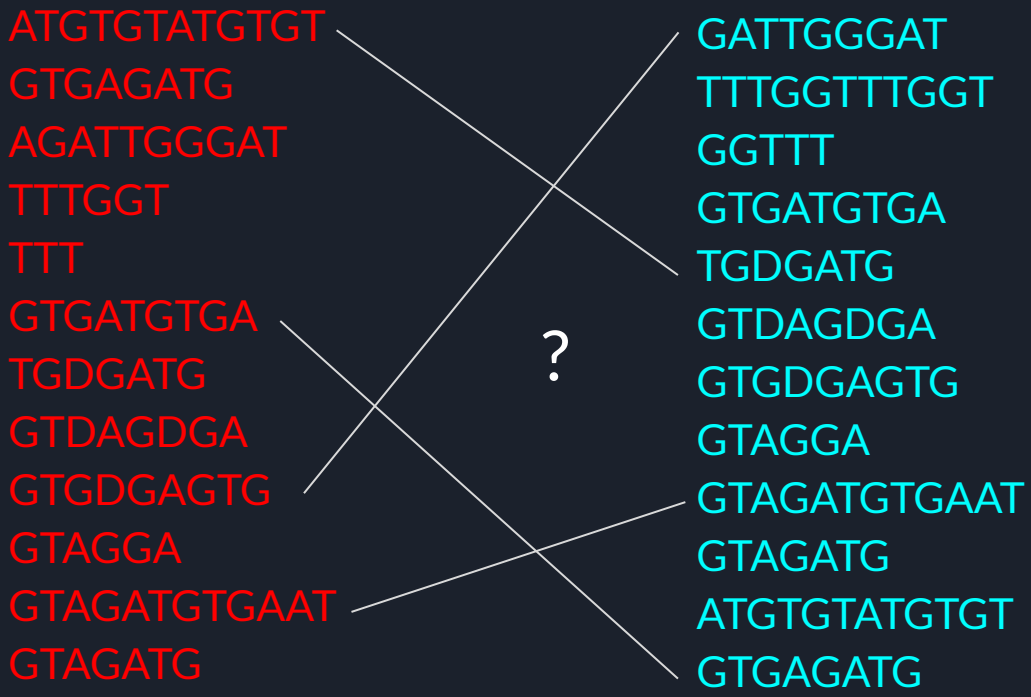
Problem B Avast

Micha Nowak, Stephan Hartmann


ATGTGTATGTGT
GTGAGATG
AGATTGGGAT
TTTGGT
TTT
GTGATGTGA
TGDGATG
GTDAGDGA
GTGDGAGTG
GTAGGA
GTAGATGTGAAT
GTAGATG
?
GATTGGGAT
TTTGGTTTGGT
GGTTT
GTGATGTGA
TGDGATG
GTDAGDGA
GTGDGAGTG
GTAGGA
GTAGATGTGAAT
GTAGATG
ATGTGTATGTGT
GTGAGATG

Der **Wiederholungswert** einer Sequenz s ist gleich der Länge der kürzesten Sequenz u, so dass s einer k-fachen Wiederholung von u gleicht, für einen positiven Integer k.

ATGATG	**Wiederholungswert: 3**	2 x ATG
GGGGG	**Wiederholungswert: 1**	5 x G
ATATA	**Wiederholungswert: 5**	1 x ATATA

Matching

Matche jede Virus-Sequenz mit einer Antivirus-Sequenz

Schaden eines Paars = Quadrierte Differenz der Wiederholungswerte

Bsp: ATGATG mit ATATA

(3 - 5) ² = **4 Schadenspunkte**

Gesucht: **minimal möglicher,** über alle Paare aufsummierter Schaden

Input

- Eine Zeile mit einem Integer **n** (1 ≤ **n** ≤ 50) Anzahl der DNA Sequenzen (je n für Virus und Antivirus)
- **n** Zeilen, jede mit einer Virus DNA Sequenz
- **n** Zeilen, jede mit einer Antivirus DNA Sequenz

Jede DNA-Sequenz ist ein **nicht leerer** String mit einer Länge von **höchstens 250** und besteht aus den Kleinbuchstaben **a-z** und Großbuchstaben **A-Z.**

