pythonlp-5.0

1. Un compresseur de fichier rudimentaire

Le but de cet exercice est de programmer un compresseur de fichier basé sur la méthode RLE (Run Length Encoding). Cette méthode consiste à remplacer dans un fichier des suites de caractères identiques par trois caractères. Un caractère ! qui indique le début d'une séquence de remplacement, un caractère chiffre (compris entre 4 et 9) qui indique le nombre de caractères remplacés suivi du caractère effectivement remplacé. Des suites de caractères inférieures à quatre caractères ne sont pas modifiées car le gain serait nul. Un caractère ! dans le fichier original est traduit par deux !! dans le fichier compressé.
Par exemple avec le fichier ci-dessous, le fichier compressé obtenu est :

!bonjour!!	!!bonjour!!!!
aaaazzzzzzz	!4a!7z
jkiiiiii	jk!6i
ss	ss
sssssssssssdfffff	!9sssd!5f
1111111111111111	!91!71
xxxxxggg!gnnnnnnn!	!5xggg!!g!7n!!
qqqxx	qqqxx
lm!	lm!!

Des séquences de plus de 9 caractères identiques consécutifs sont traduites par deux (ou plus) séquences compressées. Ainsi la séquence de 11 s est traduite par !9sss (9 fois s + ss) et la séquence de 16 1 est traduite par !91!71 (9 fois 1 + 7 fois 1).

1/ Écrire un programme qui compresse par la méthode décrite un fichier dont le chemin est entré au clavier. Le nom du fichier compressé est le nom du fichier de départ auquel on rajoute l'extension rle. Votre programme indiquera le gain obtenu ☼☼☼.
2/ Écrire le programme de décompression correspondant. Vérifier son fonctionnement avec la commande linux diff ☼☼.
Essayez vos programme avec différents format de fichier (txt,pdf,exe,bmp,jpeg,etc...) et notez les performances obtenues.
3/ Le gain maximal, de 6 octets, est obtenu pour une séquence de 9 caractères identiques consécutifs. Les caractères ! présents dans le fichier d'origine pénalisent l'algorithme car il doivent être doublés. On peut facilement améliorer l'algorithme de deux manières :

- en choisissant un caractère qui n'apparaît pas dans le fichier d'origine (ou le caractère le moins fréquent lorsque toutes les valeurs existent) pour indiquer le début d'une séquence de remplacement. Le caractère choisit est simplement indiqué en premier caractère du fichier compressé.
- -en choisissant un caractère supplémentaire pour des séquences de 10 à 99 caractères identiques consécutifs. Ce caractères est également choisi parmi les caractères peu fréquents du fichier d'origine et est indiqué en deuxième caractère du fichier compressé.

Améliorez vos programmes et notez le gain obtenu pour différents formats et contenus de fichier. Les performances sont relativement faibles car l'algorithme est vraiment basique ☼☼.

2. Un "générateur" de (moins mauvais) mot de passe

Les meilleurs mots de passe sont ceux générés aléatoirement. Mais ce sont aussi les plus difficiles à retenir. Un mot du langage courant est facile à retenir mais ne résistera pas longtemps à une attaque par dictionnaire. Un moyen d'obtenir un mot de passe facile à retenir et compliquant (un peu) l'attaque par dictionnaire est d'utiliser un mot du langage courant et de le couper à une position aléatoire par un caractère aléatoire imprimable et non alphabétique*. Par exemple, à partir du mot "bonjour" entré par l'utilisateur on propose le mot "bonjo!ur".
On pourra ensuite proposer un mot passe contenant deux caractères non alphabétiques. Toujours à partir de "bonjour", on obtiendrait "bonj#o'ur" ou "6bonjo&ur". (les positions des deux caractères non alphabétiques ne devant pas être consécutives)

*On définit un caractère alphabétique comme un caractère appartenant à ['a','z']U['A','Z'].
*Les caractères dont le code est inférieur 0x20 ou supérieur à 0x7F ne sont pas "imprimables".

Écrire le programme qui propose un mot de passe à partir d'un mot choisi par l'utilisateur☼

En utilisant le fichier texte dictionnaire "dico_sans_accents.txt" modifier votre programme pour qu'il choisisse aléatoirement un mot d'au moins 6 caractères dans le dictionnaire (ici pas d'accent car les programme de login ne les acceptent généralement pas)☼.

http://arlotto.univ-tln.fr/ressources/lpaii/python_ue0/tp/tp5/dico_sans_accents.txt

3. Jeux : Retrouver des mots mélangés

A partir d'un dictionnaire donné sous la forme d'un fichier texte contenant un mot par ligne, on vous demande de présenter à l'utilisateur un mot choisi aléatoirement dont les lettres ont été mélangées aléatoirement. L'utilisateur aura un certain nombre d'essais pour deviner le mot original.
Le nombre d'essais pourra être fonction de la longueur du mot à deviner.

Exemple :

dictionnaire.txt :
.....
.....
voile
voilure
voiture
voiturette
.....
.....

