DS

Python

Festival de Musique

Lorsqu'une fonction est demandée par l'énoncé, même si vous ne proposez pas de code pour la définir, vous pouvez supposer qu'elle est définie et vous en servir dans les questions suivantes

Si vous êtes amené à repérer ce qui vous semble être une erreur d'énoncé, vous le signalerez sur votre copie et devrez poursuivre votre composition en expliquant les raisons des initiatives que vous avez prises

Gestion par dictionnaire

Dans cette partie indépendante des suivantes, on considère un festival disposant de plusieurs scènes accueillant des concerts entre 12 h (heure de début) et 24 h (heure de fin) sur une seule journée. La durée d'un concert est comprise entre 1 et 2 h.
Les groupes ou artistes et leurs horaires de programmation sont notés dans un dictionnaire dont les clés sont les noms des groupes ou artistes et les valeurs correspondent à une liste de 2 élément $[a;\ b]$

- a est l'heure de début du concert qui est un **entier** compris entre 12 et 23.
- $b = 1 + a + \varepsilon$ où ε est un réel compris entre 0 et 1.

Par exemple :

```
concert = {'Jimi Hendrix' : [12, 13.5], 'Blood, Sweat and Tears' : [13, 14],
'Joan Baez' : [15, 16.5], 'Creedence Clearwater Revival' : [15, 16],
'The Band' : [17, 17.5], 'Janis Joplin' : [18, 20],
'Jefferson Airplane' :  [15, 17], ...}
```

1) Proposer une fonction `heure_list(concert:dict) -> dict` qui prend en argument un dictionnaire `concert` comme décrit précédemment et renvoie un dictionnaire dont les clés sont les heures de concerts entre $[\![12; 23]\!]$ et les valeurs sont les listes des groupes ou artistes sur scène à ces heures.
Un artiste qui joue de 13 h à 14,5 h est présent à 13 h et à 14 h.

2) En déduire une fonction `heure_nb(concert:dict) -> dict` qui prend en argument un dictionnaire `concert` et renvoie un dictionnaire dont les clés sont les heures de concerts entre $[\![12; 23]\!]$ et les valeurs sont les nombres d'artistes sur scène à ces heures.

3) Afin de déterminer les heures de grande affluence durant la journée, rédiger une fonction `maxi_heure(concert:dict) -> list` qui prend en argument le dictionnaire

`concert` et renvoie la liste des heures où il y a le maximum d'artistes sur les scènes du festival. Cette liste peut contenir un seul élément mais il est fort probable qu'il y ait plusieures heures de même affluence maximale sur scène, d'où l'utilisation d'une liste.

Position du problème d'optimisation

Présentation de la structure

On s'intéresse à un festival de musique présentant une programmation variée avec de nombreux concerts numérotés par un indice $i \in [\![0, n-1]\!]$. Le concert d'indice i est défini par l'heure de début d_i, l'heure de fin f_i et sa note moyenne v_i (ou valeur) donnée par les critiques. Un concert est ainsi caractérisé par le triplet fourni sous forme de liste : $[d_i,\ f_i,\ v_i]$. Le festival propose ainsi une liste de n concerts notés :

$$C = [[d_0,\ f_0,\ v_0],\ [d_1,\ f_1,\ v_1], \ldots, [d_{n-1},\ f_{n-1},\ v_{n-1}]]$$

Problème d'optimisation

Les différentes contraintes imposées :

- Ne disposant pas du don d'ubiquité, il n'est possible d'assister qu'à un unique concert à la fois.
- Les concerts sont écoutés entièrement de la première à la dernier seconde. L'intersection entre les intervalles de deux concerts i et j auxquels on assiste est l'ensemble vide ou réduit à un nombre ($d_i = f_j$).
- On souhaite assister à un maximum de concerts tout en optimisant la note totale de l'ensemble de ces concerts. On cherche ainsi à déterminer le sous-ensemble S^C de C tel que $\sum_{i \in S^C} v_i$ est maximal.

