



# ADN ASSEMBLER



**BOUKHEROUAA** Taha

**OUDAOUD** El Mehdi

**Groupe 2**

**Groupe 5**

Encadré par : Mr. Hassounny

# REMERCIEMENTS

2019-2020

On tient à remercier toutes les personnes qui ont contribué au succès de ce projet et qui nous ont aidé lors de la réalisation de ce projet.

On voudrait dans un premier temps remercier Mr.ELHASSOUNY qui a eu le soin d'examiner et encadrer ce travail et ainsi nous offrir une véritable opportunité d'apprentissage.

Nous aimerions aussi gratifier les efforts de nos professeurs des modules de « Structures de données » : Monsieur Abdellatif EL FAKER , et de « Techniques de programmation » : Monsieur Hatim GUERMAH pour leur disponibilité, leur savoir-faire et les méthodes de programmation et raisonnement qu'ils nous ont transmis.

# ABSTRACT

2019-2020

The DNA sequence contains necessary informations of living beings to survive and to breed. Establishing this sequence is also usefull for researchs on how organisms live. In medecine, it can be used to identify, diagnose and potentially find treatments to genetic diseases.

In this subject we look to assemble a DNA from an FQ file created using global sequencage method (or Shotgun), also to look if it contains a bacterial strain ( obtained from a database).

# PLAN

---

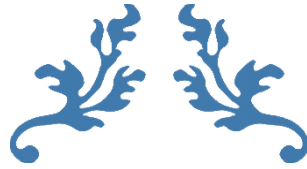
## Chapitre 1 : CADRE GENERAL ET CONCEPTION DU PROJET :