Exécution :
Devinez ce mot : ervouti
Vous avez 5 essais :
essai 1 : *vroum*
Non !
Essai2 : *etrouvi*
Non !
Essai3 : *voiture*
Bravo !
On rejoue (o/n) : *n*
Au revoir ....

Améliorations possibles :

Imaginer un système de scores basé sur le temps mis pour répondre.

Un fichier des meilleurs scores sera créé et tenu à jour par votre programme.

Imaginer la possibilité de choisir plusieurs niveau de jeu :

niveau débutant : mots limités à 6 lettres, Affichage de la première puis de la dernière lettre lorsque l'utilisateur ne trouve pas au bout de quelques essais.
niveau intermédiaire : mots limités à 8 lettres, Affichage de la dernière lettre lorsque l'utilisateur ne trouve pas au bout de quelques essais.
niveau expert : pas de mots de moins de 7 lettres, aucune aide.

Dictionnaire :
http://arlotto.univ-tln.fr/ressources/lpaii/python_ue0/tp/tp5/dicco.txt
Une version du dictionnaire sans lettres accentuées est également disponible à :
http://arlotto.univ-tln.fr/ressources/lpaii/python_ue0/tp/tp5/dico_sans_accents.txt

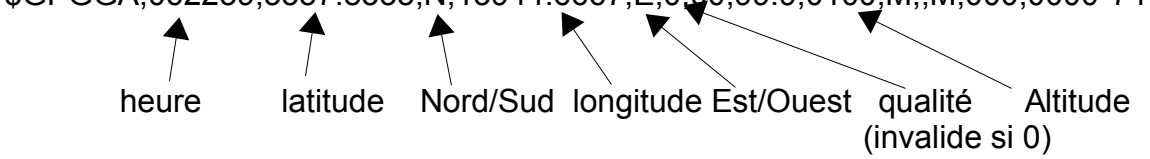
4. Affichage de parcours avec le logiciel GoogleEarth

Le logiciel **GoogleEarth** (http://earth.google.fr) est un logiciel gratuit qui permet d'afficher des images satellites de la terre à haute résolution.
KML (*Keyhole Markup Language*) est un format de fichier pour la modélisation et le stockage de caractéristiques géographiques comme les points, les lignes, les images, les polygones et les modèles pour l'affichage dans Google Earth.
Une introduction au langage kml en français se trouve à :
https://developers.google.com/kml/documentation/kml_tut

Un récepteur **GPS** (http://fr.wikipedia.org/wiki/Global_Positioning_System) délivre diverses informations (position, direction, vitesse, date, heure, nombre de satellites en vue,....) sous forme de trames (suite de caractères) au format **nmea** (http://aprs.gids.nl/nmea/) . Il est facile de connecter un gps à un ordinateur par un port série et de stocker toutes les informations délivrées par le récepteur GPS durant un parcours dans un fichier. On obtient un fichier constitué de trames qui commencent toutes par $ et une suite de 5 caractères qui identifie la nature des informations présentes dans la trame.
Voici un extrait du fichier obtenu :

```
$GPZDA,062238,03,02,2022,,*46
$GPGSV,1,1,00,,,,,,,,,,,,,,,,*79
$GPVTG,000.0,T,,M,000.0,N,000.0,K*60
$GPGGA,062239,3537.8333,N,13944.6667,E,0,00,99.9,0100,M,,M,000,0000*74
$GPGLL,3537.8333,N,13944.6667,E,062239,V*32
$GPRMC,062239,V,3537.8333,N,13944.6667,E,000.0,000.0,030222,,*04
$GPZDA,062239,03,02,2022,,*47
$GPGSA,A,1,,,,,,,,,,,,,99.9,99.9,
```

On va s'intéresser uniquement aux trames **$GPGGA** car elles contiennent les principales informations dont nous avons besoin pour afficher le parcours dans GoogleEarth. La structure de cette trame est la suivante :



Voici un exemple de fichier au format kml qui permet l'affichage dans GoogleEarth d'un parcours sous forme de lignes jaunes reliant les différents points relevés.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Document>
    <name>Paths</name>
```

