EA4 – Éléments d'algorithmique

TD n° 6 : produits de permutations, tri et sélection rapides

Exercice 1 : décomposition en produit de transpositions

On rappelle qu'une *transposition* est une permutation ayant un unique cycle de longueur 2 (et donc $n-2$ points fixes). Le but de cet exercice est de montrer plusieurs manières de décomposer une permutation en produit de transpositions, sur l'exemple de la permutation[1]

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

À partir de la représentation en cycles disjoints

1. Calculer les produits $(i_1\ i_2) \circ (i_2\ i_3) \circ (i_3\ i_4)$ et $(i_1\ i_2) \circ (i_1\ i_3) \circ (i_1\ i_4)$.

2. En déduire deux factorisations différentes du cycle $(2\ 9\ 7\ 10)$ en produit de 3 transpositions, puis (au moins) une factorisation de σ.

À partir de la représentation linéaire

3. Calculer $\sigma_1 = \sigma \circ (1\ 8)$ et $\sigma'_1 = (1\ 4) \circ \sigma$, puis comparer σ_1 et σ'_1 à σ. Quel algorithme de tri procède de cette manière ?

4. Calculer itérativement des transpositions τ_i telles que $\sigma\circ\tau_1\circ\tau_2\circ\cdots\circ\tau_\ell = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$.

5. Que peut-on en déduire pour $\tau_1 \circ \tau_2 \circ \cdots \circ \tau_\ell$? En déduire une décomposition de σ en produit de ℓ transpositions.

6. Calculer itérativement des τ'_i telles que $\tau'_\ell \circ \cdots \circ \tau'_2 \circ \tau'_1 \circ \sigma = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$. En déduire une autre décomposition de σ en produit de ℓ transpositions.

7. Au lieu de chercher à trier σ, donc à partir de σ pour obtenir la permutation identité, on peut faire l'inverse : partir de 1 2 3 4 5 6 7 8 9 10 pour *engendrer* σ en plaçant d'abord le bon élément en position 1, puis celui en position 2, etc. Calculer de cette manière une décomposition de σ en produit de transpositions. Que remarquez-vous ?

Exercice 2 : déroulement des algorithmes de tri et de sélection rapides

On considère le tableau `T` suivant :

73	19	55	64	28	37	99	82	5	91	46

(les questions étoilées sont à aborder seulement pour les TD postérieurs au Cours n° 7)

1. Décrire le déroulement des variantes suivantes du tri rapide sur le tableau `T` :

a. variante avec mémoire auxiliaire, en prenant le premier élément comme pivot ;

b. *idem*, mais en choisissant toujours pour pivot l'élément médian ;

c. (*) variante « en place », le pivot étant toujours le premier élément.

2. (*) Décrire le déroulement de l'algorithme de sélection rapide pour trouver l'élément de rang 4 du tableau `T`.

Dans chaque cas, compter précisément les comparaisons effectuées.

1. il n'est pas interdit de vérifier qu'il s'agit bien de deux représentations de la même permutation

Exercice 3 : complexité du tri rapide

1. **Évaluer** le nombre de comparaisons d'éléments effectuées par le tri rapide (avec pivot `T[0]`) dans les cas suivants :
   - `C[n] = [1, 2, ..., n-1, n]` ;
   - `D[n] = [n, n-1, ..., 2, 1]` ;
   - `M[k] = [2**k] + M[k-1] + [ i+2**k for i in M[k-1] ]` (avec `M[0] = [1]`).
2. Donner d'autres exemples de tableaux de taille 7 pour lesquels la complexité du tri rapide est la même que pour `C[7]`. Même question pour `M[2]`.
3. Quel est le nombre de sommets de l'arbre de récursion du tri rapide pour un tableau de longueur n dont tous les éléments sont distincts ? Que peut-on en conclure concernant sa hauteur minimale ? et sa hauteur maximale ?
4. À chaque sommet dans l'arbre de récursion, on associe la valeur du pivot utilisé. Soit un sommet de profondeur p dans l'arbre de récursion. À combien de pivots a-t-il été comparé ? En déduire une expression du nombre total de comparaisons effectuées par le tri représenté par un arbre donné.
5. En déduire la complexité en temps du tri rapide dans le meilleur et dans le pire cas (en supposant toujours que tous les éléments sont distincts).

Exercice 4 : hauteur de pile du tri rapide

Comme tout algorithme récursif, le tri rapide est sujet à des débordements de la pile si le nombre d'appels récursifs devient trop grand.

1. On considère l'implémentation « naïve » du tri rapide avec copies de tableaux :

```
def triRapide(T) :
    if len(T) <= 1 : return T
    pivot = T[0]
    gauche = [elt for elt in T[1:] if elt <= pivot]
    droite = [elt for elt in T[1:] if elt > pivot]
    return triRapide(gauche) + [pivot] + triRapide(droite)
```

   Quelle hauteur atteint la pile dans le pire cas ? dans le meilleur cas ?
2. Le deuxième appel récursif est terminal [2] et peut donc être dérécursivé. Quelle hauteur atteint alors la pile sur l'entrée `C[n] = [1, ..., n]` ?
   Et sur l'entrée `D[n] = [n, ..., 1]` ?
3. Proposer une solution permettant de garantir une hauteur de pile maximale en $O(\log n)$ dans tous les cas (où n est la longueur du tableau à trier).
4. Quel est alors le meilleur cas en ce qui concerne la hauteur de pile ?

2. enfin... presque... faisons comme si c'était le cas !