Gestion de mémoire

Amédée COSTES
Lin GUO

27 février 2016

Résumé

L'objet principal de ce dossier est de programmer un gestionnaire automatique de mémoire sous le langage python. La robutesse du code a été testée avec des cas tests manufacturés.

Table des matières

1 Introduction

Le langage python a été créé au milieu des années 90 afin de rassembler en un langage interprété tous les éléments pratiques et simples des autres langages pour rendre le python accessible et rapide d'utilisation. De nombreuses librairies ont ainsi été créées au fur et à mesure des années ce qui permet à ce langage de disposer aujourd'hui d'une bibliothèque parmis les plus riches.

L'aspect pratique de ce code permet de réaliser de très nombreuses tâches simplement, comme par exemple la gestion automatique de mémoire sur un ordinateur. Le présent rapport expose les résultats du projet de gestion de mémoire, effectuée à l'aide d'un code en langage python et dont le but est une fusion intelligente de deux dossiers dans un troisième.

2 Création de class Dossier

Il est apparu rapidement que la manipulation des dossiers serait largement facilitée par l'utilisation d'une classe qui a été assez logiquement nommée "Dossier". La partie ci-dessous présente l'initialisation de cette classe.

2.1 Initialisation

```
1  class Dossier:
2      def __init__(self,name,workFolder):
3          """---------- name = the name of the folder ----------"""
4          self._name = name
5          """---------- workFolder= the path of the folder : name----------"""
6          self._workFolder=workFolder
7          """---------- path= the path of the folder : name----------"""
8          self.path=self._workFolder+'/'+self._name
```

2.2 Contenu du répertoire

Les méthodes ele_dir, folder_dir, file_dir renvoient respectivement la liste des éléments (dossiers + fichiers), des dossiers et des fichiers qui sont contenus dans le répertoire, et un message d'erreur si le répertoire n'existe pas.

```
1  """---------- elements dans un repertoire ----------"""
2      def ele_dir(self,folder_name):
3          if os.path.exists(self._workFolder + '/'+folder_name+'/'):
4              return mylistdir(self._workFolder+ '/'+folder_name)
5          else:
6              print "the folder doesn't exists"
7
8      """---------- folders in the directory ----------"""
9      def folder_dir(self,folder_name):
10         if os.path.exists(self._workFolder + '/'+folder_name+'/'):
11             return listfolderdir(self._workFolder+ '/'+folder_name)
12         else:
13             print "the folder doesn't exists"
14     """---------- files in the directory ----------"""
15     def file_dir(self,folder_name):
16         if os.path.exists(self._workFolder + '/'+folder_name+'/'):
17             return listfiledir(self._workFolder+ '/'+folder_name)
18         else:
19             print "the file doesn't exists"
```

2.3 création d'un répertoire

La méthode new permet de vérifier l'éventuelle existence du répertoire que l'on souhaite créer. Dans le cas où celui-ci existe, la méthode le supprime avant de le recréer.

```
1     def new(self,folder_name):
2         if os.path.exists(self._workFolder + '/'+folder_name+'/'):
3                 shutil.rmtree(self._workFolder + '/'+folder_name+'/')
4         os.mkdir(self._workFolder + '/'+folder_name+'/')
```

3 Les hypothèses

Le programme réalisé lors de ce projet n'est sensé fonctionné que sous un certain nombre d'hypothèses vis-à-vis de la structure des contenus et des choix de traitement des données. Les hypothèses faites sont présentées ci-dessous.

Hypothèse 1

Le dossier A (resp. B) ne contient pas deux fichiers qui possèdent le même nom.

Hypothèse 2

Le dossier A (resp. B) ne contient pas deux dossiers qui possèdent le même nom.

Hypothèse 3

Les dossiers A et B continnent au plus deux dossiers(resp. fichiers) qui ont le même nom (un dans A et un dans B) S'il ne sont pas tous les deux dans un dossier qui port le même nom, l'un des deux se situe à la racine de sauvegarde.

Hypothèse 4

Si deux dossiers ont le même nom (un dans A, renommé ici d_A et un dans B, renommé ici d_B), on conserve l'emplacement du plus profond dans l'arbre de sauvegarde en conservant la totalité du contenu distinct de d_A et de d_B.

