1 TP bonus : Algorithme de Knuth

Notions pratiquées : listes, permutations, aléatoire.

1.1 Permutation d'une liste

On se propose dans un premier temps de générer une permutation d'une liste. Il existe une fonction `shuffle()` dans le module `random`, mais il est demandé de **ne pas** l'utiliser pour l'instant.

1. Écrire une fonction `echange(L,i,j)` qui prend en argument une liste L ainsi que 2 entiers i et j, et qui échange les 2 éléments de la liste L situés à ces positions (on suppose que les indices reçus sont corrects). Cette fonction ne renvoie rien (elle modifie la liste L par effet de bord).
2. Écrire une fonction `melange(L)` qui prend une liste L en paramètre et qui en permute aléatoirement les éléments, en utilisant la fonction `echange` définie ci-dessus.
3. Écrire une fonction `generer_permutations_1(n,m)` qui reçoit un entier n et un entier m, qui construit la liste L des n entiers naturels dans l'ordre croissant, puis qui en génère m permutations avec la fonction `melange` ci-dessus (on utilisera des copies de L). La fonction renvoie la liste de ces permutations (liste de m listes de n entiers).

1.2 Analyse de la distribution des permutations générées

On veut maintenant analyser une liste de permutations en terme de fréquence d'apparition de chaque permutation dans la liste. Par exemple si on analyse une liste de 7 permutations de 2 entiers (seules 2 permutations différentes existent) générées aléatoirement :

```
>>> lp = generer_permutations(2,7)
>>> lp
[[1,2], [1,2], [2,1], [2,1], [1,2], [1,2], [1,2]]
>>> frequences(lp)
[5,2]
```

1. Écrire une fonction qui reçoit une liste de permutations et qui construit une liste d'entiers qui représente la fréquence d'apparition de chaque permutation dans cette liste. La liste des fréquences est donc une liste d'entiers strictement positifs ; la position dans cette liste n'a pas de signification.
2. Utiliser le module `time` pour calculer le temps d'exécution de cette fonction, par exemple pour $n = 10$ et $m = 50000$.
3. Si votre fonction prend trop longtemps (plusieurs minutes), écrivez-en une nouvelle version mieux optimisée. En particulier, il faut éviter de parcourir plusieurs fois la liste de permutations pour y compter les éléments.
4. Utilisez à nouveau le module `time` pour vérifier combien de temps vous avez gagné sur l'exécution de votre fonction.
5. Écrire une fonction qui reçoit une liste de fréquences et affiche ses caractéristiques: valeur minimale, maximale, moyenne, écart-type (le module *statistics* fournit une fonction *stdev*).

1.3 Comparaison avec la fonction `shuffle`

Nous allons maintenant pouvoir comparer les permutations générées par votre fonction (écrite à l'exercice 1) avec celles générées par la fonction `shuffle` fournie dans le module `random`.

1. Utiliser les fonctions précédentes pour créer la liste des n premiers entiers naturels, en générer m permutations avec votre fonction *melange*, puis m permutations avec la fonction $shuffle$.
2. Utiliser les fonctions précédentes pour calculer et afficher les statistiques de ces 2 listes de permutations.
3. Faire des tests notamment avec m très supérieur à n. Que remarque-t-on ?
4. En fait la fonction $shuffle$ utilise l'algorithme de Knuth, décrit ci-dessous. Implémentez cette nouvelle fonction de permutation, et refaites les tests pour vérifier qu'on obtient bien les mêmes résultats.

```
# Algorithme de Knuth ou Fisher-Yates
def permutek(l):
  for i in range(len(l)-1, 0, -1):
    j = random.randint(0, i + 1)
    swap(l,i,j)
```

1.4 Génération de toutes les permutations possibles d'une liste

On souhaite maintenant générer de manière exhaustive toutes les permutations possibles d'une liste de n entiers. Attention, quand n augmente, le nombre de permutations augmente très vite ! Il faudra travailler avec n pas trop grand.

1. Écrire une fonction `combien(n)` qui calcule et renvoie le nombre de permutations possibles d'une liste de n éléments. *Combien y en a-t-il ?*
2. Écrire une fonction `exhaustive_p(n)` qui génère la liste de toutes les permutations possibles de la liste des n premiers entiers naturels. *Indice: procéder de manière récursive.*
3. Un premier test rapide de votre fonction consiste à vérifier qu'elle renvoie bien le bon nombre de permutations, calculé ci-dessus.
4. Le module `itertools` fournit une fonction `permutations(li,n)` qui génère toutes les permutations de n éléments de la liste li. Cette fonction renvoie un objet de type `permutations` qu'on peut convertir en liste ; les éléments dans cet objet sont des tuples et pas des listes. Utiliser cette fonction pour écrire une fonction `exhaustive_i(n)` qui reçoit un seul argument n, et qui génère et renvoie la liste exhaustive de toutes les permutations complètes de la liste des n premiers entiers naturels (liste de listes de n entiers).
5. Écrire maintenant une fonction `verifier(li,sol)` qui reçoit 2 listes de listes d'entiers, et qui vérifie que la liste li comprend bien une et une seule fois chacun des éléments de sol.
6. Utiliser cette fonction `verifier` pour tester votre fonction `exhaustive_p` de génération des permutations, par comparaison avec la liste de permutations générée par `exhaustive_i`.
7. Vous pouvez aussi utiliser la fonction `time()` du module `time` pour comparer le temps d'exécution de votre générateur de permutations avec celui fourni par le module `itertools`.

1.5 Affichage

On veut maintenant afficher sur un histogramme les fréquences des permutations générées par votre fonction `melange`, vs celles générées par la fonction `shuffle` de Python.

On travaillera avec la liste li des n premiers entiers naturels. On veillera à travailler avec un m suffisamment plus grand que n pour pouvoir observer des répétitions de certaines permutations.

1. Générer une liste $lp1$ de m permutations de li avec la fonction `melange` ; et une autre liste $lp2$ de m permutations de li avec la fonction `shuffle`.
2. Calculer la liste lpe de toutes les permutations possibles de li avec la fonction `exhaustive_p`
3. Calculer la liste $lf1$ des fréquences dans $lp1$ de toutes les permutations possibles (données par lpe). La liste $lf1$ peut donc cette fois contenir des 0. De même calculer la liste $lf2$ des fréquences des éléments de lpe dans $lp2$.
4. Afficher la distribution comparative des 2 listes de fréquences, en utilisant `matplotlib` pour tracer un histogramme.

1.6 Pour en savoir plus

Article Wikipédia sur le mélange de Fisher-Yates ou de Knuth : `https://fr.wikipedia.org/wiki/M%C3%A9lange_de_Fisher-Yates`

L'algorithme de Knuth permet aussi de limiter la taille des nombres aléatoires générés avec `randint`, ce qui en est un autre avantage.