DS 2 ITC : solution du pb. pour les Options Info CCP MP 2019

Q1 Les couples présentant une inversion pour σ sont $(0,2)$, $(1,2),(1,3),(1,5),(4,5)$, et ainsi la permutation σ présente au total 5 inversions.

Q2 On peut utiliser un `range` décroissant : avec la commande `range(len(L)-1,0,-1)` le dernier argument donne le pas de –1 l'indice du range va prendre les valeurs `len(L)-1,len(L)-2,..,1` la valeur `0` étant la première valeur non prise.

```
def tri_bulle(L) :
    for i in range(len(L)-1, 0, -1) :
        for j in range(len(L)-1, len(L)-i-1, -1) :
            if L[j] < L[j-1] :
                L[j], L[j-1] = L[j-1], L[j]
```

A défaut du `range` décroissant, on peut poser $i = n - p$ et $j = n - q$:

- on veut que i aille de $n-1$ à 1 pris donc p va de 1 à $n-1$ pris donc `p in range(1,n)`
- on veut que j aille de $n-1$ à $n-i$ pris donc q de 1 à i pris donc `q in range(1,i+1)`

```
def tri_bulle2(L):
    n=len(L)
    for p in range(1,n): # i=n-p
        i=n-p
        for q in range(1,i+1):
            j=n-q
            if L[j] < L[j-1] :
                L[j], L[j-1] = L[j-1], L[j]
```

Q3 L'algorithme échange $L[j]$ et $L[j-1]$ lorsque $L[j] < L[j-1]$: cela supprime une inversion.
Montrons qu'on en n'a pas ainsi créé d'autre.
Si le couple $(i, j-1)$ avec $i < j-1$ est une inversion de σ avant l'application de cette étape, alors après cette étape c'est le couple (i,j) et de même si le couple (i,j) avec $i < j$ était une inversion, après cette étape c'est le couple $(i, j-1)$ mais cela ne change pas le *nombre* de couples présentant une inversion de ce type.
De même avec les couples $(j-1,k)$ et (j,k) pour $k > j$.

Q4 Il suffit de rajouter un compteur pour chaque étape où le tri bulle supprime une inversion. Autrement dit :

```
def nombre_inversions(L) :
    compteur = 0
    for i in range(len(L)-1, 0, -1) :
        for j in range(len(L)-1, len(L)-i-1, -1) :
            if L[j] < L[j-1] :
                L[j], L[j-1] = L[j-1], L[j]
                compteur = compteur + 1
    return compteur
```

Q5 Ici $Tab_\sigma[0] = 1$ car il y un un seul indice $j > 0$ tel que $\sigma_j < \sigma_0$ qui est $j = 2$. On calcule de même les autres entrées de Tab_σ et on trouve que $\boxed{Tab_\sigma = [1,3,0,0,1,0].}$

Q6 Soit σ une permutation. Pour chaque $i \in [\![0, n-2]\!]$, α_i est par déf. le cardinal d'un sous-ensemble de $[\![i+1, n-1\,]\!]$ donc d'un ensemble à $n-1-(i+1)+1 = n-i-1$ éléments. Noter que le α_{n-1} défini par l'énoncé est forcément nul, car l'ensemble d'indice considéré est vide
Donc $\alpha_i \in [\![0, n-i-1]\!]$ i.e. $(\alpha_0, \ldots, \alpha_{n-1}) \in \prod_{i=0}^{n-1} [\![0, n-i-1]\!] = \prod_{k=1}^{n} [\![0, n-k]\!]$.

Cette question devait payer le soin à cause du décalage d'indice d'un cran p.r. à la page précédente

Q7 On utilisera ici plutôt la notation fonctionnelle $\sigma(0) = \sigma_0$ et plus généralement $\sigma(j) = \sigma_j$.
Par déf.

$$\begin{aligned} Tab_\sigma[0] = \mathrm{Card}(\{j \in [\![1, n-1]\!],\ \sigma(j) < \sigma(0)\} &= \mathrm{Card}(\{j \in [\![1, n-1]\!],\ \sigma(j) \in [\![0, \sigma_0[\![\} \\ &= \mathrm{Card}\,\sigma^{-1}([\![0, \sigma_0\ [\![) \end{aligned}$$

Or comme σ est *bijective*, on sait que $\mathrm{Card}\,\sigma^{-1}([\![0, \sigma_0\ [\![) = \mathrm{Card}[\![0, \sigma_0\ [\![$
Bien sûr $\mathrm{Card}[\![0, \sigma_0\ [\![= \sigma_0$, d'où l'égalité $\boxed{Tab_\sigma[0] = \sigma_0}$.

