Devoir à la maison n°2

Distribué le jeudi 19 décembre 2024,

à rendre le jeudi 9 janvier 2024

Vous devez traiter le problème 1, puis au choix, le problème 2 ou le problème 3.
Le problème 2 est une série de petits exercices pour s'entrainer à programmer avec des listes et des dictionnaires.
Le problème 3 est d'un niveau plus relevé. Soyez réaliste sur vos compétences et vos besoins au moment de choisir.

1 Problème 1

Étant donnés deux mots, on cherche la plus longue séquence commune, c'est-à-dire la plus longue chaîne de caractères dont les lettres sont dans les deux mots à la fois tout en conservant l'ordre (les lettres ne sont pas forcément consécutives dans le mot). Par exemple, pour les mots **circonstance** et **importance**, la plus grande séquence commune a pour longueur 7 avec **iotance** ou **irtance**.

1.1 Programmation récursive

1. On souhaite d'abord construire une fonction `longueur(s,t)` qui pour deux chaînes de caractères s et t va trouver la longueur de la plus grande sous-sequence commune.
   (a) Si s (ou t) est la chaîne vide, que vaut `longueur(s,t)` ?
   (b) Notons `n=len(s)` et `p=len(t)`.
   Si `s[n-1]=t[p-1]`, alors quel est le lien entre `longueur(s,t)` et `longueur(s[:n-1],t[:p-1])` ?
   (c) Si `s[n-1]!=t[p-1]`, donner une relation entre `longueur(s,t)`, `longueur(s[:n-1],t)` et `longueur(s,t[:p-1])`.
   (d) Coder alors la fonction `longueur` de façon récursive avec mémoïsation.
2. Programmer alors une fonction `PLSC(s,t)` qui trouve une séquence commune de longueur maximale.
   On distinguera les cas comme pour la fonction `longueur`.
3. On dispose d'un fichier texte comprenant 1357 mots (mots les plus utilisés de la langue française). Chaque mot est écrit sur une ligne du fichier.
   Ecrire les instructions permettant de faire afficher deux mots dont la plus longue séquence commune est la plus longue.
   Si vous souhaitez tester votre programme sur machine, ce fichier est disponible dans cahier de prépas.

Nous venons en fait de résoudre un problème d'optimisation (on veut trouver une solution "optimale", c'est-à-dire qui minimise ou maximise un problème donné). L'exercice sur le plus grand carré blanc dans une grille de pixels était également un problème d'optimisation.

Pour ces deux problèmes nous avons proposé une résolution récursive, dite **descendante ("top-down")**, où on écrit directement la résolution du problème complet, et en laissant la récursivité appeler la résolution des sous-problèmes

qui en découlent. Il est alors impératif d'utiliser **la technique de mémoïsation** pour optimiser la gestion de l'espace mémoire.

La récursivité ayant certains inconvénients (la limitation du nombre d'appels récursifs par Python notamment), nous allons aussi proposer pour le problème de la plus grande séquence commune une façon itérative, qui met en place une méthode de résolution dite **ascendante ("bottom-up")**, et cherche à résoudre le problème final en partant de cas de bases triviaux à résoudre :

1.2 Programmation itérative

Voici l'idée de la fonction `longueur2(s,t)` (pour deux chaînes de caractères s et t, cette fonction retourne la longueur de la plus grande sous-sequence commune) :

- On va construire un tableau `tab`, de dimension 2, qui comporte $n = len(s)$ lignes, et $m = len(t)$ colonnes.
- `tab[i][j]` contiendra la longueur de la plus sous-sequence commune à `s[:i+1]` et `s[:j+1]` (par exemple `tab[0][0]` contiendra 1 si les deux première lettres de s et de t sont égales, 0 sinon)
- Vous allez remplir ce tableau petit à petit, en utilisant les relations de récurrences trouvées dans la partie 1.
- Finalement, la longueur à renvoyer sera la dernière valeur calculée dans le tableau, c'est-à-dire `tab[n-1][m-1]`

4. Implémenter cette fonction.

---

2 Problème 2 : Pour s'entraîner à manipuler les listes et les dictionnaires

Exercice 1 : 1. Créer le dictionnaire ci-dessous :
`jours={"lundi":1,"mardi":2,"mercredi":3,"jeudi":4,"vendredi": 5,"samedi":6,"dimanche":8}`
2. Corriger l'erreur "dimanche" :8 en "dimanche" :7
3. Créer un programme qui affiche la liste des clés.
4. Créer un programme qui affiche la liste des valeurs .
5. Créer un programme qui affiche la liste des pairs (clé, valeur).