Output

Ein Integer: der **minimale mögliche Schaden.**

Sample input 1

2
TTTTT
TATG
TATATA
AAAGAAAG

Wiederholungswerte:
1
4
2
4

Schaden:

$(1 - 2)^2 + (4 - 4)^2 = 1 + 0 = \mathbf{1}$

$(1 - 4)^2 + (4 - 2)^2 = 9 + 4 = 13$

Modellierung

- Input in zwei String Listen umwandeln
- Wiederholungswerte für **alle** Strings ausrechnen => Zwei Integer Listen
- Diese Listen so matchen, dass Distanz minimal ist => Sortieren!
- Distanz zurückgeben

Pseudocode Model

```
def main() -> int:
    input = sys.stdin.read()
    list_1, list_2 = transform_input(input)
    list_1 = repetition_score_list(list_1)
    list_2 = repetition_score_list(list_2)
    list_1.sort()
    list_2.sort()
    return distance(list_1, list_2)
```

Wiederholungswert bestimmen

Eingabe: String

Ausgabe: Länge des kleinsten möglichen Subpatterns, der den ursprünglichen String mit Wiederholungen ausfüllt

Wiederholungswert bestimmen

Eingabe: ATGATG

A $\xrightarrow{x6}$ AAAAAA ≠ ATGATG

AT $\xrightarrow{x3}$ ATATAT ≠ ATGATG

ATG $\xrightarrow{x2}$ ATGATG = ATGATG

Länge 3 = Rückgabewert

Subpattern Pseudocode

```
# returns the repetition score for a given sequence
def repetition score(sequence: str) -> int:
    pattern = ""

    for char in sequence:
        pattern += char
        if is valid subpattern(pattern, sequence):
            return len(pattern)

    return len(sequence)
```

Sortieren = Perfektes Matching

Beweisidee: Zeige, jede beliebige Permutation der beiden Listen kann in aufsteigend sortierte Form gebracht werden, ohne dabei den Schadenswert zu verschlechtern.

Da dies dann auch für die optimale Permutation gelten muss, ist die aufsteigend sortierte Form gleich gut wie die optimale Permutation, somit ist die aufsteigende Sortierung eine optimale Lösung.

Sortieren = Perfektes Matching

V und A = Listen der Wiederholungswerte für Viren und Antiviren.

v_i und a_i = i-ten Elemente dieser Listen.

v_i wird mit a_i gematcht $\forall\ i\ \epsilon \{1..n\}$.

Liste V aufsteigend sortiert bedeutet $v_i <= v_j$ für alle $i < j$

Liste A aufsteigend sortiert bedeutet $a_i <= a_j$ für alle $i < j$

Sortieren = Perfektes Matching

```
while not (V == sorted(V) and A == sorted(A)):
    for i in range(0, len(V) - 1):
        for j in range(i + 1, len(V)):
            sort(i, j)
```

Ergibt alle möglichen Index-paare (i, j) mit $i < j$

Sortieren = Perfektes Matching

```
while not (V == sorted(V) and A == sorted(A)):
    for i in range(0, len(V) - 1):
        for j in range(i + 1, len(V)):
            sort(i, j)
```

Vertauscht v_i mit v_j, a_i mit a_j, oder beide, falls sie falsch sortiert sind, ohne den Schaden zu erhöhen.

Sortieren = Perfektes Matching

$sort(i, j)$ mit $i < j$

1. $v_i = v_j$
   - $a_i = a_j$ schon sortiert
   - $a_i < a_j$ schon sortiert
   - $a_i > a_j$ tausche a_i mit a_j → Schaden bleibt gleich
2. $v_i < v_j$
   - $a_i = a_j$ schon sortiert
   - $a_i < a_j$ schon sortiert
   - $a_i > a_j$ tausche a_i mit a_j → Schaden wird reduziert
3. $v_i > v_j$
   - $a_i = a_j$ tausche v_i mit v_j → Schaden bleibt gleich
   - $a_i < a_j$ tausche v_i mit v_j → Schaden wird reduziert
   - $a_i > a_j$ tausche v_i mit v_j und a_i mit a_j → Schaden bleibt gleich

Sortieren = Perfektes Matching

Fazit: Jede beliebige Permutation, kann in eine aufsteigend sortierte Form gebracht werden, ohne den Schaden zu verschlechtern.