Q8 Je ne sais pas ce que les concepteurs du sujet avaient en tête mais la question précédente n'aide pas beaucoup pour celle-ci, me semble-t-il, en tout cas pour démarrer une récurrence comme on pourrait s'y attendre.

(M1) Il aurait plus intéressant de faire remarquer que $\sigma^{-1}(0)$ est le plus petit indice k tel que $\alpha_k = 0$, ce qui permet de débuter une récurrence ce qui ferait une (M1) que je ne détaille pas car je la trouve fastidieuse.

(M2) La méthode suivante est exposée par D. Knuth dans la bible *The Art of Computer Programming*, en citant un article de Marshall de 1956. Nous allons expliquer suivant Knuth [1] comment passer d'un table $\mathtt{T} = [\alpha_0, \ldots, \alpha_{\mathtt{n}-1}]$ à la permutation σ telle que $T = Tab_\sigma$ en suivant l'exemple de l'énoncé.

On considère $T = [1, 3, 0, 0, 1, 0]$.

• On commence par écrire σ_5 tout seul

• Comme $\alpha_4 = 1$, on sait que le couple $(4, 5)$ est inversé, donc on écrit σ_4 après σ_5, on a : σ_5, σ_4.

• Comme $\alpha_3 = 0$, les couples $(3, j)$ avec $j > 3$ ne sont pas inversés donc on écrit σ_3 à gauche des σ_5, σ_4 : on a $\sigma_3, \sigma_5, \sigma_4,$.

• Comme $\alpha_2 = 0$, de même on écrit σ_2 à gauche, on a $\sigma_2, \sigma_3\sigma_5, \sigma_4$.

• Comme $\alpha_1 = 3$, 1 est inversé avec trois de ses successeurs, donc on intercale σ_1 après la troisième place : $\sigma_2, \sigma_3, \sigma_5, \sigma_1, \sigma_4$.

• Comme $\alpha_0 = 1$, 0 est inversé avec un de ses successeurs, donc on intercale σ_0 après la première place : on obtient $\sigma_2, \sigma_0, \sigma_3, \sigma_5, \sigma_1, \sigma_4$.

Cette suite ordonnée dit que $\sigma^{-1} = [2, 0, 3, 5, 1, 4]$, il reste alors quand même à inverser la permutation (cf. Q10).

Cette méthode, qui s'applique au cas général, donne l'unicité de l'antécédent $\sigma \in S_n$ d'un élément T de E_n. Comme $\mathfrak{S}_n$ et E_n ont même cardinal, on est sûr que cet antécédent convient et que l'application Tab est bijective.

Q9 Pas de difficulté ici :

```
def permutation_vers_table(L) :
    n = len(L)
    Tab = [0] * n
    for i in range(0, n) :
        for j in range(i+1, n) :
            if L[j] < L[i] :
                Tab[i] = Tab[i] + 1
    return Tab
```

Q10 On implémente l'idée décrite à la Q8. La première fonction renvoie la liste $[\sigma^{-1}(0), \ldots, \sigma^{-1}(n-1)]$ la seconde inverse une permutation. Il suffit alors de combiner ces fonctions.

1. même si les notations de Knuth sont différentes car Knuth appelle inversion les couples $(\sigma(i), \sigma(j))$ au lieu de (i, j) ce que fait que ses tables d'inversions sont celles de σ^{-1}

```
def table_vers_sigma_inv(T):
    n=len(T)
    L=[n-1]# on place le dernier élément
    for i in range(len(T)-2,-1,-1):
        L[T[i]:T[i]]=[i]# on met i à la T[i]-ième place.
    return L

def inverse(L):
    S=[0]*len(L)
    for i in range(len(L)):
        j=0
        while L[j]!=i:
            j=j+1
        S[i]=j
    return S

def table_vers_permutation(T):
    return inverse(table_vers_sigma_inv(T))
```

Extrait du rapport de l'épreuve CCINP 2019

PARTIE 1

Question 1, 2 : pas de problème particulier (hormis les problèmes de range).

Question 3 : de nombreuses démonstrations étaient incomplètes : on se contentait d'expliquer que lorsque l'échange a lieu entre L[i] et L[i+1], on enlève l'inversion (i,i+1). Pourtant, d'autres inversions apparaissent également : si l'inversion (i,j) avec j>i+1 est présente avant échange, après échange on obtient l'inversion (i+1,j).

Question 4, 5, 6, 7 : pas de problème particulier (hormis les problèmes de range).

Question 8 : certains parlent de noyau ou de dimension alors que ce n'est pas une application linéaire. Si on effectue une démonstration par récurrence, il est important de bien définir la proposition P que l'on veut démontrer ainsi.

Question 9 : pas de problème particulier.

Question 10 : la construction de la réciproque de la fonction Tab n'est pas toujours comprise et donc la question n'a pas toujours été bien traitée.