Алгоритм Нидлмана-Вунша. Практическая реализация.

1. Задача:

Построить выравнивание двух аминокислотных или нуклеотидных последовательностей.

2. Постановка задачи:

Имеются две строки, состоящие из некоторых повторяющихся символов (обычно - AGCT). Необходимо с помощью вставки некоторого числа символов “пропуск” преобразовать начальные строки к строкам одинаковой длины так, что функция, оценивающая “похожесть” строк, даст наилучший результат.

В качестве оценки похожести строк будет использоваться система поощрений и наказаний за взаимное состояние двух i-х символов преобразованных строк: совпадение => +1; несовпадение или “пропуск” на месте одной из них => -1.

3. Решение:

Обработка входных данных:

```
line1 = str(input())
line2 = str(input())


line1 = line1.strip()
line2 = line2.strip()
len1 = len(line1)
len2 = len(line2)
```

Создание будущей матрицы похожести строк:

```
score = [[0]*(len1+1) for i in range(len2+1)]
```

Установка весов взаимных состояний символов и создание функции для получения этих взаимных состояний:

```
weight_match = 1
weight_mismatch = -1
weight_gap = -1


def match_score(first, second):
    if first == second:
```

```
        return weight_match
    elif first == '-' or second == '-':
        return weight_gap
    else:
        return weight_mismatch
```

Создание матрицы похожести строк:

```
for j in range(0, len1 + 1):
    score[0][j] = - j
for i in range(0, len2 + 1):
    score[i][0] = - i
for j in range(1, len1 + 1):
    for i in range(1, len2 + 1):
        # Calculate the score by checking the top, left, and diagonal
cells
        match = score[i - 1][j - 1] + match_score(line1[j - 1], line2[i -
1])
        delete = score[i - 1][j] + weight_gap
        insert = score[i][j - 1] + weight_gap
        # Record the maximum score from the three possible scores
calculated above
        score[i][j] = max(match, delete, insert)
```

Построение последовательностей по полученной матрице:

```
# Setting variables to store alignment
align1 = ""
align2 = ""

# Start from bottom right cell of matrix
j = len1
```

```
i = len2


while i > 0 and j > 0:
    # (in that case we'll touch top or left edge of the matrix)
    score_current = score[i][j]
    score_diagonal = score[i - 1][j - 1]
    score_up = score[i][j - 1]
    score_left = score[i - 1][j]


    # Check to figure out which cell the current score was calculated
from,
    # then update i and j to correspond to that cell.
    if score_current == score_diagonal + match_score(line1[j - 1], line2[i
- 1]):
        align1 += line1[j - 1]
        align2 += line2[i - 1]
        i -= 1
        j -= 1
    elif score_current == score_up + weight_gap:
        align1 += line1[j - 1]
        align2 += '-'
        j -= 1
    elif score_current == score_left + weight_gap:
        align1 += '-'
        align2 += line2[i - 1]
        i -= 1


# Then we finish tracing up to the top left cell
while j > 0:
    align1 += line1[j - 1]
    align2 += '-'
    j -= 1
```

```
while i > 0:
    align1 += '-'
    align2 += line2[i - 1]
    i -= 1

# Our alignments were constructed from end to start of actual lines, so we
reverse them to get actual answer
align1 = align1[::-1]
align2 = align2[::-1]

print(align1 + "\n" + align2)
input('Press ENTER to exit')
```

4. Примеры тестов.

Ввод:

ttcttacgac

tcagtcgctt

Вывод:

ttc-ttacgac--

-tcagt-cg-ctt

Ввод:

ctttggttgcagggttggccccgttagcgt

acccttttctgactgcacgtgcttaatggtg

Вывод:

----ctt-tggttgcagggttggccccgtt-a-gcgt-

acccttttctgactgca-cg-t-g---c-ttaatg-gtg

Данные взяты с сайта bioinformatics.com - Random DNA Sequence: https://www.bioinformatics.org/sms2/random_dna.html?

5. Выводы:

Алгоритм реализован и проверен на случайных данных. Благодаря использованию системы штрафов/поощрений для оценки похожести символов, не привязанной к их значениям, вместо матрицы похожести для символов ACGT:

1. Возможно использование алгоритма для выравнивания последовательностей из любых символов.
2. Возможна менее высокая ценность алгоритма для решения реальных задач.