Tugas Pemrograman 2, DDP1

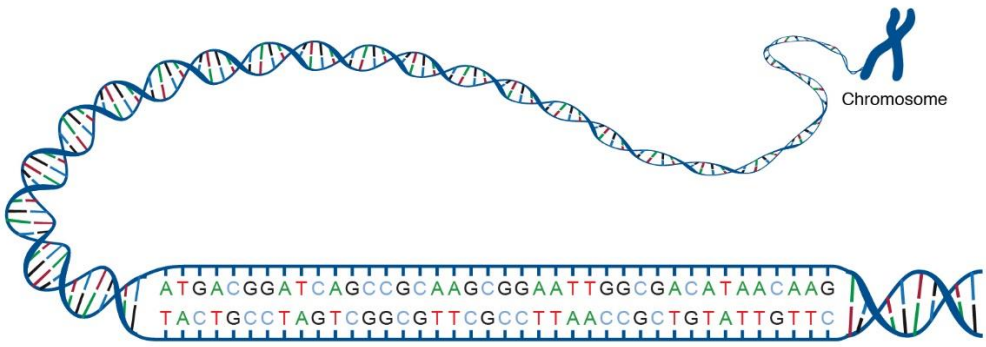
Semester Gasal 2024-2025

Alfan F. Wicaksono & Tim Asdos

DDP1 meets Bioinformatics

Catatan: dalam mengerjakan tugas pemrograman 2 ini, peserta dilarang menggunakan method pada string `count()`, `find()`, `index()`, `rfind()`, dan `rindex()`. Peserta hanya boleh menggunakan indexing, slicing, loop, dan conditional ketika memproses string pada permasalahan di Tugas Pemrograman 2 ini.

Bioinformatika adalah sebuah bidang irisan antara *computer science* dan *biology*, yang berfokus kepada penggunaan perangkat lunak atau program komputer untuk memproses data biologis yang berukuran besar. Salah satu data yang sering diolah pada bidang Bioinformatika adalah string **DNA genome** yang setiap elemennya merupakan salah satu dari 4 buah nukleotida: **adenine (A)**, **cytosine (C)**, **guanine (G)**, and **thymine (T)**. Gambar 1 menampilkan ilustrasi dari sebuah untaian **DNA genome**.

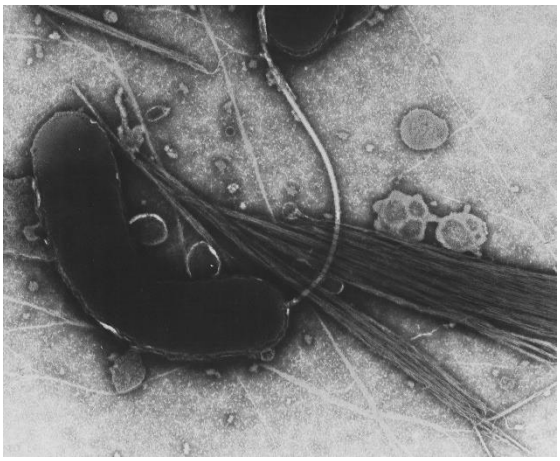


Gambar 1. Contoh sebuah untaian Genome (https://www.genome.gov/genetics-glossary/acgt)

String atau untaian molekul **DNA genome** inilah yang menjadi ciri khas atau keunikan spesies-spesies yang ada di alam semesta ini. Bersamaan dengan tugas ini dirilis, peserta mendapatkan tiga buah file sebagai berikut. Peserta bisa melihat langsung dengan membuka file-file teks berikut:

1. vibrio_cholerae.txt (~1MB)
2. ori_vibrio_cholerae.txt (~1KB)
3. e_coli.txt (~4,5MB)

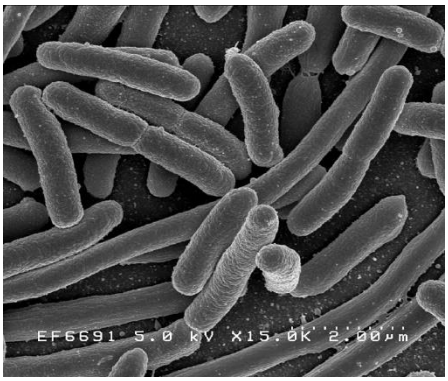
File pertama (vibrio_cholerae.txt) berisi **DNA genome** untuk **Vibrio Cholerae**, yaitu bakteri yang jika tidak terkendali dalam perut kita akan menyebabkan penyakit diare. Gambar 2 berikut adalah gambar dari bakteri tersebut.



