Progetto di Elementi di Bioinformatica A.A. 24/25

Christena Attia, Mat. 894887

1. Introduzione

Il progetto affronta l'analisi dei file FASTQ contenenti sequenze (*reads*) di uguale lunghezza (152 basi), con l'obiettivo di studiare la distribuzione dei k-mer (sottostringhe di lunghezza *k*) in funzione della posizione in cui si trovano all'interno dei reads. L'elaborazione prevede, per ogni k-mer che supera una soglia di frequenza F, la produzione di statistiche posizionali, la selezione del k-mer più ricorrente in una singola posizione, la creazione di una visualizzazione grafica e l'estrazione in formato FASTA dei reads rilevanti.

2. Parametri in input

Il notebook accetta come parametri:

- **file_path**: percorso del file FASTQ da analizzare (in questo caso è 'SRR18961685-5000.fastq').
- **k**: lunghezza del k-mer da analizzare.
- **frequency_threshold**: soglia tra 0 e 1 che definisce la frequenza minima oltre la quale un k-mer è considerato significativo.

Questi parametri possono essere modificati direttamente all'interno del notebook.

```
file_path = 'SRR18961685-5000.fastq'
k = 7
frequency_threshold = 0.001
```

3. Metodi e algoritmi implementati

3.1 parseFileFASTQ()

Il primo passo dell'analisi consiste nel leggere e validare il file FASTQ contenente le sequenze. Questa funzione verifica innanzitutto che il file specificato esista nel sistema, per evitare errori durante l'elaborazione. Successivamente procede alla lettura di tutte le sequenze contenute nel file, estrae le informazioni sulla qualità del sequenziamento associate a ogni base, e conta il numero totale di sequenze processate.

```
def parseFileFASTQ(file_path):

    if not os.path.exists(file_path):
        raise FileNotFoundError(f"Il file '{file_path}' non è stato trovato. Verificare il percorso.")

    reads = list(SeqIO.parse(file_path, "fastq"))
    sequences = [str(r.seq) for r in reads]
    qualityScores = [r.letter_annotations["phred_quality"] for r in reads]
    readLen = len(sequences[0])

    print(f"Letti {len(sequences)} reads, lunghi {readLen} basi ciascuno.")
    return reads, sequences, qualityScores, readLen

reads, sequences, qualityScores, readLen = parseFileFASTQ(file_path)
```

```
Letti 5000 reads, lunghi 152 basi ciascuno.
```

```
print("\nEsempio di record:")
print(reads[0])

print("\nEsempio di sequenza:")
print(sequences[0])

print("\nPunteggi di qualità (Phred) della prima read:")
print(qualityScores[0])
```

```
Esempio di record:
ID: SRR18961685.1
Name: SRR18961685.1
Description: SRR18961685.1 1 length=152
Number of features: 0
Per letter annotation for: phred_quality
Seq('TTAGGGTTAGGGTTAGGGTTAGGGTTAGGGTTAGGGTTAGGTTTAGGGTTAGGG...CAA')

Esempio di sequenza:
TTAGGGTTAGGGTTAGGGTTAGGGTTAGGGTTAGGGTTAGGTTTAGGGTTACGGATAGGGTAACGTGTAGCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCATAA
CCCTAACCCAA

Punteggi di qualità (Phred) della prima read:
[31, 32, 30, 32, 34, 35, 33, 36, 34, 36, 36, 36, 34, 36, 34, 36, 36, 36, 34, 36, 34, 36, 36, 29, 31, 31, 34, 36, 36, 36, 34, 36, 34, 30, 36,
36, 34, 30, 34, 36, 36, 14, 14, 36, 34, 15, 36, 36, 14, 14, 25, 15, 36, 13, 14, 14, 34, 12, 31, 28, 10, 35, 25, 15, 13, 13, 34, 8, 35, 12, 3
0, 14, 14, 14, 25, 16, 31, 31, 31, 29, 32, 33, 34, 34, 35, 33, 34, 35, 35, 35, 36, 34, 35, 35, 35, 35, 35, 34, 35, 35, 35, 35, 36, 34, 35, 3
5, 35, 35, 36, 34, 35, 35, 35, 35, 36, 34, 35, 27, 35, 11, 35, 34, 35, 35, 10, 35, 36, 34, 11, 26, 35, 11, 21, 34, 35, 27, 11, 9, 13, 34, 11,
12, 25, 11, 33, 34, 11, 12, 31, 12, 11, 31]
```

