DS 2 d'informatique - MPSI - 18/12/2023

Le but du sujet est d'aider le père Noël pour l'organisation de sa distribution annuelle de jouets. Voici quelques consignes :

- ⋆ Les calculatrices sont interdites. Numérotez vos feuilles et faites apparaître les questions dans l'ordre de l'énoncé.
- ⋆ Les différentes parties peuvent être traitées de manière indépendante.
- ⋆ Les listes et dictionnaires évoqués dans les exemples se trouvent sur la dernière page.
- ⋆ Si vous repérez une erreur d'énoncé, signalez le sur votre copie et poursuivez votre composition.

1 Gestion du stock de jouets

Depuis le début de son activité en 1855, le père Noël gère son stock de jouets avec une liste de chaînes de caractères « `L: list[str]` » répertoriant tous les jouets de son entrepôt. Par exemple, si `L = L1` avec `L1` la liste donnée en annexe, cela signifie que l'entrepôt contient 2 camions de pompiers, 3 poupées Parpie, 2 jeux de construction Lebo ... Cette représentation n'étant pas pratique, le père Noël souhaite convertir `L` en un dictionnaire `D` qui associe à chaque jouet sa quantité dans l'entrepôt. Par exemple, à partir de `L1`, il obtient `D1` (voir annexes).

1. Écrire une fonction de signature `L_to_D(L: list[str]) -> dict[str: int]` qui prend en entrée la liste `L` et renvoie le dictionnaire `D`. Par exemple, `L_to_D(L1)` s'évalue en `D1`. On ferra en sorte de ne parcourir `L` qu'une seule fois.

À partir de maintenant, le stock de jouets est représenté par le dictionnaire « `D: dict[str: int]` » décrit ci-dessus.

2. Écrire une fonction de signature `compterJouets(D: dict[str: int]) -> int` qui renvoie le nombre de jouets dans l'entrepôt du père Noël. Par exemple, `compterJouets(D1)` vaut 17.
3. Écrire une fonction de signature `maxJouets(D: dict[str: int]) -> list[str]` qui renvoie la liste des jouets les plus présents dans l'entrepôt. Par exemple pour `D1`, ce sont les poupées Parpie et les puzzles Kopémon qui sont les plus présents (3 exemplaires de chaque). Ainsi, `maxJouets(D1)` s'évalue en `['poupée Parpie', 'puzzle Kopémon']`.

Chaque jour, les lutins du père Noël construisent de nouveaux jouets, ce qui nécessite une mise à jour régulière du stock. Pour garder un historique, le père Noël fait une copie journalière du dictionnaire `D` avant sa mise à jour.

4. À l'aide d'une boucle `for`, écrire une fonction de signature

   `copieD(D: dict[str: int]) -> dict[str: int]`

   qui renvoie une copie de `D`.

À la fin de la journée, chaque lutin enregistre les jouets qu'il a construit dans un dictionnaire « `C: dict[str: int]` ». Par exemple, si `C = {'Jeu de construction Lebo': 150, 'Livre Henry Botteur': 200}`, cela signifie que le lutin a construit 150 jeux de construction Lebo et 200 livres Henry Botteur. Les jouets construits par l'ensemble des lutins sont consignés dans une liste « `L_C: list[dict[str: int]]` » dont chaque élément est le dictionnaire `C` associé à l'un des lutins. Étant donnés le dictionnaire `D` et la liste `L_C`, le père Noël souhaite mettre à jour son stock. Par exemple, à partir de `D1` et de `L_C1`, il obtient `D2` (voir les annexes).

5. Écrire une fonction

   `ajoutProd(D: dict[str: int], L_C: list[dict[str: int]]) -> dict[str: int]`

   qui ajoute la production des lutins dans le stock. Le fonction commencera par créer une copie de `D` à l'aide de la fonction `copieD`, puis modifiera cette copie (sans modifier `D`). Par exemple, `ajoutProd(D1, L_C1)` s'évalue en `D2`.

