EPU GB 4ème année BIMB — *Algorithmique pour la biologie*
Examen du 19 mai 2021 — Documents autorisés.

Inscrivez **lisiblement** vos Nom et Prénom en tête de vos copies.

Exercice 1 : (**Programmation dynamique – [7 points]**)

Soient 2 séquences A et B, de longueurs n et m. Nous nous proposons de calculer la plus longue sous-séquence M commune à ces 2 séquences A et B, c'est à dire la plus grande suite de lettres qui apparaissent dans le même ordre dans les 2 séquences. **Exemple :** `ACE` est la plus longue sous-séquence de `ABCDEFG` et de `DEAACAEA`). Pour cela, on considère $L(i, j)$ la longueur de la plus longue sous-séquence commune aux préfixes `A[1:i]` et `B[1:j]` pour tout $i \leq n$ et tout $j \leq m$.

1. En vous aidant de l'algorithme de programmation dynamique pour l'alignement de séquences, proposez, dans le cas où i et j sont strictement positifs, une récurrence exprimant $L(i, j)$ en fonction de certains $L(k, l)$. *Indication :* deux cas sont à considérer.
2. Donner les cas de base.
3. Ecrire en Python un algorithme de programmation dynamique calculant la longueur de la plus grande sous séquence commune.
4. Donner la complexité en temps et en mémoire de votre algorithme.
5. Pour en extraire la plus longue sous séquence commune, est-il nécessaire de programmer une phase de remontée ? Justifiez votre réponse. Si oui, expliquer l'algorithme qui permettrait de reconstruire la plus longue sous-séquence commune.

Exercice 2 : (**Algorithme Online pour la recherche de palindromes – [13 points]**)

Supposons que les lettres successives d'une séquence nous soient révélées lettre par lettre. Le but de l'exercice est de concevoir un algorithme qui, à chaque fois qu'une lettre nous arrive, teste de manière efficace si la séquence reçue depuis le début est un palindrome.

Exemples :

```
Entrée: "atcta"
Sortie: a Yes   % "a" est un palindrome
        t No    % "at" n'est pas un palindrome
        c No    % "atc" n'est pas un palindrome
        t No    % "atct" n'est pas un palindrome
        a Yes   % "atcta" est un palindrome
```

```
Entrée:  "aataacaataa"
Sortie:  a Yes   % "a" est un palindrome
         a Yes   % "aa" est un palindrome
         t No    % "aat" n'est pas un palindrome
         a No    % "aata" n'est pas un palindrome
         a Yes   % "aataa" est un palindrome
         c No    % "aataac" n'est pas un palindrome
         a No    % "aataaca" n'est pas un palindrome
         a No    % "aataacaa" n'est pas un palindrome
         t No    % "aataacaat" n'est pas un palindrome
         a No    % "aataacaata" n'est pas un palindrome
         a Yes   % "aataacaataa" est un palindrome
```

Considérons que la chaîne reçue à l'étape i est `str[0..i-1]`. La solution la plus simple serait de tester à chaque étape si `str[0..i-1]` est un palindrome. Cette solution nécessite à chaque étape de reparcourir le début de la chaîne.

Une meilleure solution est d'utiliser une fonction de codage, un peu comme dans l'algorithme de Rabin-Karp. L'idée est de coder, pour chaque chaîne `str[0..i-1]`, la première moitié de la chaîne (qu'on lit de droite à gauche) ainsi que la deuxième moitié (qu'on lit de gauche à droite). On ne teste si c'est un palindrome que lorsque la fonction de codage donne le même résultat pour les deux sous-chaînes. Voici l'algorithme complet :

1. Pour une chaîne de longueur $i = 1$, `str[0..i-1]` est toujours un palindrome.
2. Pour une chaîne de longueur $i = 2$, soit `str[0..i-1]`= "`ab`", initialiser la chaîne qui représente la première moitié de la chaîne (qu'on lit de droite à gauche) à "`a`", et initialiser la chaîne qui représente la deuxième moitié (qu'on lit de gauche à droite) à "`b`". Soient `code1` et `code2` les valeurs de codage de ces 2 chaînes.
3. Itérer pour une longueur $i \geq 2$:
   (a) si `code1` et `code2` sont égaux, tester caractère par caractère si les 2 chaînes sont les mêmes. Noter que l'égalité des valeurs de codage n'implique pas l'égalité des chaînes.