3.2 countKmers()

Questa funzione esamina ogni sequenza presente nel dataset ed estrae tutti i possibili k-mer di lunghezza *k* utilizzando la tecnica della "finestra scorrevole" (*sliding window*). Si tratta di un approccio che consiste nel far “scorrere” una finestra virtuale di dimensione *k* lungo la sequenza, avanzando di una base per volta. Ad ogni passo, la finestra cattura un nuovo frammento della sequenza, che viene registrato come un potenziale k-mer. Questo processo continua fino a quando la finestra non raggiunge la fine della sequenza. Per ogni k-mer individuato, tiene traccia di quante volte appare in ciascuna posizione, salvando questi conteggi in un dizionario, creando così una mappa completa della distribuzione posizionale di tutti i k-mer.

```
def countKmers(sequences, k, readLen):
    kmerCounts = {}

    for index, seq in enumerate(sequences):
        for pos in range(readLen - k + 1):
            kmer = seq[pos:pos + k]

            if kmer not in kmerCounts:
                kmerCounts[kmer] = [0] * readLen

            kmerCounts[kmer][pos] += 1

    return kmerCounts
```

```
kmerDictionary = countKmers(sequences, k, readLen)

print("\n Esempi di distribuzione posizionale di due k-mer distinti")
firstKmer = list(kmerDictionary.keys())[0]
print(f"\n Primo k-mer distinto individuato: '{firstKmer}'")
print(f"    Distribuzione posizionale di '{firstKmer}': {kmerDictionary[firstKmer]}")

secondKmer = list(kmerDictionary.keys())[1]
print(f"\n Secondo k-mer distinto individuato: '{secondKmer}'")
print(f"    Distribuzione posizionale di '{secondKmer}': {kmerDictionary[secondKmer]}")
```

```
 Esempi di distribuzione posizionale di due k-mer distinti

 Primo k-mer distinto individuato: 'TTAGGGT'
    Distribuzione posizionale di 'TTAGGGT': [1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

 Secondo k-mer distinto individuato: 'TAGGGTT'
    Distribuzione posizionale di 'TAGGGTT': [2, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 7, 1, 2, 0, 1, 1, 0, 1,
0, 0, 0, 0, 1, 1, 0, 1, 4, 2, 1, 1, 0, 1, 4, 0, 1, 2, 3, 1, 2, 1, 2, 0, 0, 1, 1, 3, 1, 1, 0, 0, 1, 0, 3, 0, 0, 5, 2, 0, 1, 0, 0, 1, 0, 0, 1,
0, 1, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 3, 0, 0, 0, 0, 1, 0, 0, 2, 2, 0, 4, 0, 0, 1, 4, 0, 1, 0, 0, 0,
2, 0, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

3.3 filterKmers()

Questa funzione calcola la frequenza relativa di ogni k-mer dividendo il numero di occorrenze del singolo k-mer per il totale delle posizioni analizzate. I k-mer che risultano troppo rari, ovvero con frequenza inferiore alla soglia prestabilita, vengono eliminati dall'analisi per concentrarsi sui pattern più significativi. Il risultato è una tabella riassuntiva che contiene solo i k-mer statisticamente rilevanti.