2 Réorganisation de l'entrepôt

Dans cette partie, on appelle ***caractère spécial*** un caractère ne faisant pas partie des 52 lettres minuscules et majuscules de l'alphabet (`"a"`, `"b"`, ..., `"z"`, `"A"`, `"B"`, ..., `"Z"`). Par exemple, les signes de ponctuation (`" "`, `"!"`, `"."`, ...), les chiffres (`"0"`, `"1"`, ..., `"9"`), les lettres accentuées (`"é"`, `"à"`, `"É"`, ...) sont des caractères spéciaux.

En Python, on pourra utiliser les fonctions « `ord(c: str) -> int` » et « `chr(i: int) -> str` » qui établissent une correspondance entre les chaînes de caractères de taille 1 et les entiers. Par exemple :

s	"A"	"B"	"C"	...	"X"	"Y"	"Z"	"a"	"b"	"c"	...	"x"	"y"	"z"
ord(s)	65	66	67	...	88	89	90	97	98	99	...	120	121	122

i	65	66	67	...	88	89	90	97	98	99	...	120	121	122
chr(i)	"A"	"B"	"C"	...	"X"	"Y"	"Z"	"a"	"b"	"c"	...	"x"	"y"	"z"

s	" "	"!"	"."	"0"	"1"	...	"9"	"é"	"à"	"É"
ord(s)	32	33	46	48	49	...	57	233	224	201

i	32	33	46	48	49	...	57	233	224	201
chr(i)	" "	"!"	"."	"0"	"1"	...	"9"	"é"	"à"	"É"

On note `M` la liste sans doublon contenant les noms de tous les jouets présents dans l'entrepôt du père Noël. Par exemple, si le stock est décrit par le dictionnaire `D2`, alors la liste correspondante est la liste `M1` définie en annexe.

6. À l'aide d'une compréhension de liste, écrire une fonction

```
D_to_M(D: dict[str: int]) -> list[str]
```

qui renvoie la liste `M` décrite ci-dessus. La liste renvoyée ne doit pas contenir de doublon et votre programme ne doit être composé que d'une unique compréhension de liste. Par exemple, `D_to_M(D2)` s'évalue en la liste `M1`.

Dans la suite, on dira qu'une liste « `M: list[str]` » est ***standardisée*** si toutes ses chaînes de caractères sont non vides et ont pour premier caractère une lettre majuscule ou bien un caractère spécial. En d'autres termes, les noms commençant par l'une des 26 lettres minuscules sont interdits.

Afin de standardiser la nomenclature de son stock, le père Noël modifie la liste `M` obtenue par la fonction `D_to_M` : lorsque le nom d'un jouet commence par une lettre minuscule, il la remplace par sa version majuscule. Par exemple, à partir de `M1`, il obtient `M2` en remplaçant trois lettres minuscules par les lettres majuscules correspondantes.

7. (a) Écrire une fonction `strVide(M: list[str]) -> bool` qui renvoie un booléen indiquant si la chaîne de caractères vide appartient à `M`.
   (b) Écrire une fonction `standardisation(M: list[str]) -> list[str]` qui prend en entrée une liste de chaînes de caractères et modifie chaque chaîne de caractères commençant par une lettre minuscule en remplaçant cette dernière par la lettre majuscule correspondante. Votre fonction déclenchera une erreur si `M` contient une chaîne de caractères vide. Par exemple, `standardisation(M1)` s'évalue en `M2`.

On suppose maintenant que la liste `M` est standardisée. Afin de gagner en productivité, le père Noël souhaite trier son stock de jouets par ordre alphanumérique (c'est à dire par ordre alphabétique en autorisant les caractères spéciaux). Il confie cette tache à 27 lutins numérotés de 0 à 26. Chacun des 26 premiers lutins est associé à une lettre de l'alphabet et va gérer les jouets dont le nom commence par cette lettre. Le 27$^{\text{ème}}$ lutin va gérer les jouets dont le nom commence par un caractère spécial.

8. Écrire une fonction `repartition(M: list[str]) -> list[list[str]]` qui renvoie une liste `R` de taille 27 telle que `R[i]` est l'ensemble des jouets attribués au lutin numéro `i`. On pourra supposer sans le vérifier que la liste est bien standardisée. Par exemple à partir de `M2`, on obtient la liste `R1` de l'annexe.

