Master Bioinformatique

Université de Montpellier

HAU724I : PROJET SYSTEME

SCORE : SAM file Characterization and Observational Report for Evaluation

Auteurs :
Mickael Coquerelle
Homero Sanchez

Matière :
Système

Professeurs :
Anna-Sophie Fiston-Lavier
Pierre Pompidor

```python
def analyse_CIGAR(d_sam):
    comptes_CIGAR = {
        'M': ["Alignés", 0], 'I': ["Insertions", 0], 'D': ["Délétions", 0],
        'N': ["Sauts de bases", 0], 'S': ["Soft Clipping", 0], 'H': ["Hard Clipping", 0],
        'P': ["Complétion", 0], '=': ["Match exact", 0], 'X': ["Mismatch", 0]}

    REGEX_CIGAR = re.compile(r'(\d+)([MIDNSHP=X])')
    TOTAL_OPE_CIG = 0

    # Calcul des sommes pour chaque motif CIGAR
    for read in d_sam.values():
        matches = REGEX_CIGAR.findall(read["CIGAR"])
        for SUM_OPE_CIG_str, OPE_CIG in matches:
            SUM_OPE_CIG = int(SUM_OPE_CIG_str)
            if OPE_CIG in comptes_CIGAR:
                comptes_CIGAR[OPE_CIG][1] += SUM_OPE_CIG
                TOTAL_OPE_CIG += SUM_OPE_CIG

    # Calcul des pourcentages et mise à jour du dictionnaire
    for OPE_CIG, (commentaire, count) in comptes_CIGAR.items():
        pourcentage = (count / TOTAL_OPE_CIG * 100) if TOTAL_OPE_CIG > 0 else 0
        comptes_CIGAR[OPE_CIG].append(pourcentage)  # Ajouter le pourcentage au dictionnaire

    return comptes_CIGAR


# Analyser les CIGAR
comptes_CIGAR = analyse_CIGAR(d_sam)

# Construction du Dataframe pour l'affichage dans le terminal.
Data_cigar = [(f"{cle_CIG}|", f"{Val_CIG[0]} |", f"{Val_CIG[1]} |", f"{Val_CIG[2]:.3f} % |") for cle_CIG, Val_CIG in
              comptes_CIGAR.items()]
t_Data_cigar = pd.DataFrame(Data_cigar,
                            columns=["Motif       ", "Nom           ", "Occurences          ", "Valeur relative"])
```

M.Coquerelle 2024

Table des matières

1 Introduction

Dans un pipeline bioinformatique, plusieurs étapes et fichiers sont générés, chacun ayant ses spécificités. Après un séquençage haut débit (SHD), les lectures *(reads)* sont alignées sur une séquence de référence et transformées en fichiers standardisés pour des analyses selon la question biologique (diagnostic, écologie, etc.). Le fichier SAM (Sequence Alignment/Map) est un élément central, stockant des métadonnées liées à l'alignement des séquences, divisées en deux catégories : l'aspect qualitatif (identification, orientation) et l'aspect quantitatif (longueur des lectures, scores de qualité).

Le fichier SAM ne sert pas uniquement à consigner ces informations, mais constitue aussi une base pour des analyses statistiques comme la distribution des lectures et la couverture génomique. Il nécessite des outils algorithmiques spécialisés pour extraire, synthétiser et visualiser les données, facilitant leur interprétation par le biologiste. Ce projet présente d'abord le fichier SAM, ses caractéristiques et son intérêt biologique, puis le programme *SCORE* (SAM file Characterization and Observational Report for Evaluation), développé en Python3, qui extrait et exploite les informations courantes d'un fichier SAM. Enfin, des améliorations futures sont proposées pour rendre cet outil plus performant et exhaustif.

2 Le Format SAM : Structure et informations techniques

Le fichier SAM se compose de deux parties : l'en-tête *(Header)* contenant les métadonnées et les paramètres de traitement, et le corps qui regroupe les données alignées. La figure ci-dessous illustre cette structure et leur séparation :

