Exercice 8

Chapitre Langages

Lycée Chateaubriand

Question a)

- Il y a $10! = 3628800$ anagrammes d'un mot de 10 lettres. Cela nécessite donc

$$10! \times 10 \times 10^{-6} \simeq 36,29s$$

- Il y a $20! \simeq 2,43 \times 10^{18}$ anagrammes d'un mot de 20 lettres. Cela nécessite donc

$$20! \times 20 \times 10^{-6} \simeq 48658040163532,8s = 1542936 \text{ années}$$

Question b)

Pour détecter si deux mots sont des anagrammes, il suffit de comparer les occurrences de chaque lettre dans chaque mot. On peut, par exemple, les stocker dans un tableau de 26 cases.

```
let detecteAnagramme m n =
  let occ = Array.make 26 0 in
  let lm = String.length m and ln = String.length n in
  let b = ref (ln=lm) in
  if !b then
    (for i = 0 to ln-1 do
       let ind = int_of_char(m.[i])-97 in
       occ.(ind) <- occ.(ind) +1
     done);
  let j = ref 0 in
  while !b && !j<ln-1 do
    let ind = int_of_char(m.[!j])-97 in
    occ.(ind) <- occ.(ind) -1;
    b := occ.(ind) >= 0;
    incr j
  done;
  !b;;
```

Question c)

On commence par écrire une fonction `ajoute_lettre : char -> string -> string list` telle que `ajoute_lettre c mot` renvoie la liste de tous les mots que l'on obtient en insérant la lettre `c` à une position dans le mot `mot`.

On va utiliser les deux fonctions suivantes :

- La fonction `String.sub` telle que `String.sub mot i k` renvoie la sous-chaine de `mot` qui commence au caractère d'indice i et qui est de longueur k.
- La fonction `Char.escaped` qui transforme un caractère en une chaine de longueur 1.

```
let ajoute_lettre c mot =
  let rec parcours i =
    if i > String.length mot then []
    else (String.sub mot 0 i)^Char.escaped(c)^(String.sub mot i (n-i))
                :: (parcours (i+1))
  in parcours 0;;
```

Question c)

On va maintenant *généraliser* la fonction précédente, en une fonction `ajoute_lettre_list : char -> string list-> string list` telle que `ajoute_lettre c lmot` renvoie la liste obtenue en concaténant les listes obtenues par la fonction précédente pour chaque mot de la liste `lmot`

```
let rec ajoute_lettre_list c lmot = match lmot with
  | [] -> []
  | mot :: q -> (ajoute_lettre c mot) @ ( ajoute_lettre_list c q);;
```

Question c)

On peut ensuite écrire notre fonction récursive. Pour calculer les anagrammes d'un mot `mot` ayant au moins une lettre, on sépare le mot en considérant sa première lettre `c` et le reste du mot queue. Il suffit alors de calculer tous les anagrammes de queue et d'y ajouter la lettre `c` par la fonction précédente.

```
let rec anagramme mot =
  let n = String.length mot in
  if n = 0 then []
  else ajoute_lettre_list mot.[0] (anagramme (String.sub mot 1 (n-1)));;
```