Hypothèse 5

Si deux fichiers ont le même nom (un dans A, renommé ici d_A et un dans B, renommé ici d_B), on conserve l'emplacement du plus profond dans l'arbre de sauvegarde et le fihcier le plus récent. Un message à l'attention du destinataire est alors affiché pour lui spécifier ces changements.

4 Algorithme

4.1 Principe de fonctionnement général

Le fonctionnement du programme créé dans le cadre de ce projet est présenté dans les sections suivantes. D'une manière générale, il a été choisit de parcourir chacun des dossier père une fois, d'abords au niveau un des dossiers contenus puis à la racine et enfin en profondeur :

4.1.1 Etape 1 : Création de liste doublon_folder

- création d'une liste doublon_folder de dossiers existant en double
- création d'une liste contenant les chemins des dossiers de doublon_folder à la racine
- création d'une liste contenant les chemins les plus longs des dossiers de doublon_folder

Grace à l'hypothèse 3, on a pu construire une liste des dossiers qui ont de même nom en comparant les dossiers situés à la racine de A aux dossiers situés au profondeur de B, et inversement.
Cette liste est une liste de trois sous-suites, la premières sous-suite contient les noms de dossers doublons, la deuxième contient les chemins à la racine , la troixième contient les chemins plus pronfonds

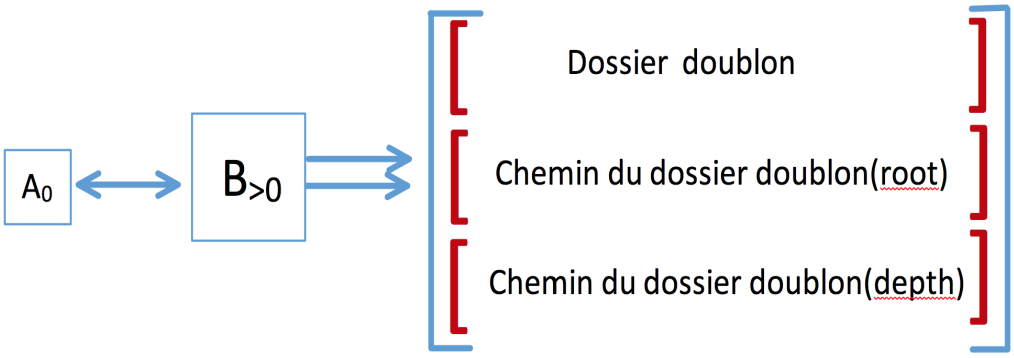

FIGURE 1 – Listes des dossiers doublons

4.1.2 Etape 2 : Copie à la racine des dossiers et création de liste doublon_file

- détermination des fichiers doublons en profondeurs
- création d'un liste doublon_file de fichiers existant en double
- création d'un liste doublon_file_path de chemins des fichiers existant en double

- Copie des dossiers à la racine du premier "grand dossier père" qui ne sont pas en double et listage des fichiers de ces dossiers

- Copie des dossiers à la racine du second "grand dossier père" qui ne sont pas en double et comparaison de leur fichiers avec la liste doublon_file. Les fichiers y figurant sont copiés uniquement s'ils sont plus récents (vérification réalisée par la fonction

file_fusion), sinon, c'est le fichier de doublon_file qui est copié. L'emplacement de la copie est le chemin le plus profond. Si le fichier est absent de la liste, il est rajouté.