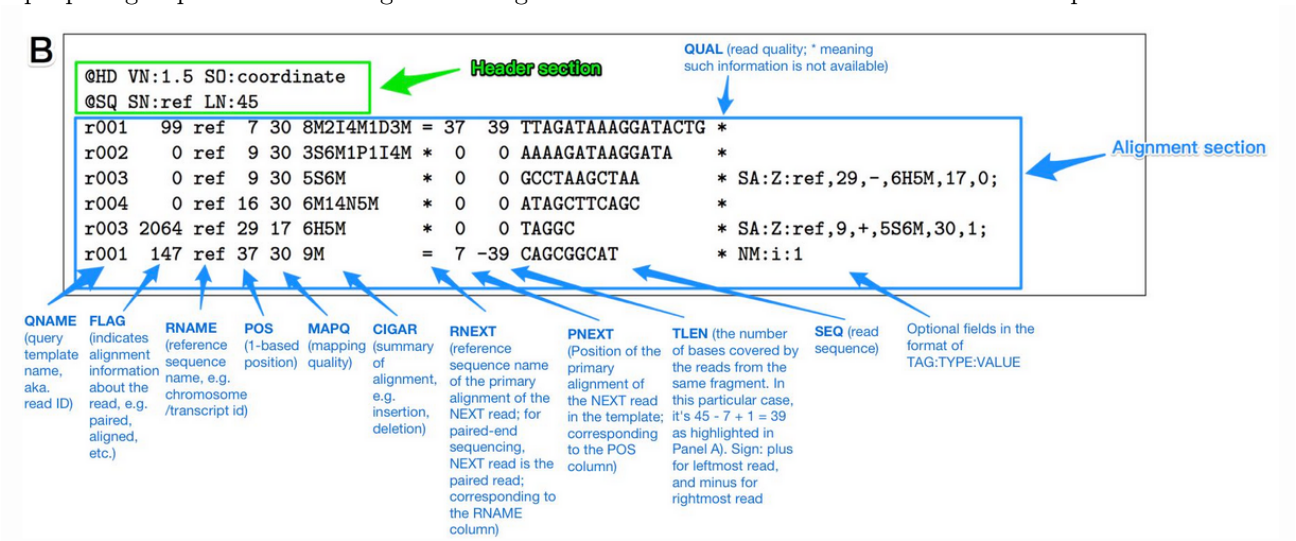

Figure 1: Format de fichier SAM, Illustrant l'en-tête et le corps. Zhuyi Xue (2017)

L'en-tête d'un fichier SAM (Sequence Alignment Map) contient des informations sur les métadonnées, incluant les détails de la référence, les programmes utilisés et d'autres paramètres d'alignement. L'exemple ci-dessous montre le *header* de notre fichier *mapping.sam*:

```
 GNU nano 6.2                                                    mapping.sam
@SQ     SN:Reference    LN:1000000
@PG     ID:bwa  PN:bwa  VN:0.7.9a-r786  CL:/home/e20140028067/Bureau/TPsnpSV/bwa-0.7.9a/bwa mem ../TPsnpSV/reference.fasta ../
```

Figure 2: En-tête du fichier SAM : *mapping.sam*

`@SQ` : Cette ligne décrit la REF utilisée dans l'alignement. Elle contient deux informations clés :

- **SN:Reference** : Le nom de la référence utilisée : "Reference".
- **LN:1000000** : La longueur de la séquence de référence : 1 000 000 paires de bases.

`@PG` : La seconde ligne indique les programmes utilisés pour effectuer l'alignement, ici :

- **ID:bwa** : L'identifiant du programme utilisé pour l'alignement "bwa".
- **PN:bwa** : Le nom du programme "bwa".
- **VN:0.7.9a-r786** : La version du programme ici 0.7.9a-r786.
- **CL:** : La commande utilisée avec les fichiers de référence et d'entrée :`/home/e20140028067/...`.

Ces lignes peuvent être complétées par d'autres métadonnées spécifiques à l'alignement, mais celles-ci sont parmi les plus courantes, définissant la référence et les outils/paramètres utilisés pour générer l'alignement[1].

Le corps du fichier SAM détaille l'alignement des séquences avec onze colonnes standardisées, offrant une description précise et exhaustive. Ce format consensuel, largement adopté par la communauté scientifique, garantit l'interopérabilité entre outils et logiciels. Sa structure claire et tabulée facilite l'analyse et le traitement algorithmique des données.

• La première colonne QNAME est le nom de la séquence. Elle va donc permettre d'identifier la lecture. Le nom dépendra de l'expérience de séquençage et du contexte biologique. Ici, *"Clone1"* est un identifiant de séquence unique suivi, à priori, d'un numéro d'incrémentation. Il est tout à fait plausible, par exemple, de faire référence à la plateforme de séquençage (SampleX_Pac_BIORead001 ...).

