RAPPORT PROGRAMMATION

2023/2024

Langage : PYTHON

AHNOUDJ Lina
SADDOUGUI Sacha

SOMMAIRE

Problème intersections de listes

Dans la première partie de cet exercice, on nous donne deux listes triées dans l'ordre croissant.

Il faut simplement créer une nouvelle liste et y ajouter chaque élément commun aux deux listes.

· Difficulté :

La difficulté principale de cet exercice est d'ajouter les éléments en commun une seule fois dans la liste finale. C'est-à-dire que si la première liste contient 3 fois le nombre « 6 », et la seconde liste contient 2 fois le nombre « 6 », dans la liste des éléments communs, le chiffre « 6 » ne devra apparaître qu'une seule fois.

· Solution :

Pour résoudre ce problème, il faut mettre en place une condition qui vérifie la présence ou non d'un élément dans la liste commune avant de l'ajouter.

· Les algorithmes à implémenter :

Dans un premier temps, nous avons implémenté une fonction auxiliaire permettant de vérifier que les listes sont bien triées dans l'ordre croissant.

```
################################################################################
Fonction verif_triee(L : liste) : booléen

    // Algorithme de vérification de liste triée
    Pour i de 0 à longueur(L) - 2 faire
        // Comparaison des éléments adjacents
        Si L[i] > L[i+1] alors
            // La liste n'est pas triée
            Retourner Faux
        Fin Si
    Fin Pour
    // La liste est triée
```

```
    Retourner Vrai
Fin Fonction
```

##

Ensuite, nous avons implémenté la fonction principale permettant de résoudre le problème.

##

```
Fonction intersection_2_listes(L1 : liste, L2 : liste) : liste
    // Vérification que les listes sont triées
    assert (verif_triee(L1) et verif_triee(L2)), "Les listes ne sont pas triées, il faut les modifier"

    // Initialisation de la liste qui va contenir les valeurs communes
    L3 = []

    // Parcours de la liste L1
    Pour i de 0 à longueur(L1) - 1 faire
        // Si la valeur de L1 est dans L2 et n'est pas dans L3, on l'ajoute à L3
        Si (L1[i] appartient à L2) et (L1[i] n'appartient pas à L3) alors
            Ajouter L1[i] à L3
        Fin Si
    Fin Pour

    // Retour de la liste L3 contenant les valeurs communes
    Retourner L3
Fin Fonction
```

##

Problème intersections de liste de liste

Dans la deuxième partie de cet exercice, on nous fournit une liste de listes, où chaque élément est une liste triée en ordre croissant. On nous demande de chercher les éléments qui sont présents dans chacune des sous-listes et de les ajouter dans une nouvelle liste.

· Difficulté :

La difficulté principale de cet exercice est le traitement de toutes les sous-listes en même temps, sachant qu'un élément peut être en première position dans une liste et en cinquième dans une autre.

· Solution :

Pour résoudre ce problème, le plus adapté est de se servir de la fonction précédente. On l'applique sur la première sous-liste, la deuxième sous-liste, et on stocke le résultat dans une liste que l'on va nommer L3. Ensuite, on applique encore la fonction précédente mais cette fois sur la liste L3 et la sous-liste suivante, que l'on stocke dans L3. Ainsi de suite…

· Les algorithmes à implémenter :

On utilise deux fonctions auxiliaires, intersection_2_listes et verif_triee, implémentées précédemment. On implémente la fonction intersection afin de résoudre le problème.

```
################################################################################
Fonction intersection(L : liste de listes) : liste
    // Vérification que chaque liste de la liste de listes est triée
    Pour i de 0 à longueur(L) - 1 faire
        assert (verif_triee(L[i])), "Les listes ne sont pas triées, il faut les modifier"
    Fin Pour

    // Initialisation de la liste qui va contenir les valeurs communes
    L3 = L[0]

    // Initialisation du compteur
    compteur = 0
```

```
        // Parcours de chaque élément de la liste L
        Tant que compteur < longueur(L) faire
                // Intersection de la liste courante avec L3
                L3 = intersection_2_listes(L3, L[compteur])

                // Incrémentation du compteur pour passer à la liste suivante
                compteur += 1
        Fin Tant que

        // Retour de la liste L3 contenant les valeurs communes
        Retourner L3
Fin Fonction
```

Problème de la conversion

Dans cet exercice, on doit implémenter la structure de données « Pile ». À l'aide de cette pile, il faut convertir un nombre décimal en nombre binaire.

