

TRABALHO FINAL

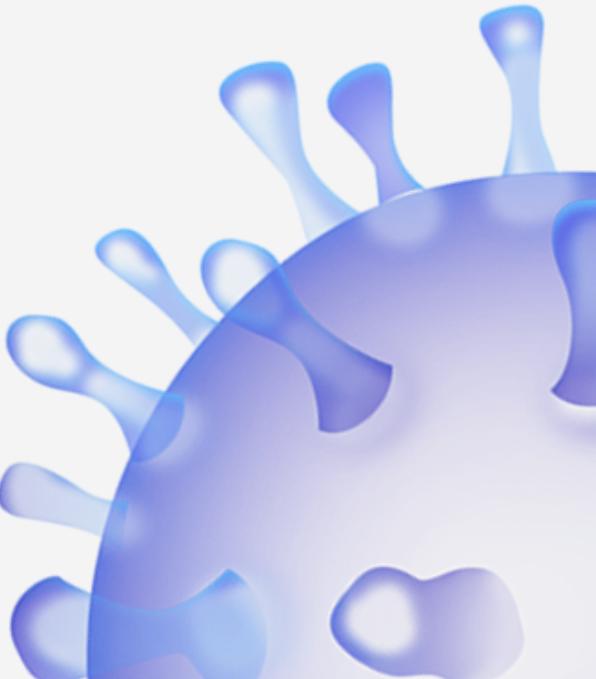
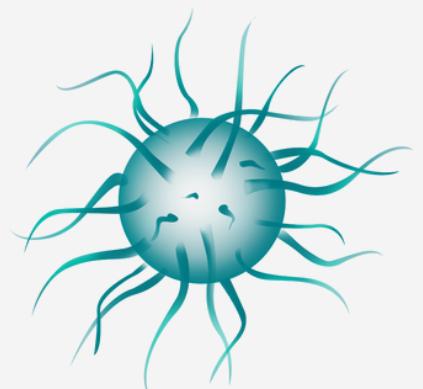
Ambientes em Computação

Fabiano Santana

Helena Lott Costa

Rossana de Oliveira Souza

Thaiane Gomes Nascimento



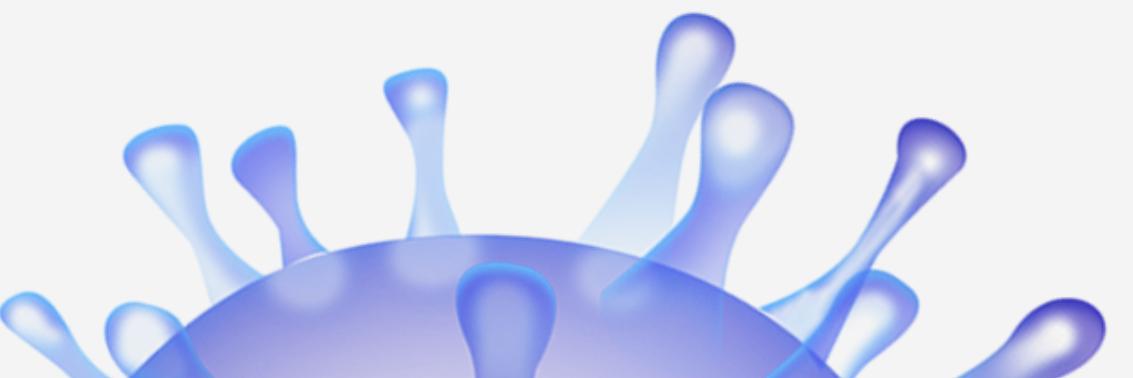
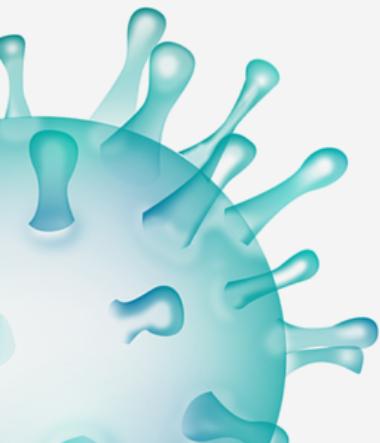
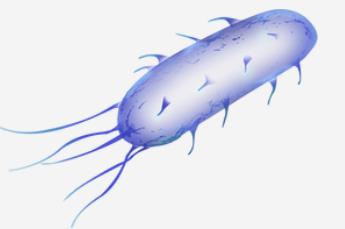
Conteúdo da apresentação