• La seconde colonne FLAG est particulièrement intéressante. Il s'agit d'un entier composé de 16 bits (soit 2 octets) qui encode diverses informations sur l'alignement du *read*. Chaque bit, s'il est activé, représente une propriété particulière, comme le montre le tableau ci-contre. Notons qu'il y a 16 bits, comme mentionné dans la documentation officielle du SAM, et ce malgré la constatation que seuls 12 bits sont utilisés pour encoder des informations. Les autres bits (13, 14 et 15) sont probablement réservés pour d'éventuelles extensions, mais ils sont actuellement inexploités dans les outils et logiciels courants. [2]

#	Decimal	Description of read
1	1	Read paired
2	2	Read mapped in proper pair
3	4	Read unmapped
4	8	Mate unmapped
5	16	Read reverse strand
6	32	Mate reverse strand
7	64	First in pair
8	128	Second in pair
9	256	Not primary alignment
10	512	Read fails platform/vendor quality checks
11	1024	Read is PCR or optical duplicate
12	2048	Supplementary alignment

Figure 3: Encodage du FLAG[2]

• Les colonnes RNAME et POS offrent une information complémentaire pour localiser une lecture alignée sur la référence. RNAME correspond au nom de la référence, comme un chromosome (*chr1*), un contig, ou une autre région spécifique du génome. Elle permet une identification générale de la zone d'alignement. POS indique précisément la position de départ où le *read* commence sur l'information donnée par RNAME.

Dans *mapping.sam*, une seule localisation est indiquée, mais ces données permettent de positionner un *read* sur des éléments génomiques spécifiques, comme un gène ou une région intronique. Elles sont cruciales pour identifier des variations génétiques, telles que les SNP, et ordonner dans l'espace les lectures par rapport à la référence.

• La colonne MAPQ représente le score de qualité de l'alignement, exprimé sur une échelle logarithmique, indiquant le niveau de confiance dans l'alignement. Par exemple, un score élevé de 60 indique une grande fiabilité [2]. Ce score est relié à la probabilité d'erreur Pe, garantissant ainsi un alignement fiable : $MAPQ = -10\log_{10}(P_e)$.

• Les colonnes RNEXT ("Read suivant") et PNEXT ("Position suivante") indiquent la séquence de référence et la position de départ du read apparié, essentielles pour associer les lectures appariées en séquençage par paires. Par exemple, un alignement sur des positions chromosomiques différentes peut révéler un réarrangement structurel. La colonne TLEN, dérivée de RNEXT et PNEXT, mesure la distance entre les extrémités d'une paire de lectures.

CIGAR Code	BAM Integer	Description
M	0	alignment match (can be a sequence match or mismatch)
I	1	insertion to the reference
D	2	deletion from the reference
N	3	skipped region from the reference
S	4	soft clipping (clipped sequences present in SEQ)
H	5	hard clipping (clipped sequences NOT present in SEQ)
P	6	padding (silent deletion from padded reference)
=	7	sequence match
X	8	sequence mismatch

Figure 4: Liste des opérations CIGAR[5]

• La colonne CIGAR (Rapport Compact d'Alignement) résume les divergences entre un *read* et sa référence à l'aide d'opérations algorithmiques. Chaque opération combine un entier (nombre de bases concernées) et une lettre (type d'opération), comme illustré ci-contre. Ce format concis réduit la taille des fichiers tout en conservant l'intégralité des informations.

• La colonne SEQ contient la séquence brute du read aligné représentant les bases azotées. Elle permet de détecter les variations génétiques par comparaison avec la référence. Son interprétation repose sur la colonne QUAL, encodée en caractères ASCII, qui fournit la qualité de chaque base via le score Phred. Par exemple, un score de 20 (1% de probabilité d'erreur) indique une précision de 99% [2]

3 Contexte biologique

Le fichier SAM joue un rôle clé dans l'analyse des données issues du SHD. Cette technologie, qui a profondément modifié la manière d'aborder la génétique, génère une avalanche de données qui implique d'adopter une stratégie adéquate pour rendre efficace leur stockage et leur utilisation. Ce format de fichier permet de détecter des variations génétiques ponctuelles, telles que des insertions ou les délétions, ou des remaniements structuraux à plus grande échelle.