· Difficulté :

La difficulté principale de cet exercice est l'implémentation et la gestion de la pile. Le code principal ne contient pas de difficulté particulière si l'on connaît la formule pour convertir un nombre décimal en binaire grâce à la division.

· Solution :

"Pour résoudre ce problème, le plus adapté est d'utiliser une classe pour implémenter la pile car dans le code principal c'est bien plus simple de l'utiliser comme ça. Ensuite, il suffit d'empiler le reste de la division et de dépiler quand on a fini toutes les divisions.

· Les algorithmes à implémenter :

On implémente la classe Pile ainsi que la fonction conversion_dec_bin afin de résoudre le problème.

```
##############################################################################
Classe Pile:
        // Implémentation de la pile
        Fonction pile_vide() : Pile
                Retourner une nouvelle pile (une liste vide)

        Fonction est_vide(p : Pile) : booléen
                Retourner vrai si la pile p est vide, sinon faux

        Fonction sommet(p : Pile) : élément
                // Vérifier que la pile n'est pas vide
                        Si longueur(p) == 0 alors
```

```
                Afficher "La pile est vide"
                Arrêter l'exécution
                Fin Si
        // Renvoyer le dernier élément de la pile
        Retourner p[dernier indice]


    Procédure empiler(p : Pile, e : élément)
        // Ajouter un élément à la pile
        Ajouter e à la fin de p

    Fonction depiler(p : Pile) : élément
        // Vérifier que la pile n'est pas vide
        Si longueur(p) == 0 alors
            Afficher "La pile est vide"
            Arrêter l'exécution
        Fin Si
    // Enlever le dernier élément de la pile
    Retourner et supprimer le dernier élément de p
Fin Classe

################################################################################
Fonction conversion_dec_bin(nombreDécimal : entier) : liste
    // Initialisation de la pile et de la liste contenant le résultat binaire
    pile = Pile()
    liste = liste_vide()
    listeBis = liste_vide()

    // Effectuer une division par 2 tant que le nombre n'est pas nul
    Tant que nombreDécimal n'est pas égal à 0 faire
        // Empiler le reste de la division par 2
        pile.empiler(liste, nombreDécimal % 2)

        // Diviser le nombre par 2
```

```
            nombreDécimal = nombreDécimal // 2
        Fin Tant que

        // Dépiler la pile et ajouter les éléments à la liste
        Tant que non pile.est_vide(liste) faire
            // Dépiler dans une liste
            listeBis.ajouter_element(pile.depiler(liste))
        Fin Tant que

        // Retourner la liste contenant la représentation binaire
        Retourner listeBis
Fin Fonction
##########################################################################################
```

Problème du mot de passe

Dans cet exercice, on doit simplement vérifier qu'un mot de passe contient une majuscule, une minuscule, un chiffre et un caractère spécial.

· Difficulté :

La difficulté principale de cet exercice est de vérifier que toutes les conditions sont remplies en même temps.

· Solution :

Pour résoudre ce problème, le plus adapté est d'utiliser la bibliothèque regex qui est très efficace pour cela.

· Les algorithmes à implémenter :