01 O problema

02 Programação diâmica

03 Algoritmo de Needleman Wunch

04 Conclusão



O problema

- Importância do alinhamento de sequências: análise de padrões e homologias entre diferentes organismos;
- Alinhamento global ou local;
 - Global:
Busca da maior subsequência comum (permite deleções e inserções);
Comparação pela função score (semelhança ou diferença entre sequências);
Parâmetros: **match** (caracteres iguais). **mismatch** (caracteres diferentes) e **gap penalty** (penalidade por lacuna).
- Algoritmo Needleman-Wunsch: custo de ordem $O(n^2)$.



O problema

- Alinhamento: comparação entre sequências (de nucleotídeos ou proteínas) de forma a se observar seu nível de identidade.



```
A - TGCAATGGTCTAA - GG
|   |   |   |   |   |   |   |   |   |
AATGCAATGG - CCTAAAGG
|   |   |   |   |   |   |   |   |
ACTGCAATGG - - CTAAAAGG
```


O problema

- Alinhamento: comparação entre sequências (de nucleotídeos ou proteínas) de forma a se observar seu nível de identidade.



- Tipos de alinhamento:

Global

Local

O problema

Global

Algoritmo Needleman-Wunsch - custo de ordem $O(n^2)$:

- Busca da maior subsequência comum (permite deleções e inserções);
- Comparação pela função score (semelhança ou diferença entre sequências);
- Parâmetros: match (caracteres iguais). mismatch (caracteres diferentes) e gap penalty (penalidade por lacuna).



Programação dinâmica

- “A programação dinâmica é um método de desenvolvimento que busca encontrar a solução de vários subproblemas para, daí então, encontrar a solução do problema geral.”
- O algoritmo de Needleman-Wunsch foi uma das primeiras aplicações a utilizar a programação dinâmica;
- Processo: inicialização, preenchimento da matriz (scoring) e alinhamento (traceback).



Algoritmo

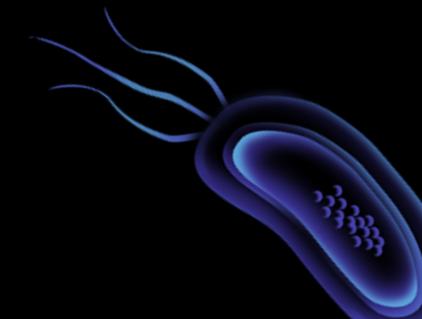
- Para fins de apresentação, o algoritmo foi dividido em 9 partes.
- Vamos passar por cada parte e discutir a implementação abordada:

```
# abre o arquivo e extrai a primeira sequencia
arquivo = open('seq1.txt')
seq1 = arquivo.readline() #TGCTCGTA

# abre o arquivo e extrai a segunda sequencia
arquivo = open('seq2.txt')
seq2 = arquivo.readline() #TTCATA

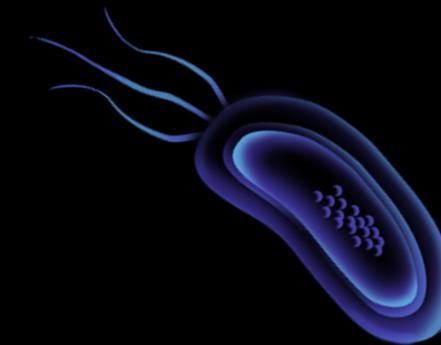
# chama a função de programação dinâmica de Needleman Wunch
needleman_wunsch(seq1, seq2)
```

Algoritmo



```
def needleman_wunsch(seq1, seq2, igual=5, diferente=-2, lacuna=-6):  
    # quantidade de linhas = tamaho da primeira sequencia + 1, pra colocar o valor da inicialização  
    linhas = len(seq1) + 1  
  
    # quantidade de colunas = tamaho da segunda sequencia + 1, pra colocar o valor da inicialização  
    colunas = len(seq2) + 1  
  
    # inicialização da matriz - todos valores iguais a zero  
    matriz = [[0] * colunas for _ in range(linhas)]  
  
    # imprimi a matriz inicial - todos valores iguais a zero  
    for linha in matriz:  
        print(linha)
```