Dies gilt auch für jede optimale Permutation, woraus folgt: Eine aufsteigende Sortierung ist gleich gut wie jede optimale Lösung, ist also auch eine optimale Lösung.

Laufzeit

n: Sequenz-Länge, **m:** Sequenz-Anzahl

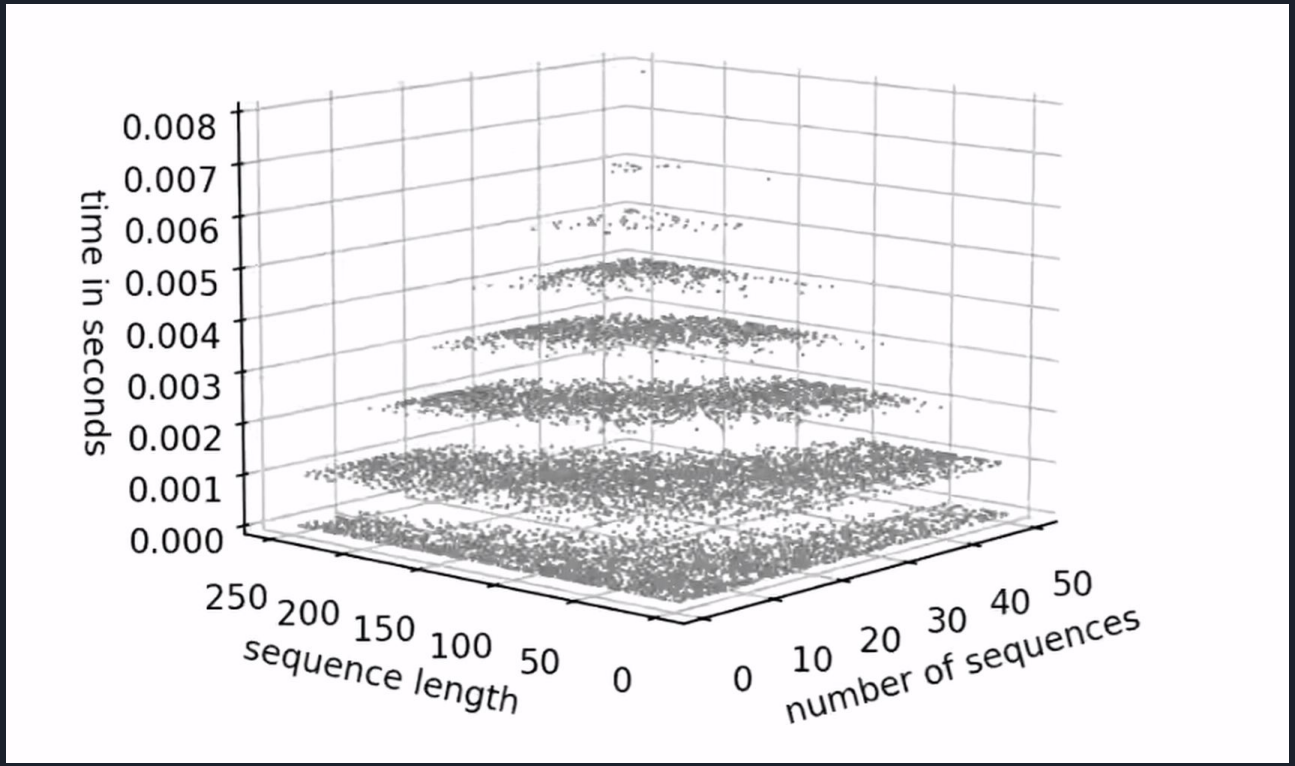
Subpattern: n^2 → **Loop: n**

Teiler
Addition
Vergleich

Listensortierung: m log m

Gesamt: $m * n^2 + m \log m$

Performance



Happy Coding!