TP 03

Recherche séquentielle dans un tableau unidimensionnel. Algorithmes opérant sur une structure séquentielle par boucles imbriquées.

Savoirs et compétences :

- *Th. 2 : Algorithmes opérant sur une structure séquentielle par boucles imbriquées.*

Consignes

1. Commencez la séance en créant un dossier au nom du TP dans le répertoire dédié à l'informatique de votre compte.
2. Après la séance, vous devez rédiger un compte-rendu de TP et l'envoyer au format électronique à votre enseignant.
3. Ayez toujours un crayon et un papier sous la main. Quand vous réfléchissez à une question, utilisez les !
4. Essayer d'être le plus autonome possible.
5. Ce TP est à faire en binôme, vous ne rendrez donc qu'un compte-rendu pour deux. Votre fichier portera un nom du type : `tp03_durif_berne.py`, où les noms de vos enseignants sont à remplacer par ceux des membres du binôme. Le nom de ce fichier ne devra comporter ni espace, ni accent, ni apostrophe, ni majuscule. Dans ce fichier, vous respecterez les consignes suivantes.
   - Écrivez d'abord en commentaires (ligne débutant par #), le titre du TP, les noms et prénoms des étudiants du groupe.
   - Commencez chaque question par son numéro écrit en commentaires.
   - Les questions demandant une réponse écrite seront rédigées en commentaires.
   - Les questions demandant une réponse sous forme de fonction ou de script respecteront pointilleusement les noms de variables et de fonctions demandés.

Le but de ce TP est l'entrainement à l'enchaînement des boucles itératives et quelques notions de complexité.

Activité 1 –

Rappels sur les dictionnaires

- Un dictionnaire est une suite de couple (clé, valeur) non ordonnée. Chaque élément est repéré par sa clé qui est donc unique. Le type d'un dictionnaire est `dict`.

- Test d'appartenance d'une clé :

`print(''PCSI'' in dico)` qui renvoie ici `True` (ou `False` dans le cas général)

- Nombre d'éléments d'un dictionnaire :

`len(dico)`

- Parcourir les éléments d'un dictionnaire avec `items()` :

```
for elt in dico.items():
    a=elt[0] # clé
    b=elt[1] # valeur
```

OU

```
for clé, valeur in dico.items():
    ...
```

- Copie d'un dictionnaire :

```
import copy
dico1=copy.deepcopy(dico)
```

(*et non* `dico1=dico` *qui va créer un alias seulement.*)

Exercice : On dispose d'un dictionnaire associant à des noms de commerciaux d'une société le nombre de ventes qu'ils ont réalisées : Par exemple :
`ventes={''Dupont'':14,''Henry'':19,''Pierre'':15,''Lucas'':21}`

Question 1 *Écrire une fonction* `Nb_Ventes(ventes:dict)` *qui prend en entrée un dictionnaire et renvoie le nombre total de ventes dans la société.*

Question 2 *Écrire une fonction* `Nom_vendeur(ventes:dict)` *qui prend en entrée un dictionnaire et renvoie le nom du vendeur ayant réalisé le plus de ventes. Si plusieurs vendeurs sont ex-aequo, la fonction devra retourner le nom de l'un d'entre eux seulement.*

Activité 2 – Recherche dans un tableau

- Un dictionnaire est une suite de couple (clé, valeur) non ordonnée. Chaque élément est repéré par sa clé qui est donc unique. Le type d'un dictionnaire est `dict`.
- Le dictionnaire vide est `dico={}`
- On peut définir un dictionnaire globalement :
  `dico={''MPSI'':46,''PCSI'':47,''PTSI'',45}` ou "MPSI" est une clé, 46 sa valeur.
- On peut ajouter des éléments à un dictionnaire :
  `dico[''MP'']=45`
  et afficher :
  `print(dico)` renvoie : `dico={''MPSI'':46,''PCSI'':47,''PTSI'',45,''MP'':45}`
  `print(dico[''MPSI''])` affiche 46
- On peut supprimer un élément :
  `del dico[''MP'']`
  `print(dico)` renvoie `dico={''MPSI'':46,''PCSI'':47,''PTSI'',45}`
- Test d'appartenance d'une clé :
  `print(''PCSI'' in dico)` qui renvoie ici `True` (ou `False` dans le cas général)
- Nombre d'éléments d'un dictionnaire :
  `len(dico)`
- Parcourir les éléments d'un dictionnaire avec `items()` :

```
for elt in dico.items():
    a=elt[0] # clé
    b=elt[1] # valeur
```

