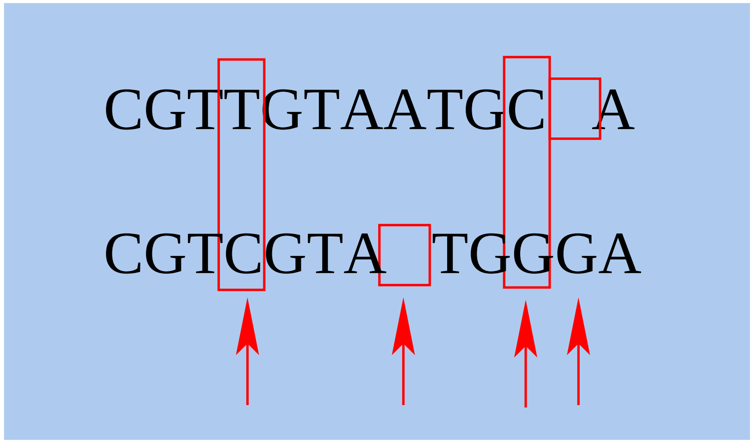
Por que comparar sequências?

1. Tem-se 2 sequências sobre o mesmo alfabeto, com quase o mesmo tamanho ($n \times 10^3$). Sabe-se que as sequências são quase iguais, diferindo apenas pela inserção, deleção e/ou substituição de alguns caracteres. Quais os lugares onde essas diferenças ocorrem.

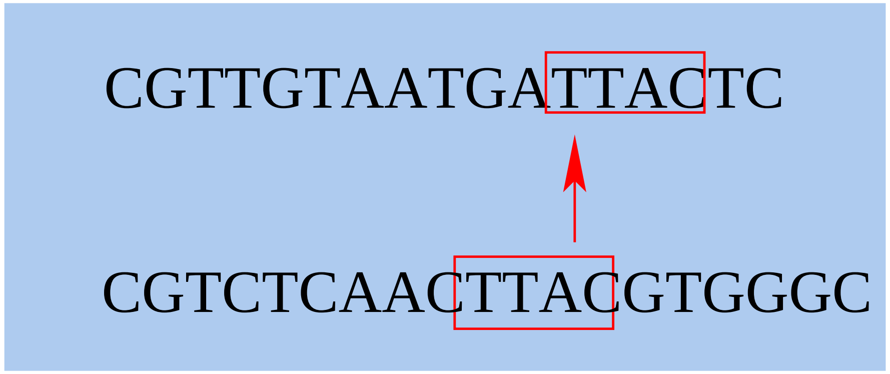


2. Tem-se 2 sequências sobre o mesmo alfabeto com tamanho $n \times 10^2$. Existe um prefixo de uma que é similar a um sufixo da outra? Em caso positivo, indicar esse prefixo/sufixo.

```
        TGGGAATGCTGCCCATGT
CGTCGTATGGGA
```

3. Tem-se o problema (2), agora com centenas de sequências a serem comparadas. Além disso, sabe-se que a maioria dos pares de sequências não estão relacionados (ou não apresentam o grau de similaridade desejado).

4. Tem-se 2 sequências sobre o mesmo alfabeto com tamanho $n \times 10^2$. Existem substrings, um de cada sequência, que são similares?



5. Tem-se o problema (4), agora com milhares de sequências a serem comparadas.

Alinhamento entre Sequências

É a inserção de espaços em posições arbitrárias nas sequências para que elas fiquem com o mesmo tamanho, levando em consideração que a mesma posição nas duas sequências não pode conter um espaço.

Escore de um Alinhamento

A cada posição associa-se um valor:

+1, caso ela contenha o mesmo caracter nas 2 sequências;

-1, caso ela contenha caracteres diferentes;

-2, caso ela contenha um branco em alguma das sequências.