- I. Introduction
- II. Cahier de charge
- III. Analyse et conception

## Chapitre 2 : MISE EN ŒUVRE ET REALISATION :

- I. Outils utilisés
- II. Exécution

## Chapitre 3 : Bibliographie et conclusion :



---

# CHAPITRE 1 : CADRE GENERAL ET CONCEPTION DU PROJET

---



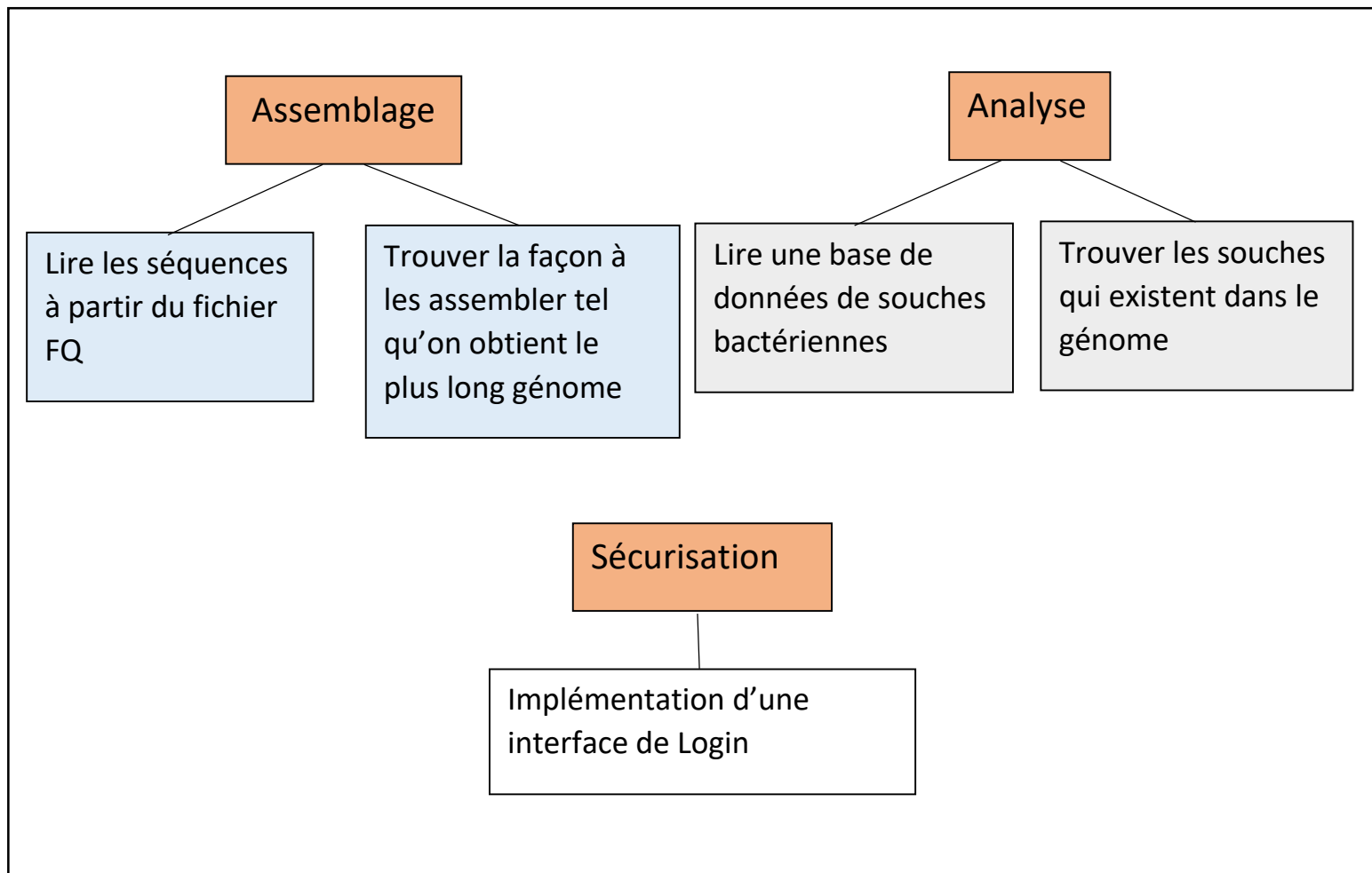
Dans ce chapitre on parlera des méthodes utilisées pour assembler l'ADN notamment la méthode « shortest superstring using greedy setcover », et La méthode de « De BRUIJN » avec une implémentation de correcteur d'erreur de lecture. Ensuite analyser

## I. INTRODUCTION :

La séquence d'ADN contient l'information nécessaire aux êtres vivants pour survivre et se reproduire. Déterminer cette séquence est donc utile aussi bien pour les recherches visant à savoir comment vivent les organismes que pour des sujets appliqués. En médecine, elle peut être utilisée pour identifier, diagnostiquer et potentiellement trouver des traitements à des maladies génétiques.

Dans ce sujet on cherche à faire l'assemblage de l'ADN à partir d'un fichier FQ crée à l'aide de la méthode de séquençage globale (ou Shotgun), ainsi que chercher s'il contient des souches bactériennes (à partir d'une base de données).

## II. CAHIER DE CHARGE



### III. ANALYSE ET CONCEPTION :

La conceptualisation du projet n'a pas été facile vu que les méthodes d'assemblage sont très nombreuses cependant nous avons décidé de nous focaliser sur : une approche naïve avec la méthode « Greedy Set Cover » qui cherche à trouver la meilleure concaténation possible entre tout élément existant dans le fichier des séquences (.fq) et refaire cette procédure jusqu'à avoir créé l'ADN (avec une marge d'erreur).

