CC : Tri de Multiplets

En plus des 2 heures que durent l'épreuve, le site où déposer votre copie vous donne 15 minutes supplémentaires pour prendre en compte le temps de télécharger l'énoncé, de scanner votre copie, et de la déposer sur le site web.

Vous ne devez utiliser l'ordinateur que pour télécharger puis afficher l'énoncé et pour déposer votre copie. En particulier il vous est demandé de ne pas utiliser python pendant l'épreuve. Lorsqu'un programme en python est demandé, l'important n'est pas de savoir s'il s'exécute correctement sans erreur de syntaxe, mais si les opérations effectuées répondent à la question.

Pour la rédaction de votre copie, gardez bien en tête que l'important est de convaincre l'examinateur que vous avez compris pourquoi le résultat est correct (il arrive qu'écrire une demi ligne suffise pour cela, mais le plus souvent écrire uniquement le résultat ne suffit pas).

I. Tri d'une liste

Soit $n \in \mathbb{N}^*$. On désigne par $\mathcal{S}_n$ l'ensemble des permutations de $[\![0,n[\![$.

On considère un nuplet $(x_0,x_1,\dots x_{n-1}) \in \mathbb{R}^n$ dont les éléments dont deux à deux distincts.

1. Montrer que
$$\exists \sigma \in \mathcal{S}_n : \forall j \in [\![0,n[\![, \forall i \in [\![0,j[\![: x_{\sigma(i)} \leqslant x_{\sigma(j)}$$

Trier un n-uplet consiste à lui associer le n-uplet $(x_{\sigma(0)},x_{\sigma(1)},\dots,x_{\sigma(n-1)})$ où σ est la permutation définie par cette propriété.

2. Justifier que si les éléments du n-uplet ne sont pas deux à deux distincts, il est néanmoins possible de le trier.

Permutations et nuplets

En python, une permutation $\sigma \in \mathcal{S}_n$ est donnée sour la forme de le liste `[`σ_0`, `σ_1`, ..., `σ_{n-1}`]`.

On rapelle que les "transpositions" sont des cas particuliers de permutations, qui échangent deux valeurs. Par exemple, si n = 5, la permutation qui échange 0 et 3 est la permutation `[3,1,2,0,4]`.

Les nuplets sont des `tuple`s en python, et ils se manipulent "presque comme des listes" : si `x` et `y` sont des tuples, alors `x[0]` est le premier élément de `x` et `x+y` est la concaténation de `x` et `y`. En revanche, il n'est possible d'utiliser "`append`" ou de changer une valeur avec `x[1]=...`.

Si vous n'êtes pas à l'aise avec les `tuple`s, vous pouvez toujours utiliser les listes à la place sur votre copie.

II. Recherche de la permutation σ

Une des façons de trier une liste consiste donc à essayer toutes les permutations et choisir celle qui satisfait la propriété de la question précente.

On écrit alors le code suivant :

```
def tri(nuplet):
    l=liste_permutations(len(nuplet))
    for p in l:
        if convient(p,nuplet): return triee(p,nuplet)
```

ce code nécessite d'avoir préalablement défini les fonctions `liste_permutations`, `triee` et `convient` définies par :

- `liste_permutations(n)` renvoit la liste des permutations de $[\![0,n[\![$ (dans un ordre arbitraire). Par exemple, `liste_permutations(3)` renvoit `[[0, 1, 2], [1, 0, 2], [2, 0, 1], [0, 2, 1], [1, 2, 0], [2, 1, 0]]` (ou bien les mêmes permutations dans un autre ordre).
- Étant donnés un nuplet x et une permutation σ, `triee(`σ`,`x`)` renvoit le nuplet $(x_{\sigma(0)},x_{\sigma(1)},\dots,x_{\sigma(n-1)})$.
- Étant donnés un nuplet x et une permutation σ, `convient(`σ`,`x`)` renvoit `True` si $\forall j\in[\![0,n[\![, \forall i\in[\![0,j[\![: x_{\sigma(i)}\leqslant x_{\sigma(j)}$ et `False` sinon.

On se focalise ici sur la fonction `liste_permutations`, que l'on définit ci-dessous :

```
def liste_permutations(n):
    liste_perms=[list(range(n))]
    trouvee=True
    while trouvee:
        derniere_permutation=liste_perms[-1]
        trouvee=False
        for i in range(1,n):
            if trouvee: break
            for j in range(0,i):
                p=[  derniere_permutation[j] if k==i else
                    (derniere_permutation[i] if k==j else
                     derniere_permutation[k])
                    for k in range(len(derniere_permutation))]
                if all(perm!=p for perm in liste_perms):
                    liste_perms.append(p)
                    trouvee=True
                    break
    return liste_perms
```

Ce programme s'appuie sur l'idée suivante :

- On considère les permutations une par une (en commencant par `[0, 1, ..., n-1]`), et on les enregistre dans `liste_perms`.
- Pour trouver la permutation suivante, on essaie de la composer par n'importe quelle transposition (ie échanger le `i`ème élément avec le `j`ème, comme le fait la commande `p=[derniere_permutation[j] if k==i ... for k in range(len(derniere_permutation))]`).
  - Si une permutation produite de cette façon n'est pas encore dans la liste des permutations déjà trouvées, alors on l'ajoute à la liste et on recommence à partir de cette nouvelle premutation.
  - Sinon, on s'arrete là, on cesse d'ajouter de nouvelles permutations à la liste.

3. Démontrer que tel qu'il est écrit, cet algorithme trouve bien toutes les permutations $\sigma\in\mathcal{S}_n$. Vous pourrez par exemple écrire une preuve par récurrence sur $n\in\mathbb{N}^*$.

4. Expliquez quel est le rôle de la variable `trouvee` et en quoi elle permet le bon fonctionnement de cette fonction.

On ne demande pas d'implémenter les fonctions `triee` et `convient`, mais on supposera pour la suite que l'on sait les définir.

5. Éstimer le temps de calcul pour trier un nuplet (en fonction de sa taille n). Cette méthodes est-elle plus rapide ou plus lente que celles vues en TD (dont le temps de calcul était n^2 et $n\log(n)$) ?