O escore de um alinhamento é a soma dos valores associados a cada posição.

```
CGTTGTAATGC□A

CGTCGTA□TGGGA
```

$$9 \times 1 + 2 \times (-1) + 2 \times (-2) = 3$$

Melhor alinhamento é chamado de **SIMILARIDADE**.

Problema: Como encontrar o melhor alinhamento entre duas sequências?

Soluções:

1. Calcular o escore de todos os possíveis alinhamentos e pegar o melhor... (grande custo computacional)

2. Usar **programação dinâmica**: Esta técnica consiste em usar soluções de partes do problema para calcular a solução do problema.

Exemplo: Dadas 2 sequências s e t de tamanhos n e m. A similaridade entre elas será calculada usando as similaridades de todos os prefixos dessas sequências.

Para obter os alinhamentos entre as sequências $s[1..i]$ e $t[1..j]$ (prefixos de s e t, respectivamente), existem 3 possibilidades:

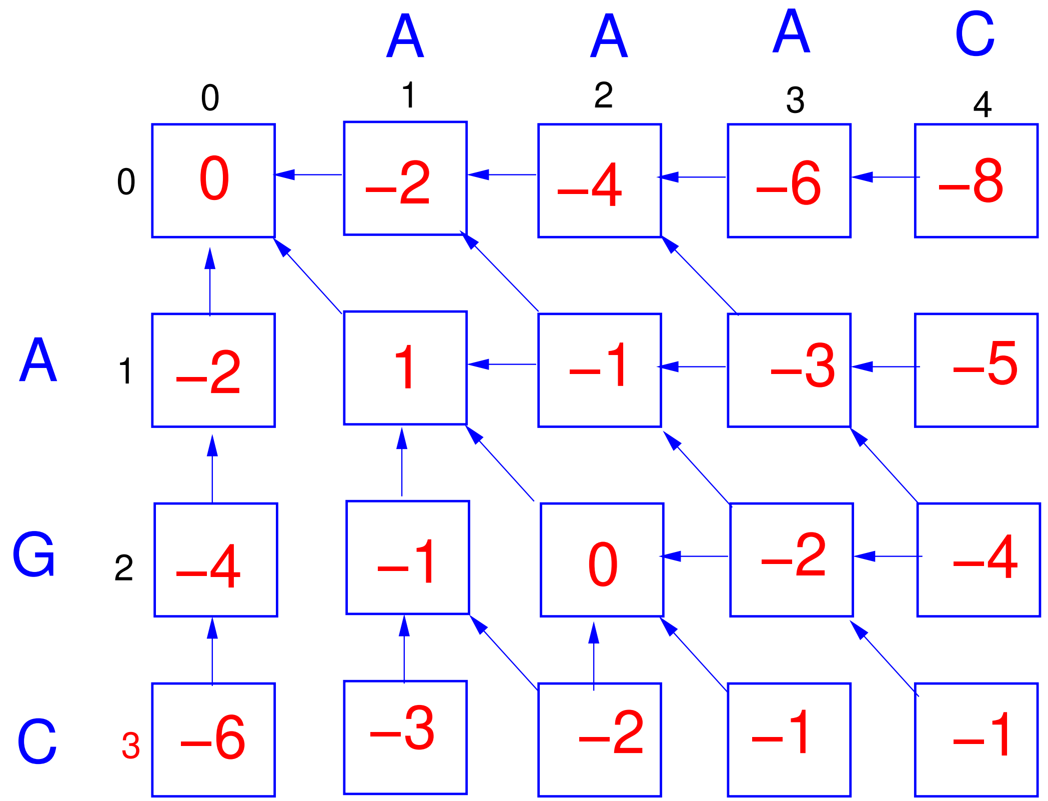
- alinhar $s[1..i]$ com $t[1..j-1]$ e casar um espaço com $t[j]$;
- alinhar $s[1..i-1]$ com $t[1..j-1]$ e casar $s[i]$ com $s[j]$;
- alinhar $s[1,i-1]$ com $t[1..j]$ e casar $s[i]$ com um espaço.

