CAPES MATHS OPTION INFORMATIQUE

EXERCICES – ALGORITHMIQUE ET PROGRAMMATION

Exercice 1 : Recherche dichotomique dans un tableau initialement trié

a. Ecrire en Python la fonction qui renvoie l'indice, de l'élément e passé en paramètre, dans un tableau trié. Cette recherche se fera de façon dichotomique.

b. Quel est le coût d'une telle recherche ?

Exercice 2 : Recherche d'un mot dans une chaîne

a. Ecrire une fonction en Python qui prend en paramètres un mot et une chaîne de caractères et détermine si le mot est dans la chaîne (sans utiliser la fonction in déjà existante en Python).

b. Estimer la complexité de l'algorithme proposé dans le pire des cas.

Exercice 3 : Conversion décimal-binaire

L'objectif est de créer un programme qui code un nombre entier naturel, donné en écriture décimale, en binaire. Les nombres binaires seront représentés par des chaînes de caractères. Pour plus de lisibilité, on insérera un espace tous les quatres caractères en partant du dernier. Ansi, le nombre décimal 525 sera traduit par la chaîne "10 0000 1101" en binaire.

a. Ecrire une fonction conversion en Python, qui prend en paramètre un nombre entier naturel en décimal et renvoie la chaîne de caractères composée uniquement de 0 et de 1 correspondant à son écriture binaire. On ne cherchera pas encore à insérer d'espaces.

b. Ecrire une fonction prenant en paramètre une chaîne de caractères et réalisant l'insertion d'un espace tous les quatres caractères en partant du dernier.

c. Modifier la fonction conversion pour prendre en compte l'insertion des espaces.

Exercice 4 : Crible d'Eratosthène et décomposition en facteurs premiers

a. On se propose de calculer tous les nombres premiers plus petits qu'un entier n donné. La méthode consiste à calculer pas à pas ces nombres en utilisant la règle suivante : si un entier k n'est divisible par aucun nombre premier plus petit que k alors il est lui-même premier. Quelles sont les structures de données qu'on peut utiliser pour résoudre ce problème ? Quelle est la plus efficace ?

b. Ecrire en Python la procédure **eratosthene** qui retourne le tableau des nombres premiers plus petits que n passé en paramètre.

c. Ecrive en Python la fonction **decompose,** qui retourne la décomposition d'un entier n par le produit de nombres premiers.

Exercice 5 : Tri mystère

Voici le début de la trace d'un algorithme de tri :

10	56	-2	52	-8	41	13
-8	56	-2	52	10	41	13
-8	-2	56	52	10	41	13
-8	-2	10	52	56	41	13

a. De quel algorithme s'agit-il ? Pourquoi ?

b. Complétez le tableau ci-dessus en indiquant les étapes suivantes, jusqu'à ce que le tableau soit complètement trié.

c. Si le tableau est de taille *n*, quelle est la complexité de cet algorithme en fonction de *n* ? Justifiez (brièvement) votre réponse.

Exercice 6 : Analyse de la complexité d'un algorithme

On considère le pseudo-code suivant, comportant deux « tant que » imbriqués. On cherche à mesurer la complexité de cette imbrication en fonction de n. Pour cela, on utilise la variable « compteur », qui est incrémentée à chaque passage dans le « tant que » interne.

```
Variables :
  n : entier
  compteur : entier
  i, j : entiers
Début
  Afficher(« Quelle est la valeur de n ? »)
  Saisir(n)
  compteur ← 0
  i ← 1
  Tant que (i < n) Faire
      j ← i + 1
      Tant que (j ≤ n) Faire
          compteur ← compteur + 1
          j ← j + 1
      Fin tantque
      i ← i * 2
  Fin tantque
  Afficher(compteur)
Fin
```

a. Quelle est la valeur finale du compteur dans le cas où n = 16 ?

b. Considérons le cas particulier où n est une puissance de 2 : on suppose que $n = 2^p$ avec p connu. Quelle est la valeur finale du compteur en fonction de p ? Justifiez votre réponse.

c. Réexprimez le résultat précédent en fonction de n.

Exercice 7 : Tri à bulles

Le tri à bulles est un algorithme de tri qui s'appuie sur des permutations répétées d'éléments contigus qui ne sont pas dans le bon ordre.

