Exercices de Programmation & Algorithmique 1

Série 10 – Compression et décompression de texte — Fichiers et Exceptions
(30 novembre 2021)

Département d'Informatique – Faculté des Sciences – UMONS

Pré-requis : Lecture et écriture de fichiers; noms de fichiers et chemins; sauvegarder des objets (pickle et shelve); lancer une exception; rattraper une exception (cours jusqu'au **Chapitre 12**).

Objectifs : Être capable de lire et d'écrire dans des fichiers ; être capable d'utiliser le mécanisme d'exceptions.

---

1 Contexte : Compression et décompression de textes

La série précédente concernait le chiffrement, dont le but est de rendre un texte illisible. Cette série concerne la compression d'un fichier texte afin d'économiser de l'espace en mémoire. Il existe de nombreux algorithmes de compression : zip, rar, ace, 7z, tar, bz2, gzip, ...

Cette section décrit deux techniques de codage.

1.1 Codage par dictionnaire, LZ (Lempel-Ziv)

Le codage par dictionnaire se base sur les mots que le texte contient et leurs occurrences dans le texte. Dans le cadre de ce TP, nous considérons comme "mot" toute séquence de caractères excluant les espaces et les retours à la ligne.

L'idée générale consiste à établir une liste numérotée des mots distincts présents dans le fichier à compresser et de remplacer chacune des occurrences d'un de ces mots par son numéro. Par exemple, considérons le texte « un plus un différent de un ». Les mots numérotés sont les suivants : `['un', 'plus', 'différent', 'de']` (l'index d'un élément représente son numéro). Le texte compressé pourrait alors s'écrire : `"0 1 0 2 3 0"` (ou, sous la forme d'une liste : `[0, 1, 0, 2, 3, 0]`).

1.2 Codage par répétition, RLE (Run-Length encoding)

Le codage par répétition profite de caractères qui sont présents plusieurs fois successivement.

Lorsqu'une lettre apparaît au moins deux fois successivement, elle est remplacée par le nombre de fois que la lettre apparaît suivi de la lettre elle-même.

Evidemment, pour pouvoir être décodé/décompressé correctement, le texte de départ ne peut contenir que des lettres (sans chiffres).

Par exemple, la chaîne « `aaabaccccaa` » est compressée en « `3aba4c2a` » et la chaîne « `ab11a3b` » est décompressée en « `abaaaaaaaaaaabbb` ».

2 Le contrat

Pour ce contrat, vous allez devoir écrire un programme unique qui permettra de compresser ou décompresser des fichiers texte en utilisant l'une ou l'autre technique de codage.

Pour fonctionner, votre programme aura besoin de différentes informations : s'il s'agit de compression ou de décompression ; le choix de la technique utilisée ; les fichiers d'entrée (ce qu'il faut compresser ou décompresser) et de sortie (où placer le résultat). Ces informations devront être transmises à votre programme par le biais des options dans la ligne de commande.

Pour ce faire, veuillez utiliser le module `docopt`. Vous trouverez des informations sur ce module sur le site web de `docopt` (`docopt.org`).

Voici l'interface attendue :
`-c` pour spécifier que l'on veut compresser (mutuellement exclusif avec `-d`) ;
`-d` pour spécifier que l'on veut décompresser (mutuellement exclusif avec `-c`) ;
`-t <TECHNIQUE>` pour spécifier la technique utilisée (obligatoire) ;
`--in <FICHIER>` pour spécifier le fichier d'entrée (obligatoire) ;
`--out <FICHIER>` pour spécifier le fichier de sortie. Si cette option est absente, on placera le résultat dans un fichier portant le même nom que le fichier d'entrée auquel on a concaténé `".out"`.

Il vous est demandé de gérer finement les erreurs pouvant se produire :
— Si le nombre de paramètres fournis n'est pas correct ou que les paramètres sont incorrects, générez une exception de type `ValueError` et affichez un message d'aide concernant l'utilisation du script.
— Si le fichier source n'existe pas, gérez l'exception concernée.
— Si le fichier source, lors de la décompression, n'est pas dans le format attendu, affichez une erreur et générez une exception du type `TypeError`.
— D'autres erreurs spécifiques aux techniques seront spécifiées plus loin.

