Algoritmo Needleman-Wunsch

Alunos: Gabriel Reis, Roberto Otavio

1) Em nossa matriz, S1 de tamanho M, será o eixo Y e S2, de tamanho N, será o eixo X.

O alinhamento global tem como objetivo analisar 2 sequências de DNA e encontrar 2 novas sequências, a partir das 2 iniciais, que sejam o mais próximas possíveis entre si. Para garantir esse alinhamento, é utilizado como possível preenchimento um “gap” no sequenciamento genético, representado por escrito como “-”. O algoritmo de needleman-wunsch utiliza de programação dinâmica para criar a matriz que possui um sistema de pontuação para apontar a melhor possibilidade de alinhamento.

A matriz terá tamanho N+1 por M+1, pois ela é montada considerando que toda cadeia tem uma possibilidade de caractere a mais, que seria o vazio. Esse vazio é inserido como a primeira posição de cada eixo (0,0). As futuras posições representam uma letra de cada sequência. Cada posição dessa matriz deve apontar a melhor pontuação para aquele conjunto de caracteres das 2 sequências.

O fator “maior pontuação” é garantido pela abordagem utilizada para preencher cada posição da matriz. As pontuações possíveis são: match, mismatch e gap, onde é um padrão para os valores que match > mismatch > gap.

A posição (0,0) é preenchida com 0. As posições (0, y) e (x, 0) são preenchidas com o valor de x ou y multiplicados pelo valor do gap. Após isso, é dado o início do preenchimento da matriz pela posição (1,1), é a partir desse momento que é aplicado o algoritmo de escolha da melhor pontuação.

É obtido os 3 valores possíveis para essa posição.

- O valor da acima do ponto atual, (x, y-1), adicionando o valor do gap;
- O valor a esquerda do ponto atual, (x-1, y), adicionando o valor do gap;
- O valor a diagonal do ponto atual, (x-1, y-1), adicionando o valor do match se os respectivos caracteres dessa posição forem iguais ou adicionando o mismatch caso os caracteres não sejam iguais.

Possuindo estes 3 valores, é obtido o maior dos 3 para a posição atual da matriz. Ao finalizar o preenchimento da matriz, a última posição preenchida possuirá a melhor pontuação que considera todos os caracteres daquelas 2 cadeias, sendo consequentemente a melhor pontuação entre S1 e S2.

Para obter as 2 novas sequências, considerando que a matriz possui somente as pontuações, é possível percorrer a matriz no sentido inverso. Inicia-se no ponto (N+1, M+1), é calculado os 3 valores novamente, cima, esquerda e diagonal e comparado com o valor da posição que estamos. Ao encontrarmos uma igualdade nessa comparação, significa que foi desta posição que viemos para a posição atual quando construímos a matriz, com essa informação fazemos a seguinte ação, a depender da posição de origem.

- Caso viemos da diagonal, adicionamos as 2 letras dos respectivos eixos as novas sequências e andamos para a diagonal na leitura da matriz.
- Caso viemos da posição cima, adicionamos a letra de cima da posição que estamos na nova S1, adicionamos um gap na sequência S2 e andamos para cima na leitura da matriz.
- Caso viemos da posição esquerda, adicionamos a letra da esquerda da posição que estamos na nova S2, adicionamos um gap na sequência S1 e andamos para esquerda na leitura da matriz.

Fazemos isso até atingir alguma borda da esquerda ou de cima da matriz. Após esse percurso, teremos as 2 novas sequências de melhor pontuação para o alinhamento.

2) O código implementado para needleman-wunsch foi o seguinte

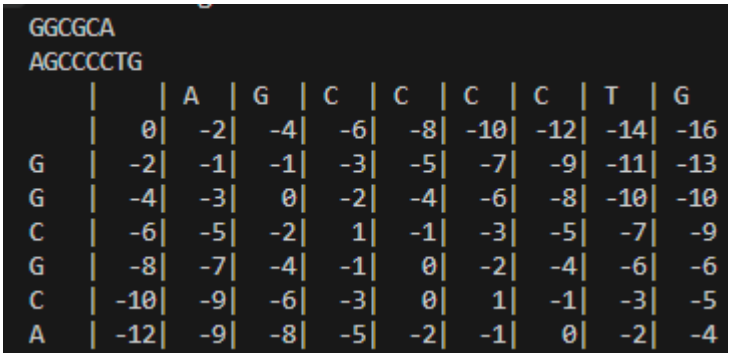
```
def needleman_wunsch(s1 : str, s2 : str, match_value : int, mismatch_value :
int, gap_value : int):
    # cria uma matriz de 0s de tamamho S1+1 por S2+1
    size_s1 = len(s1)
    size_s2 = len(s2)
    matrix = np.zeros((size_s1 + 1, size_s2 + 1))

    # Adiciona o valor nos gaps nos eixos (0,y) e (x,0), sendo
    # Nas posições (x, 0) adicionado o valor x * gap
    # Nas posições (0, y) adicionado o valor y * gap
    matrix[:,0] = np.linspace(0, size_s1 * gap_value, size_s1 + 1)
    matrix[0,:] = np.linspace(0, size_s2 * gap_value, size_s2 + 1)

    # Inicia a populacao da matriz, calculando o valor de cima, esquerda e
diagonal e escolhendo o melhor valor
    # Esquerda: Valor da esquerda anterior + gap_value
    # Cima: Valor de cima anterior + gap_value
    # Diagonal: Valor anterior de cima e esquerda +
    #   (match_value se os caracteres que represetam essa casa anterior forem
iguais
    #   mismatch_value se os caracteres que represetam essa casa anterior
forem diferentes)
    for line in range(1, size_s1+1):
        for column in range(1, size_s2+1):
            top = matrix[line, column-1] + gap_value
            left = matrix[line-1, column] + gap_value
            diagonal = matrix[line-1, column-1] + (match_value if s1[line-1]
== s2[column-1] else mismatch_value)
            matrix[line, column] = max([top, left, diagonal])
```