Gambar 2. Vibrio Cholerae (https://id.wikipedia.org/wiki/Vibrio_cholerae)

File kedua (ori_vibrio_cholerae.txt) berisi subset genome dari file pertama, dan berukuran lebih kecil. Subset genome ini merupakan bagian pada genome utama Vibrio Cholerae (vibrio_cholerae.txt) yang menjadi titik mula (*origin*) proses replikasi DNA terjadi ketika *cell* membelah diri.

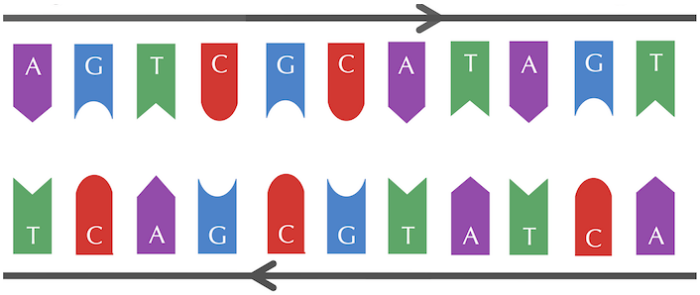
File ketiga (e_coli.txt) merupakan yang paling besar dan berisi string **DNA genome** untuk bakteri **Escherichia Coli**, yaitu sebuah bakteri yang umum berada di dalam perut manusia, dan tidak berbahaya jika banyaknya terkendali.



Gambar 3. Escherichia Coli (https://id.wikipedia.org/wiki/Escherichia_coli)

PART I. Reverse Complementary Strand (Untaian komplementer terbalik)

Seperti yang terlihat pada Gambar 1 di atas, **DNA genome** terdiri dari dua buah **strand** yang saling berikatan: **Adenin (A)** berpasangan dengan **Thymine (T)**; dan **Cytosine (C)** berpasangan dengan **Guanine (G)**. Gambar 4 berikut menampilkan strand **AGTCGCATAGT** dan komplementer-nya **ACTATGCGACT**.



Gambar 4. Complementary Strand (bawah) berjalan pada arah yang berlawanan dari Template Strand (atas). Gambar ini diperoleh dari https://www.bioinformaticsalgorithms.org/bioinformatics-chapter-1

Perhatikan bahwa komplementer dari sebuah *strand* ditulis secara terbalik (reversed) karena pergerakan yang berlawanan. Untuk contoh yang lain, *reverse complement* dari **AGT** adalah **ACT**, bukan **TCA**. Pada bagian pertama ini, tugas Anda adalah **mengimplementasikan** fungsi **`revese_pattern(pattern)`** yang menerima string *pattern* dan mengembalikan *reverse complement* dari *pattern* tersebut.

```
>>> reverse_pattern('AGT')
'ACT'

>>> reverse_pattern('AGTCGCATAGT')
'ACTATGCGACT'
```

PART II. Menghitung kemunculan K-MER

Di bioinformatika, kita biasa menggunakan istilah **k-mer** untuk menyatakan sebuah substring genome yang panjangnya **k**. Contoh, **AGT** adalah sebuah **3-mer**, dan **AGTCGCATAGT** adalah sebuah **11-mer**. Salah satu permasalahan komputasi penting yang dapat dilakukan adalah untuk menghitung berapa kali sebuah pola **k-mer** muncul pada sebuah *genome*. Ini salah satunya berguna untuk mengetahui ciri khas dari suatu *genome* atau untuk menemukan bagian spesial dari *genome* yang menjadi awalan untuk melakukan replikasi DNA.