Procédure tri_bulles(tab : tableau [1..n] de réels)

Précondition : tab est un tableau contenant n réels

Postcondition : les éléments de tab sont triés dans l'ordre croissant

Variables locales : i, j : entiers, e : réel

Début

```
1    i ← 1
2    Tant que (i <= n-1) Faire
3        j← n
4        Tant que (j >= i+1) Faire
5            Si tab[j] < tab[j-1] Alors
6                {on permute les deux éléments}
7                e ← tab[j-1]
8                tab[j-1] ← tab[j]
9                tab[j] ← e
10           Fin Si
11           j ← j-1
12       Fin Tant que
13       i ← i + 1
14   Fin Tant que
```

Fin tri_bulles

a. Soit le tableau suivant : {53.8, 26.1, 2.5, 13.6, 8.8, 4.0}. Donnez les premiers états intermédiaires par lesquels passe ce tableau lorsqu'on lui applique la procédure tri_bulles.

b. Complétez la phrase suivante de sorte à ce qu'elle corresponde à l'invariant de boucle du Tant que interne (boucle sur j, lignes 4 à 12 de l'algorithme).

> « Lorsqu'on vient de décrémenter j (ligne 11), le ________________ du sous-tableau tab[_____... _____] se trouve en position _______. De plus, si j>1, les éléments du sous-tableau tab[______...______] occupent les mêmes positions qu'avant le démarrage de la boucle sur j. »

c. Déduisez-en la propriété que présente le tableau lorsqu'on a terminé cette boucle interne, c'est-à-dire lorsqu'on arrive sur la ligne 13.

d. En utilisant la propriété précédente, on peut montrer que l'invariant de boucle du Tant que externe (boucle sur i) est le suivant : « Juste avant d'incrémenter i (ligne 13) : tab est trié entre les indices 1 et i, et tous les éléments restants sont supérieurs ou égaux à tab[i] ». Donnez la première étape de cette démonstration (initialisation). Les étapes de conservation et de terminaison ne sont pas demandées.

e. Calculez le nombre total d'affectations de réels réalisées par la procédure tri_bulles lors du tri complet d'un tableau de *n* réels, dans le cas le plus défavorable.

Exercice 8 : Tri par insertion

Le tri par insertion est l'algorithme utilisé par la plupart des joueurs lorsqu'ils trient leur « main » de cartes à jouer. Le principe consiste à prendre le premier élément du sous-tableau non trié et à l'insérer à sa place dans la partie triée du tableau.

a. Dérouler le tri par insertion du tableau [5.1, 2.4, 4.9, 6.8, 1.1, 3.0].

b. Ecrire en Python le corps de la procédure de tri par insertion, par ordre croissant, d'un tableau de réels :

> **Procédure** tri_par_insertion (tab : tableau [1..n] de réels)
> **Précondition** : tab[1], tab[2], … tab[n] initialisés
> **Postcondition** : tab[1] ≤ tab[2] ≤ … ≤ tab[n]

c. Donner l'invariant de boucle correspondant à cet algorithme, en démontrant qu'il vérifie bien les 3 propriétés d'un invariant de boucle : initialisation, conservation, terminaison.

d. Evaluer le nombre de comparaisons de réels et le nombre d'affectations de réels pour un tableau de taille *n*, dans le cas le plus défavorable (tableau trié dans l'ordre décroissant). Cet algorithme est-il meilleur que le tri par sélection (ou tri du minimum) ?

Exercice 9 : Stabilité d'un algorithme de tri

Considérons la liste de noms-prénoms suivante :

Doe, Phil
Doe, Jane
Doe, Fred
Jones, Bill
Jones, Jane
Smith, Mary
Smith, Fred
Smith, Jane

Elle est pour l'instant triée dans l'ordre alphabétique des noms de famille. Mais on veut la trier dans l'ordre alphabétique des prénoms, à l'aide du tri ci-dessous.

