SESSION 2023

CONCOURS COMMUN INP

ÉPREUVE COMMUNE – FILIÈRES TPC ET TSI

INFORMATIQUE

Durée : 3 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- *Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.*
- *Ne pas utiliser de correcteur.*
- *Écrire le mot FIN à la fin de votre composition.*

Cette proposition de sujet est une variation sur le sujet d'informatique CCINP-TSI de 2021. Elle se veut conforme au nouveau programme d'informatique des TPC et TSI.

Les calculatrices sont interdites.

Le sujet est composé de deux parties indépendantes.

Le sujet comporte :

- le texte du sujet : page 1 à page 6 ;
- le Document Réponse (DR) : 2 pages.

Le Document Réponse (DR) doit être rendu dans son intégralité avec la copie.

Optimisation de rendement d'une entreprise de livraison

Une entreprise de livraison dispose de plusieurs locaux en France et chacun possède plusieurs camions de livraisons. Celle-ci souhaite optimiser le chargement de ses camions pour diminuer ses frais de fonctionnement.

Partie I - Données liées aux livraisons conservées par l'entreprise

À chaque livraison, l'entreprise stocke des données relatives à celle-ci.
L'entreprise dispose de 20 locaux, numérotés de 1 à 20, disposant chacun d'un certain nombre de camions. Pour faciliter ses livraisons, l'entreprise découpe la France en 30 zones et associe trois zones possibles de livraisons à chaque local.
Ces données sont enregistrées dans une base de données composée de trois tables :
La table `livraison` constituée des champs suivants :

- *date :* date de la livraison au format "jj-mm-aaaa" (chaîne de caractères) ;
- *heure :* heure de la livraison au format : "hh-mm-ss" (chaîne de caractères) ;
- *id_client :* identifiant du client recevant la livraison (entier) ;
- *id_local :* identifiant du local de l'entreprise (entier compris entre 1 et 20) ;
- *id_commande :* identifiant de la commande (chaîne de caractères).

La table `client` constituée des champs suivants :

- *id :* identifiant du client (entier) ;
- *zone :* entier compris entre 1 et 30.

La table `local` constituée des champs suivants :

- *id :* identifiant du local (entier compris entre 1 et 20) ;
- *zone1 :* entier ;
- *zone2 :* entier ;
- *zone3 :* entier.

Q1. Donner une clé primaire pour la table `livraison`. Y-a-t-il d'autres possibilités ?

Q2. Écrire une requête SQL permettant d'obtenir les identifiants des clients livrés le 10 janvier 2023.

Q3. Écrire une requête SQL permettant de récupérer les heures de toutes les livraisons ayant eu lieu dans la zone 5 le 2 mars 2023.

Q4. Écrire une requête SQL permettant de compter le nombre de livraisons effectuées le 3 février 2023 par des camions dont les locaux ne livrent que dans des zones possibles inférieures ou égales à 10.

Partie II - Optimisation du chargement

Chaque camion de l'entreprise peut charger une cargaison jusqu'à un poids maximal noté P_{max}.
L'entreprise dispose de différentes informations provenant de ses fournisseurs :

- le poids de chaque produit p_i (chaque fournisseur propose un seul produit) ;
- la valeur v_i associée au transport de chaque produit : c'est-à-dire l'argent gagné par l'entreprise si elle réalise le transport de ce produit.

En considérant que l'entreprise dispose de n fournisseurs, l'entreprise cherche donc à trouver une liste d'indices notée I contenue dans $\{0, \dots, n-1\}$ telle que :

$$\sum_{i \in I} p_i \leqslant P_{max} \qquad \text{(respect du poids maximal)} \tag{1}$$

et

$$\sum_{i \in I} v_i \text{ soit maximal } \text{(optimisation du profit pour l'entreprise).} \tag{2}$$

Dans toute la suite, les poids seront donnés en centaines de kilogrammes et les valeurs en centaines d'euros.

II.1 - Un exemple

Dans cette sous-partie, on suppose que $n = 4$ et que $P_{max} = 8$ centaines de kilogrammes.
On stocke alors les différentes informations dans trois listes :

- `Produits` est la liste des produits proposés par les fournisseurs, numérotés de 0 à 3.
  Ici `Produits = [0,1,2,3]` ;
- `Poids` est la liste des poids associés : `Poids = [3,2,1,4]` ;
- `Valeurs` est la liste des valeurs associées : `Valeurs = [4,3,1,9]`.

Par exemple, le produit 1 a un poids de deux centaines de kilogrammes et une valeur de trois centaines d'euros.
Les questions **Q5.**, **Q6.** et **Q7.** se traitent à l'aide de calculs simples, à faire à la main.

Q5. Expliquer pourquoi une cargaison constituée d'un, de deux ou de quatre produits ne répond pas au problème posé, c'est-à-dire ne maximise pas le profit fait par l'entreprise en respectant la condition donnée sur le poids maximal.

