Sequence Alignment

Mark Voorhies

4/12/2018

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

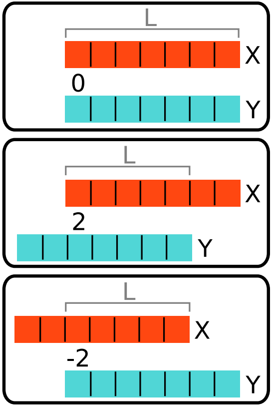
Given two sequences and a scoring matrix, find the offset that yields the best scoring ungapped alignment.

```
def score(S,x,y):
    assert(len(x) == len(y))
    s = 0
    for (i,j) in zip(x,y):
        s += S[i][j]
    return s
```

Exercise: Scoring an ungapped alignment

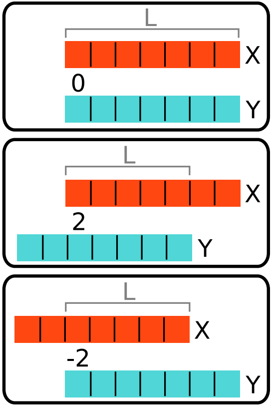
Given two sequences and a scoring matrix, find the offset that yields the best scoring ungapped alignment.

```
def subseqs(x,y,i):
    if(i > 0):
        y = y[i:]
    elif(i < 0):
        x = x[-i:]
    L = min(len(x),len(y))
    return x[:L],y[:L]
```



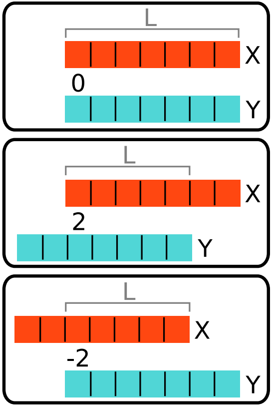
Given two sequences and a scoring matrix, find the offset that yields the best scoring ungapped alignment.

```
def alignment(x,y,i):
    if(i > 0):
        x = "-"*i+x
    elif(i < 0):
        y = "-"*(-i)+y
    L = len(y) - len(x)
    if(L > 0):
        x += "-"*L
    elif(L < 0):
        y += "-"*(-L)
    return x,y
```



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

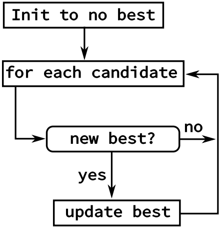
```
def ungapped(S,x,y):
    best = None
    best_score = None
    for i in range(-len(x)+1,len(y)):
        (sx,sy) = subseqs(x,y,i)
        s = score(S,sx,sy)
        if((best_score is None) or (s > best_score)):
            best_score = s
            best = i
    return best, best_score, alignment(x,y,best)
```



Exercise: Scoring an ungapped alignment

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

```
def ungapped(S,x,y):
    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((best_score is None) or (s > best_score)):
            best_score = s
            best = i
    return best, best_score, alignment(x,y,best)
```



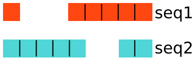
Exercise: Scoring a gapped alignment

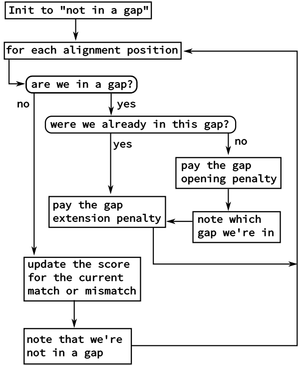
Write a new scoring function with separate penalties for opening a zero length gap (*e.g.*, G = -11) and extending an open gap by one base (*e.g.*, E = -1).

