Info
PSI *
Année 2022-2023

DS 2 Info (11/03/2023)

Le corrigé sortira assez vite sur le cahier, mais après lecture d'une dizaine de copies.

Bases de données "Philatélie"

La structure est la suivante :

– Table Timbres, dont les attributs sont **idtimbre** (entier), **pays** (str), **valeur** (flottant), **unite** (entier), **annee** (entier) et **valable** (booléen).

– Table Unites, dont les attributs : **idunite** (entier), **symbole** (str) et **nomunite** (str).

– Table Cours, dont les attributs sont **monnaie** (entier) et **euros** (flottant).

– Table Rangement, dont les attributs sont **timbre** (entier), **valise** (entier), **compartiment** (entier), **rangee** (entier) et **colonne** (entier).

Quelques explications :

– Plusieurs timbres identiques peuvent être rangés, et potentiellement empilés au même endroit (donc dans le même compartiment de la même valise, à la même colonne de la même rangée), mais parfois à côté, en tout cas la flexibilité nécessaire fait qu'il n'y a pas d'attribut pour le nombre d'exemplaires (tant mieux en vue des exercices à venir).

– Certaines timbres n'ont plus cours légal, d'où le booléen valable. Pour simplifier, cela correspond exactement à une unité dont le cours n'est pas mentionné dans la table appropriée. Pour les autres monnaies, l'attribut euros détermine combien d'euros vaut un de la monnaie en question. Il y a un enregistrement pour les euros aussi, afin d'éviter de faire échouer les requêtes associées.

– La table Cours n'est pas rassemblée avec la table Unite pour éviter d'utiliser la valeur NULL, qui est hors programme.

Bien lire ce qui va suivre pour comprendre les schémas !

Les clés primaires sont les suivantes :

– Table Timbres : idtimbre.
– Table Unites : idunite.
– Table Cours : monnaie.

– Table Rangement : timbre.

Les clés étrangères sont les suivantes :

– Table Timbres : unite vers la table Unites.
– Table Unites : aucune.
– Table Cours : monnaie vers la table Unites.
– Table Rangement : timbre vers la table Timbres.

Q1 : Écrire une requête déterminant le nombre total de timbres dont je dispose.

.

Q2 : Écrire une requête déterminant le nombre total de timbres distincts, c'est-à-dire le nombre d'enregistrements uniques en excluant l'identfiant des timbres.

.

Q3 : Ecrire clairement ce que fait cette requête :

```
SELECT valise FROM Rangement GROUP BY valise HAVING COUNT(*) = (SELECT COUNT(*) FROM Rangement GROUP BY valise ORDER BY COUNT(*) DESC LIMIT 1)
```

.

Q4 : Écrire une requête déterminant la valeur faciale, le nom de l'unité monétaire, l'endroit où est rangé et l'âge du plus vieux timbre dont je dispose.

.

Q5 : Écrire une requête déterminant le symbole de la monnaie la plus forte enregistrée dans la base de données.

.

Q6 : Écrire une requête déterminant tous les endroits où sont rangés des timbres en Hryvnia (monnaie ukrainienne).

.

Q7 : Écrire une requête déterminant la valeur d'échange de l'ensemble de ma collection, en excluant tous les timbres n'ayant plus cours légal.

.

Plus longue sous-suite commune

La fonction max est autorisée, calculette aussi, à la fin on me rend les sujets.

Définition : Soit $S = (s_0, ..., s_{n-1},$ une suite (de caractères, de mots etc...).

Une sous-suite de S (ou suite extraite dans les cours de maths) est une suite

$Z = (z_0, ..., z_{p-1}$ (évt vide) de termes de S construite de la façon suivante :

- on se donne une suite strictement croissante d'indices $0 \leqslant i_0 \leqslant i_1 \leqslant ... \leqslant i_{p-1}$.

- on pose $z_k = s_{i_k}$, $0 \leqslant k \leqslant\leqslant p-1$.

Bref $Z = (z_k)_k$, est obtenue en prélevant dans l'ordre certains éléments de S.

Soient S et T deux suites, de longueurs respectives m, n.

On voudrait trouver une suite Z qui serait ss-suite de S et de T de longueur maximale.

On dira de cette ss-suite que c'est une **plus longue ss-suite commune**

à S et T (notée PLSSC à S et T).

Un exemple : $S =$ 'formulation' et $T =$'formalisation'.

Une solution optimale est donnée avec $p = 10$, indices extraits :

$I_{10} = (0, 1, 2, 3, 5, 6, 7, 8, 9, 10)$ et $J_{10} = (0, 1, 2, 3, 5, 8, 9, 10, 11, 12)$.

La ss-suite commune correspondante est : $Z =$'formlation'.

Pour programmer tout ça, on note $L(p, q)$ la longueur d'une PLSSC .