Prenons l'exemple d'une maladie génétique rare quelconque : l'identification des mutations responsables de cette maladie commence souvent par le séquençage d'individus atteints, puis logiquement par l'alignement de leurs séquences sur une référence soigneusement sélectionnée . Par exemple avec la colonne POS, on peut localiser précisément la position d'une mutation dans le génome et, par comparaison avec des séquences de contrôles sains, identifier les différences significatives. Cela va nous permettre de caractériser le variant génétique causal et d'en étudier l'impact au niveau moléculaire. Un autre exemple cette fois phylogénétique. L'alignement des séquences provenant de différents individus ou groupes d'individus géographiquement éloignés permet d'émettre des hypothèses sur la conservation génétique entre les populations, d'étudier leur adaptation locale à des caractères environnementaux spécifiques, ou encore de retracer l'histoire démographique de ces dernières. La diversité des informations contenues dans ce type de fichier va aider à comprendre ces interactions entre les populations. C'est en fournissant une représentation standardisée et riche en informations du processus d'alignement que le format SAM est devenu incontournable pour les biologistes, les généticiens et de manière générale les chercheurs, en facilitant l'étude de la variation génétique à différentes échelles, d'un individu isolé à une espèce dans son intégralité.

4 Présentation du programme *SCORE*

Le programme prend en entrée un fichier SAM. Ce fichier doit être au format standard avec les items présentés plus haut. Avant de lancer le programme il faut s'assurer que Python version 3.6 minimum soit installé ainsi que les paquets *pandas, argparse, sys et re*. Ces opérations peuvent être effectuées à l'aide de la commande :

```
pip install -r requirements.txt # Installation via pip des dependances du requirement
```

4.1 Script de lancement : *main.sh*

Ce premier script va vérifier la conformité du fichier SAM avant d'appeler le second script codé en Python (*NGS.py*. Il teste plusieurs conditions sur le fichier passé en argument ("$1"). En cas d'erreur, un message s'affiche et le script s'arrête (*exit 1*). Sinon, le script Python s'exécute. Les options *-z, -d, -f et -s* [4] sont utilisées pour effectuer des vérifications courantes sous la forme de disjonctions booléennes successives . Si des erreurs sont détectées, elles peuvent être concaténées et affichées à l'utilisateur via *echo* dans le terminal :

```
#!/bin/bash
if [ -z "$1" ] || [ ! -f "$1" ] || [ -d "$1" ] || [ ! -s "$1" ]; then
    [ -z "$1" ]   && echo "Erreur: Aucun fichier SAM specifie."
    [ ! -f "$1" ] && echo "Erreur: Le fichier n'existe pas."
    [ -d "$1" ]   && echo "Erreur: Le fichier est un repertoire."
    [ ! -s "$1" ] && echo "Erreur: Le fichier est vide."
    exit 1
fi # Verification des conditions de l'input
```

Dans un second temps, nous avons élaboré un système d'options à passer en argument (détaillées dans le *README*). Le script bash va vérifier ici la présence d'options supplémentaires pour tous éléments passés après le second paramètre dans la commande : *"$@:2"*[4]. le cas échéant faire l'intégralité de l'analyse avec *–all* :

```
options="${@:2}" # Tous les arguments apres le premier ($1 est le fichier SAM)

if [ -z "${@:2}" ]; then
    echo "Aucune option specifique fournie, toutes les analyses seront effectuees par defaut."
    python3 "$Script1" "$1" --all
else
    echo "Options specifiees : $options"
    python3 "$Script1" "$1" $options
fi
```

Enfin, une condition post-exécution est ajoutée pour évaluer le succès ou l'échec du script Python en se basant sur la valeur de retour de la commande précédent "[$? -ne 0]". Un message adapté est affiché pour conclure sur l'éxécution :

```
if [ $? -ne 0 ]; then
    echo "Erreur lors de l'extraction des flags et du comptage des occurrences, faire une verification du
        nom et du contenu du script python"
    exit 1
else
    echo -e "\nExtractions des Flags et comptages des occurrences effectues avec succes\n"
fi
```

4.2 Traitement du fichier avec *NGS.py* illustration avec les motifs CIGAR

Tout d'abord on va extraire les éléments tabulés successivement du fichier SAM et les stocker dans une structure de données de type dictionnaire. Pour ce faire nous parcourons les lignes à l'aide d'une boucle *for* et découpons chacune d'elle en colonne avec un sépérateur grace à la méthode *split()*[3]:

```
    def dico_extraction1(fichier_sam):
    file = open(fichier_sam, 'r')        # Ouverture en mode lecture
    d_sam = {}                           # Creation du dictionnaire
    id_ligne = 1                         # Initialisation de l'ite rateur
    for i_ligne in file:
        if i_ligne[0] != "@":            # Nous excluons les lignes du Header
            l_colonnes = i_ligne.split()# On coupe les lignes en colonnes
            QNAME = l_colonnes[0]        # Extraction successive de chaque champs.
            FLAG = l_colonnes[1] ... }
            d_sam[id_ligne] = {          # On complete sucessivement le dictionnaire.
                "QNAME": QNAME,
                "FLAG": FLAG, ... }
            id_ligne += 1                # On incremente de 1 pour chaque tour de boucle.
    file.close()                         # On referme le fichier SAM
    return d_sam

# Appeler la fonction dico_extraction1 pour obtenir le dictionnaire d_sam
d_sam = dico_extraction1(fichier_sam) #Stocjage du dictionnaire dans la variable
```

