

## 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  $\sigma$  et  $\tau$  en notation cyclique (c'est-à-dire comme produits de cycles disjoints).
2. Calculer les produits  $\sigma\tau (= \sigma \circ \tau)$  et  $\tau\sigma$ .
3. Calculer les inverses  $\sigma^{-1}$ ,  $\tau^{-1}$  et  $(\sigma\tau)^{-1}$ .
4. Calculer  $\sigma^{2022}$ .

#### Exercice 2 : 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  $T$  représente 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  $T2$  de même taille représentant deux permutations, 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 tableau d'entiers  $T$  représentant une permutation et qui renvoie la décomposition de la permutation comme produit de transpositions selon la seconde méthode vue à l'exercice précédent. On pourra renvoyer une liste de listes  $[[i_1, j_1], [i_2, j_2], \dots]$  où  $[i, j]$  représente la transposition  $(i \ j)$ . L'ordre des transpositions dans la liste devra être le même que celui du produit donnant la permutation.

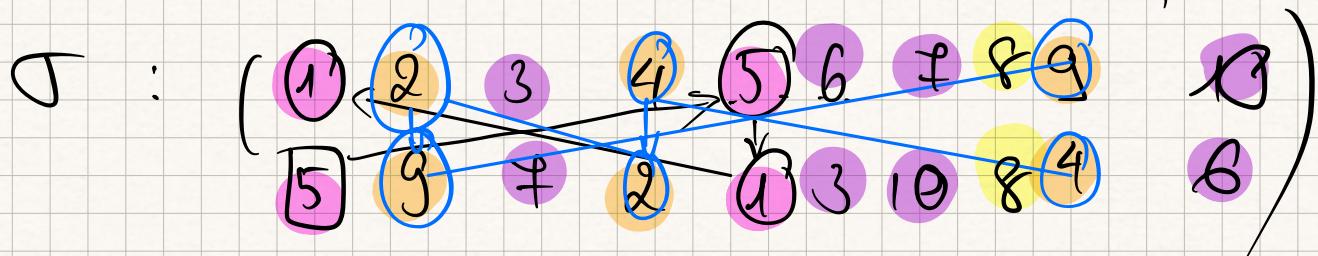
#### Exercice 3 : tri fusion

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  $T$  de taille  $n$  en temps  $\Theta(n \log n)$ .

Ex1

Permutation:

$$\sigma : \{1, \dots, n\} \longrightarrow \{1, \dots, n\}$$



$$\sigma(1) = 5, \sigma(2) = 9$$

$$\sigma(3) = 7$$

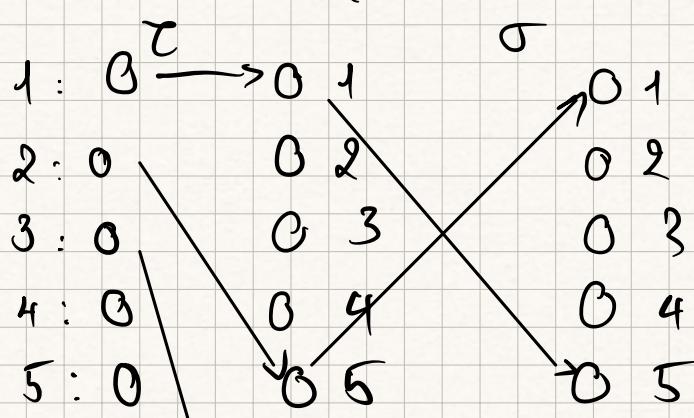
$$\sigma = (1\ 5)(2\ 9\ 4)(3\ 7\ 10\ 6)(8)$$

$$\tau = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10)$$

$$\begin{aligned}\tau &= (1)(2\ 5)(3\ 8)(4)(6\ 7)(9)(10) \\ &= (2\ 5)(3\ 8)(6\ 7)\end{aligned}$$

$$\sigma\tau = \tau\sigma$$

$$\sigma\tau(i) = \sigma(\tau(i))$$



$$\begin{array}{l}
 6 : 0 \quad 0 6 \\
 7 : 0 \quad 0 7 \\
 8 : 0 \quad \sqrt{0} 8 \longrightarrow 0 8 \\
 9 : 0 \quad 0 9 \\
 10 : 0 \quad 0 10
 \end{array}$$