Q1. Pouvez-vous expliquer (si besoin, admettre pour la suite) :

$$(\text{RR})\begin{cases} \text{Si} \quad p = 0 \quad \text{ou} \quad q = 0, \quad \text{alors} \quad L(p,q) = 0 \\ \text{Si} \quad s_{p-1} = t_{q-1}, \quad \text{alors} \quad L(p,q) = L(p-1,q-1) + 1 \\ \text{Si} \quad s_{p-1} \neq t_{q-1}, \quad \text{alors} \quad L(p,q) = max(L(p-1,q), L(p,q-1)) \end{cases} .$$

.

On va écrire 2 programmes qui utilisent RR pour calculer la longueur d'une **PLSSC**.

Puis (un peu plus difficile) comment reconstituer une **PLSSC**.

Q2. Version itérative, programme suivant à compléter (les ? ?) .

```
def PLSSC1(S,T): ### #S, T : str; #Renvoie un dictionnaire ###
    m, n = len(S), len(T)
    D = { p: {} for p in range(0, ?    ?)}
    for q in range(0, ?    ?):
        D[0][q] = 0
    for p in range(0, ?   ?):
        D[p][0] = 0
    for p in range(1, ?   ?):
        for q in range(1, ?   ?):
            if S[p-1] == T[?   ?]:
                D[p][q] = D[?   ?][?    ?][0]+1
            else:
                ??
```

```
            ??
    return ?   ?
```

Q3. Version récursive, programme suivant à compléter (les ? ?)

```
def PLSSC2(S, T, D):
    m, n = len(S), len(T)
    if m not in D:
        D[m] = {}
    if m == 0 or n == 0:
        D[m][n] = 0
        return ?       ?
    elif S[-1] == T[-1]:
        if m-1 not in D or n-1 not in D[m-1]:
            r =?                                  ?
        else:
            r = ?                                 ?
        D[m][n] = r + 1
        return D[m][n][0]
    else:
        if m-1 not in D or n not in D[m-1]:
            r1 = PLSSC2(?                          ?)
        else:
            r1 = ?                                  ?
        if n-1 not in D[m]:
            r2 = PLSSC2(?                          ?)
        else:
            r2 = ?                                  ?
        if r1 > r2:
           ??


           ??
        return ?         ?
```

Comment utiliser ce code ?

.

Q4. Donner un variant de boucle (ss preuve) pour justifier la terminaison.

.

Quelle grande idée du cours est utilisée dans ces 2 programmes ?

.

Q5. Donner un majoration de la complexité en mémoire, temps, explications efficaces.

.

Q6. On va maintenant reconstruire explicitement une **PLSSC** à S et T à partir des informations contenues dans le dictionnaire crée par **PLSSC1(S,T)**, mais il faut la modifier.

On va compléter le dictionnaire D qui est créé dans cette fonction.

Exemple $D[p][q] = D[p-1][q-1][0]+1, (-1,-1)$, l'information de droite $(-1,-1)$ sera remplacée (aux bon moments) par $(-1,0),(0,-1),(0,0)$ suivant le cas utilisé dans RR.

Compléter votre code **PLSSC1(S,T)** pour l'adapter à cette idée,

avec une couleur **différente** de Q2.

Puis compléter le code récursif suivant pour réaliser notre objectif.

```
def PLSSC(S,T):
    m, n = len(S), len(T)
    D=? ?
    def reconstruirePLSSC(p, q):
        if p == 0 or q == 0:
            return ""
        else:
            if D ?    ?==(-1,-1):
                return reconstruirePLSSC(p-1, q-1) + T[?    ?]
            elif D ?    ?==(-1, 0):
                return reconstruirePLSSC(?    ?, q)
            elif D ?    ?==(0, -1) :
```

```
            return reconstruirePLSSC(?   ?, ?    ?)

    return reconstruirePLSSC(m,n)
```

Éléments de correction

Bases de données

Remarques générales.

Sur cet exercice, l'un des vrais problèmes était le sens de chaque attribut.

Tu as le droit de te méprendre dans la compréhension des attributs, mais ça doit rester cohérent et dans le bon ordre.

Bref, je paye la cohérence et l'ordre des "choses".

Apprendre à faire de vraies jointures !

Jamais de Having sans le group by et la fonction d'agrégat.

Par défaut le group by est ASC, mais c'est mieux de le marquer.

Q1 : SELECT COUNT(*) FROM Timbres 1 pt

Q2 : SELECT COUNT(DISTINCT pays, valeur, unite, annee, valable) FROM Timbres 3 pt

Rq : ne fonctionne pas sur tous les logiciels.

Q3 : Requête déterminant les valises contenant le plus de timbres. 2 pt

Q4 : SELECT valeur, nomunite, valise, compartiment, rangee, colonne, 2023 - annee FROM Timbres JOIN Unites ON unite = idunite JOIN Rangement ON idtimbre = timbre WHERE annee = (SELECT MIN(annee) FROM Timbres) 4 pt

Q5 : SELECT symbole FROM Unites JOIN Cours ON idunite = monnaie ORDER BY euros DESC LIMIT 1 3 pt

Q6 : SELECT valise, compartiment, rangee, colonne FROM Rangement JOIN Timbres ON timbre = idtimbre JOIN Unites ON idunite = unite WHERE nomunite = "hryvnia" 3 pt

Q7 : SELECT SUM(valeur * euros) FROM Timbres JOIN Cours ON unite = monnaie WHERE valable 3 pt

Plus longue sous-suite commune

Applications : plagiats , génétique...