Algoritmo



```
# inicialização das bordas com penalidades de lacuna
for linha in range(linhas):
    # penalidades das linhas
    matriz[linha][0] = linha * lacuna

for coluna in range(colunas):
    # penalidades das colunas
    matriz[0][coluna] = coluna * lacuna

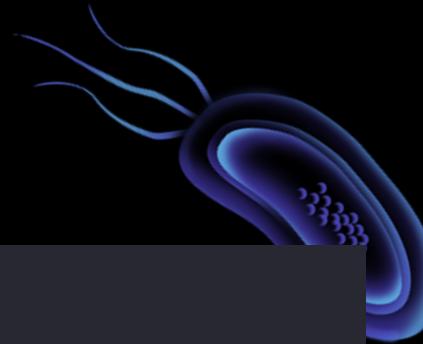
# imprimi a matriz com os valores das penalidades
for linha in matriz:
    print(linha)
```

Saída:

```
[0, -6, -12, -18, -24, -30, -36]
[-6, 0, 0, 0, 0, 0, 0]
[-12, 0, 0, 0, 0, 0, 0]
[-18, 0, 0, 0, 0, 0, 0]
[-24, 0, 0, 0, 0, 0, 0]
[-30, 0, 0, 0, 0, 0, 0]
[-36, 0, 0, 0, 0, 0, 0]
[-42, 0, 0, 0, 0, 0, 0]
[-48, 0, 0, 0, 0, 0, 0]
```

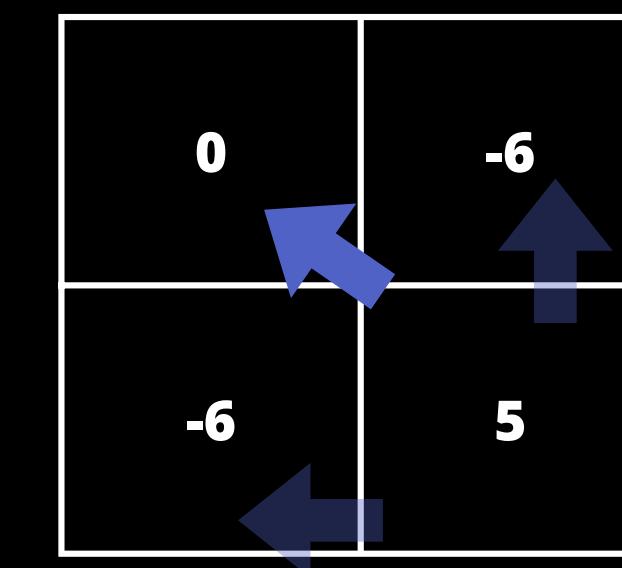
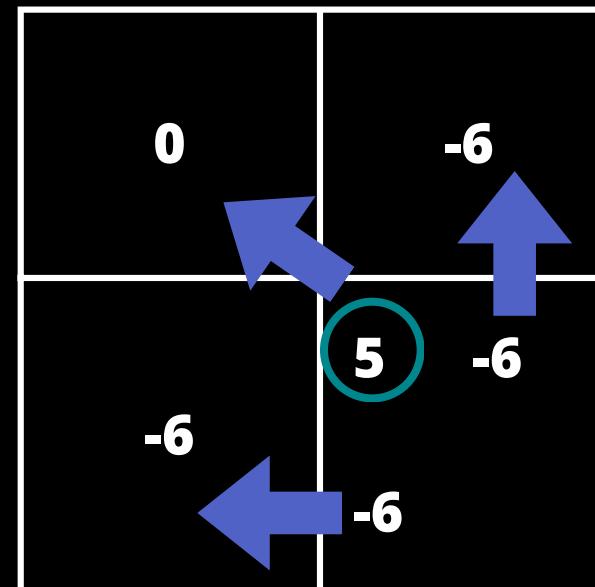
Algoritmo

