Informatique *DM*
Programmation fonctionnelle avec OCaml Ipesup

Distance d'alternance

On travaille sur l'alphabet $\Sigma = \{A, B\}$, avec

```
type lettre = A | B
type mot = lettre list
```

On définit la *distance d'alternance* d_{alt} entre deux mots $u, v \in \Sigma^*$ de la façon suivante. Si $u = (u_1, \ldots, u_n)$ et $v = (v_1, \ldots, v_m)$, on aligne les deux mots à gauche et l'on pose :

$$d_{\text{alt}}(u, v) = \#\{\, 1 \leq i \leq \min(n, m) \;:\; u_i \neq v_i \,\} \;+\; |n - m|.$$

Autrement dit, on compte les **désaccords positionnels** sur le préfixe commun le plus long, puis on ajoute la **différence de longueur**.

Exemple

$$d_{\text{alt}}([A; B; A], [A; B; B]) = 1 \quad \text{(désaccord en position 3)},$$
$$d_{\text{alt}}([A; A; B], [A; B]) = 1 + |3 - 2| = 2,$$
$$d_{\text{alt}}([], [B; B; A]) = 3.$$

Questions

1. Vérifier sur des exemples supplémentaires que $d_{\text{alt}}(u, v) = 0$ si et seulement si $u = v$, et que $d_{\text{alt}}(u, v) = d_{\text{alt}}(v, u)$.
2. Montrer que d_{alt} vérifie l'inégalité triangulaire, donc définit bien une *distance*.

1 Implémentation en OCaml

Écrire une fonction OCaml `d_alt : mot -> mot -> int` calculant d_{alt} et verifier :

```
d_alt [A;B;A] [A;B;B] = 1
d_alt [A;A;B] [A;B]   = 2
d_alt []      [B;B;A] = 3
d_alt [A;B]   [A;B]   = 0
```

2 Classement de mots par distance

Étant donnés un mot cible `w : mot` et une liste `L : mot list`, trier `L` par ordre croissant de `d_alt (.) w` et afficher en sortie la liste des paires `(mot, distance)`.

Exemple.
Si `w = [A;B;A]` et `L = [[A;B;B]; [A;B]; [A;B;A]; []]`, on attend : `[1; 2 ; 0 ; 3 ]`.

3 Boule de rayon r

Écrire une fonction `boule_alt : int -> mot -> mot list -> mot list` qui renvoie tous les mots de `L` à distance au plus `r` du centre `w` pour la distance d_{alt}.

4 Plus proches voisins.

Écrire `k_ppv : int -> mot -> mot list -> mot list` qui renvoie les `k` mots de `L` les plus proches de `w` (en cas d'égalité, définissez un ordre de tri à votre convenance en ajoutant un commentaire en OCaml pour expliquer le tri que vous avez choisi).

5 Chemin d'édition élémentaire pour d_{alt}

Dans d_{alt}, la transformation minimale de u vers v peut se décrire par :

— (Substitutions) pour chaque $i \leq \min(|u|, |v|)$ tel que $u_i \neq v_i$, remplacer u_i par v_i ;
— (Extensions) si $|u| < |v|$, ajouter les lettres manquantes de v en fin ;
— (Coupures) si $|u| > |v|$, supprimer les lettres excédentaires en fin.

Écrire `script_alt : mot -> mot -> mot list` qui renvoie la *suite de mots* depuis u jusqu'à v où chaque étape correspond à une opération élémentaire ci-dessus, et où la longueur de la suite moins 1 est exactement $d_{\text{alt}}(u, v)$.

Exemple.
Si u=[A ;A ;B] e v=[A ;B], alors le chemin est $[A; A; B] \to [A; B; B] \to [A; B]$.