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

Exercise: Scoring a gapped alignment

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

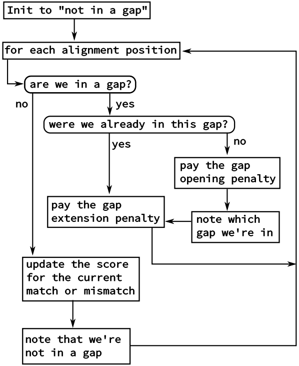




Exercise: Scoring a gapped alignment

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

```python
def gapped_score(seq1, seq2,
                 s, g = 0, e = -1):
    gap = None
    score = 0
    for pair in zip(seq1,seq2):
        assert(pair != ("-","-"))
        try:
            curgap = pair.index("-")
        except ValueError:
            score += s[pair[0]][pair[1]]
            gap = None
        else:
            if(gap != curgap):
                score += g
                gap = curgap
            score += e
    return score
```



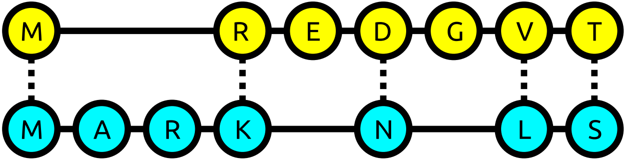
Exercise: Scoring a gapped alignment

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

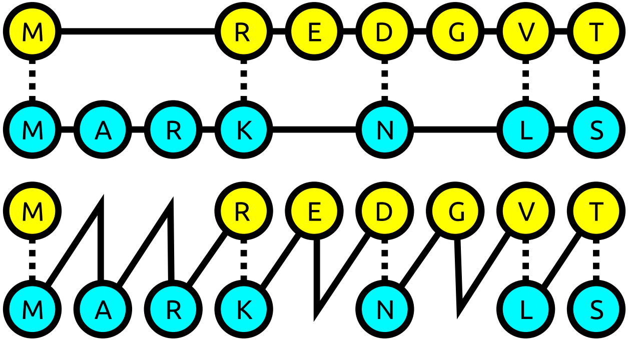
```
def gapped_score(seq1, seq2,
                 s, g = 0, e = -1):
    gap = None
    score = 0
    for pair in zip(seq1,seq2):
        assert(pair != ("-","-"))
        try:
            curgap = pair.index("-")
        except ValueError:
            score += s[pair[0]][pair[1]]
            gap = None
        else:
            if(gap != curgap):
                score += g
                gap = curgap
            score += e
    return score
```

```
def gapped_score(seq1, seq2,
                 s, g = 0, e = -1):
    gap = None
    score = 0
    for (c1,c2) in zip(seq1,seq2):
        if((c1 == "-") and (c2 == "-")):
            raise ValueError
        elif(c1 == "-"):
            if(gap != 1):
                score += g
                gap = 1
            score += e
        elif(c2 == "-"):
            if(gap != 2):
                score += g
                gap = 2
            score += e
        else:
            score += s[c1][c2]
            gap = None
    return score
```

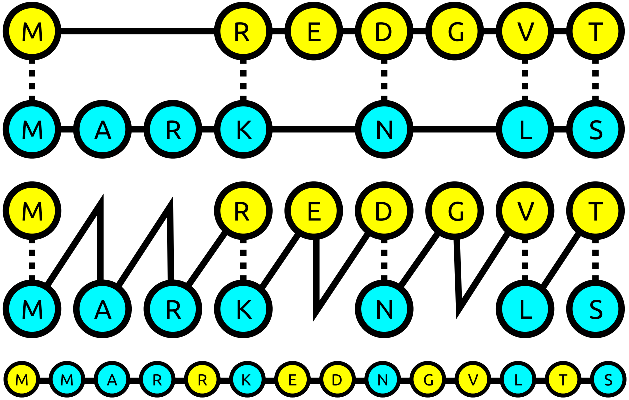
How many ways can we align two sequences?



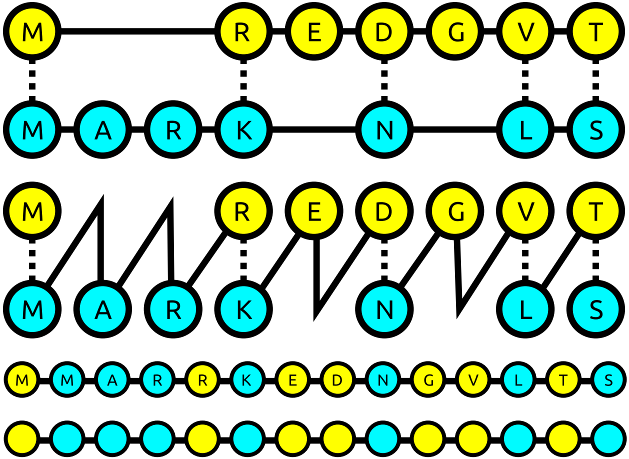
How many ways can we align two sequences?