Como existem $n+1$ prefixos de s e $m+1$ prefixos de t, podemos usar uma matriz $(n+1) \times (m+1)$ para colocar as similaridades dos prefixos, tal que a posição $[i,j]$ conterá a similaridade entre as sequências $s[1..i]$ e $t[1..j]$. O valor de cada posição da matriz será dado por:

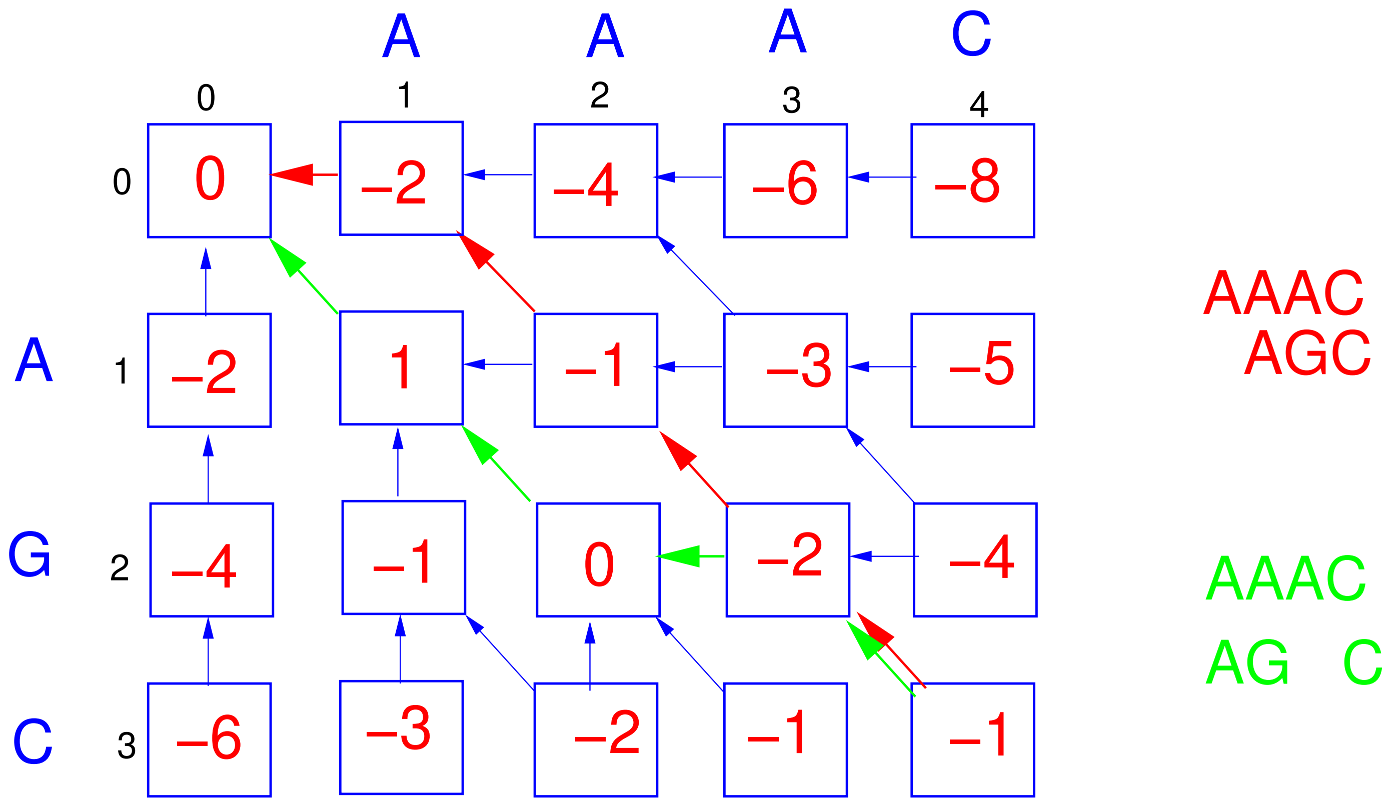
$$a[i,j] = max \begin{cases} a[i,j-1] - 2 \\ a[i-1,j-1] + p(i,j) \\ a[i-1,j] - 2 \end{cases}$$

onde $p(i,j) = \begin{cases} +1, caso\, s[i] = t[i] \\ -1, caso\, s[i] \neq t[i] \end{cases}$