> **Procédure tri** (tab : tableau [1…n] de Personnes)
> Précondition : tab[1], tab[2], … tab[n] initialisés
> Postcondition : tab[1] ≤ tab[2] ≤ … ≤ tab[n]
> Variables locales :
>   i, j : entiers
>   elt_a_placer : Personne
> **Début**
>   Pour i allant de 2 à n par pas de 1 Faire
>   elt_a_placer ← tab[i]
>   j ← i – 1
>   Tant que (j > 0) et (*elt_a_placer.prenom est strictement*
>   *avant tab[j].prenom dans l'alphabet*) Faire
>   tab[j+1] ← tab[j]
>   j ← j – 1
>   Fin TantQue
>   tab[j+1] ← elt_a_placer
>   Fin Pour
> **Fin tri_par_insertion**

a. De quel algorithme de tri s'agit-il ?

b. Quelle liste obtient-on au final avec ce tri ?

c. Un algorithme de tri est dit « stable » s’il préserve toujours l’ordre initial des ex-aequos. Dans notre exemple, l’algorithme est stable si les personnes avec le même prénom (qui vont être groupées ensemble lors du tri par prénom) restent dans l’ordre alphabétique des noms de famille. L’algorithme de tri par insertion est-il stable ? (La preuve n’est pas demandée).

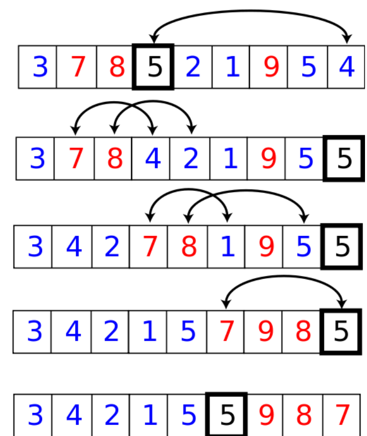
d. Combien de comparaisons de prénoms ferait-on avec cet algorithme s’il était appelé sur un tableau de taille n qui serait déjà dans le bon ordre (c’est-à-dire, ici, déjà trié dans l’ordre alphabétique des prénoms) ?

Exercice 10 : Tri rapide *(quick sort)*

Cette méthode de tri consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite. Cette opération s'appelle le *partitionnement*. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété récursivement, jusqu'à ce que l'ensemble des éléments soit trié.

Concrètement, pour partitionner un sous-tableau :

- on place le pivot arbitrairement à la fin (peut être fait aléatoirement), en l'échangeant avec le dernier élément du sous-tableau (étape 1 ci-contre)
- on place tous les éléments inférieurs au pivot en début du sous-tableau (étapes 2 et 3)
- on place le pivot à la fin des éléments déplacés (étapes 4 et 5)



Ecrire en Python la procédure de partitionnement et la procédure récurvise de tri rapide.

Exercice 11 : Tri par fusion (interne)

Soit un fichier binaire de *n* octets. Ecrire une version itérative (c’est-à-dire non récursive) de l’algorithme du tri par fusion d’un tableau de réels.

Exercice 12 : Tri par comptage

Soit N un entier naturel non nul. On cherche à trier une liste L d’entiers naturels strictement inférieurs à N.

a. Ecrire une fonction comptage, d’arguments L et N, renvoyant une liste P de longueur N dont l’élément d’indice k désigne le nombre d’occurrences de l’entier k dans la liste L.

b. Utiliser la liste P pour déduire une fonction triComptage, d’arguments L et N, renvoyant la liste L triée dans l’ordre croissant.

c. Tester la fonction triComptage sur une liste de 20 entiers inférieurs ou égaux à 5, choisis aléatoirement.

d. Quelle est la complexité temporelle de cet algorithme ? La compararer à la complexité d’un tri par insertion ou d’un tri fusion.

Exercice 13 : Calcul d'un polynôme en un point

Soit un polynôme P tel que $P(x) = \sum_{k=0}^{n} a_k x^k$.

On veut écrire la fonction qui retourne la valeur de $P(x)$ pour une valeur de *x* passée en paramètre, les coefficients du polynôme étant également passés en paramètre dans un tableau. Pour cela, une méthode efficace est la méthode de Horner, qui utilise la réécriture suivante de *P*(*x*) :

$$P(x) = \Big(\big(\ldots\big((a_n x + a_{n-1})x + a_{n-2}\big)x + \cdots\big)x + a_1\Big)x + a_0$$

La méthode consiste donc à multiplier le coefficient de plus haut degré par *x* et à lui ajouter le coefficient suivant. On multiplie alors le nombre obtenu par *x* et on lui ajoute le troisième coefficient, etc., jusqu'à avoir ajouté le coefficient constant.