Algoritmo



```
# preenchimento da matriz
for i in range(1, linhas):
    for j in range(1, colunas):
        # pega o valor do canto superior esquerdo e soma - se der igual pega a pontuação igual
        # (match), se não, pega a pontuação diferente
        alinhar = matriz[i-1][j-1] + (igual if seq1[i-1] == seq2[j-1] else diferente)
        # pega o valor acima e soma a pontuação da lacuna
        deletar = matriz[i-1][j] + lacuna
        # pega o valor da esquerda e soma a pontuação da lacuna
        inserir = matriz[i][j-1] + lacuna
        # coloca na matriz o maior valor entre
        matriz[i][j] = max(alinhar, deletar, inserir)
```

```
for linha in matriz:
    print(linha)
```



Saída:

[0,	-6, -12, -18, -24, -30, -36]
[-6,	5, -1, -7, -13, -19, -25]
[-12,	-1, 3, -3, -9, -15, -21]
[-18,	-7, -3, 8, 2, -4, -10]
[-24,	-13, -2, 2, 6, 7, 1]
[-30,	-19, -8, 3, 0, 4, 5]
[-36,	-25, -14, -3, 1, -2, 2]
[-42,	-31, -20, -9, -5, 6, 0]
[-48,	-37, -26, -15, -4, 0, 11]