On importe la bibliothèque regex et on implémente la fonction mot_de_passe. On implémente également une fonction test.

```
# On importe la librairie regex qui est très efficace pour ce genre de cas
import re

# Définition de la fonction mot_de_passe avec un paramètre passwd
fonction mot_de_passe(passwd):
    # On vérifie que le mot de passe contient une minuscule
    minuscule = re.search("[a-z]", passwd)

    # On vérifie que le mot de passe contient une majuscule
    majuscule = re.search("[A-Z]", passwd)
```

```
        # On vérifie que le mot de passe contient un chiffre
        chiffre = re.search("[0-9]", passwd)
        # On vérifie que le mot de passe contient un caractère spécial
        special = re.search("[!@#$%^&*()_+={}|:;'<>.?]", passwd)

        # Si toutes les conditions sont respectées, on retourne True
        si minuscule et majuscule et chiffre et special:
                retourner True
        sinon:
                retourner False

# Définition de la fonction test_mot_de_passe
fonction test_mot_de_passe():
        # On teste la fonction avec des cas spécifiques
        si mot_de_passe("azertyuiop") est faux alors
                afficher "Verification 1 exercice 3 a échoué"
        si mot_de_passe("AZERTYUIOP") est faux alors
                afficher "Verification 2 exercice 3 a échoué"
        si mot_de_passe("azertyuiopAZERTYUIOP") est faux alors
                afficher "Verification 3 exercice 3 a échoué"
        si mot_de_passe("azertyuiopAZERTYUIOP1234567890") est faux alors
                afficher "Verification 4 exercice 3 a échoué"
        si mot_de_passe("azertyuiopAZERTYUIOP1234567890!") est vrai alors
                afficher "Verification 5 exercice 3 a réussi"
        si mot_de_passe("azertyuiopAZERTYUIOP1234567890!@#$%^&*()_+={}|:;'<>.?") est vrai
        alors
                afficher "Verification 6 exercice 3 a réussi"
        si mot_de_passe("") est faux alors
                afficher "Verification 7 exercice 3 a échoué"
        retourner vrai
```

Problème HTML

Dans la première partie de cet exercice, on nous demande de vérifier qu'un code HTML est bien équilibré et que les balises correspondent entre elles. Dans un second temps, on nous demande de compter le nombre d'occurrences de chaque balise présente dans le code HTML.

· Difficulté :

La difficulté principale de cet exercice, selon nous, est de ne prendre en compte que la balise et pas les arguments qui suivent, mais aussi de s'assurer qu'une balise ouvrante correspond bien à sa balise fermante.

· Solution :

Pour résoudre ce problème, il faut veiller à mettre les bonnes conditions au bon endroit ainsi que choisir une méthode qui nous convient pour ce problème. En effet, il y a énormément de manières de résoudre ce problème. Selon nous, le plus simple est de comparer une liste de balises ouvrantes et une liste de balises fermantes. Pour compter les occurrences, avec une bonne utilisation du dictionnaire, le problème n'est pas compliqué.

· Les algorithmes à implémenter :

Dans un premier temps, nous avons implémenté la fonction verif_html qui permet de vérifier que le fichier est bien équilibré. Puis, dans un second temps, nous avons compté les occurrences de chaque balise dans ce fichier à l'aide de la fonction compteur_occurrence.

```
fonction verif_HTML(htmlTXT)
        fichier <- ouvrir_fichier(htmlTXT, "r")
        fichiertxt <- lire_fichier(fichier)

        listeManip <- liste_vide
        listeBaliseOuvrante <- liste_vide
        listeBaliseFermante <- liste_vide
```

```
pour i de 0 à longueur(fichiertxt) faire
    si fichiertxt[i] = "<" et fichiertxt[i+1] != "/" alors
        i <- i + 1
        tant_que fichiertxt[i] != ">" faire
            ajouter_element(listeManip, fichiertxt[i])
            i <- i + 1

            si fichiertxt[i] = "/" alors
                listeManip <- liste_vide
                sortir du boucle
            fin si

            si fichiertxt[i] = " " alors
                sortir du boucle
            fin si
        fin tant_que

        si concatener(listeManip) ≠ "meta" et concatener(listeManip) ≠ "!DOCTYPE"
        alors
            ajouter_element(listeBaliseOuvrante, concatener(listeManip))
            listeManip <- liste_vide
        fin si
        listeManip <- liste_vide
    sinon si fichiertxt[i] = "<" et fichiertxt[i+1] = "/" alors
        i <- i + 2
        tant_que fichiertxt[i] != ">" faire
            ajouter_element(listeManip, fichiertxt[i])
            i <- i + 1
        fin tant_que
        ajouter_element(listeBaliseFermante, concatener(listeManip))
        listeManip <- liste_vide
    fin si
```

```
        fin pour

        si longueur(listeBaliseFermante) = longueur(listeBaliseOuvrante) alors
                pour i de 0 à longueur(listeBaliseFermante) faire
                        si listeBaliseFermante[i] dans listeBaliseOuvrante alors
                                supprimer_element(listeBaliseOuvrante, listeBaliseFermante[i])
                        fin si
                fin pour
        fin si

        si longueur(listeBaliseOuvrante) = 0 alors
                retourner Vrai
        sinon
                retourner Faux
        fin si
fin fonction

###################################
fonction compteur_occurrence(htmlTXT)
        fichier <- ouvrir_fichier(htmlTXT, "r")
        fichiertxt <- lire_fichier(fichier)

        si verif_HTML(htmlTXT) ≠ Vrai alors
                retourner "Le fichier n'est pas un fichier html"
        sinon
                listeBalise <- liste_vide
                listeBaliseOuvrante <- liste_vide

                pour i de 0 à longueur(fichiertxt) faire
                        si fichiertxt[i] = "<" et fichiertxt[i+1] ≠ "/" alors
                                i <- i + 1
                                tant_que fichiertxt[i] ≠ ">" faire
                                        ajouter_element(listeBalise, fichiertxt[i])
```

```
                                        i <- i + 1

                                        si fichiertxt[i] = " " alors
                                                sortir du boucle
                                        fin si
                                fin tant_que


                                ajouter_element(listeBaliseOuvrante, concatener(listeBalise))
                                listeBalise <- liste_vide
                        fin si
                fin pour

                dico <- dictionnaire_vide

                pour i de 0 à longueur(listeBaliseOuvrante) faire
                        si listeBaliseOuvrante[i] dans dico alors
                                dico[listeBaliseOuvrante[i]] <- dico[listeBaliseOuvrante[i]] + 1
                        sinon
                                dico[listeBaliseOuvrante[i]] <- 1
                        fin si
                fin pour

                retourner dico
        fin si
        fin fonction

###################################
```