Maintenant que chacun des 27 lutins est en charge d'une partie de l'entrepôt, sa mission est de trier les jouets qui lui sont attribués. Tout d'abord, il faut être capable de comparer deux chaînes de caractères suivant l'ordre alphanumérique. Pour comparer deux chaînes de caractères `s1` et `s2`, on utilise le même principe que pour l'ordre alphabétique d'un dictionnaire papier. En Python, on considérera que le caractère `c1` est avant `c2` dans l'ordre alphanumérique lorsque `ord(c1)` $\leqslant$ `ord(c2)`.

9. Écrire une fonction `comp(s1: str, s2: str) -> int` qui renvoie :

$$\begin{cases} -1 & \text{si } \texttt{s1} \text{ est strictement avant } \texttt{s2} \text{ dans l'ordre alphanumérique} \\ 0 & \text{si } \texttt{s1} \text{ et } \texttt{s2} \text{ sont égaux} \\ 1 & \text{si } \texttt{s1} \text{ est strictement après } \texttt{s2} \text{ dans l'ordre alphanumérique} \end{cases}$$

Remarque : en Python, on pourrait comparer `s1` et `s2` directement avec les opérateurs booléens habituels (`<`, `<=`, `==`, ...), mais on se l'interdit dans cette question. Les seules comparaisons autorisées sont les comparaisons entre deux entiers `ord(c1)` et `ord(c2)` où `c1` et `c2` sont des caractères.
Par exemple :

s1	"A"	"B"	""	""	"abcd"	"abc"	"abed"
s2	"b"	"a"	""	"abcd"	""	"abcd"	"abcd"
comp(s1, s2)	-1	-1	0	-1	1	-1	1

Étant donnée une liste « `M: list[str]` » dont les éléments sont distincts deux à deux, on souhaite construire une liste `T` ayant les mêmes éléments que `M` et triée par ordre alphanumérique. Pour cela, on va utiliser le ***tri par insertion*** dont voici le pseudo-code :

(Étape 1) Initialement, la liste `T` est vide.
(Étape 2) On parcourt tous les éléments `e` de `M`. Pour chaque `e` :
  (Étape 2.a) On définit une variable `i` de la manière suivante :
    – Si tous les éléments de `T` sont strictement avant `e` dans l'ordre alphanumérique, on pose `i = len(T)`.
    – Sinon, `i` est défini comme le plus petit indice tel que `comp(T[i], e) = 1`.
  (Étape 2.b) On insère `e` à l'indice `i` dans `T`.

Voici l'évolution de ces différentes variables lors du tri de la liste `M = ["Abc", "Az", "Abb", "Abcd", "Abd"]` :

	e	i	T
Initialisation	×	×	[]
Fin du 1er tour de boucle	"Abc"	0	["Abc"]
Fin du 2ème tour de boucle	"Az"	1	["Abc", "Az"]
Fin du 3ème tour de boucle	"Abb"	0	["Abb", "Abc", "Az"]
Fin du 4ème tour de boucle	"Abcd"	2	["Abb", "Abc", "Abcd", "Az"]
Fin du 5ème tour de boucle	"Abd"	3	["Abb", "Abc" "Abcd", "Abd", "Az"]

10. (a) À l'aide d'une boucle `while`, écrire une fonction de signature

`indIns(T: list[str], e: str) -> int`

qui renvoie l'entier `i` comme défini à l'étape 2.a. On pourra supposer sans le vérifier que `T` est triée et que `e` n'apparaît pas dans `T`.

(b) En supposant que `e` contient au plus 100 caractères, quelle est la complexité de la fonction `indIns` ? Justifier.

11. (a) Écrire une fonction de signature `ins(T: list[str], e: str, i: int) -> list[str]` qui insère l'élément `e` dans `T` à l'indice `i` (voir étape 2.b). Votre fonction renverra la liste obtenue après insertion et ne devra pas modifier la liste initiale `T`. Si la condition `0` $\leqslant$ `i` $\leqslant$ `len(T)` n'est pas respectée, votre fonction déclenchera une erreur.