T T C A T A

T G C T C G T A

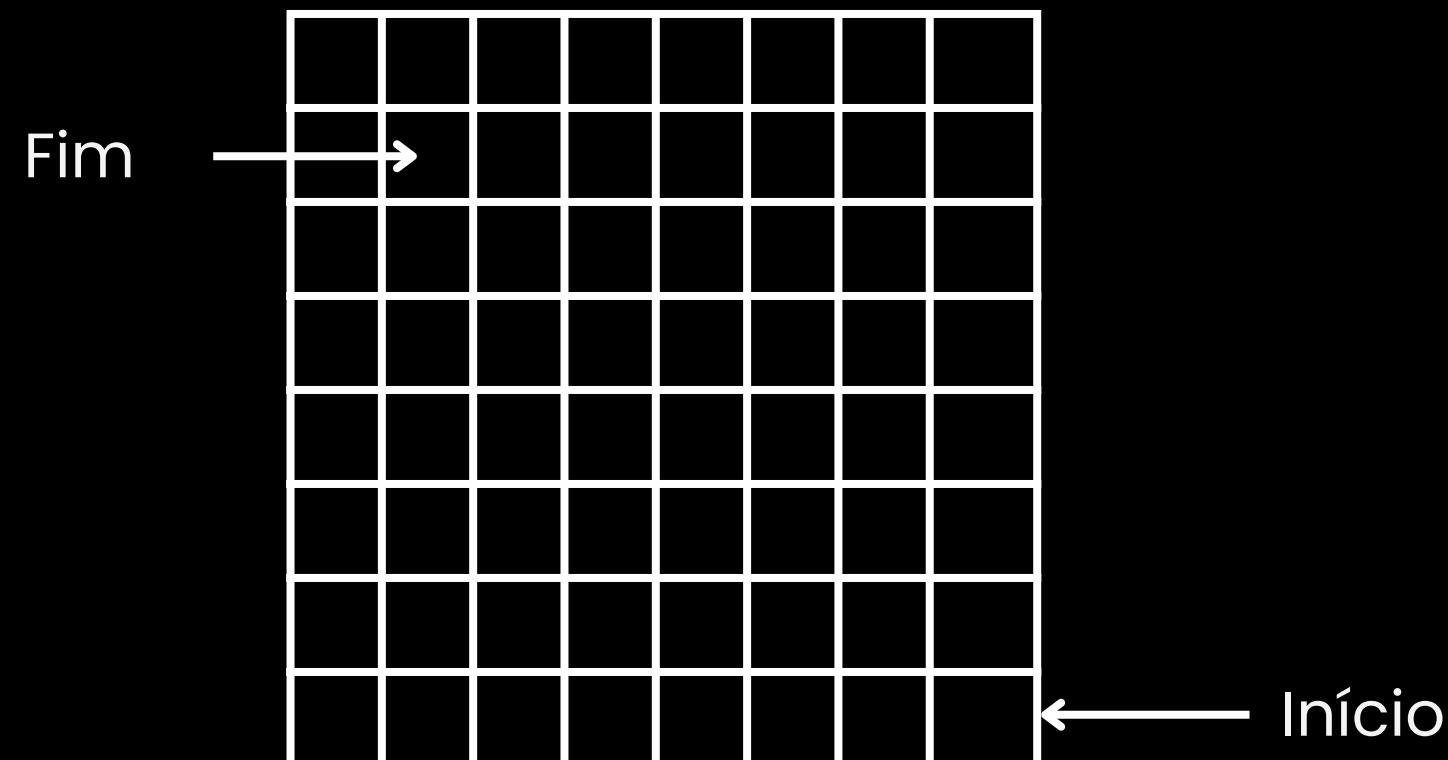
Algoritmo



```
# Rastreamento (Backtracking)
alinhamento_1, alinhamento_2 = "", "" # inicializa as variáveis de alinhamento 1 e 2

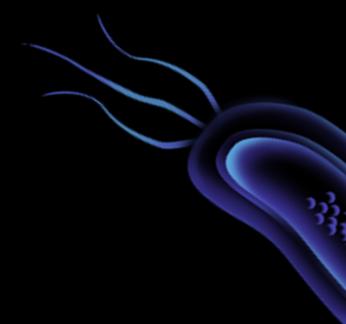
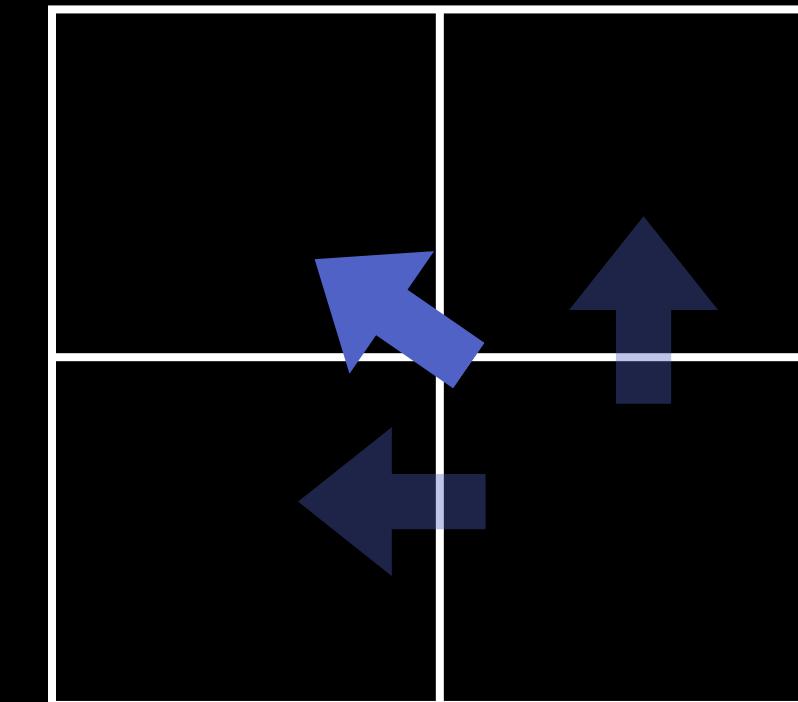
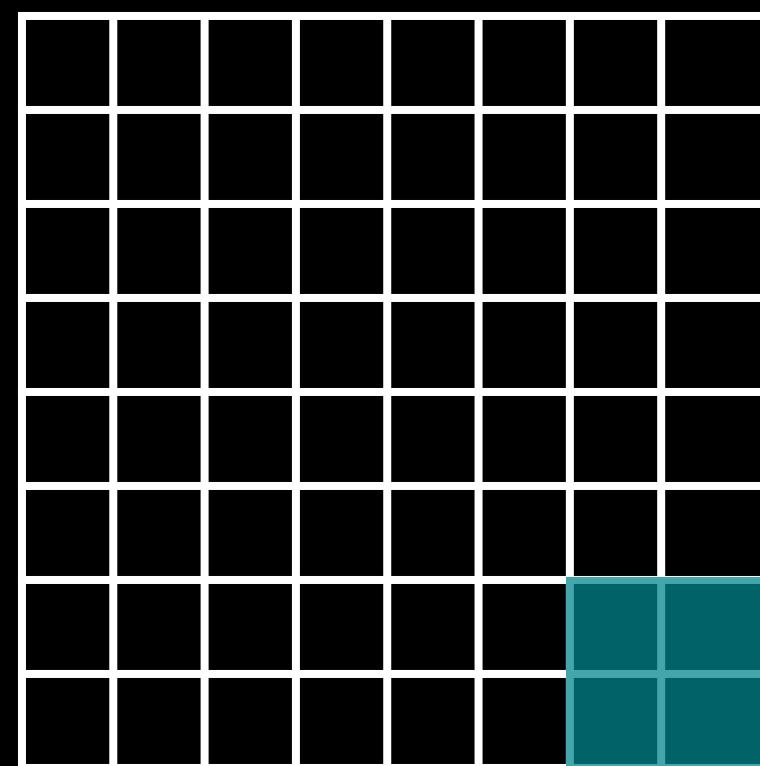
# para cada linha de linhas e para cada coluna de colunas
linha, coluna = linhas - 1, colunas - 1

# loop enquanto ambos os índices linha e coluna forem maiores que 0,
# o que significa que ainda não chegamos à borda superior ou à esquerda da matriz.
while linha > 0 and coluna > 0:
```



