Mardi 12 décembre 2022. MP 2023/2024.

T.P. d'informatique n° 13

Plus longue sous-suite strictement croissante

Problématique :

À partir d'une liste d'entiers ou de réels $(S_i)_{i \in [\![0,n-1]\!]}$, il s'agit de déterminer la plus longue sous-suite d'éléments strictement croissante issue de S.

Pour tester les fonctions, on rappelle que l'on peut générer une liste de N entiers compris entre 0 et N avec le code suivant :

```
import random
N=20
S=[random.randint(0,N) for i in range(N)]
>>> S = [8, 5, 3, 4, 9, 5, 19, 9, 18, 2, 20, 6, 3, 18, 5, 10, 9, 16, 1, 14]
```

Utilisation de l'algorithme du TP11

Si plusieurs sous-séquences possèdent la taille la plus longue, l'algorithme peut retourner n'importe laquelle.

On note `S` la séquence d'entrée complète (de `n` éléments indicés de `0` à `n-1`), `S[i]` son (i+1)-élément et `S[0:i+1]` son (i + 1)-ème préfixe (c'est-à-dire la séquence constituée seulement des i+1 premiers éléments de `S[0]` à `S[i]`).

Une première idée consiste à se ramener à l'algorithme de recherche de la plus longue sous-séquence commune à deux séquences (cf TP 11) ; on utilisera le même algorithme adapté à une liste à la place d'une chaîne de caractères : voir le fichier TP13ini.py.

La recherche de la plus longue sous-séquence strictement croissante peut alors s'écrire ainsi :

```
def plusLongueSousSequenceStrictementCroissante(S):
    return plssc(S, supprimeMultiples(triCroissant(S)))
```

où `triCroissant` permet de trier une liste par ordre croissant et `supprimeMultiples` transforme une liste d'éléments en ne gardant q'une fois ceux qui apparaissent successivement plusieurs fois (ce qui nous permettra ici de transformer une séquence croissante en une séquence strictement croissante de même image).

Question 1 : Donner le nom d'un algorithme de tri qui pourrait être utilisé ici et indiquez la complexité de cet algorithme de tri en fonction de la longueur de la liste passée en argument. Écrire la fonction Python `triCroissant` correspondant ; *attention, un tri classique modifie son argument, or on veut ici conserver la séquence S, et elle ne doit donc pas être modifiée par votre fonction.*

Question 2 : Écrire la fonction Python `supprimeMultiples` en prenant bien soin qu'elle soit de complexité linéaire.

Question 3 : En déduire la complexité de `plusLongueSousSequenceStrictementCroissante` pour l'algorithme de tri que vous avez choisi en fonction de `n = len(S)`.

On rappelle que pour deux chaines de longueurs respectives m et n, l'algorithme de recherche de la plus longue sous-suite commune est de complexité $m.n$

Testez votre hypothèse à l'aide de la fonction `time` de la bibliothèque `time` (`time.time()` renvoie une date en secondes avec initiation au 1^{er} janvier 1970 à 00h00).

Vous détaillerez les résultats validant votre hypothèse quant à la complexité de cet algorithme.

Programmation dynamique

Pour `k < n`, notons `L[k]` la longueur de la plus longue sous-séquence strictement croissante de `S[0:k+1]` qui se termine par `S[k]`. On a bien sûr `L[0] = 1`. Selon l'ordre de `S[0]` et `S[1]`, la valeur de `L[1]` sera soit `2=L[0]+1` (lorsque `S[0] < S[1]`), soit 1 (lorsque `S[1]` $\leqslant$ `S[0]`).

Question 4 : Donner la relation de récurrence qui exprime `L[k]` en fonction des termes précédents de `L`, et des éléments de `S`.

Question 5 : Une fois que les `L[0],L[1], ...,L[n - 1]` ont été calculés, comment peut-on trouver la taille de la plus longue sous-séquence strictement croissante de `S` ? (Cette plus longue sous-séquence ne se termine pas forcément par `S[n - 1]`).

Question 6 : Déterminer la complexité du calcul de la taille de la plus longue sous-séquence strictement croissante d'une séquence de `n` éléments, avec un algorithme de programmation dynamique basé sur les remarques précédentes.

Question 7 : Écrire une fonction Python qui implémente cet algorithme de programmation dynamique.

Question 8 : Adapter la fonction précédente pour retourner la sous-séquence strictement croissante elle-même : le principe, dit de **mémoïsation**, consiste à mémoriser dans une liste `sol` des solutions correspondants aux longueurs de `L` ; ainsi, au moment du calcul de `L[k]`, on enregistre dans `sol[k]` une plus longue sous-séquence strictement croissante de `S` de longueur `L[k]` se terminant par `S[k]`.