(b) Quelle est la complexité de la fonction `ins` ? Justifier.

12. (a) À l'aide des questions précédentes, écrire une fonction de signature

```
triIns(M: list[str]) -> list[str]
```

qui renvoie la liste triée `T` obtenue en appliquant un tri par insertion sur `M`.

(b) En supposant que chaque élément de `M` contient au plus 100 caractères, quelle est la complexité de la fonction `triIns` ? Justifier.

Dans le but d'améliorer le temps d'exécution de la fonction `triIns`, on propose de réécrire l'étape 2.a à l'aide d'une recherche dichotomique.

13. (a) Écrire une fonction de signature

```
indInsDicho(T: list[str], e: str) -> int
```

qui renvoie l'entier `i` comme défini à l'étape 2.a à l'aide d'une recherche dichotomique. On garantira une complexité logarithmique en la taille de `T`.

(b) On appelle `triInsDicho` la fonction obtenue en prenant le code de `triIns` et en y remplaçant les appels à `indIns` par des appels à `indInsDicho`. En supposant que chaque élément de `M` contient au plus 100 caractères, quelle est la complexité de la fonction `triInsDicho` ? Justifier.

3 Lettres au père Noël

Soit $n \in \mathbb{N}$ un entier. Pour s'entraîner à gérer les lettres des enfants, le père Noël demande à sa femme, la mère Noël, de générer toutes les combinaisons possibles de jouets. Ainsi, étant donnée une liste de jouets « `J: list[str]` » sans doublon, la mère Noël doit créer une liste de listes « `Comb: list[list[str]]` » dont les sous-listes sont de taille n et représentent toutes les combinaisons possibles de n jouets. Attention, si deux listes contiennent les mêmes jouets (éventuellement dans un ordre différent), alors elles sont considérées comme identiques ; une seule de ces deux listes doit donc apparaître dans `Comb`. De plus, chaque jouet peut apparaître au plus une fois dans une sous-liste de `Comb`. Par exemple, si $n = 3$ et `J = J1`, alors `Comb = Comb1` où `J1` et `Comb1` sont définies en annexe.

14. Écrire une fonction de signature `makeComb(J: list[str], n: int) -> list[list[str]]` qui construit la liste `Comb`.

4 Optimisation de la tournée

Le jour de Noël, le père Noël ne peut pas prendre tous les jouets dans sa hotte en une seule fois, mais souhaite tout de même la remplir le plus possible. Formellement, on associe à la hotte une capacité $C \in \mathbb{N}$ et à chaque jouet un entier naturel représentant le volume qu'il occupe. Ainsi, si on note $k \in \mathbb{N}$ le nombre total de jouets, on dispose d'une liste triée $P = [p_0, p_1, \ldots, p_{k-1}]$ où $p_i \in \mathbb{N}$ est le volume du jouet numéro i. Le père Noël souhaite donc choisir un sous-ensemble $I \subset [\![0, k-1]\!]$ tel que :

$$\mathcal{A}(I) : \sum_{i \in I} p_i \leqslant C$$

La stratégie du père Noël est dite ***optimale*** si pour tout $C \in \mathbb{N}$ et toute liste P, cette stratégie lui permet de construire un ensemble I vérifiant $\mathcal{A}(I)$ et tel que pour tout $J \subset [\![0, k-1]\!]$ vérifiant $\mathcal{A}(J)$, on ait :

$$\sum_{j \in J} p_j \leqslant \sum_{i \in I} p_i$$

Stratégie gloutonne 1. La première stratégie consiste à emporter en priorité le jouet avec le plus petit volume. Le père Noël emporte donc les jouets 0, 1, 2, ... et s'arrête lorsque l'objet suivant ne rentre plus dans la hotte.

Par exemple, avec $C = 7$ et $P = [3, 3, 5, 6]$, on obtient $I = \{0, 1\}$. En effet, les objets d'indices 0 et 1 prennent un volume total de $3 + 3 = 6$. Même si la hotte n'est pas complètement remplie ($6 < C$), il n'est plus possible d'y ajouter d'objet.