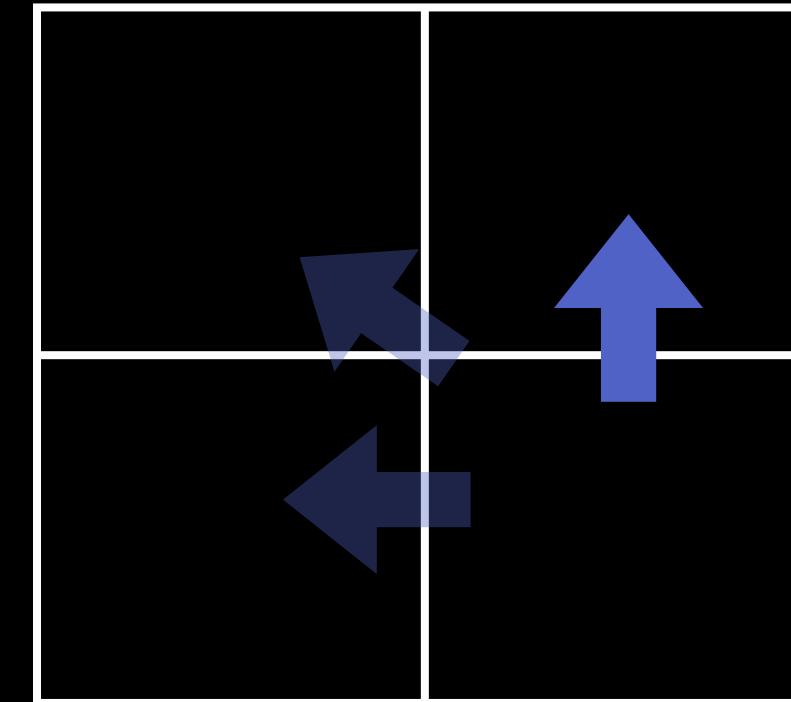
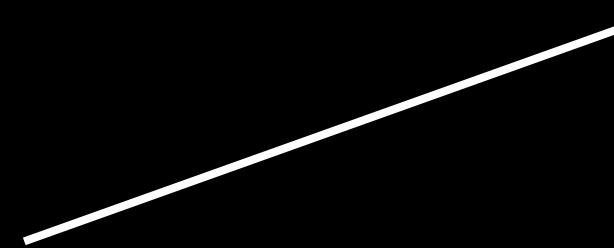
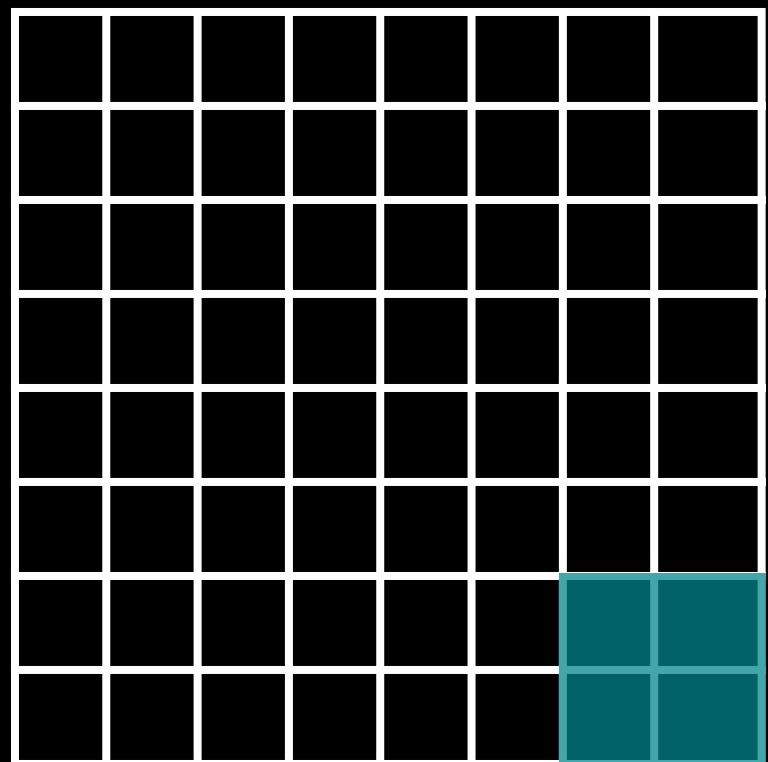
Algoritmo

```
# verifica se a pontuação atual na matriz é igual à pontuação da diagonal superior esquerda mais a pontuação  
# correspondente ao caractere atual das duas sequências.  
# Se for verdadeiro, significa que o caractere está alinhado (match ou mismatch).  
if matriz[linha][coluna] == matriz[linha-1][coluna-1] + (igual if seq1[linha-1] == seq2[coluna-1] else diferente):  
  
    # adiciona os caracteres correspondentes das sequências seq1 e seq2 às strings de alinhamento.  
    alinhamento_1 = seq1[linha-1] + alinhamento_1  
    alinhamento_2 = seq2[coluna-1] + alinhamento_2  
  
    # move os índices linha e coluna para a posição da diagonal superior esquerda na matriz.  
    linha -= 1  
    coluna -= 1
```



