Greedy algorithm for aligning DNA sequences

1. Постановка задачі :

Впорядковування послідовностей ДНК це процес в якому дві послідовності, які можуть починатися однаково, але скоріше за все мають помилку в послідовності. Мета впорядкування це пошук такого вирівнювання, яке максимально зблизить ці послідовності. Це використовується біологами для побудови філогенетичних дерев, які показують взаємозв'язки між живими організмами. В кожному організмі присутні білки які постійно сканують ДНК та проводять процес репарації(синтезують нові нуклеотиди, вирізають зайві, тощо).

Динамічний підхід хоч і працює швидше, ніж повний перебір всіх можливих комбінацій, все ще вимагає багато пам'яті та часу, в той час як жадібний алгоритм для вирівнювання послідовностей працює за поліноміальний час, і знаходить найкраще впорядкування.

Нехай ***A*** **=** $\{a_1, \dots a_n\}$, а ***B*** **=** $\{b_1, \dots b_k\}$, ***N*** та ***M*** їх довжини, а змінна ***score*** це загальний рахунок близькості цих послідовностей.

Існує три можливі випадки схожості кожного елементу ***A*** та ***B***:

1) $a_i = b_i$ – match (співпадіння). В такому випадку до загального рахунку схожості додається два бали (***+2***)
2) $a_i \neq b_i$ – mismatch (неспівпадіння, помилка). В цьому випадку від рахунку віднімається одиниця (***-1***)
3) GAP (пропуск). В випадку коли довжини послідовностей різні, або для кращого результату потрібно додати додати пропуск, від рахунку віднімається двійка. (***-2***).

2. Власний опис алгоритму (що відбувається на кожному кроці).

Головний клас DNAAlignment приймає два аргументи : стрічки dna1 та dna2. Для початку підрахування викликається метод get_best_score. Спочатку він підраховує довжини послідовностей, і перевіряє чи вони:

1) Рівні нулю
    - Тоді метод повертає порожній екземпляр допоміжного класу Alignment.
2) Друга послідовність має довжину нуль.
    - Відбувається рекурсивний виклик функції з обрізаною першою послідовнісю.
    - З отриманої змінної result (змінна типу Alignment) викликається метод add_match, який в даному випадку додає пропуск (gap) до другої послідовності.
3) Перша послідовність має довжину нуль.
    - Відбувається те саме що в другому випадку, але в іншому порядку.
4) Якщо обидві послідовності не дорівнюють нулю.

a. Змінна first – рекурсивний виклик функції, в яку передається обрізана перша послідоність (dna1[1:]), та повна друга. Далі до first додається match виклоком методу add_match.
b. Змінна second – рекурсивний виклик функції, в яку передається обрізана друга послідоність (dna2[1:]), та повна перша. Далі до second додається match виклоком методу add_match.
c. Змінна both - рекурсивний виклик функції, в яку передаються обидві обрізані послідовності (dna1[1:], dna2[1:]).

Після цього повертається змінна з найбільшим рахунком. Рахунок визначається способом, який описаний вище.

3. Оцінка складності алгоритму :

(Для розгляду брався псевдокод із запропонованої статті).

1) За допомогою while циклу знайти першу позицію, в якій $a_{i+1} \neq b_{i+1}$. Складність цього кроку ***O(max(N, M)),*** де ***N*** це дожина першої стрічки, а ***M*** це довжина другої.
2) Наступний крок – ***do while*** цикл, яикй переривається тоді коли ***L > U + 2***, де ***L*** це нижня межа, а ***U*** – вища. Складність цього кроку – ***L***
3) В ***do while*** циклі другого кроку присутній ще один while цикл, який працює так само як цикл з першого кроку.
4) Отже, загальна складність алгоритму виходить : $\boldsymbol{O(d_{max}^2 + L)}$

4. Опис експерименту :

Для проведення експерименту ми дослідили кілька послідовостей нуклеотидів в ДНК та порівнювали послідовність нуклеотидів здорового ДНК і ДНК ураженого захворюванням.