```python
def filterKmers(kmerDictionary, threshold, totalPos):
    filtered = {}
    rows = []

    for kmer, counts in kmerDictionary.items():
        total_occurrences = sum(counts)
        frequency = total_occurrences / totalPos
        included = frequency >= threshold

        if included:
            filtered[kmer] = counts

        rows.append({
            "k-mer": kmer,
            "Totale occorrenze": total_occurrences,
            "Frequenza": round(frequency, 6),
            "Incluso": "Sì" if included else "No"
        })

    table_filterd = pd.DataFrame(rows)
    table_filterd.sort_values(by="Frequenza", ascending=False, inplace=True)
    table_filterd.reset_index(drop=True, inplace=True)

    display(table_filterd)

    return filtered, rows
```

```python
totalPos = (readLen - k + 1) * len(sequences)

filteredDict, table_rows = filterKmers(kmerDictionary, frequency_threshold, totalPos)
```

	k-mer	Totale occorrenze	Frequenza	Incluso
0	GGCAGAG	1615	0.002212	Sì
1	GCAGAGG	1615	0.002212	Sì
2	GCGGGCA	1404	0.001923	Sì
3	CTCACTT	1175	0.001610	Sì
4	GGGCTGG	1172	0.001605	Sì
...	...	...	...	...
11007	GTGCAGG	1	0.000001	No
11008	CATGAAC	1	0.000001	No
11009	ATGAACG	1	0.000001	No
11010	CTCGGCT	1	0.000001	No
11011	GACTTGT	1	0.000001	No

11012 rows × 4 columns

```python
print(f"Rimasti {len(filteredDict)} k-mer dopo il filtro su {len(kmerDictionary)} totali.\n")
```

```
Rimasti 130 k-mer dopo il filtro su 11012 totali.
```

3.4 findMaxKmer()

Tra tutti i k-mer che hanno superato il filtro di frequenza, il programma identifica quello che presenta la concentrazione massima in una singola posizione. Questo k-mer "dominante" rappresenta il pattern più caratteristico del dataset analizzato e viene utilizzato per le analisi successive. La funzione

registra l'identità del k-mer dominante, la posizione specifica di massima concentrazione, e il valore del conteggio massimo.

```
def find_maxKmer(filteredDict):
    bestKmer, bestKmer_pos, max_count = None, None, 0

    for kmer, counts in filteredDict.items():
        for i, count in enumerate(counts):
            if count > max_count:
                max_count = count
                bestKmer = kmer
                bestKmer_pos = i

    if bestKmer is None:
        print("Nessun k-mer dominante trovato.")
    return bestKmer, bestKmer_pos, max_count
```

```
bestKmer, bestKmer_pos, max_count = find_maxKmer(filteredDict)

table_best_kmer = pd.DataFrame({
    "k-mer dominante": [bestKmer],
    "Posizione massima": [bestKmer_pos],
    "Occorrenze massime": [max_count]
})

display(table_best_kmer)
```

	k-mer dominante	Posizione massima	Occorrenze massime
0	GCAGAGG	135	47

3.5 top5_bestKmers()

Questa funzione crea una classifica dei 5 k-mer che hanno il picco più alto di occorrenze in una posizione. Per ognuno mostra il totale delle occorrenze, la posizione del picco e il valore massimo in quella posizione. Questi dati vengono organizzati in una tabella ordinata che evidenzia i pattern più rilevanti del dataset.

```
def top5_bestKmers(filteredDict):
    table = []

    for kmer, counts in filteredDict.items():
        max_count = max(counts)
        bestPos = counts.index(max_count)
        total = sum(counts)
        table.append((kmer, total, bestPos, max_count))

    df = pd.DataFrame(table, columns=["k-mer", "Totale occorrenze", "Posizione max", "Occorrenze max"])
    top5_table = df.sort_values(by="Occorrenze max", ascending=False).head(5).reset_index(drop=True)
    display(top5_table)
    return top5_table