Ensuite, nous procédons au traitement successif des éléments tabulés extraits du fichier SAM. Le programme est conçu pour analyser les éléments suivants :

- Une analyse détaillée des motifs CIGAR.
- Les fréquences nucléotidiques.
- La distribution des FLAG.
- Une analyse des positions chromosomiques.
- Une analyse de la qualité de mapping.

Nous présentons ici l'analyse des motifs CIGAR afin de respecter la contrainte de forme du rapport. L'objectif est d'illustrer la stratégie générale appliquée à un type spécifique d'information. Le traitement des autres données du fichier SAM suit une méthodologie sensiblement identique similaire. Pour une description détaillée des approches algorithmiques utilisées les autres éléments du fichier SAM, veuillez vous référer au fichier *README*. La fonction *analyse_ CIGAR* examine les motifs CIGAR dans le dictionnaire *d_ sam* pour calculer la répartition des opérations, en initialisant un dictionnaire associant chaque motif à une description et un compteur d'occurrences.

```
    comptes_CIGAR = { # Dictionnaire qui decrit les operations et initialise le compteur des occurences :
    'M': ["Alignes", 0],          'I': ["Insertions", 0],    'D': ["Deletions", 0],
    'N': ["Sauts de bases", 0],   'S': ["Soft Clipping", 0], 'H': ["Hard Clipping", 0],
    'P': ["Completion", 0],       '=': ["Match exact", 0],   'X': ["Mismatch", 0] }
```

La stratégie choisie est d'extraire chaque motif à l'aide d'une expression régulière , on recherche les couples nombre et lettres parmis les caractères CIGAR (*"REGEX_ CIGAR"*), puis, on réitère cette recherche sur chaque motif de chaque ligne en mettant à chaque tour de boucle à jour les compteurs d'occurences du dictionnaire compte_CIGAR. La difficulté ici était de gérer les potentiels tuples successifs "nombre et motif", pour chaque lecture :

```
def analyse_CIGAR(d_sam):
    # On recherche chaque chiffre suivi d'une operation CIGAR :
    REGEX_CIGAR = re.compile(r'(\d+)([MIDNSHP=X])')
```

```
    TOTAL_OPE_CIG = 0  # Compteur global des operations

    for read in d_sam.values():                          # Parcours les reads dans d_sam
        matches = REGEX_CIGAR.findall(read["CIGAR"])     # findall nous permet de traiter tous les motifs
        for SUM_OPE_CIG_str, OPE_CIG in matches:         # On traite chaque motif CIGAR individuellement
            SUM_OPE_CIG = int(SUM_OPE_CIG_str)           # Convertir en entier le premier element du tuple
            if OPE_CIG in comptes_CIGAR:                 # Verifier si le motif est present dans le dico
                comptes_CIGAR[OPE_CIG][1] += SUM_OPE_CIG # Sommation
                TOTAL_OPE_CIG += SUM_OPE_CIG             # MAJ du total global pour le motif courant

    for OPE_CIG, (commentaire, compte) in comptes_CIGAR.items():
        pourcentage = (count / TOTAL_OPE_CIG * 100) if TOTAL_OPE_CIG > 0 else 0
        comptes_CIGAR[OPE_CIG].append(pourcentage)       # Ajouter le pourcentage
    return comptes_CIGAR                                 # Retourner le dictionnaire final
```