Figure : Fichier contenant l'ensemble de lectures obtenues à partir d'un séquenceur

```
1  @r0
2  GTCCAGCAGAGCAAGTGATGCGAGAGCTGCCCATCCTCCAACCAGCATGCCCTAGACATTGACACTGCATCGGAGTCAGGCCAAG
3  +
4  I
5  @r1
6  GGAGTACGACTTCAGAGATCTCACTTGGTGTATCAACCCGCCAGAGAGAATCAAATTGGATTATGATCAATACTGTGCAGATGTGG
7  +
8  I
9  @r2
10 GCAAAATTTTGATCTCTCTTGGCTTCAACAATCAATTCAACCATGACCCGAGATGTAGTCATACCCCTCTCACAAACAACGATCTCT
11 +
12 I
13 @r3
14 GAGTTAATTGAAGCCCTAGATTACATTTTCATAACTGATGACATACATCTGACAGGGGAGATTTTCTCATTTTTCAGAAGTTTCGG
15 +
16 I
17 @r4
18 AATGACAGAGACCGCTATGACCATTGATGCTAGGTATGCAGAATTCTAGGAAGAGTCAGATACATGTGGAAACTGATAGATGGTT
19 +
```

Figure : Base de données des pseudo-souches bactériennes

Cependant cette méthode s'avère assez complexe en terme de temps (NP-hard ou complexité exponentielle), ainsi on a pensé à implémenter une méthode plus avancée (utilisée dans les assembleurs modernes). C'est la méthode des graphes de De Bruijn.



Et puis après avoir assembler le genome on va chercher l'analyser détectant ainsi les souches bactériennes qui y réside.

Pour accomplir cette tache on a penser à créer une base de données de souches bactériennes (virus pour simplifier).

```

1  virus_0_107 : AACCTGTTAGGTTTCGGGCAAATTGGCACTCGCGAGTACGTGCTTCCCTGCCCGTTTGGTTATCGAGGCGGCATAGGTATCTCCATCCAGTTGCTT
2  virus_1_113 : AACGTACGTTGATCTAGAGCCTGTCAAGTATATGACACTTAAAGGCAGCCTAAAGACTGCGCGCCCAGGCTATGAGGAAAAGTTGTACGTTTGAACCG
3  virus_2_116 : AACGTACCATACCGCCTCTGGATATGCTCCGTTTCGGCATCATCTCTACGCGGCTCACGCACGAATTCGCGAGCCCATTAATCTCCGCACAGCCCCG
4  virus_3_107 : AAGTATTGTTACAAAGACCCCATGGCTACAGCAGGTGCCATTGGGGTTAATCAAGCACAAAGAGAGACCCCCGACGAGAGGATAGGCTTTTGCCACTGC
5  virus_4_192 : ACATGCGTTTAAACCCCATCGACAACAGCGAGCCGCTGCTGTTATACATTTAATCGGCAGTGTATAATGTGTATGCTGATTGGGCGACAAGCTGGTGT
6  virus_5_183 : ACCCACCCTGGTTCTCTGTTTCAGAGTCAAACCTATAATCTCTAGATGCTTCTTGGGATTAAACGCATGGTCAGGTCGGTGAAGGAGCCAGGTAAAGTAA
7  virus_6_197 : ACCTGAACACTGAGGTGCTCTAATCAGTTATGGACGCAATGGCTATTCTGTTGCTGTGAGCGTAGGTCTTACGGGCGAGATTTAGTCTGGGCAATAA
8  virus_7_131 : ACGTGGGAGTTGCGAATGCCCTGGCGTCTGTTACTTCAGGGACGGTGATGACCGTCGACAAGTAGCAGAGGCATGGTCTACTGCTTCGAACAAAGCCG
9  virus_8_114 : ACTACGCTGGCGTCTTTGATTACAGTTGGGCAATCCTTCGTGGGTTAGCGAGCCAACCTCCGGGGCGCCACAACACTTGGCAACGGCTTGAACCTAACG
10 virus_9_129 : AGAGCAGCGAACGGTTACGAATCCATAGTAGCCGATACAGTATGGACCTTCAATTCGGGACCTATTGCGAGTTAGGTGGAAGTACAGAACTCCGGCTATG
11 virus_10_153 : AGAGTTAACTCTATCGGGATGAGTCTAGTGGTGCTTTTGTCTCCAAATGGTGTGGATAGGATAGACTGATGCACGATATTCCTCTGCAAGGATTGA
12 virus_11_123 : AGATCAAGGAGAAGGTACTAGGTGGCTTCAGTATAGCATTTGCGAGTCAAGTGACATCAATACGTAATAAGGCTTGGTCAGAGCAGACTCTCGATG
13 virus_12_128 : AGCATTGCGGCTAACTTCAGTATACTACCTTACCCAAATCGTTTTTGAAGTCATACTCGTCTGGTAAGTCGAGCACGGATGGTCGCTCCGGCTTACC
14 virus_13_103 : AGCCTTACATGTGACTGGCGCTCGTCTCACTTCTCAAGTGTTCTACTGGGATCGCTAACCGTTTCAGGACAGACGCTACCACACTGTGAGGACTCTG
15 virus_14_194 : AGCTACGGTCTTATCGTAGCGATACCTGGGGTAATAACTTCTATTGAGAGTCATTTTGGCGTATCAATGCTACAGTAGAGATCCTCAATCATCA
16 virus_15_104 : AGCTGACGGTAATACAGCATAATAGAGGCTAGACGATAGACGACGAGTCCCACTTGTACTCCACGGCTTCAACGATCGCACATCCATGGTCTACCGGG
17 virus_16_154 : AGGGGGGTTAGTTGCATTATGCTCCTTGACATCTATTGATCCTGAAATCACAGTGTATGACAGATGCAATCCTGTATAGATTCTGTCGGGACAGCC