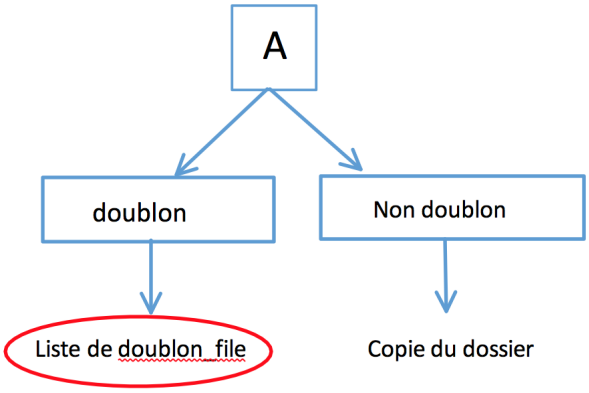

FIGURE 2 – Copie à la racine des dossiers A

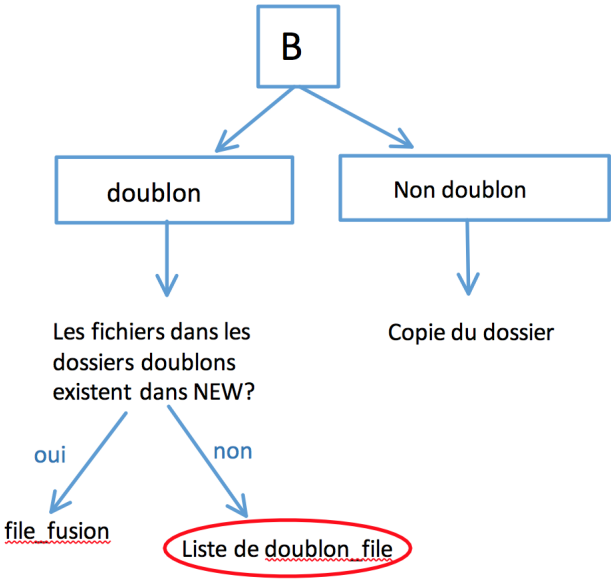

FIGURE 3 – Copie à la racine des dossiers B

4.1.3 Etape 3 : Copie en profondeur des dossiers

- Copie des dossiers existants en double. La totalitée du contenu est conservé et l'emplacement du plus profond est conservée.

4.1.4 Etape 4 : Copie à la racine des fichiers

- Comparaison des fichiers présents à la racine du premier "grand dossier père" avec les fichiers présent à la racine de l'autre. Si le fichier est en double, la fonction file_fusion est utilisé pour copier le plus récent. Si le fichier n'existe pas dans l'autre,

alors il est comparé aux éléments de doublon_file (il est donc regardé s'il existe en double). Si le fichier est dans la liste, alors la fonction file_fusion est appelée pour copier le plus récent à l'emplacement le plus profond, sinon, le fichier est copié et ajouté à la liste des fichiers doublon.

- Comparaison des fichiers présents à la racine du second "grand dossier père" avec les éléments de la liste doublon_file si ce fichiers n'est pas dans le premier "grand dossier père". Si le fichier est contenu dans la liste, alors la fonction file_fusion est utilisé pour copier le plus récent à l'emplacement le plus profond, sinon il est copié et ajouté à la liste doublon_file.

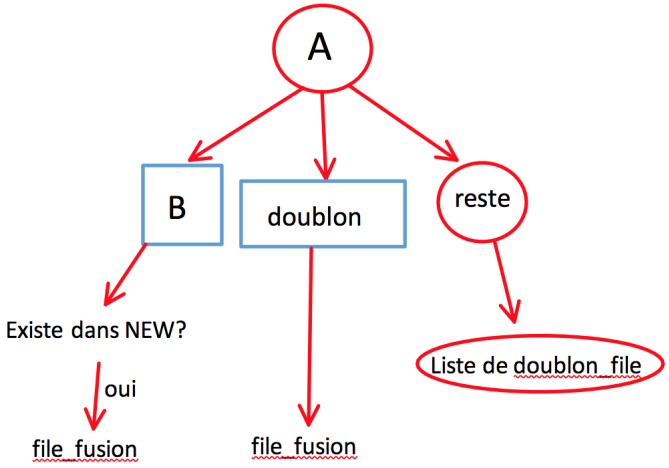

FIGURE 4 – Copie à la racine des fichiers

4.1.5 Etape 5 : Copie en profondeur des fichiers

- Comparaison des fichiers présents en profondeur dans le premier "grand dossier père" avec les éléments de doublon_file. Si le fichier y figure, la fonction file_fusion est utilisée pour copier le plus récent à l'emplacement le plus profond et l'on supprime à l'emplacement le moins profond ce fichier s'il est présent par mégarde. Si le fichier n'est pas dans la liste doublon_file, il est copié et ajouté à cette liste.