Муковісцидоз (Cystic Fibrosis)

Спадкова хвороба, що вражає легені, травну систему та потові залози. Вона може бути спричинена багатьма різними мутаціями, та ми розглянемо мутацію, яка спричиняє 70% захворювань.

Правильна послідовність “здорових” амінокислот повинна виглядати так:

A T C T T T G G T

Ile Phe Gly

При захворюванні зникає амінокислота Фениланін(Phe), та вона не просто випадає послідовності, а забирає один C нуклеотид Леуцину замість свого T. Після ураження послідовність виглядатиме так:

A T T G G T

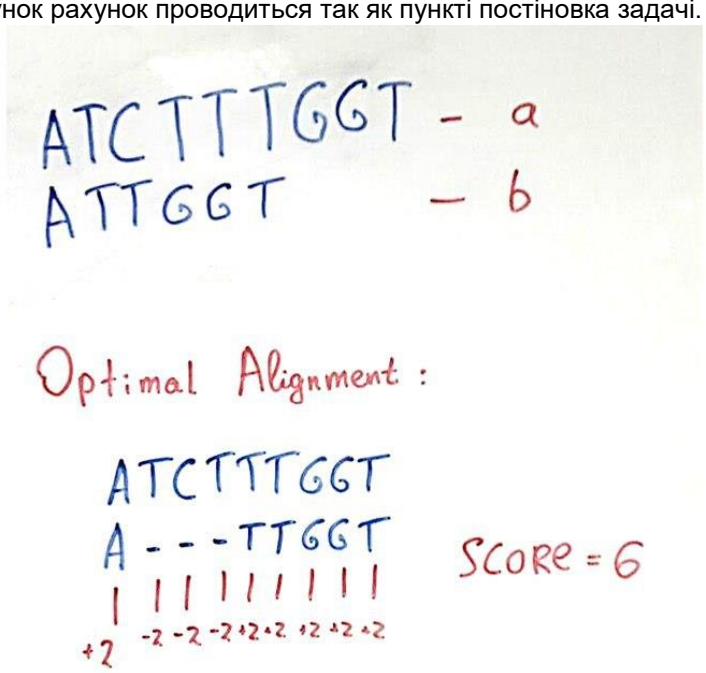
Ile Gly

Алгоритм знехтує тим, що обидві послідовності починаються з нуклеотидів A і T (що на перший погляд виглядає неправильно). Оскільки алгоритм жадібний, він надасть пріоритет ціннішій послідовності TTGGT, а не AT та TGGT і поверне такі послідовності:

ATCTTTGGT

A---TTGGT

Обрахунок рахунок проводиться так як пункті постіновка задачі. Наочний приклад:



Рахунок: 6

Анемія(Sickle Cell Anemia)

Поширене спадкове захворювання крові. Послідовність, яка може бути вражена:

При захворюванні Гемоглубін(Glu) міняє свій нуклеотид A на T і цим перетворюється у Валін(Val). Послідовність стає такою:

У даному випадку наш алгоритм поверне не змінені послідовності, бо найкращий рахунок(15) буде саме при таких послідовностях: CCTGAGGAG, CCTGTGGAG.

4. Висновок :

1) При розв’язанні задачі ми навчильсь з нуля розібрати дану проблему, та способи її вирішення. Не маючи сильного фундаменту в генетиці ми змогли реалізувати алгоритм вирівнювання послідовностей ДНК.

2) Найбільшу складність викликав псевдокод, представлений в статті. Змінні в ньому не несуть багато інформації про себе, і чіткого опису роботи цього жадібного алгоритму знайти не вдалось. Також сам процес вирівнювання послідовносновностей вимагає знань гететики та біології, тому зрозуміти те, навіщо вирівнювати послідовності було трохи складно.

3) Незрозумілим залишилось те як реалізувати даний жадібний алгоритм за поліноміальний час, згідно запропонованого псевдокоду.