Tugas Anda pada bagian 2 ini adalah mengimplementasikan fungsi **`count_k_mer(genome, pattern)`** yang menerima string *genome* yang panjang dan juga string *pattern*. Fungsi kemudian mengembalikan integer yang menyatakan berapa kali sebuah *pattern* muncul pada string *genome*. Yang perlu diperhatikan adalah "matching" tidak hanya dilakukan terhadap *pattern* asli, tetapi juga terhadap **"reverse complement-nya"**. Jadi, ketika kita menghitung kemunculan untuk **AGT**, kita juga harus akumulasikan kemunculan dari **ACT**.

Perhatikan contoh berikut:

```
>>> count_k_mer('ACAACTATGCATACTATCGGGAACTATCCTATAGT', 'ACTAT')
4
```

Perhatikan bahwa contoh di atas mengembalikan 4 karena ada 3 buah **ACTAT** dan 1 **ATAGT** (*reverse complement*).

Perhatikan juga contoh kedua berikut:

```
>>> count_k_mer('CGATATATCCATAG', 'ATA')
5
```

Contoh kedua mengembalikan 5, karena ada 3 kali kemunculan **ATA** (kita mempertimbangkan kemunculan *pattern* yang tumpang tindih), dan ada 2 kali kemunculan **TAT** (*reverse complement*).

PART III. Most Frequest K-MERs (K-MER paling sering muncul)

Pada bagian ke-3 ini, peserta diminta untuk mengimplementasikan fungsi **`frequent_k_mer(genome, k)`** yang menerima string *genome* dan bilangan bulat **k**. Fungsi ini kemudian mengembalikan daftar **k-mer** dengan frekuensi kemunculan tertinggi yang disimpan dalam bentuk Python's **list**. Untuk mengimplementasikan fungsi ini, peserta perlu mengetahui terlebih dahulu tentang struktur data **list** pada Python. List adalah struktur data linier yang mempunyai operasi yang mirip dengan string. Contoh berikut menampilkan bagaimana sebuah list dapat dibangun dari kondisi kosong, dan diisi satu persatu dengan method **`append()`**:

```
>>> mers = []      # list kosong
>>> mers.append('ACT')
>>> mers
['ACT']
>>> mers.append('GAT')
>>> mers.append('ATA')
>>> mers
['ACT', 'GAT', 'ATA']
```

Sama seperti string, operasi *indexing* dan *slicing* (zero-based index) serta membership **in** juga dapat diterapkan pada List:

```
>>> mers[1]
'GAT'
>>> mers[1:3]
['GAT', 'ATA']
>>> 'AAA' in mers
False
>>> 'GAT' in mers
True
```

Sekarang, saatnya Anda mengimplementasikan fungsi **`frequent_k_mer(genome, k)`** memanfaatkan struktur data List yang sudah dijelaskan sebelumnya. Perhatikan contoh *genome* berikut (diambil dari ori_vibrio_cholerae.txt):

```
ATCAATGATCAACGTAAGCTTCTAAGCATGATCAAGGTGCTCACACAGTTTATCCACAACCTGAGTGGATGA
CATCAAGATAGGTCGTTGTATCTCCTTCCTCTCGTACTCTCATGACCACGGAAAGATGATCAAGAGAGGATG
ATTTCTTGGCCATATCGCAATGAATACTTGTGACTTGTGCTTCCAATTGACATCTTCAGCGCCATATTGCGC
TGGCCAAGGTGACGGAGCGGGATTACGAAAGCATGATCATGGCTGTTGTTCTGTTTATCTTGTTTTGACTGA
GACTTGTTAGGATAGACGGTTTTTCATCACTGACTAGCCAAAGCCTTACTCTGCCTGACATCGACCGTAAAT
TGATAATGAATTTACATGCTTCCGCGACGATTTACCTCTTGATCATCGATCCGATTGAAGATCTTCAATTGT
TAATTCTCTTGCCTCGACTCATAGCCATGATGAGCTCTTGATCATGTTTCCTTAACCCTCTATTTTTTACGG
AAGAATGATCAAGCTGCTGCTCTTGATCATCGTTTC
```