```

## Analyse fonctionnelle : Greedy set cover

STRUCTURE	UTILITE
<b>Element</b>	C'est un nœud d'une liste chaînée composée d'un ensemble de chaines de caractère (représente l'ensemble des substrings).
<b>Liste</b>	C'est un pointeur sur la tete de la liste chaînée d' « Element ».
<b>Tuple</b>	Représente un nœud d'une combinaison entre de chaine de caractère différentes (sera utile pour trouver le superstring le plus long)
<b>Dictionnaire</b>	C'est un pointeur sur la tete de la liste chaînée de « Tuples »

L'implementation de cette methode est simple :

Trouver le plus grand « overlap » (GTA et TAC on pour overlap 2) entre deux lecture du fichier.fq et les concaténer en respectant le principe (CAAT+ATGG = CAATGG) et sauvegarder le résultat enfin.

Fonction	Utilité
<b>initialisation_liste</b>	Initialise une liste d' « élément » et la renvoie
<b>initialisation_dico</b>	Initialise une liste de « Tuple » et la renvoie
<b>insertion_liste</b>	Insère (brutement) un « element » dans la liste à partir d'une chaine de caractère.
<b>insert_dico</b>	Insère (sans répétition*) un « Tuple » dans le dictionnaire à partir de (Gauche,Droit) .
<b>Recherche_liste</b>	Cherche si une chaine existe dans la table d'éléments.
<b>lire_sequence</b>	Lit le fichier de format spéciale (.fq)
<b>chargement_input</b>	Convertit les lectures dans .fq en des éléments dans une liste.
<b>Find_overlap_start</b>	Trouve l'element à parti de quelle on a overlap sinon retourne -1
<b>subString</b>	Retourne une sous_chaine qui commence en début et fini en fin.
<b>overlap</b>	Calcule l'overlap entre deux chaine.
<b>permutations</b>	Retourne un Dictionnaire contenant toute combinaison possible de séquence
<b>Overlap_maximale</b>	Chercher dans un Dictionnaire la plus grande valeur d'overlap entre gauche et droit
<b>Greedy_setcover</b>	L'aplication de l'algorithme
<b>Sauvegarde_superstring</b>	Sauvegarde le superstring dans un fichier

MALHEUREUSEMENT CETTE METHODE EST TRES INEFFICACE (ON NE PEUT PAS VRAIMENT ASSEMBLER LE GENOME EN ENTIER).