para printar a matriz foi utilizado o seguinte algoritmo

```python
def print_matrix(matrix : list, s1 : str, s2 : str):
    matrix_s1 = "   " + s2
    matrix_s2 = " " + s1
    print(" |".join(f" {l} " for l in matrix_s1))
    pos = 0
    for line in matrix:
        l = "|".join(f"{column:4.0f}" for column in line)
        print(f"{matrix_s2[pos]}   |{l}")
        pos += 1
```

Para as seguintes cadeias:
GGCGCA
AGCCCCTG
é exibida a seguinte matriz:

		A	G	C	C	C	C	T	G
	0	-2	-4	-6	-8	-10	-12	-14	-16
G	-2	-1	-1	-3	-5	-7	-9	-11	-13
G	-4	-3	0	-2	-4	-6	-8	-10	-10
C	-6	-5	-2	1	-1	-3	-5	-7	-9
G	-8	-7	-4	-1	0	-2	-4	-6	-6
C	-10	-9	-6	-3	0	1	-1	-3	-5
A	-12	-9	-8	-5	-2	-1	0	-2	-4



3) Para realizar o backtracking, foi utilizado o seguinte algoritmo

```python
def backtracking(s1 : str, s2 : str, matrix : list, match_value : int,
mismatch_value : int, gap_value : int, print_path=False):
    # Obtem as informações necessárias
    line_position = len(s1)
    column_position = len(s2)
    reversed_s1 = []
    reversed_s2 = []
```

```
    # Percorre a matriz ate alcancar alguma borda da esquerda ou de cima da
matriz
    # O ponto de partida é o ponto inferior direito da matriz
    # Essa posicao possui a melhor pontuacao do alinhamento dessas cadeias
    while line_position > 0 or column_position > 0:
        # Para saber de onde o valor daquela posição veio, é obtido o valor
da posicao que estamos
        # e recalculado os valores das posicoes adjacentes
        # Ao encontrar que o valor veio da:
        # * Diagonal: Mantem as 2 letras, de S1 e S2
        # * Cima: Adiciona um gap a nova S2 e mantem a letra em S1
        # * Esquerda: Adiciona um gap a nova S1 e mantem a letra em S2

        actual = matrix[line_position, column_position]
        back_score = 0
        path = ""

        # Chegou na primeira coluna
        if column_position == 0:
            path = "up"
            back_score = matrix[line_position-1][column_position]

            reversed_s1.append(s1[line_position-1])
            reversed_s2.append("-")
            line_position -= 1
            continue
        # Chegou na primeira linha
        if line_position == 0:
            path = "left"
            back_score = matrix[line_position][column_position-1]

            reversed_s1.append("-")
            reversed_s2.append(s2[column_position-1])
            column_position -= 1
            continue

        # Checa se veio da diagonal
        if (actual == matrix[line_position-1][column_position-1] +
(match_value if s1[line_position-1] == s2[column_position-1] else
mismatch_value)
        ):
            path = "diagonal"
            back_score = matrix[line_position-1][column_position-1]
```

```
            reversed_s1.append(s1[line_position-1])
            reversed_s2.append(s2[column_position-1])
            line_position -= 1
            column_position -= 1

        # Checa se veio de cima
        elif (actual == matrix[line_position-1][column_position] + gap_value
        ):
            path = "up"
            back_score = matrix[line_position-1][column_position]

            reversed_s1.append(s1[line_position-1])
            reversed_s2.append("-")
            line_position -= 1

        # Veio da esquerda
        else:
            path = "left"
            back_score = matrix[line_position][column_position-1]

            reversed_s1.append("-")
            reversed_s2.append(s2[column_position-1])
            column_position -= 1

        print(f"From {actual} moves {path} to {back_score}") if print_path
else None

    # As string finais obtidas estão invertidas, aqui invertemos elas
    aligned_s1 = "".join(reversed_s1)[::-1]
    aligned_s2 = "".join(reversed_s2)[::-1]

    return aligned_s1, aligned_s2
```