Exemple : calcul de $4x^3 - 7x^2 + 3x - 5 = \big((4x - 7)x + 3\big)x - 5$
pour *x* = 2 :

- première étape : 4*2 – 7 = 1
- deuxième étape : 1*2 + 3 = 5
- troisième étape : 5*2 – 5 = 5

Voici le code Python de la fonction qui calcule $P(x)$ selon cette méthode :

```
def valeur_polynome (x, coefs, n) :
# Précondition : coefs contient les (n+1) coefficients du polynôme, ainsi coef[0]
#                contient a₀, coef[1] contient a₁, etc.
# Résultat : retourne P(x), la valeur du polynôme au point x
1   y = 0.0
2   k = n
3   while k >= 0 :
4       y = y * x + coefs[k]
5       k = k - 1
6   return y
```

a. Combien de fois passe-t-on par les lignes 4 et 5 si le polynôme est de degré *n* ?

b. Complétez le tableau suivant en fonction de *n*, le degré du polynôme. Vous prendrez bien sûr en compte le nombre de fois que l'on passe sur chaque ligne lors de l'exécution complète de la fonction.

Ligne	Nombre d'affectations	Nombre d'additions ou de soustractions	Nombre de multiplications	Nombre de comparaisons
1				
2				
3				
4				
5				
6				
Total				

c. Déduisez-en le temps d'exécution $T(n)$ de la fonction en microsecondes, en supposant qu'une affectation prend t_{aff} microsecondes, une addition ou une soustraction t_{add} microsecondes, une multiplication t_{mult} microsecondes, et une comparaison t_{comp} microsecondes.

d. De quelle fonction mathématique de n ce temps d'exécution $T(n)$ est-il « grand O » ?

e. Complétez l'invariant de boucle de l'algorithme de Horner : « Au début de chaque itération de la boucle while (juste avant d'exécuter la ligne 4), on sait que (complétez les pointillés) : $y = \sum_{i=0}^{n-(k+1)} a_{\dots\dots} x^i$ ».

f. A l'aide la propriété de terminaison de cet invariant, montrez que l'algorithme de Horner permet bien d'obtenir la valeur du polynôme au point x.

g. Ecrivez en langage Python la version « naïve » de la fonction valeur_polynome, qui calcule $P(x)$ selon la formule $P(x) = \sum_{k=0}^{n} a_k x^k$, c'est-à-dire en calculant pour chaque terme la puissance de x via autant de multiplications que nécessaire (on n'utilisera donc pas la fonction `pow()` ou l'opérateur `**`).

h. Combien de multiplications fait-on dans cette version naïve, si le polynôme est de degré n ?

Exercice 14 : Points fixes

Soit $n \in \mathbb{N}^*$. On s'intéresse aux points fixes de fonctions $f : E_n \to E_n$, où $E_n = [0; n-1]$. On représente une fonction par une liste L de taille n telle que pour tout x, $L[x] = f(x)$. Pour tout k entier naturel, on note f^k l'itérée k de f, c'est-à-dire $f^k = f \circ f \circ \dots \circ f$.

a. Ecrire une fonction admetPointFixe qui prend en argument une liste L de taille n et renvoie vrai si la fonction f représentée par L admet un point fixe, et faux sinon.

b. Ecrire une fonction nbPointFixe qui prend en argument une liste L de taille n et renvoie le nombre de points fixes de la fonction f représentée par L.

c. Ecrire une fonction itere qui prend en premier argument une liste L de taille n représentant une fonction f, en deuxième et troisième arguments un entier x et un entier k, et renvoie $f^k(x)$.

d. Ecrire une fonction nbPointFixeItere qui prend en premier argument un liste L de taille n représentant une fonction f, en deuxième argument un entier naturel k et renvoie le nombre de points fixes de f^k.

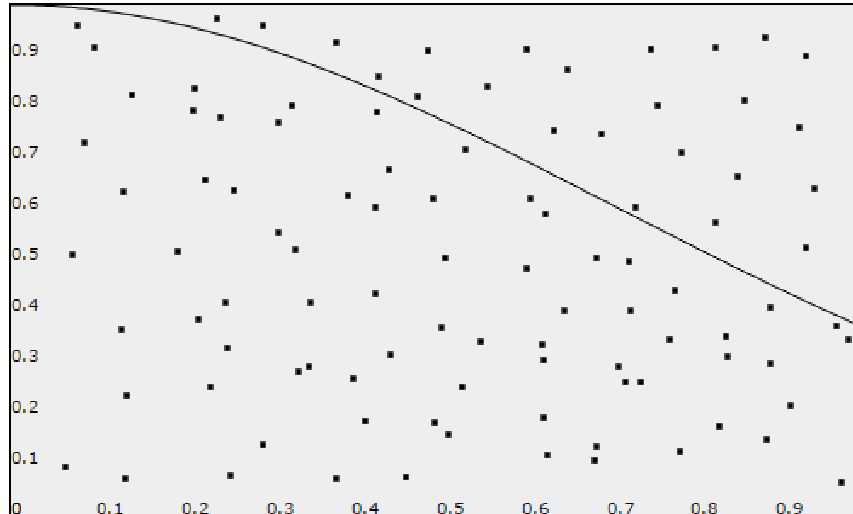
Exercice 15 : Calcul d'intégrale par la méthode de Monte-Carlo

