring matrix, calculate an alignment score with a -1 penalty for each base aligned to a gap.
2. Write a new scoring function with separate penalties for opening a zero length gap (*e.g.*, G = -11) and extending an open gap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

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

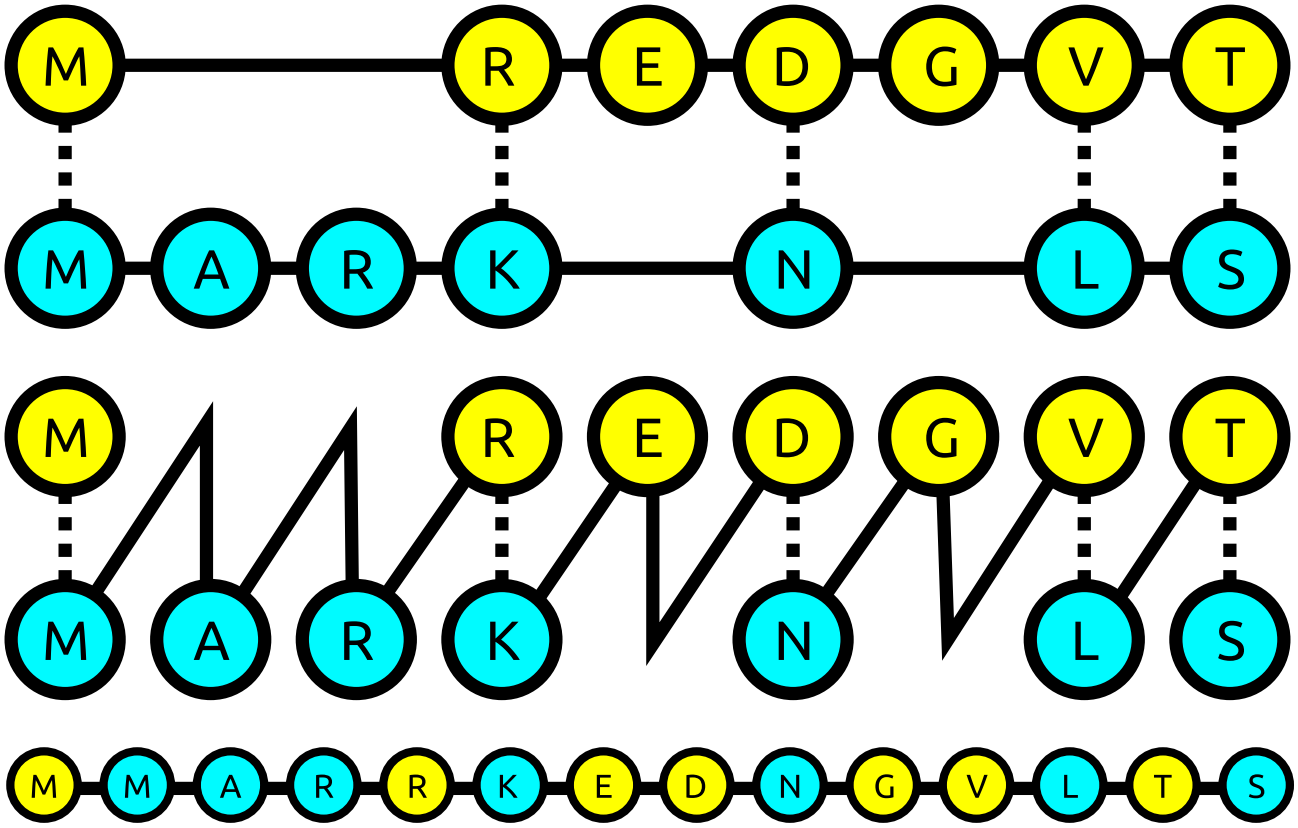
How many ways can we align two sequences?