Problème de la date du fichier

Dans cet exercice, on nous demande simplement d'afficher les fichiers présents dans un répertoire dont la date est supérieure à celle que l'on donne. Cependant, il y a plusieurs contraintes : utiliser les modules os et datetime, saisir le répertoire au clavier et la date limite.

· Difficulté :

La difficulté principale de cet exercice est d'afficher uniquement les fichiers qui ont été modifiés après la date donnée

· Solution :

Pour résoudre ce problème, la solution est de bien gérer les valeurs de l'affichage de la date. Nous avons implémenté la fonction auxiliaire gestion_date pour gérer ce problème.

· Les algorithmes à implémenter :

Dans un premier temps, nous avons implémenté la fonction auxiliaire gestion_date, puis nous avons implémenté la fonction principale fichier_repertoire.

```
###################################

fonction gestion_date(date)
   liste_date <- diviser_chaine(date, " ")

   si liste_date[1] = "Jan" alors
      liste_date[1] <- "01"
```

```
    sinon si liste_date[1] = "Feb" alors
        liste_date[1] <- "02"
    sinon si liste_date[1] = "Mar" alors
        liste_date[1] <- "03"
    sinon si liste_date[1] = "Apr" alors
        liste_date[1] <- "04"
    sinon si liste_date[1] = "May" alors
        liste_date[1] <- "05"
    sinon si liste_date[1] = "Jun" alors
        liste_date[1] <- "06"
    sinon si liste_date[1] = "Jul" alors
        liste_date[1] <- "07"
    sinon si liste_date[1] = "Aug" alors
        liste_date[1] <- "08"
    sinon si liste_date[1] = "Sep" alors
        liste_date[1] <- "09"
    sinon si liste_date[1] = "Oct" alors
        liste_date[1] <- "10"
    sinon si liste_date[1] = "Nov" alors
        liste_date[1] <- "11"
    sinon si liste_date[1] = "Dec" alors
        liste_date[1] <- "12"
    fin si

    pour i de 0 à longueur(liste_date) - 1 faire
        si liste_date[i] = "" alors
            supprimer_element(liste_date, liste_date[i])
        fin si
    fin pour

    retourner concatener(liste_date[2], "/", liste_date[1], "/", liste_date[4])
fin fonction
```

```
##################

fonction fichier_repertoire()
   // L'utilisateur saisit le nom du répertoire, son chemin, et une date de modification
   chemin_du_repertoire <- saisir("Entrez le nom du répertoire et son chemin: ")
   date <- saisir("Entrez la date de modification du fichier: ")

   // Liste les fichiers et dossiers dans un répertoire donné
   element_du_doss <- lister_contenu_repertoire(chemin_du_repertoire)

   // Parcourt la liste des fichiers et dossiers
   pour i de 0 à longueur(element_du_doss) faire
      // Récupération de la date de modification du fichier en utilisant la fonction auxiliaire
gestion_date
      temps_modification <- obtenir_date_modification(element_du_doss[i])
      dateFinal <- gestion_date(temps_modification)

      // Comparaison des dates
      dateComp <- diviser_chaine(dateFinal, "/")
      dateDemande <- diviser_chaine(date, "/")

      cond <- dateComp[2] > dateDemande[2]
      cond0 <- dateComp[2] >= dateDemande[2]
      cond1 <- dateComp[1] > dateDemande[1]
      cond2 <- dateComp[1] >= dateDemande[1]
      cond3 <- dateComp[0] >= dateDemande[0]

      // Affichage du fichier si la date de modification est après la date demandée
      si (cond) ou (cond0 et cond1) ou (cond0 et cond2 et cond3) alors
         afficher("Le fichier " + element_du_doss[i] + " correspond à la demande")
      fin si
   fin pour
```

```
fin fonction

###################################
```