Seandainya string *genome* di atas disimpan dalam variable **gen**, fungsi **frequent_k_mer(gen, 9)** akan mengembalikan list **['ATGATCAAG', 'CTTGATCAT']**. Anda bisa menganggap **ATGATCAAG** dan **CTTGATCAT** adalah sama karena mereka saling komplemen. Kemunculan mereka adalah **6** dan ini adalah nilai kemunculan **9-mer** yang paling tinggi.

```
>>> frequent_k_mer(gen, 9)
['ATGATCAAG', 'CTTGATCAT']

>>> frequent_k_mer(gen, 3)
['ATC', 'TCA', 'TGA', 'GAT']
```

Perhatikan bahwa Anda boleh saja menggunakan fungsi **count_k_mer(genome, pattern)** di dalam fungsi **frequent_k_mer(genome, k)**.

PART IV. Program utama

Di bagian terakhir ini, Anda diminta untuk membuat program utama yang membungkus penggunaan tiga fungsi di atas dalam sebuah menu program. Pertama, program akan meminta input nama file yang berisi string *genome*. Kedua, program kemudian menampilkan menu yang terdiri dari tiga buah opsi (terkait tiga fungsi di atas):

```
Genome file name: ori_vibrio_cholerae.txt [enter]
Choose an option:
[1] Compute a reverse complement of a k-mer pattern
[2] Count a k-mer pattern
[3] Find most frequent k-mer patterns

Select an operation [1/2/3]: _
```

```
Genome file name: ori_vibrio_cholerae.txt [enter]
Choose an option:
[1] Compute a reverse complement of a k-mer pattern
[2] Count a k-mer pattern
[3] Find most frequent k-mer patterns

Select an operation [1/2/3]: 1 [enter]
Input your pattern: GAT [enter]
ATC
```

```
Genome file name: ori_vibrio_cholerae.txt [enter]
Choose an option:
[1] Compute a reverse complement of a k-mer pattern
[2] Count a k-mer pattern
[3] Find most frequent k-mer patterns

Select an operation [1/2/3]: 2 [enter]
Input your pattern: GAT [enter]
42
```

```
Genome file name: ori_vibrio_cholerae.txt [enter]
Choose an option:
[1] Compute a reverse complement of a k-mer pattern
[2] Count a k-mer pattern
[3] Find most frequent k-mer patterns

Select an operation [1/2/3]: 3 [enter]
Input your value of k: 3 [enter]
ATC
TCA
TGA
GAT
```

Silakan bereksplorasi dan coba jalankan kode kalian untuk input file yang berukuran lebih besar seperti vibrio_cholerae.txt dan e_coli.txt.

PART V. Bagaimana jika ada Input yang salah? Apakah perlu divalidasi?

Ya, Anda perlu validasi input yang salah. Proses validasi diserahkan kepada peserta masing-masing 😊 Gunakan kreativitas Anda di sini.

****

Penilaian:

- 50% correctness (Part I – V mempunyai bobot yang sama)
- 40% terkait penjelasan saat presentasi/demo dengan asisten dosen
- 10% dokumentasi program

Deadline pengumpulan adalah **Selasa 8 Oktober 2024 (11:55 PM Scele time)**. Fungsi-fungsi di atas dan program utama dibungkus dalam sebuah file *.py dengan format nama:

"<kode_asdos>_<nama_mhs>_<NPM>_<kelas>_TP02.py"

Begitu Anda submit, segera hubungi Asdos Anda masing-masing untuk melakukan demo. Asdos Anda akan memberikan nilai demo; perhatikan bahwa demo bersifat wajib dan Anda akan mendapat nilai 0 jika tidak mendemokan program Anda.

Jika Ada pertanyaan, silakan hubungi Dosen Anda atau Asdos Anda.