OU

```
for clé, valeur in dico.items():
    ...
```

- Copie d'un dictionnaire :

```
import copy
dico1=copy.deepcopy(dico)
```

(*et non* `dico1=dico` *qui va créer un alias seulement.*)

Exemple 1

On dispose d'un dictionnaire associant à des noms de commerciaux d'une société le nombre de ventes qu'ils ont réalisées :

Par exemple :

```
ventes={''Dupont'':14,''Henry'':19,''Pierre'':15,''Lucas'':21}
```

Question 3 *Ecrire une fonction* `Nb_Ventes(ventes:dict)` *qui prend en entrée un dictionnaire et renvoie le nombre total de ventes dans la société.*

Question 4 *Ecrire une fonction* `Nom_vendeur(ventes:dict)` *qui prend en entrée un dictionnaire et renvoie le nom du vendeur ayant réalisé le plus de ventes. Si plusieurs vendeurs sont ex-aequo, la fonction devra retourner le nom de l'un d'entre eux seulement.*

Exemple 2

On se donne une liste (ou un tableau) `L` constituée d'entiers et contenant au moins deux éléments.

On cherche à identifier un couple `L[p]` et `L[q]` d'éléments les plus proches parmi ceux de la liste, c'est-à-dire vérifiant les conditions :

- $0 \leqslant p < q \leqslant$ `len(L)` -1
- $|L[p]-L[q]| = \min\{|L[i]-L[j] \text{ tq } 0 \leqslant i < j \leqslant \texttt{len(L)}-1\}$.

Question 5 *On appelle distance minimale, la distance entre deux éléments les plus proches (éventuellement égaux) et d'indices distincts. Écrire une fonction* `distance_min(L)` *qui renvoie la distance minimale de* `L`.

Par exemple, pour `T=[10, 10,10, 8, 8, 0, 5,1, 9, 5, 9, 9]`. la fonction devra renvoyer 0.

Question 6 *Écrire une fonction* `indices_distance_min2(L)` *qui renvoie un couple d'indices réalisant la distance minimale*

Par exemple, pour `T=[10, 10,10, 8, 8, 0, 5,1, 9, 5, 9, 9]`. la fonction pourra renvoyer $(0,8)$.

Une autre méthode pour trouver un couple d'indices solution consiste à procéder de la façon suivante :

- Réserver un dictionnaire `D`.
- Pour chaque indice $0 \leqslant i \leqslant$ `len (L)` -1, placer dans `D[i]` le couple $(j, |L[j]-L[i]|)$ qui réalise le minimum de la distance $|L[i]-L[j]|$ pour $j > i$.
- Une fois ce dictionnaire construit, le parcourir pour déterminer un couple (p,q) solution.

Question 7 *Écrire une fonction* `indices_distance_min3(L)` *qui, étant donnée une liste* `L`, *réalise ces opérations et renvoie un couple solution*

Activité 3 – Recherche d'un mot dans un texte

Question 8 *Écrire une fonction* `est_ici(texte,motif,i)` *qui, étant données deux chaines de caractères* `texte`, `motif` *et un indice* `i`, *renvoie* `True` *ou* `False` *selon que* `motif` *est ou n'est pas dans* `texte` *au rang* `i`. *On utilisera une boucle* `while`.

Question 9 *Écrire une fonction* `est_sous_mot(texte,motif)` *qui renvoie* `True` *ou* `False` *selon que* `motif` *est dans* `texte` *ou pas. On utilisera une boucle* `while`.