```
    <description> Mon parcours </description>
    <Style id="yellowLineGreenPoly">
      <LineStyle>
        <color>7f00ffff</color>
        <width>4</width>
      </LineStyle>
      <PolyStyle>
        <color>7f00ff00</color>
      </PolyStyle>
    </Style>
    <Placemark>
      <name>Absolute Extruded</name>
      <description> </description>
      <styleUrl>#yellowLineGreenPoly</styleUrl>
      <LineString>
        <extrude>1</extrude>
        <tessellate>1</tessellate>
        <altitudeMode>absolute</altitudeMode>
        <coordinates>
                  6.175453,43.115388,3
                  6.175447,43.115353,3
                  6.175480,43.115350,3
                  6.175508,43.115388,3
                  6.175493,43.115428,3
                  6.175415,43.115462,3
                  6.175327,43.115482,3
                  6.175203,43.115522,4
        </coordinates>
      </LineString>
    </Placemark>
  </Document>
</kml>
```

longitude, latitude, altitude

Le fichier se compose d'un en-tête, de balises décrivant le type et la couleur des traits et enfin entre les balises <coordinates> et </coordinates> des valeurs des coordonnées (dans l'ordre longitude,latitude,altitude) des points à relier.

La latitude doit être en degrés décimaux comprise entre -90 et +90 (positive nord).
La longitude doit être en degrés décimaux comprise entre -180 et +180 (positive est).
L'altitude doit être en mètre.

Un fois chargé dans GoogleEarth (par Fichier->Ouvrir) ce fichier produit l'affichage du parcours superposé à l'image satellite de la zone.

Votre travail consiste à écrire un programme capable d'extraire les informations latitude, longitude et altitude présentes dans les trames $GPGGA des fichiers fournis par un gps et de créer un fichier kml afin d'afficher le parcours dans GoogleEarth.
Attention la première et la dernière ligne peuvent contenir des trames incomplètes.

Voici quelques indications sur la manière de procéder :

On commence par créer un fichier kml contenant la partie invariante (depuis le début jusqu'à <coordinates> (par exemple en recopiant un fichier préalablement créé). Ensuite il faut ensuite lire chaque ligne du fichier pour repérer les trames $GPGGA. Pour les lignes $GPGGA, il faut d'abord voir si la qualité est correcte (champ qualité non nul). Si c'est le cas il faut lire les informations latitude,longitude et altitude, les convertir dans le format kml (degrés décimaux) et les recopier dans le fichier kml. Lorsqu'on atteint la fin du fichier gps, il suffit d'écrire la fermeture des balises dans le fichier kml (</coordinates> </LineString> </Placemark> </Document></kml>).

kml utilise les degrés décimaux alors que le gps donne des degrés/minutes décimales.

Par exemple, on a dans le fichier GPS 4306.9122 pour 43° et 06.9122 minutes (minutes décimales) qu'il faut convertir en 43.11520 degrés décimaux soit 43+(6.9122/60) à stocker dans le fichier KML.

Crypter , Décrypter et Casser un code simple.

Un code très simple mais néanmoins encore utilisé lorsque la sécurité est peu critique est le codage par OU exclusif avec une clé (mot de passe) constituée d'une chaîne de caractères.
On utilise la propriété de réversibilité d'un OU exclusif :

Si code = clair ⊕ clé alors clair = code ⊕ clé (en C a⊕b est noté a^b)

La même opération est alors utilisée pour coder et décoder un message.

Par exemple, pour coder le message "May the force be with you" avec la clé "Yoda", on code chaque caractère du message avec un caractère de la clé en recommençant avec le premier caractère de la clé lorsque on atteint le dernier :

```
YodaYodaYodaYodaYodaYodaY
May the force be with you
```

Le message codé sera : 'Y'⊕'M' , 'o'⊕'a', 'd'⊕'y', 'a'⊕' ', 'Y'⊕'t', 'o'⊕'h','d'⊕'e','a'⊕' ',.....
Bien sûr, le message codée n'est pas constitué uniquement de caractères ascii imprimables, mais ce n'est pas un problème car il n'est destiné à être lu tel quel. Cette méthode s'applique à n'importe quel type de fichier. Le décodage se fait en appliquant le même principe.

Écrire un programme qui crypte un fichier par cette méthode (utilisez les modes d'ouvertures "rb" et "wb") . Votre programme demandera un mot de passe de 8 à 16 caractères. ☼

Écrire un programme qui, après avoir demandé le mot de passe, décrypte un fichier crée par le programme précédent. ☼

On va montrer qu'il est assez facile si l'on dispose d'un fichier assez volumineux de retrouver le mot de passe par une méthode statistique en supposant connue la longueur de la clé et en supposant que le message original contient en majorité du texte.
En effet, la faiblesse de l'algorithme est que pour une clé de longueur N, un octet sur N est toujours codé avec le même caractère de la clé.
Ainsi avec l'exemple précédent,

le caractère 'Y' code les octets de rang 0,4,8,12,....
le caractère 'o' code les octets de rang 1,5,9,13,....

Le caractère de rang i de la clé code donc les octets de rang i, i+N, i+2N,....

Pour les mots de la langue française on sait que le caractère le plus fréquent est le e. Mais en fait lorsqu'on écrit un texte informatique c'est l'espace qui est le caractère le plus fréquent.
Il suffit donc de rechercher le caractère le plus fréquent parmi les caractère de rang 0,N,2N,.... du fichier crypté et d'en faire un ou exclusif avec le code ascii de l'espace pour retrouver le premier caractère de la clé.
En procédant de même pour les caractères de rang 1,1+N,1+2N,...., on trouve le deuxième caractère de la clé. Et ainsi de suite jusqu'à dernier caractère de la clé. Ensuite le décodage est trivial....

Python
LP SARI

Écrire un programme qui décrypte un message codé sans connaître la clé, mais en demandant sa longueur (entre 8 et 16 caractères).
Pour les essais prendre un fichier texte assez gros (au moins 10 à 20ko).

Il est possible de casser ce type de code sans connaître la longueur de la clé (méthode de Kasiski (fin 19ème siècle) ou par force brute (essais systématiques de toutes les combinaisons)
Écrire un programme qui décrypte un message codé sans connaître la clé ni sa longueur.