Exemplo: Matriz de similaridades



Exemplo: Melhor alinhamento



Algoritmo de Similaridade

```
Algorithm Similarity
input sequences s and t
output similarity between s and t
m ← |s|;
n ← |t|;
for i ← 0 to m do a[i,0] ← i × g;
for j ← 0 to n do a[0,j] ← j × g;
for i ← 1 to m do
      for j ← 1 to n do
```

$$a[i,j] \leftarrow max \begin{cases} a[i,j-1]-g \\ a[i-1,j-1]+p(i,j) \\ a[i-1,j]-g \end{cases}$$

```
return a[m,n]
```

onde g é a penalidade e p a função de escore.

Ordem de complexidade: $O(n^2)$

Algoritmo de Alinhamento

```
Algorithm Align
input indices i, j, array a given by algorithm Similarity
output alignment in alignS, alignT and lenght in len
if i = 0 and j = 0 then len ← 0
else if i > 0 and a[i, j] = a[i − 1, j] + g then
      Align(i − 1, j, len);
      len ← len + 1;
      alignS[len] ← s[i];
      alignT[len] ← −
else if i > 0 and j > 0 and a[i, j] = a[i − 1, j − 1] + p(i, j) then
      Align(i − 1, j − 1, len);
      len ← len + 1;
      alignS[len] ← s[i];
      alignT[len] ← t[j]
else // j > 0 and a[i, j] = a[i.j − 1] + g
      Align(i, j − 1, len);
      len ← len + 1;
      alignS[len] ← −;
      alignT[len] ← t[j]
```

