Projet C (Smith-Waterman)

Samuel Murail

2024-03-21

Projet en programmation : Alignement de séquences avec l'algorithme de Smith-Waterman

Dans ce projet, vous devrez implémenter un programme en langage C qui permettra d'aligner deux séquences d'acides aminés en utilisant l'algorithme de Smith-Waterman.

L'algorithme de Smith-Waterman est l'un des algorithmes les plus couramment utilisés pour l'alignement de séquences et permet de déterminer la similarité entre deux séquences. Il est décrit dans l'article de Smith et Waterman de 1981 [**1**].

L'algorithme de Smith-Waterman est un algorithme de programmation dynamique utilisé pour l'alignement local de séquences. Il fonctionne en comparant toutes les sous-séquences possibles de deux séquences données et en trouvant la sous-séquence ayant le meilleur score de similarité.

Pour ce faire, l'algorithme utilise une matrice de scores pour calculer le score de similitude entre les caractères de chaque position des deux séquences. La matrice est remplie en commençant par la position de début et en se déplaçant dans toutes les directions possibles. Le score pour chaque position est calculé en comparant les caractères de cette position dans les deux séquences et en ajoutant le score de similitude correspondant à la matrice de scores.

Ensuite, la matrice est parcourue à nouveau pour trouver la sous-séquence ayant le meilleur score de similitude. Pour ce faire, on cherche le score maximal dans la matrice et on remonte le chemin correspondant dans la matrice pour récupérer la sous-séquence alignée.

Références :

1. Smith, Temple F. & Waterman, Michael S. (1981). *"Identification of Common Molecular Subsequences"*. Journal of Molecular Biology. 147 (1): 195–197.
2. https://en.wikipedia.org/wiki/Smith%E2%80%93Waterman_algorithm

Votre programme prendra en entrée deux séquences de protéines et calculera l'alignement optimal entre les deux séquences au format `fasta` en utilisant l'algorithme de Smith-Waterman.

```
> ./align_sequences sequence1.fasta sequence2.fasta
```

Fonctions à implémenter

Voici les différentes fonctions que les étudiants devront implémenter pour réaliser ce projet :

1. `read_sequence` : Cette fonction prend en entrée le nom d'un fichier de séquence et retourne une chaîne de caractères représentant la séquence. *Attention* vous devrez surement parcourir une première fois le fichier pour déterminer la taille de la séquence afin de pouvoir allouer la mémoire nécessaire pour la chaîne de caractères.

   ```c
   char* read_sequence(char* filename);
   ```

- Arguments :
  - `filename` : une chaîne de caractères représentant le nom du fichier de séquence.
- Sortie :
  - La fonction retourne une chaîne de caractères représentant la séquence.

2. `read_substitution_matrix` : Cette fonction prend en entrée le nom d'un fichier de matrice de substitution et retourne une matrice de d'entiers représentant la matrice de substitution. Vous pourrez utiliser la matrice BLOSUM62 disponible sur le site du NIH (https://www.ncbi.nlm.nih.gov/Class/FieldGuide/BLOSUM62.txt).

```
int** read_substitution_matrix(char* filename);
```

- Arguments :
  - `filename` : une chaîne de caractères représentant le nom du fichier de matrice de substitution.
- Sortie :
  - La fonction retourne un pointeur vers la matrice de substitution.

3. `initialize_matrix` : Cette fonction initialise une matrice de taille n x m avec des valeurs initiales.

```
int** initialize_matrix(int n, int m, int value);
```

- Arguments :
  - `n` : un entier représentant le nombre de lignes de la matrice.
  - `m` : un entier représentant le nombre de colonnes de la matrice.
  - `value` : un entier représentant la valeur initiale de la matrice.
- Sortie :
  - La fonction retourne un pointeur vers la matrice initialisée.

4. `compute_score` : Cette fonction calcule le score de similarité/différence entre deux caractères en utilisant une matrice de substitution. Attention, la matrice de substitution étant uniquement numérique, il faut faire attention à la manière dont on accède aux valeurs de la matrice. Il faudra d'abord convertir les caractères en entier en utilisant l'ordre des acides aminés définit dans la matrice de substitution (heureusement celui ci est toujours: `A R N D C Q E G H I L K M F P S T W Y V B Z X *`). N'utilisez que les acides aminés standarts (pas `B Z X *`).

```
int compute_score(char *a, char *b, int** substitution_matrix);
```

- Arguments :
  - `a` : un caractère représentant le premier caractère.
  - `b` : un caractère représentant le deuxième caractère.
  - `substitution_matrix` : un pointeur vers la matrice de substitution.
- Sortie :
  - La fonction retourne un entier représentant le score de similarité/différence entre les deux caractères.

5. `calculate_alignment` : Cette fonction prend en entrée deux séquences et calcule l'alignement optimal entre les deux séquences en utilisant l'algorithme de Waterman-Smith.

- la fonction calculera la matrice de scores en utilisant l'algorithme de Smith-Waterman et la fonction `compute_score`.
- la fonction parcourera la matrice de scores pour trouver le meilleur score de similarité.
- on remonte le chemin correspondant dans la matrice pour récupérer les sous-séquence alignée.
- Enfin, la fonction retournera le score de similarité des sous-séquence alignée et les deux séquences alignées.

```c
int calculate_alignment(char* seq1, char* seq2, int** substitution_matrix,
    int gap_penalty, char** aligned_seq1, char** aligned_seq2);
```

   - Arguments :
     * `seq1` : une chaîne de caractères représentant la première séquence.
     * `seq2` : une chaîne de caractères représentant la deuxième séquence.
     * `substitution_matrix` : un pointeur vers la matrice de substitution.
     * `gap_penalty` : un entier représentant le score de pénalité pour un gap.
     * `aligned_seq1` : un pointeur vers une chaîne de caractères représentant la première séquence alignée.
     * `aligned_seq2` : un pointeur vers une chaîne de caractères représentant la deuxième séquence alignée.
   - Sortie :
     * La fonction retourne un entier représentant le score de similarité entre les deux séquence.
     * les deux séquences alignées sont retournées dans les variables `aligned_seq1` et `aligned_seq2`.

5. `print_alignment` : Cette fonction prend en entrée deux séquences alignées et les imprime à l'écran.

```c
void print_alignment(char* aligned_seq1, char* aligned_seq2);
```

   - Arguments :
     - `aligned_seq1` : une chaîne de caractères représentant la première séquence alignée.
     - `aligned_seq2` : une chaîne de caractères représentant la deuxième séquence alignée.
   - Sortie :
     - La fonction ne retourne rien.

Simplifications du problème

Dans le cas ou l'implémentation de l'algorithme de Smith-Waterman est trop complexe, vous pouvez simplifier le problème en n'utilisant pas de matrice de substitution et en considérant que le score de similarité entre deux caractères est de 1 si les deux caractères sont identiques et de -1 sinon. Vous pouvez également considérer que le score de pénalité pour un gap est de -2.

Vous pouvez également utiliser l'outil en ligne suivant pour vérifier votre implémentation:

- https://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Smith-Waterman

Aller plus loin

Si vous avez terminé le projet, vous pouvez essayer d'implémenter les fonctionnalités suivantes:

1. Ajouter un argument `gap_extension` à votre programme pour permettre de spécifier le score de pénalité pour l'extension d'un gap. Ceci permet de favoriser des gaps plus long par rapport à une multitude de gaps plus courts.