On souhaite calculer une valeur approchée de $I = \int_0^1 e^{-x^2} \, dx$ c'est-à-dire de l'aire sous la courbe de la fonction positive f définie par $f(x) = e^{-x^2}$ entre $x = 0$ et $x = 1$.
Cette fonction f est continue, positive et strictement croissante sur [0 ;1]. La valeur maximale prise par f sur [0 ;1] est $f(0) = 1$. Soit $(0; \vec{\imath}, \vec{\jmath})$ un repère orthonormal. On considère les points : $A(0,1), B(1,1), C(1,0), D(0,0)$. On note D le domaine défini par la courbe représentative de f, la droite d'équation $x = 0$, celle d'équation $x = 1$ et l'axe des abscisses.
Soit l'expérience aléatoire suivante : on tire au hasard un point de coordonnées (x,y) dans le rectangle ABCD. On associe à cette expérience aléatoire la variable aléatoire X qui prend la valeur 1 en cas de succès (appartenance du point au domaine D) et 0 en cas d'échec. On est donc en présence d'une épreuve de Bernoulli. On note p la probabilité de succès de cette expérience aléatoire.

a. Exprimer p en fonction de I et de l'aire du rectangle ABCD.

b. On cherche à estimer au mieux p à partir de la répétition de l'expérience aléatoire. On répète donc n fois cette expérience et on note N_n la variable aléatoire qui prend pour valeur le nombre de succès obtenus lors de cette répétition de n expériences indépendantes et identiques. Quelle est la loi suivie par N_n ?

c. Ci-dessous est représenté le tirage aléatoire, de manière indépendante, de 100 points dans le rectangle ABCD.



Par lecture graphique, déterminer la valeur prise par la variable aléatoire N_n à l'issue de ces 100 tirages. En déduire une estimation de p et de I.

d. Comparer ces résultats avec la valeur exacte de I, en admettant que $I \approx 0.746824132812$.

On généralise la méthode précédente, appelée méthode de Monte-Carlo, à toute fonction f continue, positive et décroissante sur un intervalle [a ;b]. On note D le domaine défini par la courbe représentative de f, la droite d'équation $x = a$, celle d'équation $x = b$, et l'axe des abscisses.

e. Quelles sont les coordonnées des points A, B, C et D dans ce cas ? Quelle est l'aire du rectangle ABCD ?

f. Comment caractériser l'appartenance d'un point de coordonnées (x,y) au domaine D ?

g. On rappelle que l'on dispose de la fonction random du module random pour générer un nombre aléatoire pris entre 0 et 1. Quelle instruction permet d'obtenir un nombre aléatoire pris entre a et b ?

h. Ecrire la fonction monteCarlo qui prend en paramètres la fonction continue, décroissante et positive f, les nombres réels a et b, et l'entier n non nul, et qui renvoie une valeur approchée de l'intégrale I par la méthode de Monte-Carlo.

i. Donner des valeurs approchées de I obtenues pour $n = 100, n = 1000, n = 10000, n = 100000, n = 1000000$.

Contrairement à d'autres méthodes d'intégration numérique, cette méthode est stochastique (aléatoire). Elle retourne un résultat différent à chaque appel de la fonction monteCarlo.

j. Comment modifier l'algorithme si f est continue, positive et croissante ?

Exercice 16 : Compression RLE

Soit un fichier pouvant contenir des séquences répétitives d'entiers, par exemple {12, 6, 6, 6, 6, 6, 1, 3, 8, 8, 10, 2, 53, 53, 53, 53, 53, 6, 6, 6, 13}.