## Analyse fonctionnelle : De BRUIJN Graphs

Implemtons une structure de graphe :

STRUCTURE	UTILITE
<b>DBN_noeud</b>	Représente un nœud du graphe de De Bruijn
<b>arc_list</b>	Représente une liste des arcs du graphe de De Bruijn
<b>Nœud_list</b>	Représente une liste des nœuds du graphe de De Bruijn
<b>DBN_GRAPH</b>	Représente le graphe de De Bruijn

Cette méthode passe par 3 phases clefs :

- 1- Création du graphs à partir k-1 mers de chaque read (voir documents) .
- 2- Correction d'erreurs de lecture commise pas le séquenceur ainsi que la correction de la topologie du graph(assurer l'unicité du chemin eulerien).
- 3- Parcours du graph à l'aide de l'algorithme de parcours eulerien.

Fonction	Utilité
<b>create_noeud</b>	Créer un nœud
<b>init_DBN_graph</b>	cette fonction recherche un str dans le graph et retourne le sommet contenant ce str
<b>add_to_universe</b>	Ajoute un sommets à l'ensemble des sommets
<b>read_seq</b>	Lire le .fq
<b>append_arc</b>	Ajoute un arc entre deux nœud

<b>DBN_graph_builder</b>	Créer le DBN graph
<b>calculer_moyenne_occurences</b>	Calcule le nombre d'occurrence moyen des sommets distinguer les sommets erroné (moins frequent).
<b>is_brother</b>	Affirme si deux chaine ne diffère que pas un seul nucléotide (taux d'erreur suposée 1% au max)
<b>arc_sortant_list_union</b>	Fait l'union de deux liste d'arc_sortant
<b>arc_entrant_union</b>	Fait l'union de deux liste d'arc_entrant
<b>replace_noeud</b>	Remplace un nœud moins fréquent par son frère plus frèquent
<b>corriger_erreur_type_read</b>	Implementation de la correction d'erreur (remplacement de neoud)
<b>read_chemin_eulerien</b>	La recherche du chemin eulerien
<b>chemin_eulerien_makeNread</b>	Le parcours du chemin eulerien et sauvegarde du superstring

## Analyse fonctionnelle : Recherche de souche malade

Il suffit de chercher si une souche est une sous-chaine du genome assemblé

Fonction	Utilité
<b>recherche_souche_super</b>	Cherche si une souche est une sous-chaine du genome.
<b>read_adn</b>	Lit le fichier assemblé (ADN)
<b>recherche</b>	Fait le parcours toute les souches dans la base de données virus et affirme s'ils existent où non

## Analyse fonctionnelle : Sécurité

Pas une grande sécurité car le login et le mot de passe sont « hard codé » mais le procédure d'authentification marche très bien

Login : oudaoud

Mot de passe : taha123



---

## CHAPITRE 2 : MISE EN ŒUVRE ET REALISATION

---



Dans ce chapitre on parlera des outils informatiques et logiciels utilisées pour accomplir ce projet ainsi qu'une simulation de l'exécution du programme

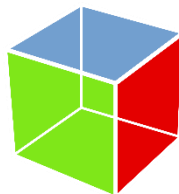
# I. OUTILS UTILISES

Afin de réaliser ce projet on a compté sur plusieurs outils informatiques autre que le langage C dont :