Questions préliminaires

4) Dans l'hypothèse où l'on dispose d'une liste de concerts valides (c'est-à-dire qui ne se chevauchent pas dans le temps), écrire une fonction `valeur(concerts:list) -> float`. Cette fonction prend en argument une liste `concerts` au format présenté et renvoie la note totale de l'ensemble des concerts de la liste.

5) Pour les algorithmes présentés dans la suite, il est nécessaire de trier les concerts par ordre **croissant** d'heure de fin. On utilise le tri fusion.

a) Proposer une fonction `fusion(gauche:list, droite:list) -> list` (récursive ou itérative) qui prend en argument deux listes de concerts triés par ordre croissant d'heure de fin et renvoie la liste triée (toujours selon l'heure de fin) de l'ensemble des éléments des deux listes.

b) En déduire une fonction récursive `tri(concerts:list) -> list` qui prend en argument une liste de concerts et renvoie une liste triée des concerts par ordre croissant d'heure de fin.

c) On note C_n la complexité de l'algorithme précédent qui s'applique sur une liste de longueur n. Établir une relation de récurrence sur C_n puis conclure quant à la complexité.

d) Parmi les autres tris, on peut citer le tri par insertion. Comparer le tri par insertion et le tri fusion en complétant le tableau ci-dessous reproduit sur la copie à rendre :

	en place (oui/non)	complexité au pire	complexité au mieux
Tri fusion			
Tri insertion			

6) En s'appuyant notamment sur le tri précédent, proposer une fonction PYTHON `compatible(concerts:list) -> bool` qui renvoie un booléen précisant si la liste `concerts` donnée en argument est bien valide c'est-à-dire que la liste ne présente aucun concert qui se chevauche dans le temps.

Méthode gloutonne

7) L'algorithme glouton trie les concerts par instant de fin décroissant et choisit le concert se terminant le plus tard, puis le concert précédent compatible se terminant au plus tard, et ainsi de suite.

a) Proposer un code `glouton(concerts:list) -> list` qui prend en argument la liste des concerts et renvoie une liste des concerts compatibles obtenue en gloutonnant.

b) Quelle est la complexité de ce code ?

c) Quel est l'inconvénient de cette méthode ? et son avantage ?

Force brute

Une autre idée naïve consiste à tester toutes les combinaisons de concerts, puis de vérifier que les concerts ne se chevauchent pas pour enfin prendre celle de valeur maximale.

8) Dans le cas d'une liste d'entiers par exemple `[1,2,3]`, l'ensemble des sous-listes s'obtient en concaténant ou non `[1]` à l'ensemble des sous-listes de `[2,3]` qui sont `[]`, `[2]`, `[3]` et `[2,3]`.

a) Proposer une fonction récursive `tousous(concerts:list) -> list` renvoyant la liste de l'ensemble des sous-listes de la liste `concerts` donnée en argument.

b) Combien de combinaisons de concerts obtient-on pour une liste initiale de n concerts ?

9) En parcourant l'ensemble des sous-listes d'une liste de concerts tout en vérifiant la compatibilité des listes (c'est-à-dire le non chevauchement des horaires), on peut regrouper l'ensemble des listes de concerts compatibles.

a) En déduire une fonction naïve `toucompatible(concerts:list) -> list` renvoyant la liste des listes de concerts compatibles.

b) Quelle est la complexité de ce code ?

10) Un dernier parcours de liste permet enfin de conclure.

a) En déduire une fonction `maxichoix(concerts:list)->(float, list)` qui prend en argument la liste des concerts et renvoie la note maximale et la liste associée.

b) Quelle est la complexité de ce code ?

c) Quel est l'inconvénient de cette méthode ? et son avantage ?

Annexe

Exemple de fonctions, methodes et manipulations sur une liste `liste`.

Opération	Exemple
Accès direct de l'élément d'indice i	`liste[i]`
Longueur	`len(liste)`
Concaténation	`liste + [1, 2]`
Ajout en fin de liste	`liste.append(2)`
Suppression en fin de liste	`liste.pop()`
Suppression de l'élément d'indice i	`liste.pop(i)`
Extraction de tranche entre l'indice i (inclus) et j exclu	`liste[i:j]`
Duplication (n fois)	`[0] * n`
Création par compréhension	`[i ** 2 for i in range(8)]`