15. (a) À l'aide d'une boucle `while`, écrire une fonction de signature

```
strat1(C: int, P: list[int]) -> list[int]
```

qui renvoie une liste contenant les éléments de l'ensemble I obtenu avec la stratégie décrite ci-dessus. On rappelle que la liste `P` est supposée triée.

(b) Cette stratégie est-elle optimale ?

Stratégie gloutonne 2. La deuxième stratégie consiste à emporter en priorité le jouet avec le plus grand volume. Le père Noël s'intéresse donc au jouet $n-1$, puis $n-2$, …, 2, 1, 0. Pour chacun de ces jouets, s'il reste suffisamment de place dans sa hotte, le père Noël emporte le jouet, sinon il passe au jouet suivant. Par exemple, avec $C = 14$ et $P = [1, 2, 2, 5, 7, 7, 8]$, on obtient $I = \{0, 3, 6\}$.

16. (a) Écrire une fonction de signature

```
strat2(C: int, P: list[int]) -> list[int]
```

qui renvoie une liste contenant les éléments de l'ensemble I obtenu avec la stratégie décrite ci-dessus. On rappelle que la liste `P` est supposée triée.

(b) Cette stratégie est-elle optimale ?

Stratégie de type "programmation dynamique". Pour terminer, étudions une stratégie optimale pour le père Noël. Soit $P = [p_0, p_1, \dots, p_{k-1}]$ une liste d'entiers naturels. Pour chaque $k' \in [\![0, k]\!]$ et chaque $C' \in [\![0, C]\!]$, on définit l'entier $s^*(k', C')$ par :

$$s^*(k', C') = \min\left\{\sum_{i\in I} p_i : I \subset [\![0, k'-1]\!] \text{ et } \sum_{i\in I} p_i \leqslant C'\right\}$$

Dans la suite, notre objectif est de construire un ensemble $I^*(k', C') \subset [\![0, k'-1]\!]$ vérifiant :

$$s^*(k', C') = \sum_{i\in I^*(k',C')} p_i$$

On remarque que pour $k' = 0$ et pour tout $C' \in [\![0, C]\!]$, on a $s^*(0, C') = 0$. Donc $I^*(0, C') = \varnothing$ convient. De plus, si $k' > 0$, alors 2 cas peuvent se produire :

- ⋆ $p_{k'-1}$ apparaît dans $I^*(k', C')$. Dans ce cas, l'ensemble $J = I^*(k', C') \setminus \{p_{k'-1}\}$ vérifie $J \subset [\![0, k'-2]\!]$ et :

$$s^*(k'-1, C'-p_{k'-1}) = \sum_{j\in J} p_j$$

- ⋆ $p_{k'-1}$ n'apparaît pas dans $I^*(k', C')$. Dans ce cas, l'ensemble $J = I^*(k', C')$ vérifie $J \subset [\![0, k'-2]\!]$ et :

$$s^*(k'-1, C') = \sum_{j\in J} p_j$$

On en déduit que pour tout $C' \in [\![0, C]\!]$ et tout $k' \in [\![1, n]\!]$, l'ensemble $I^*(k', C')$ peut être défini par :

- ⋆ Si $p_{k'-1} > C'$ ou $s^*(k'-1, C') \geqslant s^*(k'-1, C'-p_{k'-1}) + p_{k'-1}$, alors :

$$I^*(k', C') = I^*(k'-1, C')$$

- ⋆ Sinon :

$$I^*(k', C') = I^*(k'-1, C'-p_{k'-1}) + [p_{k'-1}] \text{ (il s'agit d'une concaténation)}$$

17. En déduire une fonction de signature

```
strat3(C: int, P: list[int]) -> list[int]
```

qui renvoie une liste contenant les éléments de l'ensemble $I^*(k, C)$ obtenu avec la stratégie décrite ci-dessus.

Remarque. Le problème étudié dans cette partie est un cas particulier du "problème du sac à dos".