Algorithme des fonctions test et Main

```
###################################
# Test de la fonction intersection_2_listes
fonction test_intersection_2_listes():
    # On teste la fonction avec des cas spécifiques
    si intersection_2_listes([1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5,6,7,8,9,10]) == [1,2,3,4,5,6,7,8,9,10] alors
        afficher "verification1 exercice 1.1"
    fin si

    si intersection_2_listes([1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5,6,7,8,9]) == [1,2,3,4,5,6,7,8,9] alors
        afficher "verification2 exercice 1.1"
    fin si

    si intersection_2_listes([1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5,6,7,8]) != [1,2,4,5,6,7,8] alors
        afficher "verification3 exercice 1.1"
    fin si

    si intersection_2_listes([1,2,3,4,5,6,7,8,9,10], []) == [] alors
        afficher "verification4 exercice 1.1"
    fin si

    si intersection_2_listes([9,9,9,9,9,9], [9,9,9,9,9,9,9,9]) == [9] alors
        afficher "verification5 exercice 1.1"
```

```
    fin si

    retourner vrai
fin fonction
###################################
# Test de la fonction intersection
fonction test_intersection():
    # On teste la fonction avec des cas spécifiques
    si intersection([[1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5,6,7,8,9,10]]) == [1,2,3,4,5,6,7,8,9,10] alors
        afficher "verification1 exercice 1.2"
    fin si

    si intersection([[1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5,6,7,8,9]]) == [1,2,3,4,5,6,7,8,9] alors
        afficher "verification2 exercice 1.2"
    fin si

    si intersection([[1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5,6,7,8]]) != [1,2,4,5,6,7,8] alors
        afficher "verification3 exercice 1.2"
    fin si

    si intersection([[1,2,3,4,5,6,7,8,9,10], [1,2,5], []]) == [] alors
        afficher "verification4 exercice 1.2"
    fin si

    si intersection([[9,9,9,9,9,9], [9,9,9,9,9,9,9,9], [9,9,9]]) == [9] alors
        afficher "verification5 exercice 1.2"
    fin si

    si intersection([[1,2,3,4,5,6,7,8,9,10], [1,2,3,4,5,6,7,8], [1,2,3,4,5,6,7], [12]]) == [] alors
        afficher "verification6 exercice 1.2"
    fin si
```

```
    retourner vrai
fin fonction
 ###################################

# Test de la fonction convertion_dec_bin
fonction test_convertion_dec_bin():
    # On teste la fonction avec des cas spécifiques
    si convertion_dec_bin(10) == [1,0,1,0] alors
        afficher "verification1 exercice 2"
    fin si

    si convertion_dec_bin(0) == [] alors
        afficher "verification2 exercice 2"
    fin si

    si convertion_dec_bin(1) == [1] alors
        afficher "verification3 exercice 2"
    fin si

    si convertion_dec_bin(2) == [1,0] alors
        afficher "verification4 exercice 2"
    fin si

    si convertion_dec_bin(3) == [1,1] alors
        afficher "verification5 exercice 2"
    fin si

    si convertion_dec_bin(4) == [1,0,0] alors
        afficher "verification6 exercice 2"
    fin si
```

```
si convertion_dec_bin(5) == [1,0,1] alors
   afficher "verification7 exercice 2"
fin si

si convertion_dec_bin(6) == [1,1,0] alors