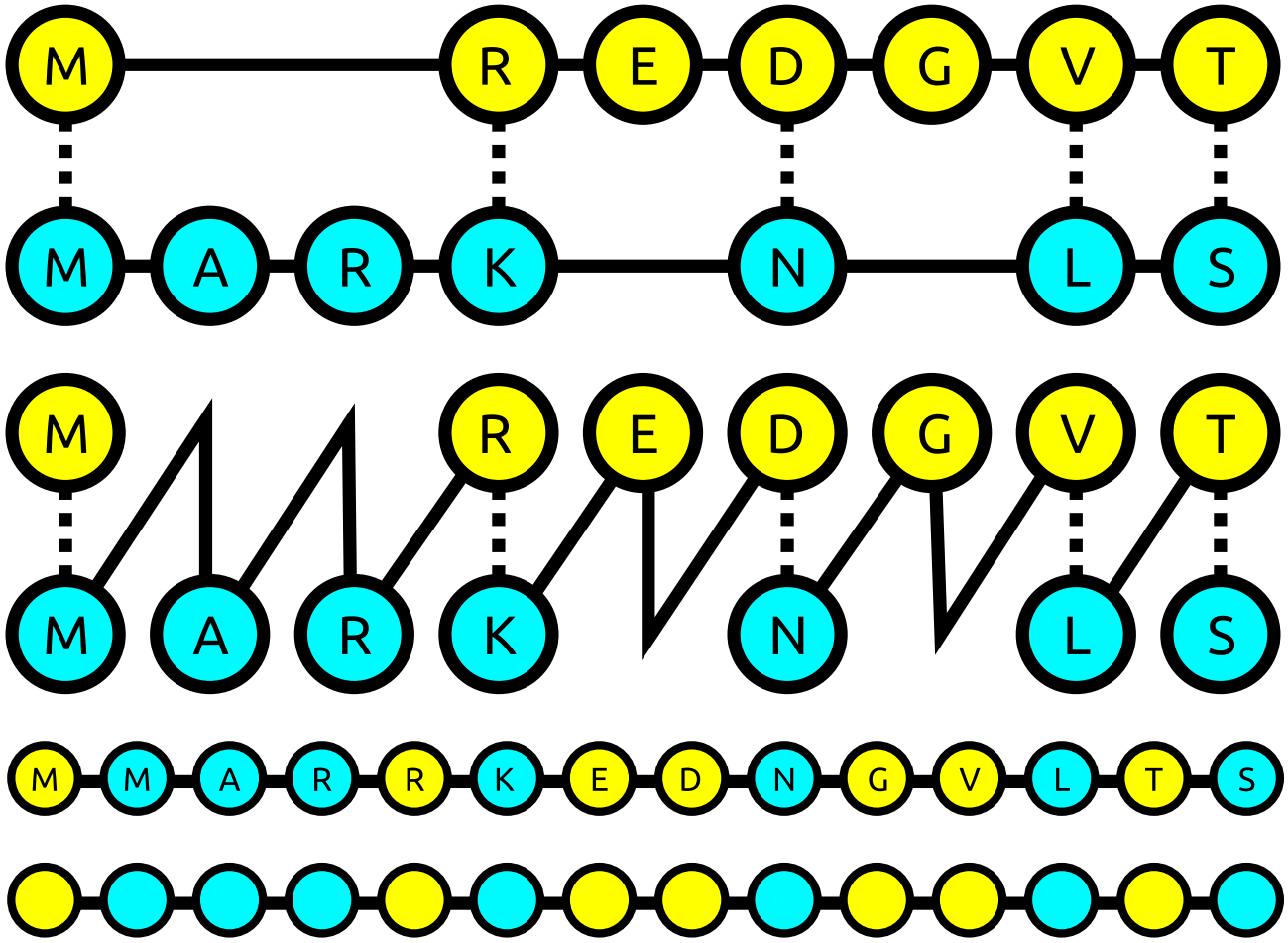
How many ways can we align two sequences?



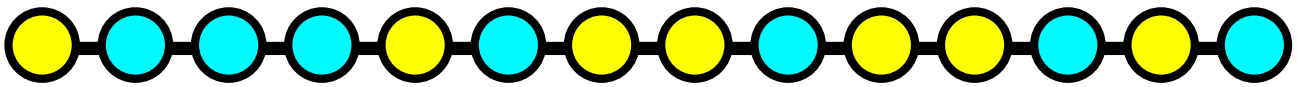
How many ways can we align two sequences?



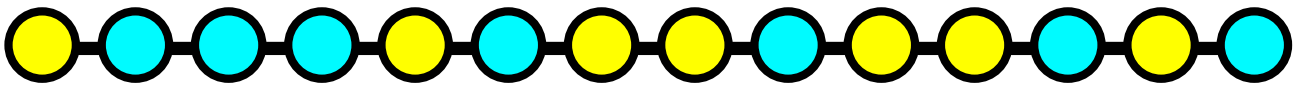
How many ways can we align two sequences?



How many ways can we align two sequences?