Enfin, on créer une structure de donnée dataframe du dictionnaire pour rendre ergonomique la sortie dans le terminal, cette opération est effectuée avec une liste par compréhension :

```
    Data_cigar = [
(f"{cle_CIG}|", f"{Val_CIG[0]} |", f"{Val_CIG[1]} |",
     f"{Val_CIG[2]:.3f} % |")
for cle_CIG, Val_CIG in comptes_CIGAR.items()]

t_Data_cigar = pd.DataFrame(Data_cigar,columns=["
    Motif", "Nom", "Occurences", "Valeur relative"])
```

```
--------------------------------------------------------------------------
2. ANALYSE DES CIGARS : COMPTAGES DES MOTIFS ET DISTRIBUTION RELATIVE
--------------------------------------------------------------------------

  Motif          Nom              Occurences          Valeur relative
0          M|         Alignés |          35000611 |      99.995 % |
1          I|      Insertions |                18 |       0.000 % |
2          D|       Délétions |                61 |       0.000 % |
3          N|  Sauts de bases |                 0 |       0.000 % |
4          S|   Soft Clipping |              1621 |       0.005 % |
5          H|   Hard Clipping |                 0 |       0.000 % |
6          P|      Complétion |                 0 |       0.000 % |
7          =|     Match exact |                 0 |       0.000 % |
8          X|        Mismatch |                 0 |       0.000 % |
```

Figure 5: Résultat de l'analyse CIGAR

5 Discussion

Ce qui caractérise notre programme, c'est sa capacité à gérer avec simplicité et efficacité tous les contrôles préliminaires (vacuité, type du fichier, etc.) avant de lancer le traitement des données avec le script Python. Ces contrôles s'effectuent avec l'affichage de messages détaillés à l'utilisateur, ce qui permet de cibler plus facilement un problème. On pourrait envisager d'étayer ce système en gérant, par exemple, les permissions ou en l'étendant au contrôle de la bonne exécution de chaque fonction Python, d'ailleurs ce partionnement en fonction, pourrait être envisagé sur le script bash Notons également, que pour amélioré sa lisibilité et sa réutilisation. Le support des options, permet à l'utilisateur de personnaliser l'utilisation du traitement et d'adapter à ses besoins l'éxécution. Notamment pour le traitement de données plus complexes.

En effet, nous sommes ici en présence d'un "petit" fichier SAM, des tests complémentaires serait intéréssant pour le tester dans des conditions plus réalistes avec une expérience de SHD. Il également possible d'optimiser notre code pour minimiser l'utilisation de la mémoire, par exemple en traitant les fichiers ligne par ligne ou en utilisant des bibliothèques optimisées pour le traitement de gros ensembles de données. En ce sens, l'ajout du temps de traitement dans le script bash nous permet un bon moyen de mesurer la performance du programme, l'étendre au script python après chaque fonction permettrait d'identifier avec précisions les points d'améliorations. En effet l'avantage d'utiliser python ici est son extensibilité, on peut faire des amélioration en ajoutant des calculs plus complexe par exemple, ou des représentations graphiques des résultats avec des librairies dédiées. Il en est de même pour sa portabilité, Python est un langage consensuel présent sur toutes les plateformes modernes. [3]

Une fois ces différentes améliorations effectuées et testé, il serait intéréssant d'envisager une interface graphique plus ergornomique pour l'utilisateur. Malgrès les différents inconvénients que nous avons identifiés ici, nous pensons que ce programme de première intension donne une bonne base de travail robustes pour extraires les informations pertinente du fichier SAM avec facilité et efficacité.

Bibliographie compilée

[1] Li Heng, Handsakerr Bob, and et al. *Sequence Alignment/Map format and SAMtools | Bioinformatics | Oxford Academic*. Aug. 2019. URL: https://academic.oup.com/bioinformatics/article/25/16/2078/204688 (visited on 11/21/2024).

[2] Petr Danacek. *Samtools Flags Documentation*. 2023. URL: https://www.htslib.org/doc/samtools-flags.html (visited on 11/21/2024).

[3] Pierre Pompidor. *Initiation à la programmation en Python*. fr. Pages 21 – 23. Université de Montpellier, 2024. URL: https://moodle.umontpellier.fr/pluginfile.php/1471751/mod_resource/content/1/cours_Python_Pierre_Pompidor.pdf (visited on 11/06/2024).

[4] Pierre Pompidor. *Support du cours Utilisation des Systèmes Informatiques - Commandes UNIX/Linux, outils, initiation au scripting en Bash*. fr. Pages 24 – 29. Université de Montpellier, 2024. URL: https://moodle.umontpellier.fr/pluginfile.php/1768462/mod_resource/content/5/Cours_Unix_Linux_2022_Pierre_Pompidor.pdf (visited on 11/01/2024).

[5] *Sequence alignment*. en. Page Version ID: 1254539960. Oct. 2024. URL: https://en.wikipedia.org/w/index.php?title=Sequence_alignment&oldid=1254539960#Representations (visited on 11/21/2024).

Liste des figures