Une technique de compression consiste à écrire pour chaque entier le nombre de répétitions supplémentaires. On pourrait donc remplacer {53, 53, 53, 53, 53} par {53, 4}, ce qui signifie : l'entier 53, suivi d'encore 4 fois l'entier 53. Le problème de cette idée est que la séquence {53, 53, 53, 53, 53} va bien être compressée, mais qu'au contraire, toutes les séquences non répétitives vont être dilatées : par exemple, {10} serait réécrit en {10, 0}.

Une meilleure idée consiste à ne transformer une séquence que si un entier est immédiatement répété au moins une fois. Ainsi, {53, 53, 53, 53, 53} serait réécrite en {53, 53, 3}, {10} serait inchangée et {8, 8} serait réécrite en {8, 8, 0}. La séquence donnée en exemple serait donc réécrite de la façon suivante : {12, 6, 6, **3**, 1, 3, 8, 8, **0**, 10, 2, 53, 53, **3**, 6, 6, **1**, 13}. Lors de la décompression, c'est le fait d'avoir deux entiers identiques à la suite qui indique que l'entier suivant est un nombre d'occurrences supplémentaires et non un entier du fichier de départ. **C'est à cette seconde idée que vous allez vous intéresser dans cet exercice**. Il s'agit de la compression dite « RLE » (Run Length Encoding). Notez que bien que ce second algorithme soit la plupart du temps meilleur que le premier, il n'est pas parfait pour autant : il existe tout de même des cas pour lesquels il dilate au lieu de compresser. Il favorise les successions de plus de trois caractères identiques, il défavorise les successions de deux et laisse inchangé les simples occurrences.

a. Supposons que le fichier d'entrée contient 10 entiers. Donnez un exemple de contenu de fichier qui donnerait la plus petite taille possible pour le fichier de sortie, puis un exemple de contenu qui donnerait la plus grande taille possible en sortie.

b. Donner le code Python de la procédure de compression, dont l'entête est la suivante :

```
def compresser (nomFichierEntree, nomFichierSortie)
# Préconditions : nomFichierEntree est le nom d'un fichier contenant une séquence
# d'entiers
# Postconditions : un nouveau fichier nommé comme spécifié dans nomFichierSortie est
# créé, son contenu correspond à la séquence d'entiers compressée par l'algorithme RLE
```

c. Donner le code Python de la procédure de décompression, dont l'entête est la suivante :

```
def decompresser (nomFichierEntree, nomFichierSortie)
# Préconditions : nomFichierEntree est le nom d'un fichier compressé selon l'algo RLE
# Postconditions : un nouveau fichier nommé comme spécifié dans nomFichierSortie est
# créé, son contenu correspond à la décompression du fichier d'entrée
```

Exercice 17 : Hauteur d'un arbre binaire

Ecrire en langage Python la fonction membre qui détermine la hauteur d'un arbre binaire. La classe Arbre étant défini ainsi :

```
class Noeud :
    def __init__(self,info,fg,fd):
        self.info = info
        self.fg = fg
        self.fd = fd

class Arbre :
    def __init__(self):
        self.adRacine = []

    def hauteur(self):
        # TODO
```

Exercice 18 : Tester si un arbre binaire est un arbre binaire de recherche

Ecrire en langage Python une fonction membre qui renvoie vrai si un arbre binaire est un arbre binaire de recherche et faux sinon. Rappel : pour qu'un arbre soit un arbre binaire de recherche (ABR) la valeur de chaque noeud doit être comprise entre le maximum du fils gauche et le minimum du fils droit.

Exercice 19 : Inversion d'une File en utilisant une Pile

Le but de cet exercice est d'écrire en Python une procédure qui inverse une file d'entiers (FIFO) qui lui est passée en paramètre. On demande de ne pas utiliser de tableau ou de liste de travail pour effectuer l'inversion, mais d'utiliser plutôt une pile (LIFO). Il existe en effet une méthode très simple pour inverser une file en utilisant une pile.