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

$$\tau \sigma = \tau \circ \sigma$$

$$\tau \sigma(i) = \tau(\sigma(i))$$

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

$$\sigma^{-1} \circ \tau = \text{id}$$

$$\sigma^{-1}(\sigma(i)) = i$$

i = 1 . \quad \sigma^{-1}(\sigma(1)) = 1

$$\sigma^{-1} \circ \sigma^{-1}(5) = 1$$

$$\begin{array}{l}
 1 : 0 \quad \rightarrow 0 1 \\
 2 : 0 \quad 0 2 \\
 3 : 0 \quad 0 3 \\
 4 : 0 \quad 0 4 \\
 5 : 0 \quad \sqrt{0} 5 \quad \rightarrow 0 5
 \end{array}$$

$$\begin{array}{l}
 6 : 0 \quad 0 6 \\
 7 : 0 \quad 0 7 \\
 8 : 0 \quad \sqrt{0} 8 \quad 0 8 \\
 9 : 0 \quad 0 9 \\
 10 : 0 \quad 0 10 \quad 0 10
 \end{array}$$

$$\sigma^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 5 & 8 & 4 & 2 & 7 & 6 & 3 & 9 & 10 \end{pmatrix}$$

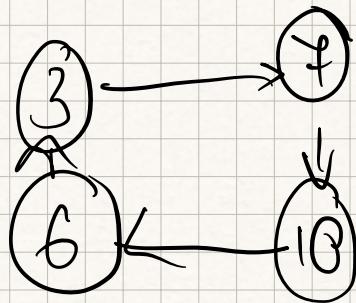
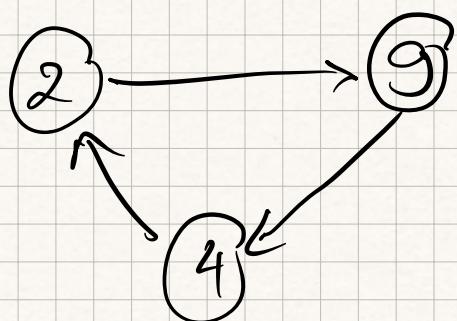
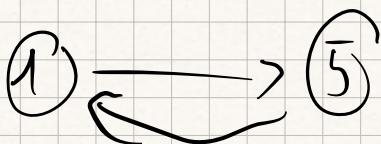
$$\tau^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 5 & 4 & 6 & 9 & 1 & 10 & 3 & 8 & 2 & 7 \end{pmatrix}$$

$$= (1 \ 5)(2 \ 4 \ 9) \cancel{(3 \ 6 \ 10 \ 7)}$$

(8)

$$(5 \ 7)^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 4 & 7 & 3 & 1 & 10 & 8 & 3 & 5 & 6 \end{pmatrix}$$

21



$$0^{2022} = (15)^{2022} \left( -294 \right)^{2022} (3 + 106)^{2022}$$

$$2022 \equiv 0 \pmod{2}$$

$$(1 \ 5)^{2022} = id$$

$$2022 \equiv 0 \pmod{3}$$

$$(294)^{2022} = \text{id}$$

$$2022 \equiv 2 \pmod{4}$$

$$(3 + 106)^{2022} = (3 + 106)^2$$

$$= (3 \ 10)(6 \ 7)$$

$$\sigma^{2022} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 10 & 4 & 1 & 7 & 6 & 8 & 9 & 3 \end{pmatrix}$$

$$\sigma \rightarrow \tau \quad \tau[i] = \sigma(i)$$

Ex 2.

1) Inverse ( $\tau$ )

2) Compose ( $\tau_1, \tau_2$ )

3) Vérifier est perm ( $\tau$ )

    Temps linéaire

$$\tau[i] = \sigma(i + 1)$$

} de vérification

2)

( $\tau_1$ ) as not estPerm ( $\tau_2$ ) : return None

```
def inverse(T)
```

```
    T2 = [0] * len(T)
```

```
    for i in range (len[T])
```

```
        tmp = T[i]
```

```
        T2[tmp] = i+1
```

```
Return R2
```

```
def composePerm(T1,T2)
```

```
    n= len(T1)
```

```
    if len(T2) != n return None
```

```
    if not est Perm
```