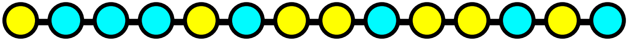
How many ways can we align two sequences?



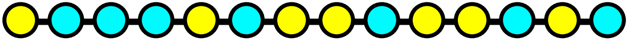
How many ways can we align two sequences?



How many ways can we align two sequences?



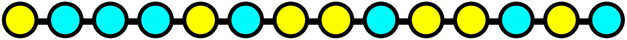
How many ways can we align two sequences?



Binomial formula:

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

How many ways can we align two sequences?

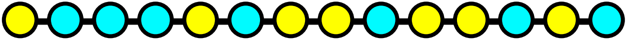


Binomial formula:

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

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

How many ways can we align two sequences?



Binomial formula:

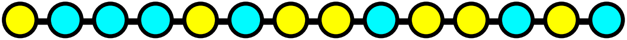
$$\binom{k}{r} = \frac{k!}{(k-r)!r!}$$

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

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:

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

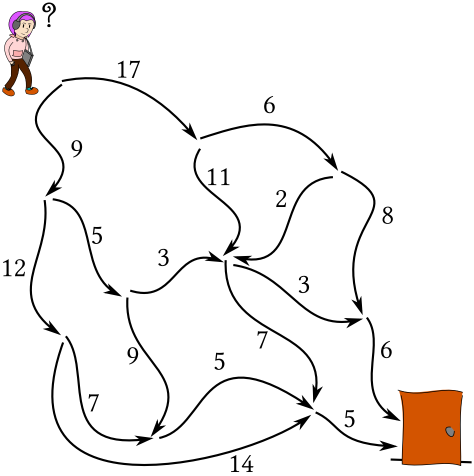
$$\binom{2n}{n} = \frac{(2n)!}{n!n!}$$

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

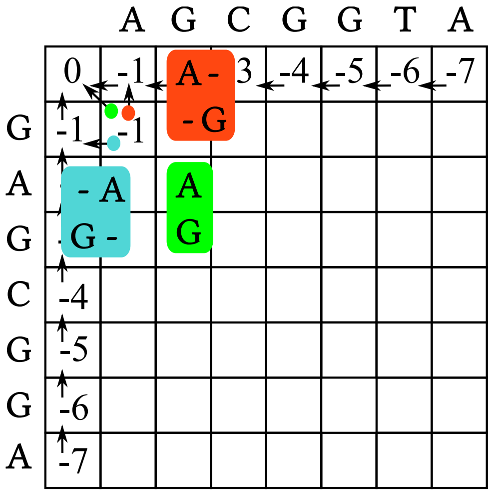
Needleman-Wunsch

		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



		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							

		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

Needleman-Wunsch

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

Smith-Waterman

The implementation of local alignment is the same as for global alignment, with a few changes to the rules:

- Initialize edges to 0 *(no penalty for starting in the middle of a sequence)*
- The maximum score is never less than 0, and no pointer is recorded unless the score is greater than 0 *(note that this implies negative scores for gaps and bad matches)*
- The trace-back starts from the highest score in the matrix and ends at a score of 0 *(local, rather than global, alignment)*

Because the naive implementation is essentially the same, the time and space requirements are also the same.

Smith-Waterman

		A	G	C	G	G	T	A
	0	0	0	0	0	0	0	0
G	0	0	1	0	0	1	0	0
A	0	1	0	0	0	0	0	1
G	0	0	2	1	1	1	0	0
C	0	0	1	3	2	1	0	0
G	0	0	0	2	4	3	2	1
G	0	0	1	1	3	5	4	3
A	0	1	0	0	2	4	4	5

Final 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, try implementing Smith-Waterman local alignment and/or non-zero gap opening penalties.