Le langage python pour générer les souches bactériennes



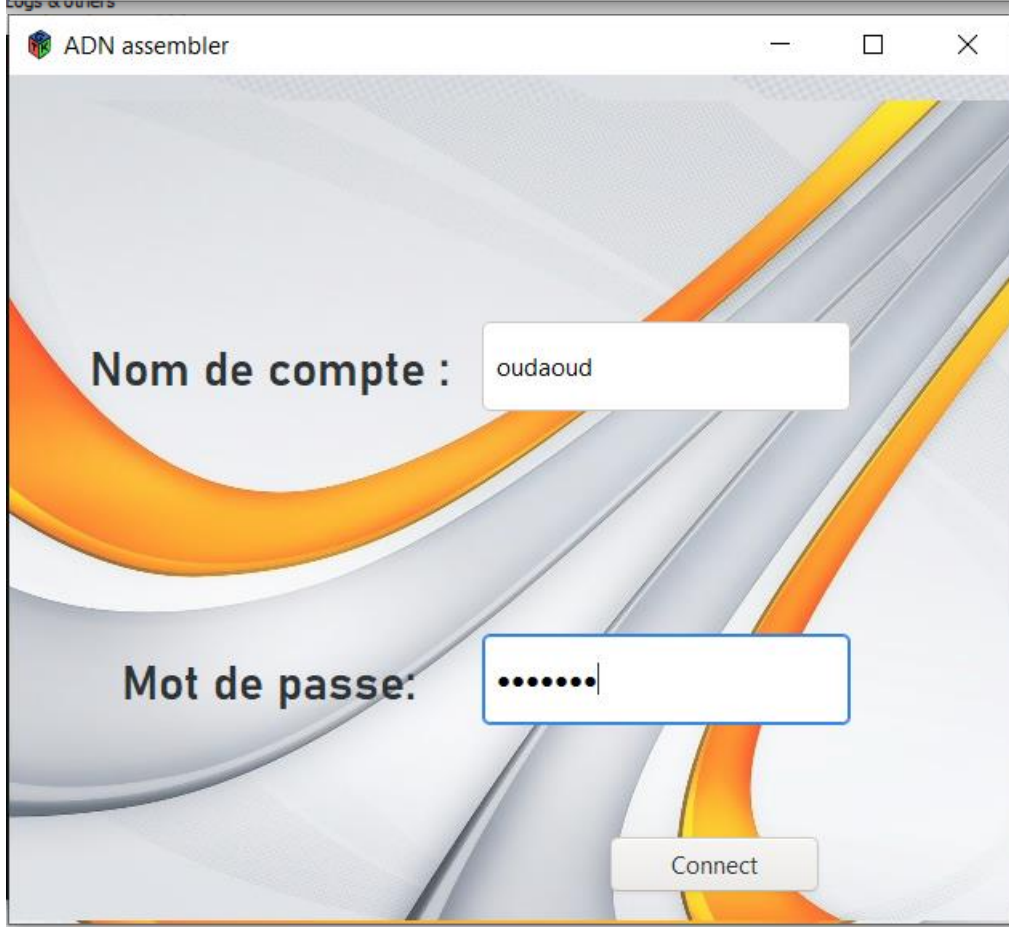
La bibliothèque GTK pour l'interface graphique.



Code Blocks pour la compilation et linkage du projet.