top5_table = top5_bestKmers(filteredDict)
```

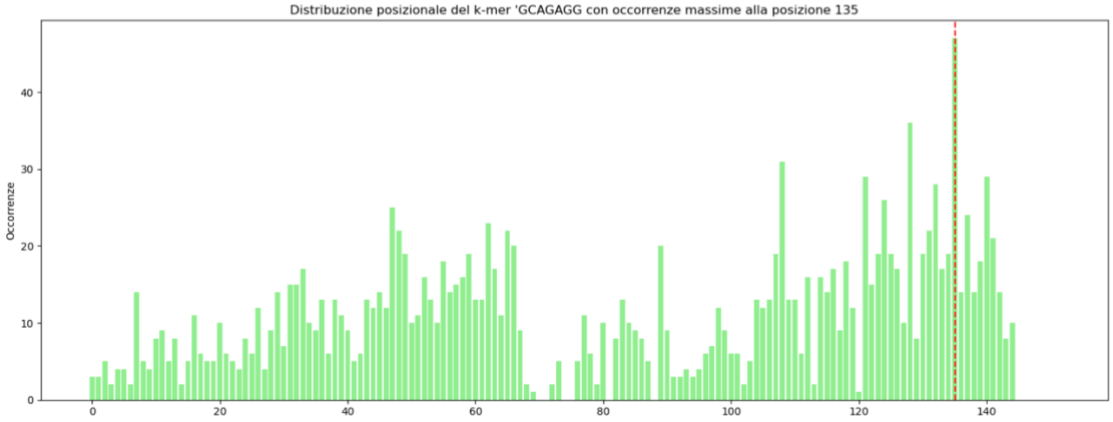
	k-mer	Totale occorrenze	Posizione max	Occorrenze max
0	GCAGAGG	1615	135	47
1	GGCAGAG	1615	134	42
2	GCGGGCA	1404	133	40
3	CTCACTT	1175	123	40
4	GGAGCCT	1011	141	39

3.6 distributionDiagram()

I risultati numerici vengono trasformati in una rappresentazione grafica che mostra la distribuzione del k-mer dominante lungo tutte le posizioni analizzate. Il grafico utilizza barre colorate per rappresentare il numero di occorrenze in ogni posizione e una linea verticale rossa tratteggiata evidenzia il punto di massima concentrazione.

```
def distributionDiagram(kmer, counts, bestKmer_pos):
    plt.figure(figsize=(15, 6))
    plt.bar(range(len(counts)), counts, color='lightgreen')
    plt.axvline(x=bestKmer_pos, color='red', linestyle='--')
    plt.xlabel("Posizione")
    plt.ylabel("Occorrenze")
    plt.title(f"Distribuzione posizionale del k-mer '{kmer} con occorrenze massime alla posizione {bestKmer_pos}")
    plt.tight_layout()
    plt.show()

if not filteredDict:
    print("Nessun k-mer supera la soglia di frequenza da analizzare. Cambiare i parametri di configurazione.")
else:
    distributionDiagram(bestKmer, filteredDict[bestKmer], bestKmer_pos)
```



3.7 fastaOutput()

Come output finale, il programma crea un nuovo file FASTA contenente esclusivamente le sequenze che presentano il k-mer dominante nella posizione di massima concentrazione. Ad ogni sequenza nel file di output viene inoltre aggiunta anche la qualità media del sequenziamento.

```
def fastaOutput(reads, qualityScores, bestKmer, bestKmer_pos, k, output_filename):
    output = []

    for r, qual in zip(reads, qualityScores):
        seq = str(r.seq)

        if seq[bestKmer_pos:bestKmer_pos+k] == bestKmer:
            average = round(stats.mean(qual), 2)
            newRecord = SeqRecord(Seq(seq), id=f"{r.id} {average}", description="")
            output.append(newRecord)

    write(output, output_filename, "fasta")
    print(f"Salvati {len(output)} reads in '{output_filename}'.")

if bestKmer is not None and bestKmer_pos is not None:
    fastaOutput(reads, qualityScores, bestKmer, bestKmer_pos, k, "output.fasta")
else:
    print("Nessun k-mer dominante valido trovato. File FASTA non generato.")
```

