Number of alignments

List all alignments of $s =$ AC and $t =$ GA.

You should have got these 13 al's:

```
-AC   A-C   --AC   A--C   -A-C
GA-   GA-   GA--   -GA-   G-A-

AC    A-C   -AC
GA    -GA   G-A

AC-   AC-   AC--   -AC-   A-C-
-GA   G-A   --GA   G--A   -G-A
```


Number of alignments

Question
How many alignments are there in general for two strings s and t?

Observation
The number of alignments depends only on the length of s and t.

Def.
Let $N(n, m)$ = number of al's of two strings of length n and m.

We know:

- $N(2, 2) = 13$
- $N(1, 1) = 3$
- $N(n, 0) = 1, \; N(0, m) = 1$
- we set: $N(0, 0) = 1$ (empty alignment)

Number of alignments

$N(n, m)$	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1	3				
2	1		13			
3	1					
4	1					
5	1					

Number of alignments

Look at the last column of the alignments:

```
-AC   A-C   --AC   A--C   -A-C
GA-   GA-   GA--   -GA-   G-A-

AC    A-C   -AC
GA    -GA   G-A

AC-   AC-   AC--   -AC-   A-C-
-GA   G-A   --GA   G--A   -G-A
```


Number of alignments

We have a recursive formula:

- $N(n, 0) = N(0, m) = 1 \qquad$ for $n, m \geq 0$
- and for $n, m > 0$:

$$N(n, m) = N(n-1, m) + N(n-1, m-1) + N(n, m-1)$$

Number of alignments

$N(n, m)$	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1	3				
2	1		13			
3	1					
4	1					
5	1					

Number of alignments

$N(n,m)$	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1	3	5	7	9	11
2	1	5	13	25	41	61
3	1	7	25	63	129	231
4	1	9	41	129	321	681
5	1	11	61	231	681	1683

Number of alignments

Let's look at the case $n = m$:

n	0	1	2	3	4	5	...	1000
$N(n,n)$	1	3	13	63	321	1683	...	$\approx 10^{767}$

Number of alignments

Let's look at the case $n = m$:

n	0	1	2	3	4	5	...	1000
$N(n,n)$	1	3	13	63	321	1683	...	$\approx 10^{767}$

In fact, it can be shown that $N(n,n)$ grows exponentially.

Running time of exhaustive search:

For any al. $\mathcal{A}$, we have $\max(n,m) \leq |\mathcal{A}| \leq (n+m)$, thus:

$$N(n,m) \cdot \max(n,m) \leq \text{no. of steps of algo.} \leq N(n,m) \cdot (n+m)$$

Therefore, it has exponential running time: too slow!

A Dynamic Programming Algorithm

Dynamic Programming

- is a class of algorithms (like greedy, divide and conquer, ...)
- applicable when solution can be constructed from solutions of subproblems
- subproblem solutions re-used several times
- uses a matrix ("DP-table") for storing subproblem solutions

Smaller subproblems

Crucial idea

If $\mathcal{A}$ is an optimal alignment, then $\mathcal{B}$, the same alignment without the last column, is also optimal.

Proof

By contradiction (see board).

Smaller subproblems

Crucial idea

If $\mathcal{A}$ is an optimal alignment, then $\mathcal{B}$, the same alignment without the last column, is also optimal.

Proof

By contradiction (see board).

So we will compute the scores of optimal alignments of all pairs of prefixes of s and t, and construct an optimal alignment from that!

The DP-table

Algorithm 2: Needleman-Wunsch algorithm for global alignment

- construct a DP-table D of size $(n+1) \times (m+1)$ s.t.

$$D(i,j) = sim(s_1 \dots s_i, t_1 \dots t_j)$$

  (We will see in a moment how!)
- return $D(n,m)$

Constructing solutions from smaller subproblems

Look at an alignment of s and t. There are 3 cases:

1. last column is $\binom{s_n}{-}$
2. last column is $\binom{s_n}{t_m}$
3. last column is $\binom{-}{t_m}$

Recall that if $\mathcal{A}$ is optimal, then so is $\mathcal{B} = (\mathcal{A}$ without last column)!