Q6. Donner toutes les cargaisons de trois produits respectant le poids maximal. On donnera à chaque fois le profit fait par l'entreprise.

Q7. Quelle est la cargaison maximisant le profit de l'entreprise ? Que vaut le profit dans ce cas ?

II.2 - Une approche exhaustive

Soit $n \in \mathbb{N}^*$. On garde les notations de la sous-partie précédente dans le cas général :

- `Produits = [0,1,2,...,n-1]` est la liste des produits ;
- `Poids` = $[p_0, \dots, p_{n-1}]$ est la liste des poids associés aux produits ;
- `Valeurs` = $[v_0, \dots, v_{n-1}]$ est la liste des valeurs associées aux produits.

Q8. Définir une fonction `listeProduits` ayant pour argument un entier naturel non nul n créant et renvoyant la liste `Produits`.

Une combinaison de i éléments de $\{0, \dots, n-1\}$ est un sous-ensemble de $\{0, \dots, n-1\}$ contenant i éléments.
En Python, une combinaison sera représentée par une liste ordonnée dans l'ordre croissant : par exemple la combinaison $\{1, 4, 5\}$ sera représentée par la liste `[1,4,5]`.
Une approche exhaustive pour optimiser le chargement consiste à tester toutes les combinaisons de i produits parmi n et à garder celle qui optimise le rendement : les deux contraintes (1) et (2) doivent donc être respectées.

Q9. Écrire une fonction Python `teste_cargaison` qui reçoit en arguments les listes `Poids`, `Valeurs`, `Pmax`, une combinaison `C` de la liste `Produits` et renvoie :

— Une valeur numérique correspondant à la valeur de la cargaison si la combinaison `C` de produits est compatible avec la contrainte (1).

— La valeur -1 si la combinaison proposée est incompatible avec la contrainte (1).

On admet que l'on dispose d'une fonction Python `combinaisons`, d'argument une liste `L`, qui renvoie la liste de toutes les combinaisons d'éléments de la liste `L`.
Par exemple :

```
>>> L = [0,1,2]
>>> LC = combinaisons(L)
>>> LC
[[],[0],[1],[2], [0, 1],[0,2], [1,2],[0,1,2]]
```

Q10. Le DR fournit le code partiel de la fonction `optimise` d'arguments `Produits, Poids, Valeurs` et `Pmax`, qui renvoie la combinaison assurant le meilleur remplissage ainsi que la valeur de la cargaison associée.

Compléter le code de cette fonction `optimise` en fonction des indications précédentes.

Q11. On admet qu'il y a 2^n combinaisons d'éléments de la liste `Produits` (y compris la combinaison vide de 0 élément parmi n).

Si $n = 50$ et que tester la validité d'une combinaison prend 10^{-9} secondes, combien de temps faudra t-il approximativement pour optimiser le remplissage du camion ?

On prendra $2^{50} \simeq 10^{15}$ *et* 1 *heure* $\simeq 4 \times 10^3$ *secondes.*

La démarche précédente est donc inefficace pour un nombre relativement élevé d'objets.
Aussi il est préférable de mettre en place une méthode plus rapide dont on étudiera ensuite l'optimalité.

II.3 - Une méthode intuitive pour la résolution du problème

On garde les notations de la sous-partie précédente.
Une méthode intuitive pour tenter d'optimiser le profit de l'entreprise est la suivante : on calcule les ratios $\frac{v_i}{p_i}$, puis on trie les objets par ordre décroissant suivant ces valeurs. Les produits sont alors classés par rentabilité : le premier produit devient le plus rentable "au poids" et ainsi de suite. On ajoute progressivement chaque produit dans la cargaison, dans cet ordre, sans dépasser la limite du poids maximal.

Q12. Écrire une fonction `ratio` ayant pour arguments deux listes `Poids` et `Valeurs` où `Poids` correspond à la liste des poids et `Valeurs` correspond à la liste des valeurs, renvoyant la liste des ratios $\frac{v_i}{p_i}$.

La fonction suivante est associée à une méthode de tri par insertion :

```
0 def tri(L):
1     """L est une liste d'entiers ou de flottants"""
2     for i in range(1,len(L)):
3         x=L[i]
4         j=i
5         while j>0 and x<L[j-1]:
6             L[j]=L[j-1]
7             j=j-1
8         L[j]=x
9     return L
```

Q13. On exécute `tri(L)` avec `L=[3,5,2,1]`. Combien y a t-il d'itérations de la boucle `for` ? Donner la valeur de `L` à la fin de chaque itération de la boucle `for`.

Q14. Le code précédent fonctionnerait-il si `L` était un tuple d'entiers (par exemple `L=(1,5,2,3)`) ? Justifier.

