TP1 : Echauffement

- Année universitaire: 2020
- Cursus : Info2
- Cours : Outils Logiciels

Exercice 1: La suite de Fibonacci

La suite de Fibonacci F{n} est définie comme suit:

- F{0} = 0
- F{1} = 1
- F{n} = F{n-1}+F{n-2} si n>1

1. Écrivez une fonction récursive `fibonacci1(n)` recevant un entier et renvoyant F{n} . Testez la fonction pour n = 10, puis 20, 40...
2. En listant les appels récursifs pour n = 4 expliquez pourquoi la fonction `fibonacci1` est si lente.
3. Écrire une fonction `fibonacci2(n)`, qui fait la même chose que `fibonacci1` sauf qu'elle renvoie un couple de valeur (F{n},S{n}), où S{n} est le nombre total d'additions effectuées au cours du calcul.
4. En utilisant la bibliothèque upemtk, afficher un graphique qui affiche le nombre d'additions pour chaque valeur de n. Il s'agit d'afficher des lignes brisées reliant les points (n,S{n}).

   Essayer d'estimer la complexité de cette fonction, par exemple en affichant sur le même graphiques les courbes de fonctions connues: par ex.: g(n) = n; h(n) = n^2 ; i(n) = 2^n ; etc.
5. Pour améliorer les performances, écrivez une fonction récursive `fibonacci3(n)` qui prend en paramètre un entier n >= 1 et qui renvoie directement le couple (F{n-1},F{n}).
6. Comparez les performances de `fibonacci3` avec `fibonacci1` en procédant comme aux questions 3-4. (La nouvelle fonction s'appellera `fibonnaci4(n)`.)
7. Si l'on a besoin de calculer de nombreuses fois la valeur de certains termes de la suite de Fibonacci, il peut être intéressant de mémoriser toutes les valeurs déjà calculées. (Ce processus est appelé **mémoïsation**). Utiliser une variable globale `fibonacci5_memoire` pour écrire une fonction `fibonacci5(n)` qui prend en argument un entier et renvoie la même chose que `fibonacci1`. Lorsqu'on cherche F{n} , on vérifie s'il est dans la mémoire; si c'est le cas, on renvoie immédiatement sa valeur, sinon on calcule la valeur F{n} et on complète la mémoire.

Exercice 2: la distance d'édition

La distance d'édition est le nombre d'opérations qu'il faut pour passer d'une chaîne de caractère à une autre. Les opérations autorisées sont: insertion d'une lettre, suppression d'une lettre, et remplacement d'une lettre par une autre.

Par exemple, la distance d'édition entre les mots "abracadabra" et "macabre" vaut 6. En effet, on peut passer de l'un à l'autre comme suit:

```
abracadabra
||    ||        4 suppressions
  raca  bra
  |       |     2 substitutions (r devient m et a devient e)
  maca  bre
```

On peut trouver d'autres suites de 6 opérations permettant de passer de l'un à l'autre, mais c'est impossible en 5 opérations.

Pour calculer la distance d'édition, on peut procéder comme le montre les deux exemples suivants.

Exemple 1: la distance d'éditions entre "abracadabra" et "macabre" est le minimum entre:

- 1 + la distance d'édition de "bracadabra" et "acabre" (substitution de "a" par "m")
- 1 + la distance d'édition de "abracadabra" et "acabre" (insertion de "m")
- 1 + la distance d'édition de "bracadabra" et "macabre" (suppression de "a")

Exemple 2: la distance d'édition entre "abracadabra" et "acabre" est le minimum entre:

- 0 + la distance d'édition entre "bracadabra" et "cabre" (Notez qu'à la différence de l'exemple 1, on ajoute 0 au lieu de 1 car les deux mots commencent par la même lettre.)
- 1 + la distance d'édition de "bracadabra" et "abre"
- 1 + la distance d'édition de "racadabra" et "cabre"

1. Écrire une fonction `distance1(chaine1,chaine2)` qui prend en argument deux chaînes de caractères et calcule la distance d'édition entre les deux arguments.

2. Écrire une fonction `distance2(chaine1,chaine2)` qui calcule la distance d'édition mais tire partie de la mémoïsation pour s'exécuter plus rapidement. On stockera les valeurs précédemment calculé dans un dictionnaire dont les clefs sont des paires de chaînes de caractères (chaine1,chaine2), et les valeurs sont les distances d'éditions correspondantes. On pourra également tirer partie de la symmétrie de la distance d'édition:

   ```
   distance(chaine1,chaine2) = distance(chaine2,chaine1)
   ```

3. Écrire une fonction `plusProche(chaine, possibilites)` qui prend en argument une chaîne de caractère et une liste de chaînes de caractères, et qui renvoie la chaîne parmi `possibilites` la plus proche de `chaine` (en distance d'édition).

4. Tester cette fonction avec quelques chaînes. Comparer la vitesse d'exécution et l'augmentation en taille du dictionnaire de mémoïsation lors d'appels successifs à `plusProche`

5. Écrire un programme python `suggere_nom.py` qui propose le titre de série le plus proche de chacun de ses arguments. Le fichier `series_2000-2019.txt` vous donne les titres de quelques séries des années 2000 et 2010.

   Merci de faire un output qui ressemble à:

   ```
   argument1  ->  titre suggéré pour l'argument 1
   argument2  ->  titre suggéré pour l'argument 2
     ...
   ```

6. Faire en sortie de stocker dans un fichier le dictionnaire de mémoïsation et le réouvrir au début du programme (s'il existe).

   `/!\` Attention à bien gérer le cas où le fichier n'existe pas.

   Il est attendu que vous trouviez un module python pour faire ce boulot, ne créez pas votre propre format.

7. Répondez dans votre code en commentaire et en toute fin de fichier aux questions suivantes.

a. Discutez la pertinence de l'amélioration proposé à la question 6.
b. Dans certains cas, la distance d'édition ne donne pas la suggestion qu'un humain trouverait "naturelle".
   Trouver un tel cas, essayer d'expliquer pourquoi ce phénomène arrive, et ébaucher (sans implémentation) une solution.