Exercice 2 : On considère les deux dictionnaires suivants :
dico1={"paris" :75,"La Rochelle" :17} et dico2={"Rennes" :35,"Orléans" :45}
Ecrire une fonction permettant de concaténer en seul dictionnaire les clés et valeurs de deux dictionnaires donnés en paramètres d'entrée (qui ne doivent pas être modifiés lorsque l'on execute la focntion).
Tester votre fonction avec dico1 et dico2.

Exercice 3 : On considère un dictionnaire donnant la moyenne de quelques étudiants.
dico={"etudiant1" :15,"etudiant2" :9,"etudiant3" :12,"étudiant4" :5}.
Ecrire une fonction permettant de partitionner ce dictionnaire en deux dictionnaires : un dictionnaire des élèves ayant la moyenne et un dictionnaire des élèves n'ayant pas la moyenne.

Exercice 4 : Écrire une fonction en Python qui prend en paramètre une liste de nombres entiers et qui renvoie un dictionnaire dont les clés sont les entiers de la liste et dont les valeurs sont 'pair' ou 'impair' selon la parité du nombre.

Exercice 5 : Ecrire une fonction comptage qui prend en paramètre un mot et renvoie un dictionnaire qui pour chaque caractère du mot associe le nombre d'occurrences de chaque lettre. On suppose le texte écrit en lettres capitales non accentuées.

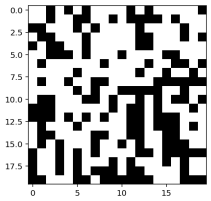
Exercice 6 : Étant donné un dictionnaire d et une clé k, il est facile de trouver la valeur correspondante `v=d[k]`. Cette opération s'appelle une recherche. Mais que faire si vous avez la valeur v et que vous voulez trouver la clé k ? Ecrire la fonction `recherche_inverse(d,v )` qui renvoie la première clé trouvée qui correspond à cette valeur. Modifier cette fonction en `rech_inv_liste(d,v)` pour qu'elle renvoie la liste des clés ayant cette valeur v.

3 Problème 3 : Chemin dans un labyrinthe

Pour ce problème vous pourrez importer les modules `numpy`, `random` et `matplotlib.pyplot`.

1. *Création d'un labyrinthe :*
   Un labyrinthe sera représenté par un tableau `numpy` à n lignes et n colonnes. Chaque cellule du labyrinthe est un entier qui vaut 0 si cette cellule est un mur, 1 si cette cellule correspond à un passage.

   Ecrire une fonction `creation_lab(n,p)`, qui à un entier naturel non nul n, et à un réel $p \in]0,1[$, associe un labyrinthe de taille n (i.e. n lignes et n colonnes), dont chaque cellule aura la valeur 1 avec une probabilité p, et 0 avec une probabilité $(1-p)$. (vous pourrez utiliser la fonction `random()` du module `random`), qui retourne un réel aléatoire entre 0 et 1 (selon une loi uniforme))
2. *Affichage d'un labyrinthe :*
   Ecrire une fonction `affiche(lab)`, de paramètre un labyrinthe donné sous forme d'un tableau numpy, qui affiche ce labyrinthe sous la forme ci-dessous. (vous pourrez utiliser la fonction `imshow` du module `matplotlib.pyplot`)



3. *Existence d'un chemin entre deux cellules :*
   Ecrire une fonction `existe_chemin(lab,x1,y1,x2,y2)`, qui retourne `True` s'il existe un chemin de la cellule d'indices $(x1, y1)$ à la cellule d'indices $(x2, y2)$, `False` sinon. Si jamais l'une de ces deux cellules est un mur on convient de retourner `False`. Pour construire votre fonction, vous effectuerez un parcours en profondeur à partir de la cellule de départ, en stockant les cellules à visiter dans une pile.
4. *Complexité :*
   Quelle est la complexité (en fonction de la taille n du labyrinthe) de la fonction `existe_chemin` ?
5. *Obtention d'un chemin entre deux cellules :*
   En modifiant la fonction précédente, écrire une fonction `chemin(lab,x1,y1,x2,y2)`, qui retourne `False` s'il n'existe pas de chemin de la cellule d'indices $(x1, y1)$ à la cellule d'indices $(x2, y2)$, et qui sinon renvoie un chemin possible, sous forme d'une liste de couples d'indices (désignant les cellules par où il faut passer). Pour cela, vous utiliserez un tableau auxiliaire `predecesseur` qui, lorsqu'un sommet est ajouté à la pile, indique le sommet qui a été visité précédemment. A la fin pour reconstituer le chemin, il suffira de remonter la chaîne des prédécesseurs depuis $(x2, y2)$.
6. *Si vraiment vous avez le temps :* Vous remarquerez que la fonction précédente renvoie un chemin qui parfois est très loin d'être optimal, sauriez-vous améliorer cela ?