Задача 1 (Выравнивание последовательностей - алгоритм Нидлмана-Вунша)

Стельмах Татьяна

Январь 2021

Содержание

1 Введение

Задача: Необходимо продемонстрировать алгоритм Нидлмана-Вунша, провести выравнивание для двух последовательностей большой длины ($> 10^5$ нуклеотидов), проверить необходимо на меньших.

2 Описание метода

Для начала составим таблицу, каждая ячейка которой хранит веса оптимальных выравниваний, а строится она следующим образом:
Сначала заполняем первый столбец и первую строку:

```
# Заполняем первый столбец
    for i in range(0, m + 1):
        score[i][0] = d * i

# Заполняем первую строчку
    for j in range(0, n + 1):
        score[0][j] = d * j
```

А остальные ячейки заполнятся следующим образом:

$$F(i,j) = \max \begin{cases} \text{F(i-1,j-1) + g(i,j)}, \\ \text{F(i-1,j) - 1}, \\ \text{F(i,j-1) - 1}. \end{cases}$$

$$\text{где } g(i,j) = \begin{cases} \text{1, if s1[i] = s2[j]}, \\ \text{-1 , if s1[i] } \neq \text{ s2[j]} \end{cases}$$

А затем выполняется обратный обход для нахождения ответа.

3 Описание данных

Корректность написанного алгоритма мы будем проверять с помощью вручную сгенерированных тестов (малой длины), затем воспользуемся популярной биологической базой данных от *NCBI*, чтобы проверить работу программы на цепочках большей длины.

4 Описание постановки эксперимента

Взглянем на код (Посмотрим на то, как получается таблица):

```
n = len(seq1)
m = len(seq2)

score = zeros(m + 1, n + 1)

# Заполняем первый столбец
for i in range(0, m + 1):
    score[i][0] = d * i

# Заполняем первую строчку
for j in range(0, n + 1):
    score[0][j] = d * j

# Вычисляем табличку
for i in range(1, m + 1):
    for j in range(1, n + 1):
        match = score[i - 1][j - 1] + match_score(seq1[j - 1], seq2[i - 1])
        delete = score[i - 1][j] + d
        insert = score[i][j - 1] + d
        # находим максимальный
        score[i][j] = max(match, delete, insert)
```

А теперь на обратный обход и восстановление ответа:

```
align1 = ""
align2 = ""

# Начинаем с нижнего правого конца матрицы
i = m
j = n

while i > 0 and j > 0:  # конец, когда придём к первой строке или столбцу
    score_current = score[i][j]
    score_diagonal = score[i - 1][j - 1]
    score_up = score[i][j - 1]
```

```
            score_left = score[i - 1][j]

            # Посмотрим на то, как пришли в эту ячейку
            # Затем обновим i и j
            if score_current == score_diagonal + match_score(seq1[j - 1], seq2[i - 1]):
                align1 += seq1[j - 1]
                align2 += seq2[i - 1]
                i -= 1
                j -= 1
            elif score_current == score_up + d:
                align1 += seq1[j - 1]
                align2 += '-'
                j -= 1
            elif score_current == score_left + d:
                align1 += '-'
                align2 += seq2[i - 1]
                i -= 1

    # Перейдём к левому верхнему углу
    while j > 0:
        align1 += seq1[j - 1]
        align2 += '-'
        j -= 1
    while i > 0:
        align1 += '-'
        align2 += seq2[i - 1]
        i -= 1


    # Развёрнем получившиеся строки
    align1 = align1[::-1]
    align2 = align2[::-1]
```

5 Результаты

Результат применения алгоритма на цепочках малой длины:

		G	A	A	C
	0	-1	-2	-3	-4
C	-1	-1	-2	-3	-2
A	-2	-2	0	-1	-2
A	-3	-3	-1	1	0
G	-4	-2	-2	0	0
A	-5	-3	-1	-1	-1
C	-6	-4	-2	-2	0

Рисунок 1. Тест 1

```
-GA-AC
CAAGAC
```

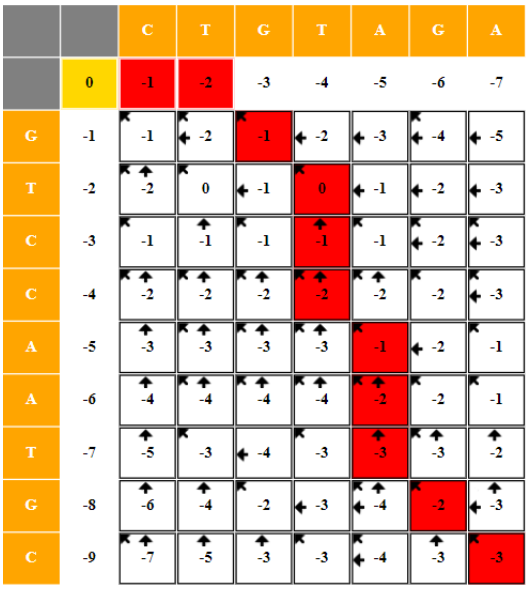

Рисунок 2. Тест 2

```
CTGTA--GA
GTCCAATGC
```

А вот результат применения алгоритма к части днк двух представителей семейства обезьян:

```
1: T--T-TC-TGCCAG--GAC-TCT-TGA--TGA-TGTGC-GGT--TTG-CTT--TCAGGGATAG-GAAG-----ATAAAAACGCT--AA--AAGAAGACAA
2: TGCTGTCTTGGCTGCTGCCATGTAAGATGTGACTTTGCTCCTCCTTGCCTTCCACAATGATTGTGAGGCCTCCCTAACCATG-TGGAACTATG-AGTCAA
1: --AAA-TAT-TAACACAATAATAAA-AAAAAAAAGC-CAGAAAAAATAGTAGTTT-TTCAAGAACACT-TCAGAATCCTGATACTTTACATGA-AC-CAA
2: TTAAACT-TCTTTC-C-TTTATAAATCACTCAGA-CTCAG---GTATA-T-CTTTATT--AG--CAGTCTGAGAA--C--AGAC--TA-AT-ACACTCTT
1: GAAATATAGACATAA--A--AAA---GAG-ACACACACCTGTAAGGAGAGATGAGACAC-AGGACTGGGT--TACTTTTGATGAAGAGGCTTCTTA-CCA
2: TTTATATA-AC-TAATCACCAAATTCTAGTTTTC-C-CCT-T----CCAGAT--GTCTCTA-GA-TTTGTCCTTCTTTTGAT----ATTTTTATCACCCT
1: GTTACA-ACTG-AAGT-GAGTAAGAAGAAATTAGCTACTTTTTTCAAGATCT-GT-GGTAGATAAAATGTAGGTCAGTTTGAGTCTGAAATAAATGGAGG
2: GTTTCAGCCTGCCA-TCCAGT--G--CACACT---TA--TTGGACACCATCTCCTAGG-A-ACAACAT-TA-G-C----TGA-TGGGGGATATATCAAGG
1: AGTGA---AAT-ACAGAAGTACAGGGA--TAGA-GAGACA-GAACATAAATCTAAGTTACCAACATCAGGAATAAAAAAGAGGACAT--TAATACAGACA
2: ATTAACCCAATCCCTG-CCTTCAGGGAGCTA-ACCAG-CAGGGAGAT-GGTC-AAG-TA--AACAGCAGCCAT-AAGCAG-GG-TGTGGTAA-A-TG-CT
1: GAAATTATGAATATCAGAATATAATAGATGTTAT-GAACAATTCTTTAGCACTAATTTTG-ACACTTTA---ATGAAGTGGACAAAATAGTTAACACACA
2: G--TTTA-G-A-GGCTG--T-GCAGGGGTGCTATGGAGC-A--C---AG-A-GAA-CTTGAAAAC-CTAGCCCTG-AG-GGGC---CT--CT-GCACATA
```

6 Краткие выводы

Была написана программа для выравнивания цепочек нуклеотидов любой длины, проверка корректности её работы выполнена на цепочках малой и большей длины.

Источники

- http://experiments.mostafa.io/public/needleman-wunsch/index.html **- визуализация данных**
- http://iitp.ru/upload/userpage/146/1_alignment.pdf **- описание алгоритма**
- https://www.ncbi.nlm.nih.gov/genbank **- база данных**