   afficher "verification8 exercice 2"
fin si

si convertion_dec_bin(7) == [1,1,1] alors
   afficher "verification9 exercice 2"
fin si

si convertion_dec_bin(8) == [1,0,0,0] alors
   afficher "verification10 exercice 2"
fin si

si convertion_dec_bin(9) == [1,0,0,1] alors
   afficher "verification11 exercice 2"
fin si

si convertion_dec_bin(10) == [1,0,1,0] alors
   afficher "verification12 exercice 2"
fin si

si convertion_dec_bin(11) == [1,0,1,1] alors
   afficher "verification13 exercice 2"
fin si

si convertion_dec_bin(12) == [1,1,0,0] alors
   afficher "verification14 exercice 2"
```

```
    fin si

    retourner vrai
###################################

# Test de la fonction mot_de_passe
fonction test_mot_de_passe():
    # On teste la fonction avec des cas spécifiques
    si mot_de_passe("azertyuiop") == faux alors
        afficher "verification1 exercice 3"
    fin si

    si mot_de_passe("AZERTYUIOP") == faux alors
        afficher "verification2 exercice 3"
    fin si

    si mot_de_passe("azertyuiopAZERTYUIOP") == faux alors
        afficher "verification3 exercice 3"
    fin si

    si mot_de_passe("azertyuiopAZERTYUIOP1234567890") == faux alors
        afficher "verification4 exercice 3"
    fin si

    si mot_de_passe("azertyuiopAZERTYUIOP1234567890!") == vrai alors
        afficher "verification5 exercice 3"
    fin si

    si mot_de_passe("azertyuiopAZERTYUIOP1234567890!@#$%^&*()_+={}|:;'<>.?") == vrai alors
        afficher "verification6 exercice 3"
    fin si
```

```
    si mot_de_passe("") == faux alors
        afficher "verification7 exercice 3"
    fin si

    retourner vrai
fin fonction

################################################
# Test de la fonction verif_HTML
fonction test_verif_HTML():
    # On teste la fonction avec des cas spécifiques
    si verif_HTML("test.txt") == vrai alors
        afficher "verification1 exercice 4"
    fin si

    si verif_HTML("test1.txt") == faux alors
        afficher "verification2 exercice 4"
    fin si

    retourner vrai
fin fonction
###################################
# Fonction principale
fonction main():
    si test_intersection_2_listes() alors
        afficher "test_intersection_2_listes() : OK"
    sinon
        afficher "test_intersection_2_listes() : ERREUR"
    fin si
```

```
si test_intersection() alors
   afficher "test_intersection() : OK"
sinon
   afficher "test_intersection() : ERREUR"
fin si

si test_convertion_dec_bin() alors
   afficher "test_convertion_dec_bin() : OK"
sinon
   afficher "test_convertion_dec_bin() : ERREUR"
fin si

si test_mot_de_passe() alors
   afficher "test_mot_de_passe() : OK"
sinon
   afficher "test_mot_de_passe() : ERREUR"
fin si

si test_verif_HTML() alors
   afficher "test_verif_HTML() : OK"
sinon
   afficher "test_verif_HTML() : ERREUR"
fin si
```

Répartition du travail

Nous avons réparti le travail de la manière suivante :

Lina :

- Exercices 1 et 2
- Chaque membre du groupe a travaillé sur ses propres exercices dans le rapport
- Mise en page une fois le rapport fini

Sacha :

- Exercices 3, 4 et 5
- Chaque membre du groupe a travaillé sur ses propres exercices dans le rapport