(b) Mettre à jour `code1` et `code2` pour la prochaîne itération :
   — si i est impair, ajouter le caractère du milieu de la chaîne au début de la première chaîne, et ajouter le nouveau caractère à la fin de la seconde chaîne. Mettre à jour les valeur de codage des 2 chaînes.
   — si i est pair, ne modifier que la seconde chaîne de caractères : supprimer le premier caractère et ajouter le nouveau caractère à la fin.

Déroulement de l'algorithme pour la chaîne "`abcba`"

— Le préfixe de longueur 1, "`a`", est un palindrome.
— Valeurs initiales de `code1` et `code2` : `code1` = `codage`('a'), `code2` = `codage`('b')
— Commencer la boucle pour une longueur de préfixe égale à 2, *i.e.* $i = 2$
   1. Comparer `code1` et `code2`, ils diffèrent, ce n'est pas un palindrome.
   2. Calculer les nouvelles valeurs de codage. Puisque i est pair, `code1` ne change pas et `code2` devient `codage`('c').

— i = 3
   1. Comparer `code1` et `code2`, ils diffèrent, ce n'est pas un palindrome.
   2. Calculer les nouvelles valeurs de codage. Puisque i est impair, `code1` devient `codage`('ba') et `code2` devient `codage`('cb').

— i = 4
   1. Comparer `code1` et `code2`, ils diffèrent, ce n'est pas un palindrome.
   2. Calculer les nouvelles valeurs de codage. Puisque i est pair, `code1` ne change pas et `code2` devient `codage`('ba')

— i = 5
   1. Comparer `code1` et `code2`, ils sont égaux, comparer lettre à lettre, c'est un palindrome.
   2. Comme c'est le dernier indice, on ne met pas à jours les différentes variables.

L'idée d'utiliser une fonction de codage permet de calculer les nouvelles valeurs de codage des deux demi-chaînes à partir de leurs valeurs d'avant en $O(1)$, c'est-à-dire en temps constant (indépendant de la longueur des chaînes).

Les différentes questions suivantes permettent d'implémenter cet algorithme.

1. **[1 point]** Écrire en python une fonction `codage(l)` qui prend en argument une lettre et renvoie un entier qui code cette lettre. Pour simplifier, on ne considèrera que l'alphabet {`'a','t','g','c'`}. La taille de l'alphabet est donc $d = 4$ et on notera q le nombre premier qui servira à calculer les modulos.
2. **[1 point]** Écrire une fonction `chainesEgales(c1,c2)` qui prend 2 chaînes de caractères et qui renvoie un booléen qui dit si les chaînes sont égales. On ne veut utiliser que la comparaison lettre à lettre.
3. **[1 point]** Quelle est la formule pour passer de `codage(ch)` à `codage(ch+l)` où `l` est une lettre que l'on ajoute à la fin de la chaîne `ch` ? Cette formule est à utiliser pour la mise à jour de `code2`.
4. **[2 points]** Quelle est la formule pour passer de `codage(ch)` à `codage(ch[1:])` où `ch[0]` est la lettre que l'on enlève au début de la chaîne `ch` ? Cette formule est à utiliser pour la mise à jour de `code2`. *Attention :* cette formule dépend de la longueur de la chaîne `ch`.
5. **[2 points]** Quelle est la formule pour passer de `codage(ch)` à `codage(l+ch)` où `l` est une lettre que l'on ajoute au début de la chaîne `ch` ? Cette formule est à utiliser pour la mise à jour de `code1`. *Attention :* cette formule dépend de la longueur de la chaîne `ch`.
6. **[6 points]** Écrire en python une fonction `quelsPrefixesSontDesPalindromes(ch)` qui prend en argument une chaîne et qui, pour chaque préfixe `ch[0:i]`, dit s'il s'agit d'un palindrome ou non.