## II. EXECUTION





# ADN ASSEMBLER



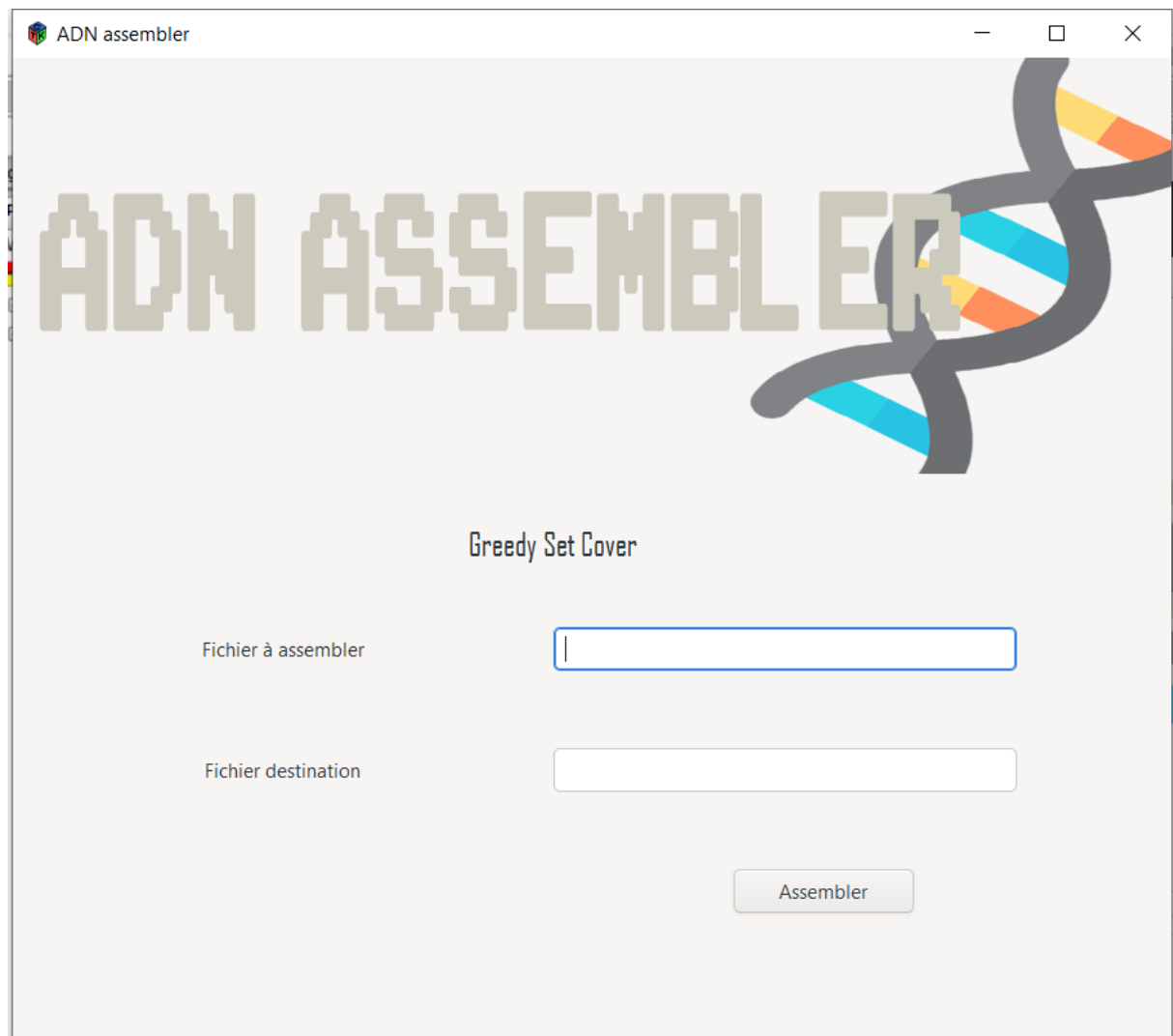
## Debruijn graph method

Fichier à assembler

Fichier destination

taille des souschaînes (k\_mers) :

Assembler



Le motif: GGAGATCCACGAGCTCCTGAACTCCAATCCA  
Le motif: TCCACGAGCTCCTGAACTCCAATCCAGAG  
Appuyez sur une touche pour continuer...

ADN assembler

# ADN ASSEMBLER

Analyse du genome

Fichier ADN assemble

Base de donnée virus

Fichier Edition Format Affichage Aide

```
GGAGATCCACGAGCTCCTGAACTCCAATCCAGAGGCAACAACCTTCCGAAGCTTGGC  
GCAATTGTGGGAGGCGCAGTTAGTAGAGGTGATATCAACCCTATTCTGAAAAAAGCTT/  
TCATACCCCTCCTCACAAACAACGATCTCTTAATAAGGATGGCACTGTTGCCAGGAG/  
ACTGATAGATGGTTTCTTCCCTGCACTCGAGTTAATTGAAGCCCTAGATTACATTTT
```