
ITC 2° ANNÉE - TP 3 - PROG DYN : RECONSTRUCTION DE SOLUTION

Fichier fourni

Vous trouverez sur <https://sbautista.fr/itc2/> un fichier `itc2_tp_3_squelette.py` qui contient des fonctions fournies.

Exercice 1 (Rendu de monnaie)

(Pour le contexte de cet exercice, voir l'exercice 2 du TP 1)

Est fournie une fonction `memo_rendu` qui étant donnés un nombre de centimes `a_rendre` et un système monétaire `valeurs_pieces` renvoie un dictionnaire qui à différentes valeurs de centimes à rendre associe le nombre minimal de pièces requises pour les rendre.

Coder une fonction `pieces_rendues(a_rendre, valeurs_pieces)` qui renvoie une liste de pièces (dont chaque valeur appartient à `valeurs_pieces`) permettant de rendre `a_rendre` centimes de façon optimale.

Par exemple, l'appel `pieces_rendues(79, [1, 3, 6, 12, 24, 30])` doit renvoyer `[1, 24, 24, 30]` (ou n'importe quelle permutation de cette liste).

Exercice 2 (Le problème de la grenouille)

Est fournie une fonction `memo_grenouille` qui prend en entrée la liste des hauteurs des différentes pierres et qui renvoie un dictionnaire qui à différents indices de pierres i associe le coût pour la grenouille d'aller de la pierre i jusqu'à la dernière pierre.

Coder une fonction `longueurs_sauts` qui prend en entrée la liste des hauteurs des différentes pierres, et renvoie en sortie la liste des longueurs de sauts que la grenouille doit effectuer pour aller de la première à la dernière pierre en un coût minimum.

Par exemple, l'appel `longueurs_sauts([10, 11, 8, 12, 6, 13, 4])` doit renvoyer `[2, 2, 2]`.

On rappelle le contexte du problème de la grenouille : On s'imagine n pierres alignées, ayant différentes hauteurs $h_0, \dots, h_{n-1}$. Une grenouille commence à la pierre d'indice 0 et veut se rendre jusqu'à la pierre d'indice $n - 1$. À chaque étape, si la grenouille est sur la pierre d'indice i , elle peut

- soit se rendre sur la pierre d'indice $i + 1$ avec une coût $|h_{i+1} - h_i|$
- soit se rendre sur la pierre d'indice $i + 2$ avec un coût $|h_{i+2} - h_i|$

Exercice 3 (Somme maximale descendant une pyramide d'entiers)

(Pour le contexte de cet exercice, voir vos notes du cours 2)

Est fournie une fonction `memo_pyramide` qui prend en entrée une pyramide d'entiers et renvoie en sortie un dictionnaire qui à différentes coordonnées (`ligne`, `colonne`) dans la pyramide associe la somme maximale de la sous-pyramide en question.

Codez une fonction `chemin_pyramide` qui prend en entrée une pyramide d'entiers et renvoie le chemin des directions à prendre en descendant de la pyramide pour réaliser une somme maximale.

Par exemple, l'appel `chemin_pyramide(ex_pyramide_1)` doit renvoyer `["droite", "droite", "gauche", "gauche"]`

Exercice 4 (Distance d'édition entre deux mots)

(Pour le contexte de cet exercice, voir le cours 3)

Est fournie une fonction `memo_levenshtein` qui prend en entrée deux mots `u` et `v` (appelons leurs tailles respectives n et m) et qui renvoie un dictionnaire qui à tout couple (i, j) (avec $i \in \llbracket 0, n \rrbracket$ et $j \in \llbracket 0, m \rrbracket$) associe la distance d'édition entre `u[0:i]` et `v[0:j]`.

Coder une fonction `modifications_mots` qui prend en entrée deux chaînes de caractères et qui renvoie en sortie les modifications (d'insertion, suppression ou remplacement d'une lettre à la fois) qu'il faut faire pour passer de l'une à l'autre.

Par exemple, l'appel `modifications_mots("Dikshtre", "Dijkstra")` peut renvoyer la liste `['remplacer e par a', 'supprimer h', 'insérer j']`.