- Comparaison des fichiers présents en profondeur dans le second "grand dossier père" avec les éléments de doublon_file. Si le fichier y figure, la fonction file_fusion est utilisée pour copier le plus récent à l'emplacement le plus profond et l'on supprime à l'emplacement le moins profond ce fichier s'il est présent par mégarde. Si le fichier n'est pas dans la liste doublon_file, il est copié et ajouté à cette liste.

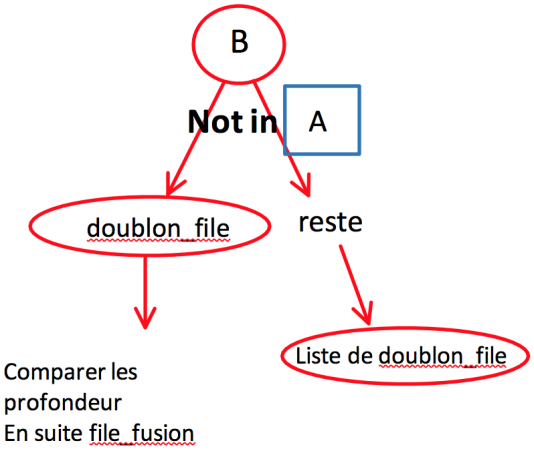

FIGURE 5 – Copie en profondeur des fichiers

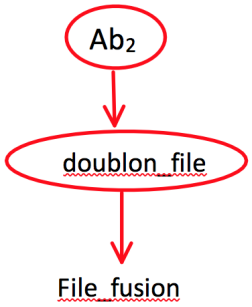

FIGURE 6 – Listes de doublons

5 Outils utilisés

5.1 Liste en profondeur de dossiers

On a écrit une fonction L_lv(A,P_A) renvoyant une liste de 3 éléments dont le premier est la profondeur du répertoire , le deuxième la liste des noms de dossiers par niveaux dont la structure est présentée sur la figure 7 et le troisième la liste des chemins des répertoires contenus dans le deuxième élément de la liste. La fonction prend deux arguments : le nom du répertoire A et le chemin P_A de sa position.