- in case 1, $\mathcal{B}$ is an opt. al. of $s_1 \dots s_{n-1}$ and $t_1 \dots t_m$
- in case 2, $\mathcal{B}$ is an opt. al. of $s_1 \dots s_{n-1}$ and $t_1 \dots t_{m-1}$
- in case 3, $\mathcal{B}$ is an opt. al. of $s_1 \dots s_n$ and $t_1 \dots t_{m-1}$

Constructing solutions from smaller subproblems

So to compute $sim(s,t) = D(n,m)$, we need to know

- $sim(s_1 \dots s_{n-1}, t_1 \dots t_m) \quad = D(n-1, m)$
- $sim(s_1 \dots s_{n-1}, t_1 \dots t_{m-1}) \quad = D(n-1, m-1)$
- $sim(s_1 \dots s_n, t_1 \dots t_{m-1}) \quad = D(n, m-1)$

and add the score of the last column!

Constructing solutions from smaller subproblems

So to compute $sim(s,t) = D(n,m)$, we need to know

- $sim(s_1 \dots s_{n-1}, t_1 \dots t_m) \quad = D(n-1, m)$
- $sim(s_1 \dots s_{n-1}, t_1 \dots t_{m-1}) \quad = D(n-1, m-1)$
- $sim(s_1 \dots s_n, t_1 \dots t_{m-1}) \quad = D(n, m-1)$

and add the score of the last column!

$$D(n,m) = \max \begin{cases} D(n-1,m) + gap \\ D(n-1,m-1) + \begin{cases} match & \text{if } s_n = t_m \\ mismatch & \text{if } s_n \neq t_m \end{cases} \\ D(n,m-1) + gap \end{cases}$$

Constructing solutions from smaller subproblems

Now we can compute all entries of D:

- $D(i,0) = i \cdot gap \quad$ for $i \geq 0$
- $D(0,j) = j \cdot gap \quad$ for $j \geq 0$
- recursion (for $i,j > 0$):

$$D(i,j) = \max \begin{cases} D(i-1,j) + gap \\ D(i-1,j-1) + \begin{cases} match & \text{if } s_i = t_j \\ mismatch & \text{if } s_i \neq t_j \end{cases} \\ D(i,j-1) + gap \end{cases}$$

Recall s = ACCT, t = CAT match: 2, mismatch: -1, gap: -1

$D(i,j)$			C	A	T
		0	1	2	3
	0	0	−1	−2	−3
A	1	−1	−1	1	
C	2	−2			
C	3	−3			
T	4	−4			

$D(1,1) = \max\{-1-1, 0-1, -1-1\} = -1 \quad D(1,2) = \max\{-2-1, -1+2, -1-1\} = 1$

$s =$ ACCT, $t =$ CAT match: 2, mismatch: -1, gap: -1

$D(i,j)$			C	A	T
		0	1	2	3
	0	0	-1	-2	-3
A	1	-1	-1	1	0
C	2	-2	1	0	0
C	3	-3	0	0	-1
T	4	-4	-1	-1	2

Needleman-Wunsch DP algorithm for global alignment

Variant which outputs $sim(s,t)$ only.

Algorithm *DP algorithm for global alignment*
Input: strings s, t, with $|s| = n, |t| = m$; scoring function f
Output: value $sim(s,t)$

for $j = 0$ to m **do** $D(0,j) \leftarrow j \cdot g$;
for $i = 1$ to n **do** $D(i,0) \leftarrow i \cdot g$;
for $i = 1$ to n **do**
  for $j = 1$ to m **do**

$$D(i,j) \leftarrow \max \begin{cases} D(i-1,j) + g \\ D(i-1,j-1) + f(s_i, t_j) \\ D(i,j-1) + g \end{cases}$$

return $D(n,m)$;

Needleman-Wunsch DP algorithm for global alignment

- Algorithm first introduced by Needleman & Wunsch (1970).
- Different orders of computation are possible: necessary to compute $D(i-1,j), D(i-1,j-1)$, and $D(i,j-1)$ before $D(i,j)$
- Time: $O(n \cdot m)$
  (initialize first row and column in constant time, for the remaining $n \cdot m$ cells, we have 3 lookups and additions, so a constant number of operations)
- Space: $O(n \cdot m)$
  (matrix of size $(n+1)(m+1)$)
- for $n = m$, we get time and space $O(n^2)$, hence this is called a quadratic (time and space) algorithm
- Space-saving variant exists (later)