EA4 – Éléments d'algorithmique
TP n° 4 : Permutations, compteurs, dichotomie

Modalités de rendu : À chaque TP, vous devrez rendre les exercices marqués par un symbole ✍. Le rendu de l'exercice K du TP N doit être inclus dans le fichier `tpN_exK.py`, à télécharger depuis la section « Énoncés de TP ». Vous devez remplir les zones marquées par le commentaire `A REMPLIR` dans ce fichier. Ne modifiez pas les autres fonctions du fichier, sauf demande explicite de l'énoncé. Chaque fichier contient une fonction `main` qui teste les fonctions que vous devez programmer et qui vous affiche un score donné par le nombre de tests passés avec succès. Pour passer ces tests, vous devez exécuter le programme écrit. La date limite du rendu d'un TP est le dimanche à 16 heures (heure de Paris).

Convention pour l'étude de la complexité Dans ce TP, on considèrera essentiellement le nombre de comparaisons produites par un algorithme, entre deux éléments d'un tableau. Les opérations concernant les affectations de tableaux, ou la récupération de leur longueur, ne seront pas à prendre en compte.

Exercice 1 : opérations sur les permutations

Pour chacune des questions de cet exercice, seule une quantité insuffisante de tests vous est fournie. Vous veillerez donc à en ajouter systématiquement, en particulier pour vous assurer que tous les cas de figure sont testés.

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. ✍ Modifier votre fonction `estPerm` en une nouvelle fonction `estPermOps` qui, en plus de renvoyer un booléen, renvoie également le nombre de comparaisons effectuées par l'algorithme.
3. ✍ 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`.
4. ✍ Modifier votre fonction `inversePerm` en une nouvelle fonction `inversePermOps` qui, en plus de renvoyer un tableau, renvoie également le nombre (cumulé) de comparaisons et d'affectations effectuées par l'algorithme.
5. ✍ 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`.
6. ✍ Modifier votre fonction `composePerm` en une nouvelle fonction `composePermOps` qui, en plus de renvoyer un tableau, renvoie également le nombre (cumulé) de comparaisons et d'affectations effectuées par l'algorithme.

Exercice 2 : Compteurs

Pour chacune des questions de cet exercice, vous observerez comment sont réalisés les tests et chercherez à ajouter des tests supplémentaires.

1. ✍ Écrire une fonction `occurrence` prenant en paramètres un tableau d'entiers `T` et un entier `x`, et qui renvoie le nombre de fois où `x` apparaît dans `T`.
2. ✍ Modifier la fonction `occurrence`, et écrire la fonction `occurrenceOps` qui compte le nombre de comparaisons effectuées par l'algorithme.
3. ✍ En utilisant la fonction `occurrence`, écrire une fonction `compteNaif(T, m)`, qui renvoie un tableau d'entiers `S` de taille m, de sorte que `S[i]` contienne le nombre d'éléments égaux à i dans le tableau `T`.
4. ✍ Modifier votre fonction `compteNaif` en une nouvelle fonction `compteNaifOps(T,m)` qui, en plus de renvoyer le tableau `S`, renvoie également le nombre de comparaisons effectuées par l'algorithme.

Exercice 3 : Dichotomie

Dans cet exercice, les tableaux donnés en entrée seront toujours triés (dans l'ordre croissant), avec éventuellement des répétitions. De plus, seule une quantité insuffisante de tests vous est fournie. Vous veillerez donc à en ajouter systématiquement, en particulier pour vous assurer que tous les cas de figures sont testés.

1. ✍ Écrire une fonction récursive `trouve(T,x, begin=0, end=None)`, qui effectue une recherche dichotomique de l'entier `x` dans le tableau `T`, entre les indices `begin` et `end`. Si `end` vaut None, il est initialisé par la fonction à `len(T)-1`. La fonction renverra `True` si `x` est présent, `False` sinon.
2. ✍ Modifier la fonction précédente en une fonction `trouveOps` afin qu'elle renvoie également le nombre de comparaisons effectuées.
3. ✍ Écrire (ou plutôt, modifier la fonction précédente) une fonction `trouvePremier(T,x,begin=0,end=None)` qui renvoie l'indice de la première occurrence de `x` si `x` est présent dans le tableau `T`, et renvoie `False` sinon. La recherche doit se faire par dichotomie !
4. ✍ Modifier la fonction précédente en une fonction `trouvePremierOps(T,x,begin=0,end=None)` qui renvoie le même résultat ainsi que le nombre de comparaisons effectuées.
5. ✍ Écrire une fonction `occurenceDichotomie(T,x)`, basée sur un algorithme dichotomique, qui renvoie le nombre d'occurrences de `x` dans le tableau `T`.
6. ✍ Modifier la fonction précédente en une fonction `occurenceDichotomieOps(T,x)` qui, en plus de renvoyer le résultat de `occurenceDichotomie`, renvoie également le nombre de comparaisons effectuées.
7. Comparez – à l'aide d'exemples – la complexité des algorithmes `occurrence` (de l'exercice précédent) et `occurrenceDichotomie` en termes de nombre de comparaisons entre éléments du tableau.