1 Veuillez vous assurer d'avoir compris le fonctionnement de `docopt` et l'exemple suivant tiré de la documentation (`https://github.com/docopt/docopt/blob/master/examples/naval_fate.py`) :

```
"""Naval Fate.
Usage:
  naval_fate.py ship new <name>...
  naval_fate.py ship <name> move <x> <y> [--speed=<kn>]
  naval_fate.py ship shoot <x> <y>
  naval_fate.py mine (set|remove) <x> <y> [--moored|--drifting]
  naval_fate.py -h | --help
  naval_fate.py --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --speed=<kn>  Speed in knots [default: 10].
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.
"""
from docopt import docopt


arguments = docopt(__doc__, version='Naval Fate 2.0')
print(arguments)
```

2 Veuillez implémenter l'algorithme de compression et de décompression selon la technique du codage par dictionnaire (section 1.1). L'option pour spécifier la technique sera `-t LZ`.

Une fois la compression effectuée (c'est-à-dire la liste des mots et la version compressée créées), sauvez le résultat dans le fichier de sortie.

La décompression d'un tel fichier se fait simplement en associant, à chaque numéro, le mot correspondant.

Testez votre application en comparant un fichier texte "original" avec celui obtenu après compression et décompression successive. Vous pouvez télécharger des fichiers textes de grandes tailles sur `http://gallica.bnf.fr/ebooks/` et par exemple `http://gallica.bnf.fr/ebooks/fleursdumal.txt`.

Exemple d'utilisation si votre script se nomme `compress.py`. L'appel suivant permettra de compresser le fichier `fleursdumal.txt` et de placer le résultat de la compression dans le fichier `fleursdumal.Ztxt` :

```
python compress.py -t LZ -c -in fleursdumal.txt -out fleursdumal.Ztxt
```

3 Veuillez implémenter l'algorithme de compression et de décompression selon la technique du codage par répétition (section 1.2). Votre algorithme doit être adapté au contexte particulier des séquences d'ADN. L'option pour spécifier la technique sera `-t ADN`.

Une séquence d'ADN peut être représentée par la séquence des quatre bases spéciales qui composent cette molécule. Ces bases sont nommées A, T, G, et C selon les initiales de leur nom.

Un fichier contenant des bouts de séquences d'ADN contiendra donc uniquement des lignes composées de A, T, G, et C. Si le fichier d'entrée ne respecte pas ce format, vous génererez une exception `badADNformat` que vous gérerez également.

Testez votre application en comparant un fichier contenant des bouts de séquences d'ADN "original" avec celui obtenu après compression et décompression successive. Vous pouvez utiliser le fichier `adn.txt` disponible sur moodle.

3 Exercices complémentaires

★☆☆ 4 **Replace**

Veuillez créer un script `replace.py` qui permet de faire des remplacements de mots dans un fichier texte. Le script prend quatre arguments :
— le fichier d'entrée ;
— le fichier de sortie ;
— une chaîne de caractères contenant tous les mots à remplacer, séparés par un espace ;
— une chaîne de caractères contenant les nouveaux mots (respectifs), séparés par un espace.
Le script doit pouvoir être utilisé de la sorte :

```
python replace.py in.txt out.txt "mot ancien algo" "phrase nouveau prog"
```

Dans l'exemple donné, le script doit considérer le texte du fichier `in.txt`. Il doit remplacer toutes les occurrences du mot "`mot`" par "`phrase`", de "`ancien`" par "`nouveau`" et de "`algo`" par "`prog`". Le résultat de ce remplacement doit être écrit dans le fichier `out.txt`.

★★☆ 5 **Suite de mots**

Veuillez écrire une fonction `suite_de_mots` qui prend 3 arguments :
— `filename`, le nom d'un fichier texte (contenant un mot par ligne, comme `words.txt`) ;
— `n`, la taille de la suite à générer ;
— `c`, la taille de la contrainte.
La fonction doit générer une suite de `n` mots tirés du fichier nommé `filename` en respectant les contraintes suivantes :
— les mots de la suite doivent au moins être de taille `c`+1.
— si `w1` et `w2` sont deux mots consécutifs de la suite générée, alors les `c` derniers caractères du mot `w1` sont identiques aux `c` premiers caractères du mot `w2`.
La suite doit ensuite être affichée à l'écran.