```
Salvati 47 reads in 'output.fasta'.
```

4. Scelte effettuate e criteri utilizzati

Per lo sviluppo del progetto si è adottato un approccio modulare e leggibile, suddividendo il flusso di elaborazione in più funzioni, ognuna con uno scopo preciso e ben definito. Questo ha facilitato sia la comprensione che il debugging del codice. L'analisi si è focalizzata sull'estrazione e sullo studio dei k-mer, ovvero sottostringhe di lunghezza fissa, per individuare pattern ricorrenti e significativi all'interno delle sequenze.

- Nella fase iniziale, è stata implementata una funzione per il parsing del file FASTQ, da cui sono state estratte le sequenze e i relativi punteggi di qualità.
- Per l'estrazione dei k-mer si è utilizzata la tecnica della "finestra scorrevole", che garantisce l'identificazione di tutte le sottostringhe di lunghezza desiderata. È stata progettata una struttura dati che consente di registrare, per ogni k-mer, il numero di occorrenze in ogni posizione possibile: ciò ha permesso una successiva analisi dettagliata della distribuzione posizionale.
- È stato applicato un filtro per escludere i k-mer troppo rari.
- Per identificare il k-mer dominante, si è scelto di confrontare le occorrenze posizionali di ciascun k-mer e selezionare quello che appare con frequenza massima in una determinata posizione.
- Sono stati poi analizzati i cinque k-mer con il massimo numero di occorrenze per dare una visione più completa della distribuzione nel dataset.
- È stato incluso un diagramma a barre che visualizza la distribuzione posizionale del k-mer dominante, con evidenziazione grafica del picco di occorrenza.
- Infine, si è deciso di esportare in un file FASTA tutte le sequenze che contengono il k-mer dominante nella posizione specifica in cui è più frequente. In fase di esportazione, si è incluso nel nome di ogni sequenza il valore medio della qualità (calcolata come media aritmetica dei punteggi Phred associati).

5. Librerie utilizzate

```
import matplotlib.pyplot as plt
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.SeqIO import write
import statistics as stats
import pandas as pd
import os as os
```

1) Bio (Biopython)

Moduli usati: SeqIO, Seq, SeqRecord, write

- Per il parsing del file FASTQ (SeqIO.parse) e l'estrazione di sequenze e qualità.
- Per la creazione e formattazione delle sequenze da salvare nel file FASTA di output (Seq, SeqRecord).
- Per la scrittura del file FASTA di output (write).

2) matplotlib.pyplot

- Per la generazione del diagramma di distribuzione posizionale del k-mer dominante.

3) statistics

- Per il calcolo della qualità media di ciascuna sequenza (media dei punteggi Phred) da inserire nel nome delle sequenze nel file FASTA.

4) pandas

Per la costruzione e la visualizzazione delle tabelle:

- Tabella di tutti i k-mer con le rispettive frequenze.
- Tabella dei top 5 k-mer con più occorrenze in una singola posizione.
- Tabella riassuntiva del k-mer dominante.

5) os

- Per la verifica dell'esistenza del file FASTQ prima del parsing.

6. Output finale

L'output finale del progetto consiste in un file FASTA generato dal programma. Questo file contiene tutte le sequenze del dataset che contengono il k-mer dominante esattamente nella posizione in cui esso risulta più frequente.

Ogni record del file FASTA è composto da:

- l'ID originale del read
- la qualità media della sequenza, calcolata come media aritmetica dei punteggi di qualità Phred associati alla read.
- Una riga di sequenza (composta da A, T, C, G) associata a quel read.

```
(base) christenaattia@MacBookAir progettoBioinfo_Attia % cat output.fasta
>SRR18961685.883 34.77
GGAGACGGTGTTTGTCATGGGCCTGGTCTGCAGGGATCCTGCTACAAAGGTGAAACCCAG
GAGAGTGTGGAGTCCACAGATGAGTCTGCTCCGTCTCGCTTCTAAGACGGTGGGCAGGAC
AAATGGCAGCCTCCTGCAGAGGCCCAGTGAGA
>SRR18961685.884 34.8
GGAGACGGTGTTTGTCATGGGCCTGGTCTGCAGGGATCCTGCTACAAAGGTGAAACCCAG
GAGAGTGTGGAGTCCACAGATGAGTCTGCTCCGTCTCGCTTCTAAGACGGTGGGCAGGAC
AAATGGCAGCCTCCTGCAGAGGCCCAGTGAGA
>SRR18961685.885 34.65
GTGTGGAGTCCAGAGTGTTGCCAGGACCCAGGCACAGGCATTAGTGCCCGTTGGAGAAAA
CAGGGGAATCCCGAAGCAGATGAGTCTGCTCCGTCTCGCTTCTAAGACGGTGGGCAGGAC
AAATGGCAGCCTCCTGCAGAGGCCCAGTGAGA
```