EA4 – Éléments d'algorithmique
TD n° 5 : permutations – tri par fusion

Exercice 1 : permutations (décomposition en cycles, produit, inverse, puissances)

On considère les permutations suivantes :

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 5 & 9 & 7 & 2 & 1 & 3 & 10 & 8 & 4 & 6 \end{pmatrix} \text{ et } \tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 5 & 8 & 4 & 2 & 7 & 6 & 3 & 9 & 10 \end{pmatrix}.$$

1. Écrire σ et τ en notation cyclique (c'est-à-dire comme produits de cycles disjoints).

2. Calculer $\sigma\tau$ et $\tau\sigma$.

3. Calculer σ^{-1}, τ^{-1} et $(\sigma\tau)^{-1}$.

4. Calculer σ^{2022}.

Exercice 2 : permutations (décomposition en produit de transpositions, tri)

On appelle *transposition* une permutation ayant un unique cycle de longueur 2 (et donc $n-2$ points fixes).

1. Si τ est une transposition, quelle est son inverse τ^{-1} ?

Toute permutation peut être décomposée comme produit de transpositions de plusieurs façons.

2. *Une première méthode.*

- Comment décomposer un cycle $(a_1\ a_2\ a_3)$ de longueur 3 comme produit de transpositions ?
- Généraliser pour un cycle de longueur quelconque.
- Utiliser ce résultat pour calculer une décomposition en produit de transpositions de la permutation σ de l'exercice précédent.

3. *Une autre méthode.*

- Soit σ une permutation quelconque et soit la transposition $(1\ \ \sigma^{-1}(1))$ (où la valeur $\sigma^{-1}(1)$ est bien entendu la position de l'entier 1 dans le mot représentant σ), que peut-on dire de la permutation $\sigma' = \sigma \cdot (1\ \ \sigma^{-1}(1))$?
- Que devient l'identité $\sigma' = \sigma \cdot (1\ \ \sigma^{-1}(1))$ si on multiplie ses deux membres par $(1\ \ \sigma^{-1}(1))$ à droite ?
- Utiliser ce résultat pour calculer itérativement une décomposition d'une permutation comme produit de transpositions et effectuez ce calcul sur la permutation σ de l'exercice précédent.
- Quel algorithme de tri suivrait le même procédé pour trier le tableau qui représente la permutation ?

Exercice 3 : permutations (encodage)

On peut modéliser les permutations par des tableaux d'entiers : un tableau `T` de longueur n représente une permutation si et seulement s'il contient tous les entiers compris entre 1 et n (nécessairement exactement une fois chacun).

1. Écrire une fonction `estPerm` qui prend en paramètre un tableau d'entiers `T` et qui renvoie `True` si `T` est une permutation et `False` sinon.

2. Écrire une fonction `inversePerm` qui prend en paramètre un tableau d'entiers `T` représentant une permutation et qui renvoie l'inverse de cette permutation. Si `T` n'est pas une permutation, la fonction doit renvoyer `None`. Vérifier ou modifier la fonction, de manière à s'assurer que la vérification se fait en temps linéaire.
3. Écrire une fonction `composePerm` qui prend en paramètre deux tableaux d'entiers `T1` et `T2` représentant deux permutations de même taille, et qui renvoie la composée de ces deux permutations. Si les deux permutations ne sont pas de même taille ou si l'un des deux tableaux n'est pas une permutation, la fonction doit renvoyer `None`.
4. (S'il reste du temps ou à faire à la maison.) Écrire une fonction `decomposePerm` qui prend en paramètre un tableaux d'entiers `T` représentant une permutation et qui renvoie la décomposition de la permutation comme produit de transpositions selon le seconde méthode vue à l'exercice précédent. On pourra renvoyer une liste de listes `[[i1,j1],[j2,j2],...]` où `[i,j]` représente la transposition $(i\ j)$. L'ordre des transposition dans la liste devra être le même que celui du produit donnant la permutation.

Exercice 4 : tri fusion

1. Appliquer à la main l'algorithme de tri fusion sur le tableau `[4,2,5,6,1,4,1,0]`. Compter précisément le nombre de comparaisons effectuées.
2. Écrire une version itérative `fusionIterative(T1, T2)` (de complexité linéaire) de la fonction de fusion de tableaux.
3. On rappelle qu'une *inversion* de `T` est un couple d'éléments `T[i] < T[j]` mal ordonnés, c'est-à-dire dont les positions vérifient $i > j$.

   Modifier la fonction précédente en une fonction `nbInversionsEntre(T1, T2)` qui compte les inversions dans le tableau `T1+T2` (en supposant `T1` et `T2` triés).
4. En déduire un programme `nbInversions(T)` qui calcule le nombre d'inversions d'un tableau `T` en temps $\Theta(n \log n)$.