Alignement global

Le but de ce premier projet (développé en C) est de réaliser une bibliothèque fournissant une méthode de calcul d'un alignement global entre deux séquences génétiques. Cette bibliothèque sera réutilisée dans le second projet (développé en Java). L'idée d'interfacer Java et C est de profiter des capacités du C pour réaliser des programmes rapides, notamment au niveau de la gestion mémoire. Votre bibliothèque devra donc fournir une méthode très efficace de calcul.

1 Cahier des charges

La bibliothèque que vous développez fournira une méthode possédant la signature suivante :

```
int global_alignment (
  char *query,
  char *target,
  char *alphabet,
  float **substitution,
  float open,
  float extension,
  float *score,
  struct alignment *result
);
```

avec :

`query`	la première séquence
`target`	la seconde séquence
`alphabet`	alphabet de la matrice de score
`substitution`	la matrice de substitution
`open`	pénalité d'ouverture de gap
`extension`	pénalité d'extension de gap
`score`	le score de l'alignement calculé
`result`	sauvegarde du résultat de l'alignement (si ≠`null`)

la valeur retournée est un code d'erreur :

0	tout s'est bien passé
1	erreur d'allocation mémoire
2	erreur d'alphabet
3	erreur

Si `result` vaut `null` à l'appel de la fonction, cela signifie que l'utilisateur n'est intéressé que par le score : votre méthode doit alors s'exécuter plus rapidement (ou tout au moins en utilisant moins de mémoire).

1.1 La matrice de substitutions

La matrice de substitutions contient le score associé à chaque couple de lettres mises en correspondance dans l'alignement. La variable `alphabet` permet simplement de savoir dans quel ordre sont donnés ces scores. Par exemple, si `alphabet`="ATCG", alors `substitution`[0][1] contient le score d'alignement de 'A' et 'T'.

Si une des séquences contient une lettre qui n'est pas dans l'alphabet, le programme doit s'arrêter proprement.

1.2 La structure d'alignement

La structure de sauvegarde du résultat sera déclarée ainsi :

```
struct alignment {
  char *query;
  char *target;
  float score;
  int length;
}
```

avec :

`query`	la première séquence ajoutée de '-'
`target`	la seconde séquence ajoutée de '-'
`score`	le score d'alignement
`length`	la longueur de l'alignement

Cette structure ne permet de sauver qu'un seul alignement. Bien sûr il se peut que plusieurs alignements obtiennent le meilleur score. Vous n'en retiendrez qu'un seul.

2 Première version (pour le 30 octobre)

Dans la première version que vous allez implémenter, on n'utilisera pas une fonction de gap affine mais une fonction de gap linéaire. Cela signifie que le coût d'ouverture ou d'extension de gap est le même. Cette première étape vous permettra de vous (re-)familiariser avec la programmation dynamique et plus particulièrement avec la remontée de la matrice de score pour obtenir un alignement associé au meilleur score.

Vous devez donc écrire une fonction dont la signature est la suivante :

```
int global_alignment_linear (
  char *query,
  char *target,
  char *alphabet,
  float **substitution,
  float indel,
  float *score,
  struct alignment *result
);
```

où `indel` est le coût associé à une gap d'une lettre.

Dans cette version, l'optimisation mémoire ne doit pas être le principal souci.

3 Seconde version (pour le 12 novembre)

Dans cette seconde version, nous allons utiliser une fonction de gap affine. Cela rend les choses (à peine) plus compliquées au niveau du calcul mais la gestion de la remontée de la matrice de scores nécessite un peu de réflexion.

Pour rappel, les formules de récurrence pour le calcul de l'alignement global avec gaps affines sont les suivantes :

`Ouv`	:	pénalité d'ouverture de gap
`Ext`	:	pénalité d'extension de gap

$$a(i,j) = Sub(i,j) + \max \begin{cases} a(i-1,j-1) \\ b(i-1,j-1) \\ c(i-1,j-1) \end{cases}$$

$$b(i,j) = \max \begin{cases} \texttt{Ouv} + \texttt{Ext} + a(i,j-1) \\ \texttt{Ext} + b(i,j-1) \\ \texttt{Ouv} + \texttt{Ext} + c(i,j-1) \end{cases} \qquad c(i,j) = \max \begin{cases} \texttt{Ouv} + \texttt{Ext} + a(i-1,j) \\ \texttt{Ouv} + \texttt{Ext} + b(i-1,j) \\ \texttt{Ext} + c(i-1,j) \end{cases}$$

avec $a(i,j)$ score maximal d'un alignement entre $U(1..i)$ et $V(1..j)$ qui termine par un match ou un mismatch entre $U(i)$ et $V(j)$; $b(i,j)$ score maximal d'un alignement entre $U(1..i)$ et $V(1..j)$ qui termine par l'insertion de $V(j)$ et $c(i,j)$ score maximal d'un alignement entre $U(1..i)$ et $V(1..j)$ qui termine par la délétion de $U(i)$. Les matrices étant initialisées de la manière suivante :

$$\begin{array}{lll} a(0,0) = 0 & & \\ a(i,0) = -\infty & b(i,0) = -\infty & c(i,0) = \texttt{Ouv} + \texttt{Ext} \times i \\ a(0,j) = -\infty & b(0,j) = \texttt{Ouv} + \texttt{Ext} \times j & c(0,j) = -\infty \end{array}$$

Etant donné que les formules de récurrence ne font appel qu'à des valeurs de la matrice se trouvant dans la colonne ou la ligne précédent la case en cours de calcul, il est possible pour le calcul du score de l'alignement, de ne se souvenir que de deux lignes de chaque matrice. On réduit de cette manière considérablement l'espace occupé par le calcul du score. Cependant, pour construire le chemin correspondant au score, il est nécessaire de se souvenir de l'opération effectuée pour chaque case . . .

4 Test de votre bibliothèque

Afin de pouvoir tester cette méthode, vous réaliserez également un petit programme simple de test et d'affichage du résultat. Ce programme n'entre pas dans la notation, mais vous devez quand même le rendre avec votre projet. Pour vous rendre la vie plus facile, un module de lecture de fichiers de matrices et de fichiers de séquences au format FASTA sont accessibles à partir de `http://www.lifl.fr/~varre/enseignement`.

5 Ce que vous devez rendre (pour le 12 novembre)

Une archive `.tgz` contenant :
- un fichier README
- un fichier TODO si tout n'a pas été réalisé correctement
- un Makefile
- un répertoire `src` contenant les sources
- un exécutable

Tout ce qui est développé doit être écrit et commenté en anglais.

Vous serez noté sur ...

D'abord la qualité de la mise en oeuvre de l'algorithme. Ensuite sur le comportement face aux erreurs éventuelles. Puis sur la qualité d'écriture du code. Enfin sur les respect des règles données pour rendre le projet.