5 Annexe

```
L1 = ['Camion de pompiers', 'poupée Parpie', 'Jeu de construction Lebo',
      'poupée Parpie', 'poupée Parpie', 'Manga Houane Pice', 'Écharpe bleue',
      'Jeu de société SkyDassin', 'puzzle Kopémon', 'Jeu de société SkyDassin',
      'figurine heroman', 'Jeu vidéo Delza', 'puzzle Kopémon', 'écharpe verte',
      'Jeu de construction Lebo', 'Camion de pompiers', 'puzzle Kopémon']


D1 = {'Camion de pompiers': 2, 'poupée Parpie': 3, 'Jeu de construction Lebo': 2,
      'Manga Houane Pice': 1, 'Écharpe bleue': 1, 'Jeu de société SkyDassin': 2,
      'puzzle Kopémon': 3, 'figurine heroman': 1, 'Jeu vidéo Delza': 1,
      'écharpe verte': 1}


# len(L_C1) = 3, il y a donc 3 lutins.
L_C1 = [
    {'Jeu de construction Lebo': 150, 'Livre Henry Botteur': 200},
    {'Manga Houane Pice': 500},
    {'Camion de pompiers': 100, 'poupée Parpie': 100, 'Jeu de construction Lebo': 100}]


D2 = {'Camion de pompiers': 102, 'poupée Parpie': 103, 'Jeu de construction Lebo': 252,
      'Manga Houane Pice': 501, 'Écharpe bleue': 1, 'Jeu de société SkyDassin': 2,
      'puzzle Kopémon': 3, 'figurine heroman': 1, 'Jeu vidéo Delza': 1,
      'écharpe verte': 1, 'Livre Henry Botteur': 200}


M1 = ['Camion de pompiers', 'poupée Parpie', 'Jeu de construction Lebo',
      'Manga Houane Pice', 'Écharpe bleue', 'Jeu de société SkyDassin',
      'puzzle Kopémon', 'figurine heroman', 'Jeu vidéo Delza', 'écharpe verte',
      'Livre Henry Botteur']


M2 = ['Camion de pompiers', 'Poupée Parpie', 'Jeu de construction Lebo',
      'Manga Houane Pice', 'Écharpe bleue', 'Jeu de société SkyDassin',
      'Puzzle Kopémon', 'Figurine heroman', 'Jeu vidéo Delza', 'écharpe verte',
      'Livre Henry Botteur']


R1 = [[], [], ['Camion de pompiers'], [], [], ['Figurine heroman'], [], [], [],
      ['Jeu de construction Lebo', 'Jeu de société SkyDassin', 'Jeu vidéo Delza'],
      [], ['Livre Henry Botteur'], ['Manga Houane Pice'], [], [],
      ['Poupée Parpie', 'Puzzle Kopémon'], [], [], [], [], [], [], [], [], [], [],
      ['Écharpe bleue', 'écharpe verte']]


J1 = ['Camion de pompiers', 'poupée Parpie', 'Jeu de construction Lebo',
      'Manga Houane Pice', 'Jeu de société SkyDassin']


Comb1 = [['Camion de pompiers', 'poupée Parpie', 'Jeu de construction Lebo'],
         ['Camion de pompiers', 'poupée Parpie', 'Manga Houane Pice'],
         ['Camion de pompiers', 'poupée Parpie', 'Jeu de société SkyDassin'],
         ['Camion de pompiers', 'Jeu de construction Lebo', 'Manga Houane Pice'],
         ['Camion de pompiers', 'Jeu de construction Lebo', 'Jeu de société SkyDassin'],
         ['Camion de pompiers', 'Manga Houane Pice', 'Jeu de société SkyDassin'],
         ['poupée Parpie', 'Jeu de construction Lebo', 'Manga Houane Pice'],
         ['poupée Parpie', 'Jeu de construction Lebo', 'Jeu de société SkyDassin'],
         ['poupée Parpie', 'Manga Houane Pice', 'Jeu de société SkyDassin'],
         ['Jeu de construction Lebo', 'Manga Houane Pice', 'Jeu de société SkyDassin']]
```