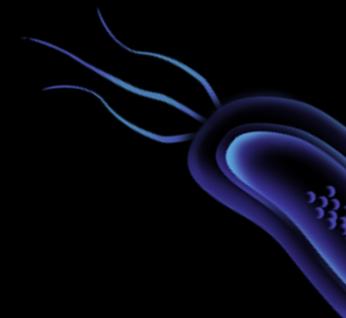
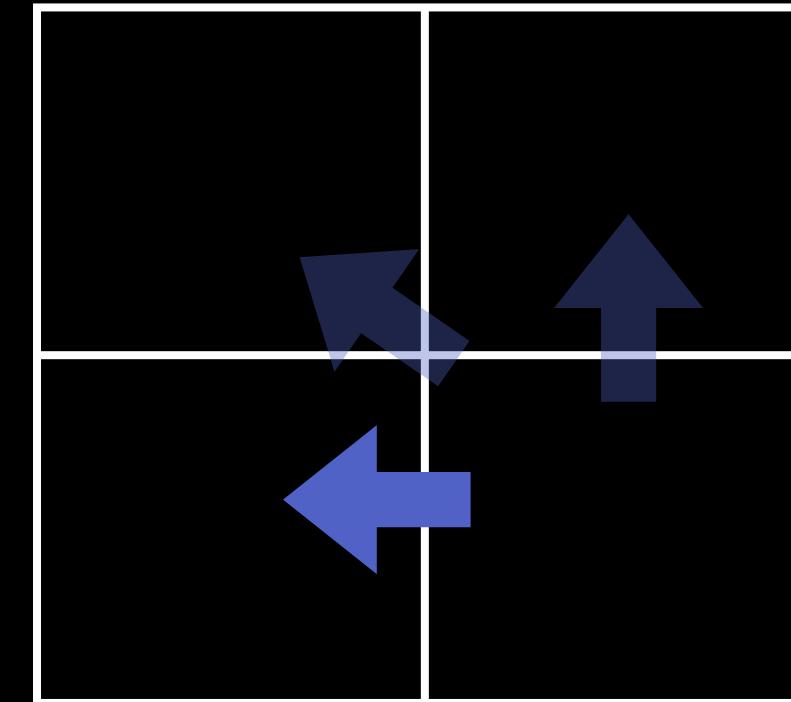
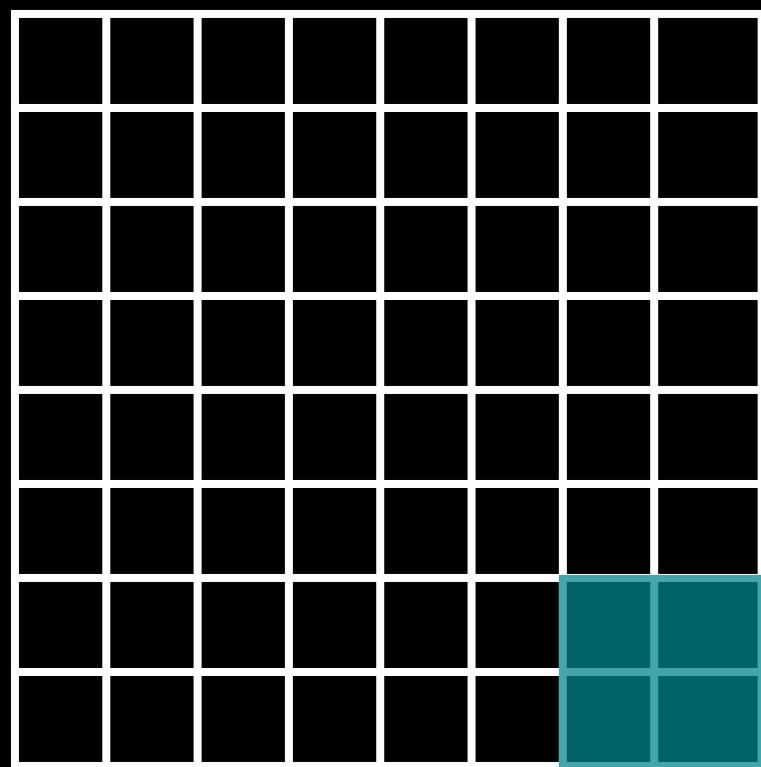
Algoritmo

```
# verifica se a pontuação atual na matriz é igual à pontuação da célula acima mais a penalidade de lacuna.  
# isso indica que foi introduzida uma lacuna na sequência seq1.  
elif matriz[linha][coluna] == matriz[linha-1][coluna] + lacuna:  
    #Adiciona um caractere de lacuna à sequência seq1 e um caractere correspondente de seq2 à string de alinhamento.  
    alinhamento_1 = seq1[linha-1] + alinhamento_1  
    alinhamento_2 = "-" + alinhamento_2  
  
    # move o índice linha, caminhando para cima na matriz  
    linha -= 1
```



Algoritmo

```
# verifica se a pontuação atual na matriz é igual à pontuação da célula à esquerda mais a penalidade de lacuna.  
# isso indica que foi introduzida uma lacuna na sequência seq2.  
else:  
    # Adiciona um caractere de lacuna à sequência seq2 e um caractere correspondente de seq1 à string de alinhamento.  
    alinhamento_1 = "-" + alinhamento_1  
    alinhamento_2 = seq2[coluna-1] + alinhamento_2  
  
    # move o índice coluna, caminhando para a esquerda na matriz  
    coluna -= 1
```



Algoritmo

```
print("Alinhamento 1:", alinhamento_1)
print("Alinhamento 2:", alinhamento_2)
```

Saída:

Alinhamento 1: TGTCGTA
Alinhamento 2: T - - TCATA



Conclusão

- Algoritmo de Needleman Wunch: problema central da bioinformática;
- Possui uma abordagem de programação com uma solução ótima;
- Identificação de regiões similares entre duas sequências: relações funcionais, estruturais e evolutivas em proteínas e microrganismos de diferentes espécies.

Por isso, sem dúvida, o torna uma técnica de extrema importância na bioinformática.

OBRIGADA!