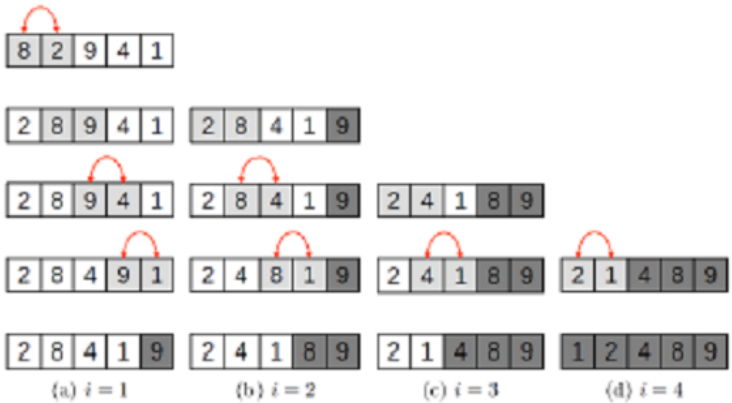
Question 10 *Écrire une fonction* `position_sous_mot(texte,motif)` *qui renvoie la liste de toutes les occurences de l'indice de position de la première lettre du mot* `motif` *dans* `texte`. *On utilisera une boucle* `for`.

Activité 4 – Tri à bulles

Le tri à bulles est un algorithme de tri classique. Son principe est simple, et il est très facile à implémenter mais il n'est pas très performant.

On considère un tableau de nombres T, de taille N. L'algorithme parcourt le tableau, et dès que deux éléments consécutifs ne sont pas ordonnés, les échange. Après un premier passage, on voit que le plus grand élément se situe bien en dernière position. On peut donc recommencer un tel passage, en s'arrêtant à l'avant-dernier élément, et ainsi de suite.

Au i-ème passage on fait remonter le i-ème plus grand élément du tableau à sa position définitive, un peu à la manière de bulles qu'on ferait remonter à la surface d'un liquide, d'où le nom d'algorithme de tri à bulles.



Question 11 *Appliquer l'algorithme de tri à Bulles «à la main» au tableau ci-dessous :*

2	1	6	9	8	4

Question 12 *Ecrire une fonction* `est_trie(T)` *qui renvoie* `True` *ou* `False` *selon que le tableau* `T` *est trié ou pas.*

Question 13 *Écrire une fonction* `TriBulles(T)` *triant le tableau* `T` *par l'algorithme de tri à bulles*

On rappelle que l'on peut créer un tableau aléatoire grâce à la fonction `randint` de la bibliothèque `random`. Étant donnés deux entiers `a` et `b`, randint.random(a,b) renvoie un entier aléatoire compris entre `a` et `b` inclus. Ainsi, la suite d'instructions :

```
import random as rd
L=[rd.randint(0,100) for i in range(50)]
```

crée une liste de 50 nombres entiers aléatoires compris entre 0 et 100 inclus.

Question 14 *Vérifier que la fonction ci-dessus est correcte en utilisant la fonction* `EstTrie` *sur un tel tableau de nombres aléatoires.*

Question 15 *Pour un tableau T de longueur n, calculer le nombre de comparaisons $C(n)$ faites dans la fonction* `TriBulles(T)` *et vérifier que la suite* $\left(\frac{C(n)}{n^2}\right)$ *a une limite finie donc est bornée (on dit alors que la* ***complexité est quadratique****);*

Question 16 *Ecrire une fonction améliorée* `TriBulles2(T)` *qui sort de la fonction dès que le tableau a été parcouru sans faire d'échange (auquel cas, il est trié et donc inutile de continuer).*

FACULTATIF

Une variante du tri à bulles est le tri cocktail : il consiste à changer de direction à chaque passage. Lors du premier parcours, on se déplace du début du tableau vers la fin. Puis lors du second parcours on part de la fin du tableau pour arriver au début. A la troisième itération on part du début et ainsi de suite ... C'est une légère amélioration car il permet non seulement aux plus grands éléments de migrer vers la fin de la série mais également aux plus petits éléments de migrer vers le début.

Question 17 *Écrire une fonction* `TriCocktail(T)` *qui étant donné un tableau* `T` *le renvoie trié selon la méthode du tri cocktail.*

Question 18 *Justifier en quoi cette variante peut être intéressante selon le type de tableau à trier mais vérifier néanmoins que la complexité reste quadratique.*