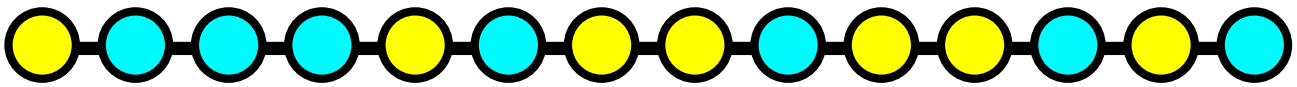
How many ways can we align two sequences?



Binomial formula:

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

How many ways can we align two sequences?

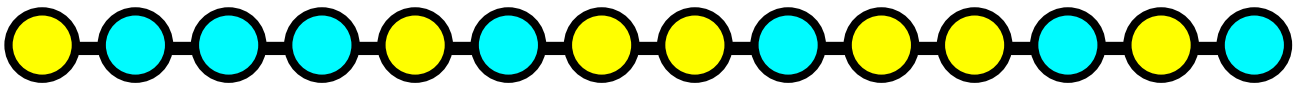


Binomial formula:

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

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

How many ways can we align two sequences?



Binomial formula:

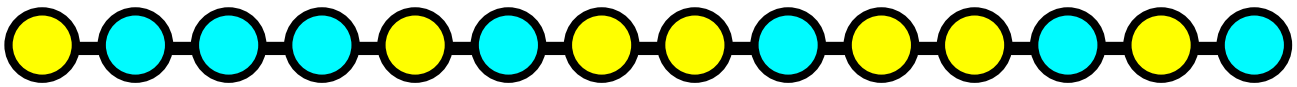
$$\binom{k}{r} = \frac{k!}{(k-r)!r!}$$

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

Stirling's approximation:

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

How many ways can we align two sequences?



Binomial formula:

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

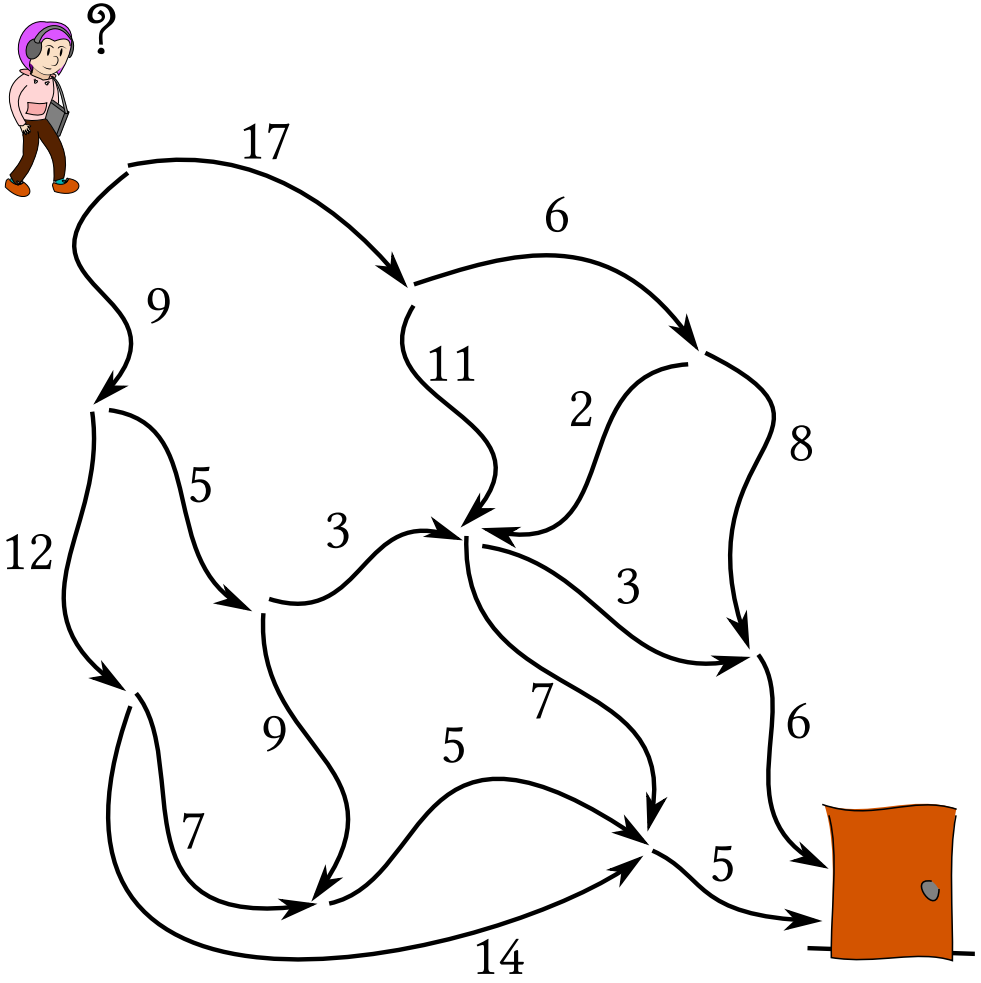
$$\binom{2n}{n} = \frac{(2n)!}{n!n!}$$

Stirling's approximation:

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

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

Dynamic Programming



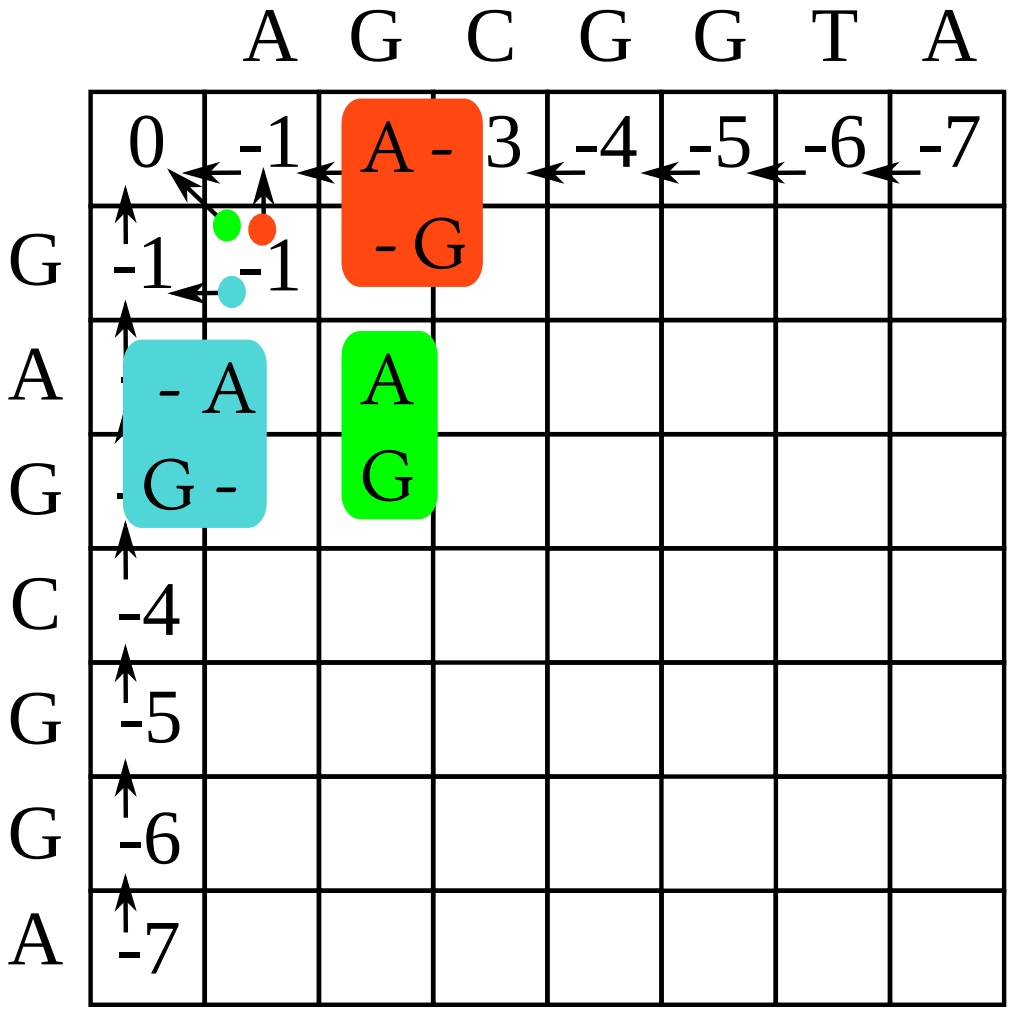
Needleman-Wunsch

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

Needleman-Wunsch

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

Needleman-Wunsch



Needleman-Wunsch

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

Needleman-Wunsch

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

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

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

Needleman-Wunsch

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

Needleman-Wunsch

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

Needleman-Wunsch

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

Needleman-Wunsch

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

Needleman-Wunsch

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

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.