Corrigé

Longest repeated substring

Recherche par force brute

Question 1. Dans la fonction qui suit, on permute si nécessaire i et j pour avoir $i \leqslant j$ de sorte que le plus petit des deux suffixes est $s[j:]$.

```
def prefixe(s, i, j):
    k = 0
    if j < i:
        i, j = j, i
    while j + k < len(s) and s[i+k] == s[j+k]:
        k += 1
    return k
```

La recherche par force brute consiste alors à tester tous les couples (i, j) de suffixes distincts possibles :

```
def lrs1(s):
    n = len(s)
    u, m = n, 0
    for i in range(n-1):
        for j in range(i+1, n):
            k = prefixe(s, i, j)
            if k > m:
                u, m = i, k
    return s[u:u+m]
```

Le coût de la fonction prefixe est un $O(k)$, k désignant la longueur maximale des facteurs présents au moins deux fois dans s. Il en résulte immédiatement que le coût de la fonction `lrs1` est un $O(kn^2)$.
La fonction suivante va nous permettre de calculer facilement la durée d'exécution de cette fonction ainsi que de la suivante.

```
def test(fichier, methode):
    f = open(fichier, 'r')
    l = f.readline()
    f.close()
    d = -time()
    s = methode(l)
    d += time()
    print("lrs = {}, durée = {}.".format(s, d))
```

On obtient ensuite :

```
>>> test('essai1000.txt', lrs1)
lrs = 'ggcctagcct', durée = 0.5785138607025146.

>>> test('essai10000.txt', lrs1)
lrs = 'acccgtcaggtgt', durée = 59.04449510574341.
```

Comme on peut le constater, la longueur du plus long facteur présent deux fois croît assez faiblement avec n, ce qui va nous permettre de considérer l'entier k comme « presque constant » vis-à-vis de n. Avec cette hypothèse, la complexité est quadratique en n et on peut estimer le temps d'exécution de la fonction `lrs1` sur le fichier `essai100000.txt` à environ 6 000 secondes, soit 1h40.

Tableau des suffixes

Question 2. On observe qu'un facteur présent deux fois dans s est préfixe de deux suffixes de s *consécutifs* dans le tableau trié des suffixes. La recherche du plus long facteur se résume donc à appliquer la fonction `prefixe` à deux suffixes voisins de t :

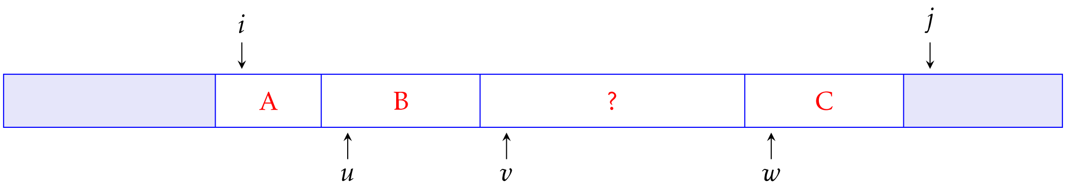
```
def lrs2(s, t):
    n = len(s)
    u, m = n, 0
    for i in range(n-1):
        k = prefixe(s, t[i], t[i+1])
        if k > m:
            u, m = t[i], k
    return s[u:u+m]
```

Le coût devient dès lors un $O(kn)$.

Calcul du tableau des suffixes

On notera que le tri *3-Way radix quicksort* utilisé ici est récent (il date de 1997) et est à l'heure actuelle le plus rapide pour trier des chaînes de caractères.

Question 3. La segmentation respecte l'invariant suivant :



$\forall x \in A$, la $(k+1)^{e}$ lettre de $s[x:]$ est strictement inférieure au pivot p ;
$\forall x \in B$, la $(k+1)^{e}$ lettre de $s[x:]$ est égale au pivot p ;
$\forall x \in C$, la $(k+1)^{e}$ lettre de $s[x:]$ est strictement supérieure au pivot p.

```
def segmente(s, t, i, j, k):
    u, v, w = i, i + 1, j
    if t[i] + k < len(s):
        p = s[t[i]+k]
    else:
        p = ' '
    while v < w:
        if t[v]+k >= len(s) or s[t[v]+k] < p:
            t[u], t[v] = t[v], t[u]
            u += 1
            v += 1
        elif s[t[v]+k] == p:
            v += 1
        else:
            w -= 1
            t[v], t[w] = t[w], t[v]
    return u, v
```

Question 4. Le tri adopte une démarche naturellement récursive :

```
def quick3way(s, t, *args):
    if len(args) == 0:
        i, j, k = 0, len(t), 0
    else:
        i, j, k = args
    if i + 1 < j:
        u, v = segmente(s, t, i, j, k)
        quick3way(s, t, i, u, k)
        quick3way(s, t, u, v, k+1)
        quick3way(s, t, v, j, k)
```

Il reste à appliquer ce tri au tableau non trié des suffixes de s :

```python
def suffixes(s):
    t = [i for i in range(len(s))]
    quick3way(s, t)
    return t
```

La fonction résolvant le problème LRS s'écrit enfin :

```python
def lrs(s):
    return lrs2(s, suffixes(s))
```

Il reste à la tester sur nos trois fichiers exemples :

```
>>> test('essai1000.txt', lrs)
lrs = 'ggcctagcct', durée = 0.019194841384887695.

>>> test('essai10000.txt', lrs)
lrs = 'acccgtcaggtgt', durée = 0.23460102081298828.

>>> test('essai100000.txt', lrs)
lrs = 'ccaggtgagcgctcca', durée = 2.8753879070281982.
```