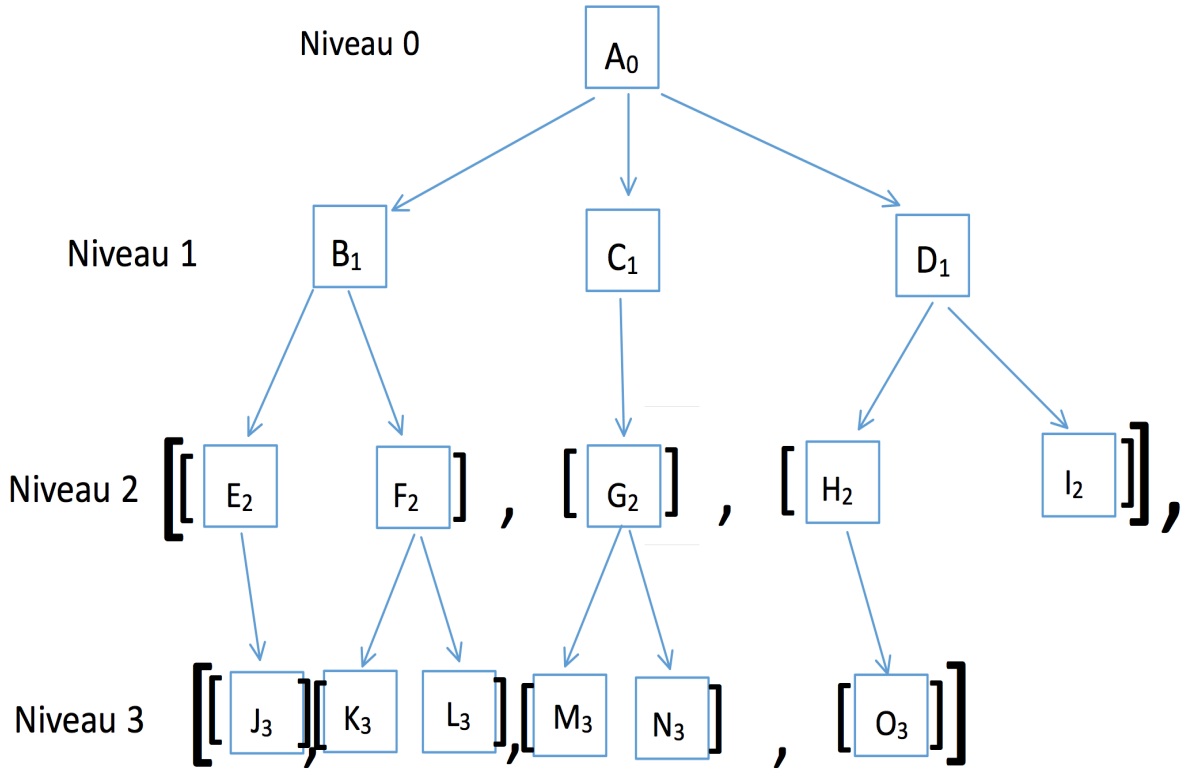

FIGURE 7 – deuxième élément de la liste

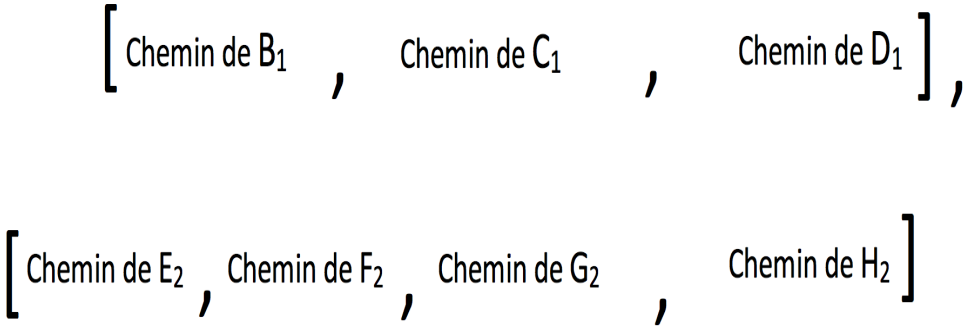

FIGURE 8 – troisième élément de la liste

5.1.1 Comparaison de la date de modification de deux fichiers - file_fuse

Lorsqu'un fichier est présent dans les deux dossiers qui doivent être rassemblés, seul le contenu du plus récent est concervé mais l'emplacement est celui du plus profond des arborescences des deux fichiers. La fonction file_fusion a été définie ci-dessous pour réaliser cette opération.

6 Cas test

6.1 Les Cas test validés

Tous les cas tests à l'exception du cas AR_3 ont été validé par le code. Les vérifications consitant à comparer le dossier produit par le programme avec un dossier de référence dont la disposition serait la réponse attendue. Le code ne permet cependant pas de réaliser une copie d'un fichier seulement dans le cas où celui-ci existe dans chacun des "grand dossier père", opération qui aurait nécéssité de créer d'autres cas tests.

6.2 Les Cas test non validés

Le cas test AR_3 n'a pas été validé par notre code parcequ'il ne respecte pas la seconde hypothèse qui fixe le cadre de fonctionnement du programme, à savoir qu'il ne peut y avoir deux dossiers (respectivement fichiers) contenus dans le même "grand dossier père" que l'on souhaite "fusionner" avec l'autre "grand dossier père".

6.3 Limites du code

Le code réalisé dans ce projet ne permet pas d'effectuer tous les cas test proposer, ce qui signifie que des améliorations peuvent être effectuées. Par ailleurs, il se peut qu'il soit très sous optimal en temps, un choix de parcours global et complet pour répertorier tous les dossiers et fichiers présents pour ensuite copier au fur et à mesure aurait peut-être apporté de meilleurs résultats en terme de coût.

7 Conclusion

Le programme réalisé au cours du projet permet bien d'effectuer de manière automatique la tâche de "fusion intelligente" de deux dossier dans les limites des hypothèses délimitant le cadre d'utilisation du programme. De meilleurs performances auraient pues être atteintes avec une approche différentes.