Exercice 20 : Classe Pile

Construire une classe implémentant les piles (sans utiliser le module queue de Python), incluant :

- un constructeur de paramètres self et une liste L
- un attribut liste
- la surcharge de la méthode d'affichage
- des méthodes empiler, depiler, sommet, estVide, hauteur et pop

Exercice 21 : Classe File

Construire une classe implémentant les files (sans utiliser le module queue de Python), incluant :

- un constructeur de paramètres self et une liste L
- un attribut liste
- la surcharge de la méthode d'affichage
- des méthodes pop, push, estVide et longueur

Exercice 22 : Validité du parenthésage d'une expression

Un problème fréquent pour les compilateurs et les traitements de textes est de déterminer si les parenthèses d'une chaîne de caractères sont équilibrées et proprement incluses les unes dans les autres. On désire donc écrire une fonction qui teste la validité du parenthésage d'une expression :

– on considère que les expressions suivantes sont valides : "()", "[([bonjour+]essai)7plus-];"

– alors que les suivantes ne le sont pas : "(", ")(", "4(essai]".

Notre but est donc d'évaluer la validité d'une expression en ne considérant que ses parenthèses et ses crochets. On suppose que l'expression à tester est dans une chaîne de caractères, dont on peut ignorer tous les caractères autres que '(', '[', ']' et ')'. Écrire en Python la fonction valide qui renvoie vrai si l'expression passée en paramètre est valide, faux sinon.

Exercice 23 : Parcours préfixé itératif d'un arbre binaire

Ecrire en langage Python la procédure **itérative** du parcours préfixé dans un arbre binaire.

Exercice 24 : Recherche itérative dans un arbre binaire quelconque

Ecrire en langage Python une fonction **itérative** qui cherche un élément dans un arbre binaire quelconque. Comparez le coût de cette recherche avec celui de la recherche dans un ABR.

Exercice 25 Définition du type nombre complexe

a. Définissez le type `NbComplexe` à l'aide d'une classe. Rappel : un nombre complexe est défini par sa partie réelle (que vous pourrez appeler `re`) et sa partie imaginaire (que vous pourrez appeler `im`).

b. Ajoutez trois fonctions membres à cette classe : une pour saisir le nombre complexe au clavier (initialisation des données membres) et une pour afficher le nombre complexe à l'écran sous le format : `re +im i` (si la partie imaginaire est positive ou nulle, et `re -im i` si elle est négative), et une pour copier le contenu d'un autre nombre complexe.

c. Créez un programme principal qui crée un nombre complexe, l'affiche, puis le saisi et le réaffiche. Qu'obtenez-vous ?

Exercice 26 : Fonctions membres

a. Ajoutez à la classe `NbComplexe` une procédure membre `multiplier` qui multiplie le nombre complexe par un autre nombre complexe passé en paramètre. L'instance courante contient le résultat de la multiplication, le nombre complexe avec lequel on multiplie n'est pas modifié.

Rappel : $(re_1 + im_1\, i) \times (re_2 + im_2\, i) = (re_1 \times re_2 - im_1 \times im_2) + (im_1 \times re_2 + re_1 \times im_2)\, i$

b. Complétez le programme principal en ajoutant la création des deux nouveaux nombres complexes (vous choisirez les valeurs), les multipliez et affichez le résultat.

c. Ajoutez dans la classe `NbComplexe` une fonction `module` qui retourne le module du nombre complexe. Rappel : $|re + im\, i| = \sqrt{re^2 + im^2}$

d. Ajoutez une fonction `estPlusPetit` qui indique si le nombre complexe est plus petit qu'un autre nombre complexe passé en paramètre. La comparaison se fera sur les valeurs des modules des deux nombres.

Exercice 27 : Création d'un tableau de nombres complexes aléatoires

a. Dans le programme principal, remplissez un tableau avec des nombres complexes dont les parties réelles et imaginaires sont tirées aléatoirement dans l'intervalle [-10,10] (avec 1 décimale de précision).

b. Affichez le tableau de nombres complexes ainsi remplit et pour chaque élément du tableau affichez également le module du nombre complexe.

Exercice 28 : Tri par sélection du tableau de nombres complexes

Définissez une procédure globale (pas une procédure membre) `trierParSelection` qui prend en paramètres un tableau de nombres complexes et qui le trie du nombre le plus petit au plus grand (en termes de module), en utilisant l'algorithme de tri par sélection. Testez votre procédure dans le programme principal.

Exercice 29 : Tri par insertion du tableau de nombres complexes

Définissez une procédure globale (pas une procédure membre) `trierParInsertion` qui prend en paramètres un tableau de nombres complexes et qui le trie du nombre le plus petit au plus grand (en termes de module), en utilisant l'algorithme de tri par insertion. Testez votre procédure dans le programme principal.