Pour y parvenir, veuillez définir les deux fonctions intermédiaires suivantes :
— `commence_par(filename, s)` qui retourne une liste de tous les mots commençant par `s` dans `filename` ;
— `rec_suite(filename, debut, n, c)`, fonction *récursive* qui prend en paramètre :
  — `filename`, le nom du fichier texte qui contient les mots ;
  — `debut`, le premier mot imposé de la suite ;
  — `n`, la taille de la suite à générer ;
  — `c`, la taille de la contrainte.
  Cette fonction *récursive* retourne une suite de `n` mots (sous forme de liste) respectant les contraintes exprimées plus haut, et dont le premier mot est `debut`.
  Si une telle suite n'existe pas, la fonction doit **lancer une exception** appropriée (de votre choix).

Voici le comportement attendu :

```
>>> suite_de_mots('words.txt', 3, 3)
['aahed', 'heddles', 'lesbian']
>>> suite_de_mots('words.txt', 10, 3)
['aahed', 'heddles', 'lesbians', 'ansate', 'atechnic',
```

```
'nice', 'iceberg', 'ergastic', 'tical', 'calabash']
>>> suite_de_mots('words.txt', 10, 5)
['abound', 'bounder', 'undercover', 'covers', 'oversales',
'saleswoman', 'womanising', 'isinglass', 'glassblower', 'lowercase']
```

★★☆ 6 Ceci concerne l'exercice précédent. Dans les exemples donnés, vous aurez remarqué que le premier mot commence par `a`. Ceci est dû au fait que les mots candidats à être choisis sont pris dans l'ordre dans le fichier.

Il vous est demandé d'améliorer votre programme (version 2) pour que l'ordre dans le fichier n'ait pas d'influence sur la probabilité qu'un mot soit choisis pour être dans la liste.

D'ailleurs, deux lancements successifs de la fonction pourraient donner deux suites de mots différentes.

★★☆ 7 **Contrôleurs de versions**

Grossièrement, une des tâches d'un contrôleur de versions est de permettre à un programmeur A de voir les suppressions, les modifications et les ajouts faits par un autre programmeur B travaillant sur le même projet. On vous demande de créer un script contenant une fonction `diff(f1, f2, s)` qui prend deux fichiers `f1` et `f2` (l'un représente le fichier du programmeur A et l'autre de B) et une chaîne de caractères `s` en paramètres et qui crée un fichier nommé `s.txt` donnant les différences entre `f1` et `f2`. Pour vous faciliter le travail, nous supposons que les modifications sont toujours réalisées au sein d'une ligne (pas de suppressions, ni d'ajouts de lignes). Pour chaque différence, vous devez :

(a) indiquer le numéro de la ligne modifiée ;

(b) écrire la ligne modifiée de `f1` ; et

(c) écrire la ligne modifiée correspondante de `f2`.

La liste de ces différences sera précédée de la date de la création du fichier, ainsi que le nom du fichier `f1` (avec son chemin complet).

Spécifications supplémentaires :

— Vous pouvez utiliser la fonction `readline()`, mais pas la fonction `readlines()` ;
— Il faut que l'utilisateur puisse entrer le nom des deux fichiers à comparer ainsi que le nom du fichier qui contiendra les différences dans la console. Il vous faut donc une fonction supplémentaire qui lise les noms de fichier à comparer et qui appelle `diff` avec ceux-ci ;
— Vous devez gérer le fait que l'utilisateur entre bien deux noms de fichier existant à l'aide des exceptions ;
— Vous devez également vérifier que l'utilisateur a bien entré 3 noms de fichiers ;
— Vérifiez votre programme avec les deux fichiers disponibles sur le site e-learning. Et vérifiez que votre fichier de différences correspond avec celui proposé sur e-learning également.

Astuce : Lorsque vous êtes au bout d'un fichier texte, la fonction `readline` retourne une chaîne de caractères vide.

Exemple :

Commande entrée en console :

```
python differences.py monfichier1.txt monfichier2.txt diff_f1_f2.txt
```

Après l'exécution de votre programme, le fichier `diff_f1_f2.txt` doit avoir été créé et contenir par exemple :

```
Le 28/11/09
Diff du fichier monfichier1.txt
- Ligne 10
f1 : for i in range(len(liste)-1):
f2 : for i in range(len(liste)):
- Ligne 15
f1 : x = a + b / n
f2 : x = (a + b) / float(n)
```