Dans tout le TD, si s est une chaîne de caractères, on note $s[i:j]$ la sous-chaîne de s des indices i à j (exclu), en omettant i si $i = 0$ et j si j est le dernier indice de s (notation Python).

Exercice 1. Application de Boyer-Moore
Appliquer l'algorithme de Boyer-Moore pour chercher le mot `string` dans `stupid_spring_string`, en détaillant chaque étape.

Exercice 2. Anagramme
Deux chaînes de caractères sont des **anagrammes** s'ils sont des permutations l'un de l'autre (dit autrement : ils contiennent les mêmes lettres, mais pas forcément dans le même ordre).
Quel est la meilleure complexité que vous pouvez obtenir pour savoir si deux chaînes de caractères sont des anagrammes ? L'écrire en OCaml.

Exercice 3. Expressions bien parenthésées

1. Soit s une chaîne de caractères composée uniquement de parenthèses (`(` ou `)`). Écrire une fonction pour savoir si s est bien parenthésée, en complexité linéaire. Par exemple, `(())` et `()()` sont bien parenthésées mais pas `)()` ni `(()`.
2. On suppose maintenant avoir différents types de parenthèses (par exemple, `()` et `[]`). Expliquer comment utiliser une pile pour savoir si une telle chaîne de caractères est bien parenthésée, en complexité linéaire.

Exercice 4. Distance de Levenshtein (ou : distance d'édition)
Soient s et t deux chaînes de caractères. La **distance de Levenshtein** entre s et t est le nombre minimum de caractères qu'il faut insérer, supprimer ou substituer pour passer de s à t.

1. Quelle est la distance de Levenshtein entre $s = nuit$ et $t = bruit$?
2. Soit $d[i][j]$ le nombre minimum d'insertion, suppression, substitution pour passer de $s[:i]$ à $t[:j]$.
   Donner une équation de récurrence sur $d[i][j]$, sans la prouver (une fois n'est pas coutume).
3. En déduire une fonction en C pour trouver la distance de Levenshtein entre 2 chaînes de caractères. Quelle est sa complexité?

Exercice 5. Plus grande sous-séquence commune
Si w et s sont deux chaînes de caractères, on dit que w est une **sous-séquence** de s si les lettres de w apparaissent dans l'ordre dans s (mais pas forcément consécutivement, contrairement à un sous-mot). Par exemple, `hiats` est une sous-séquence de `this is a test` (mais pas un sous-mot).
Soit s et t deux chaînes de caractères. Écrire un algorithme par programmation dynamique permettant de trouver la longueur d'une **plus grande sous-suite commune** à s et t. Par exemple, si $s =$ `this is a test` et $t =$ `another try`, il faut renvoyer 4 (correspondant à `thet` qui est la plus grande sous-séquence de s et t).

Exercice 6. Fenêtre minimum
Soient s et t deux chaînes de caractères. Écrire un algorithme donnant un sous-mot de s de longueur minimum contenant toutes les lettres de t.
Par exemple, si $s =$ `adobecodebanc` et $t =$ `abc` alors l'algorithme doit renvoyer `banc`.
Indice : on pourra utiliser deux « pointeurs ».

Exercice 7. Séparation de mots
Étant donné une chaîne de caractères s et un ensemble d de chaînes de caractères, écrire un algorithme par programmation dynamique déterminant si s peut être obtenu par concaténation d'un nombre quelconque de mots de d.
Pour utiliser un ensemble (immutable) en OCaml :

```
let module S = Set.Make(String) in
let d = S.empty in (* ensemble vide *)
let d2 = S.add "test" d in (* ajoute un élément à d et
                              renvoie le nouvel ensemble *)
S.mem "test" d2 (* détermine si un élément
                   appartient à l'ensemble *)
```

Étant implémenté par ABR équilibré, les opérations d'un `Set` OCaml sont en $O(\log(n))$. Avec une table de hachage, ce serait $O(1)$ en moyenne.