Q1. Première ligne ..., deuxième attention notations Python, allongement de la **PLSSC**.

Troisième , on regarde avant et on prend le mieux.

Rq si $z_k \neq s_{p-1}, t_{q-1}$ alors Z est une **PLSSC** de $S[0, p-1]$ et $T[0, q-1]$.

Mieux plus loin.3pt

Q4. La somme des longueurs des chaines en traitement $\in \mathbb{N}$ strictement décroissante 2 pt.

La grande idée du cours est bien sur la mémoisation pour éviter les appels redondants.2 pt

Q5. Le dictionnaire occupera au plus $(m+1)(n+1)$ paires de places en mémoire.

Le nombre d'itérations et le nombre d'appels récursif sera

(grâce à la mémoisation) au pire en $\mathcal{O}(mn)$.3 pt

Les codes :

```
def PLSSC1(S,T): ( 5pt + 2pt pour le prog final )
### #S, T : str; #Renvoie un dictionnaire ###
    m, n = len(S), len(T)
    D = { p: {} for p in range(0, m+1)}
    for q in range(0, n+1):
        D[0][q] = 0, (0,0)
    for p in range(0, m+1):
        D[p][0] = 0, (0,0)
    for p in range(1, m+1):
        for q in range(1, n+1):
            if S[p-1] == T[q-1]:
                D[p][q] = D[p-1][q-1][0]+1, (-1,-1)
            else:
                r1 = D[p-1][q][0]
                r2 = D[p][q-1][0]
```

```
                if r1 > r2:
                    D[p][q] = r1, (-1,0)
                else:
                    D[p][q] = r2, (0, -1)
    return D
#S="formulation"
#T="formalisation"

def PLSSC2(S, T, D): 5 pt
    m, n = len(S), len(T)
    if m not in D:
        D[m] = {}
    if m == 0 or n == 0:
        D[m][n] = 0, (0,0)
        return D[m][n][0]
    elif S[-1] == T[-1]:
        if m-1 not in D or n-1 not in D[m-1]:
            r = PLSSC2(S[0:m-1], T[0:n-1], D)
        else:
            r = D[m-1][n-1][0]
        D[m][n] = r + 1, (-1,-1)
        return D[m][n][0]
    else:
        if m-1 not in D or n not in D[m-1]:
            r1 = PLSSC2(S[0:m-1], T[0:n], D)
        else:
            r1 = D[m-1][n][0]
        if n-1 not in D[m]:
            r2 = PLSSC2(S[0:m], T[0:n-1], D)
        else:
            r2 = D[m][n-1][0]
        if r1 > r2:
            D[m][n] = r1, (-1,0)
        else:
            D[m][n] = r2, (0,-1)
        return D[m][n][0]


def PLSSC(S,T): 4 pt
    m, n = len(S), len(T)
    D = {}
    D=PLSSC1(S, T)
    def reconstruirePLSSC(p, q):
```

```
        if p == 0 or q == 0:
            return ""
        else:
            if D[p][q][1]==(-1,-1):
                return reconstruirePLSSC(p-1, q-1) + T[q-1]
            elif D[p][q][1]==(-1, 0):
                return reconstruirePLSSC(p-1, q)
            elif D[p][q][1]==(0, -1) :
                return reconstruirePLSSC(p, q-1)

    return reconstruirePLSSC(m,n)
```

Q1 : Attention aux notations, lettres muettes...$p = len(Z)$.

-Soit $s_{m-1} = t_{n-1}$ et dans ce cas : $z_p = s_{m-1} = t_{n-1}$ et $Z[0 : p-1] = (z_0, \dots z_{p-2})$

est une PLSSC à $S[0 : m-1]$ et $T[0 : n-1]$ (attention : notations Python).

-Soit $s_{m-1} \neq t_{n-1}$ et dans ce cas :
(a) si $z_p \neq s_{m-1}$, alors Z est une PLSSC à $S[0 : m-1]$ et T ;
(b) si $z_p \neq t_{n-1}$, alors Z est une PLSSC à S et $T[0 : n-1]$.
(et en particulier, si $z_p \neq s_{m-1}, t_{n-1}$, alors Z est une PLSSC à $S[0 : m-1]$ et $T[0 : n-1]$).

Remarquons qu'il permet de ramener la recherche d'une PLSSC à S et T à celles de PLSSC dans des sous-chaînes de S et T. C'est dans ce sens que l'on dit que le problème a une propriété de sous-structure optimale : une solution optimale d'une certaine taille se construit à partir de solutions également optimales pour un problème de plus petite taille.

On peut faire plus rigoureux, mais...Et on reviendrait au cours du samedi ...