Ordem de complexidade: $O(len) = O(m + n)$

Comparação Local

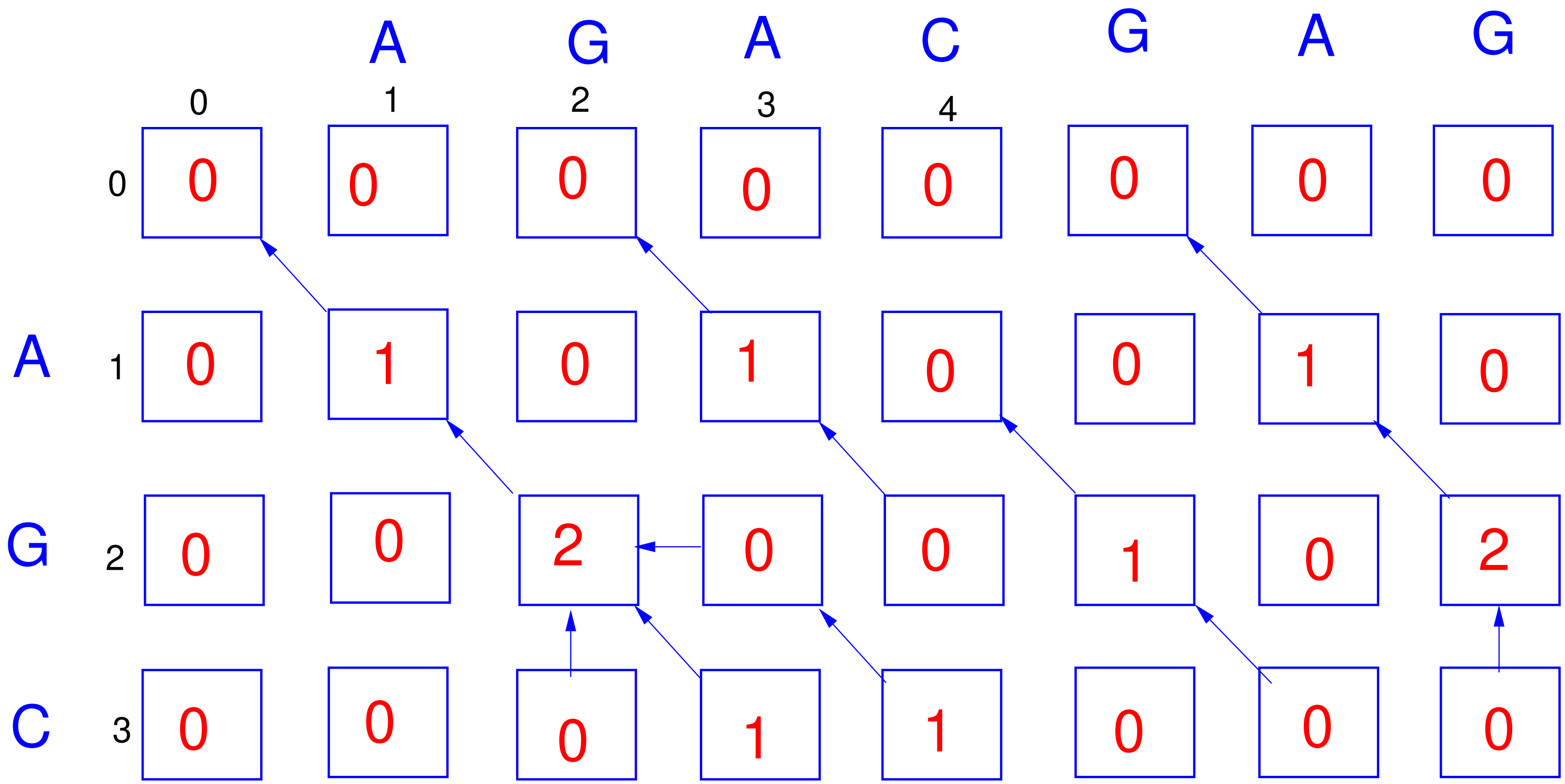
Um **alinhamento local** entre sequências s e t é um alinhamento entre um substring de s e um substring de t.

*Como encontrar o alinhamento local entre duas sequencias com **maior escore**?*

Construir uma matriz $(|s|+1) \times (|t|+1)$ onde cada posição (i,j) conterá o maior escore de um alinhamento entre um sufixo de $s[1..i]$ e um sufixo de $t[1..j]$. A primeira linha e a primeira colunas são inicializadas com zeros. O valor de cada posição é dado por:

$$a[i,j] = max \begin{cases} a[i,j-1]+g \\ a[i-1,j-1]+p(i,j) \\ a[i-1,j]+g \\ 0 \end{cases}$$

Exemplo: Alinhamento local



Comparação Semi-Global

Comparação ignorando brancos no início ou fim das sequências.

		Semiglobal
CAGCA-CTTGGATTCTCGG ---CAGCGTGG--------	sim(s,t)=–19	sim´(s,t)=3
CAGCACTTGGATTCTCGG CAGC-----G-T----GG	sim(s,t)=–12	sim´(s,t)=–12

Para desconsiderar brancos no final de s : Encontrar o melhor alinhamento entre s e um prefixo de t $\Longrightarrow$ **maior valor da última linha da matriz de similaridades**

Para desconsiderar brancos no início de s: Encontrar o melhor alinhamento entre s e um sufixo de t $\Longrightarrow$ **inicializar a primeira linha com zeros e usar a mesma fórmula para calcular as outras posições**

Posição dos espaços	Ação
Início da primeira sequência	Inicializar primeira lina com zeros
Fim da primeira sequência	Procurar o máximo na última linha
Início da segunda sequência	Iniciallizar primeira coluna com zeros
Fim da segunda sequência	Procurar o máximo na última coluna