Q15. Évaluer la complexité en nombre de comparaisons dans le pire des cas de la fonction `tri`.
On justifiera la complexité trouvée.

Q16. Écrire une fonction `renverser` ayant pour argument une liste de nombres `L` et renvoyant une nouvelle liste obtenue en lisant `L` dans le sens inverse.
Par exemple, `renverser([1, 5, 3, 4])` renverra `[4, 3, 5, 1]`.
L'utilisation de toute instruction Python comme `L[::-1]` *ou* `L.reverse()` *est proscrite ici.*

Q17. On souhaite trier une liste de poids `Poids` et une liste de valeurs `Valeurs` associées à une liste de produits en suivant l'ordre décroissant de la liste des ratios $\frac{v_i}{p_i}$. Justifier que les fonctions `ratio`, `tri` et `renverser` ne permettent pas de répondre simplement au problème posé.

Q18. Écrire, à l'aide des fonctions `ratio` et `renverser`, une fonction `tri2` ayant pour arguments une liste de poids `Poids` et une liste de valeurs `Valeurs` associées à une liste de produits. Cette fonction renverra les listes de poids et de valeurs triées par ordre décroissant de la liste des ratios.

Q19. Le DR fournit le code incomplet de la fonction `vmax` ayant pour arguments les listes de poids `Poids`, de valeurs `Valeurs`, le poids maximal `Pmax` du chargement et renvoyant la valeur maximale du profit de l'entreprise en suivant la méthode proposée. Compléter cette fonction.

Q20. On souhaite appliquer cette méthode avec les listes de poids et de valeurs de la sous-partie **II.1.** Donner la liste des ratios, les listes des poids et des valeurs obtenues à l'aide de la fonction `tri2` ainsi que le profit obtenu. Commenter le résultat.

II.4 - Une méthode récursive

Nous gardons les notations du cas général de la sous-partie **II.2** : `Produits, Poids` et `Valeurs`. On considère pour simplifier que les poids des produits sont des entiers, ainsi que `Pmax`.
Nous introduisons une méthode récursive pour résoudre le problème d'optimisation :

- pour chacun des produits, deux choix sont possibles : il fait partie de la cargaison ou non ;
- la récursivité s'effectuera sur le nombre total de produits : le premier appel de la fonction se fera avec la valeur n, puis la valeur $n-1$ et ainsi de suite jusqu'à la valeur 0 (correspondant au cas où il n'y a plus de produits) ;
- pour $i \in \{0, 1, \ldots, n\}$ et $\omega \in \{0, 1, \ldots, Pmax\}$, on note $S(i, \omega)$ la valeur maximale cumulée des produits que l'on peut placer dans un camion d'une capacité maximale (en poids) de ω avec la liste constituée des i premiers produits de `Produits`.

On pose alors la relation de récursivité suivante :

$$S(i,\omega) = \begin{cases} 0 & \text{si } i = 0 \\ S(i-1,\omega) & \text{si } i > 0 \text{ et } p_{i-1} > \omega \\ \max(S(i-1,\omega),\ v_{i-1} + S(i-1,\omega - p_{i-1})) & \text{si } i > 0 \text{ et } p_{i-1} \leqslant \omega. \end{cases} \quad (3)$$

(***Rappel*** : le i^e élément de la liste `Poids` a pour indice $i-1$. Il est noté p_{i-1} .)

Q21. Justifier les relations précédentes dans les trois cas.

Q22. Justifier la terminaison de l'algorithme associé à la relation de récursivité précédente, sachant que la première valeur donnée pour i sera n et la première valeur pour ω sera P_{max}.

Q23. Écrire une fonction `maxi` ayant pour arguments deux nombres et renvoyant le maximum parmi ces deux valeurs.
Il est interdit ici d'utiliser une quelconque fonction `max` *prédéfinie dans Python.*

Q24. Le DR fournit le code incomplet de la fonction récursive `recur` ayant pour arguments les listes de poids et de valeurs `P` et `V`, un indice `i`, un poids ω, et renvoyant $S(i,\omega)$.
En vous basant sur la relation (3), compléter ce code.

Q25. Donner une série d'instructions utilisant la fonction `recur` et permettant de déterminer le profit de la sous-partie **II.1.**

Pour optimiser la méthode récursive décrite ci-dessus, il est nécessaire d'éviter des appels récursifs inutiles. Pour cela, on va garder en mémoire les calculs déjà effectués dans un dictionnaire.
Voici le principe : nous allons stocker les valeurs $S(i,\omega)$ dans un dictionnaire `Memoire`.
Les clés de ce dictionnaire seront les couples (i,ω) : si la valeur de $S(i,\omega)$ a déjà été calculée, alors `Memoire[(i,w)]` contiendra cette valeur $S(i,\omega)$ que l'on renverra.
Sinon, on calculera cette valeur $S(i,\omega)$ en suivant la méthode récursive et on la stockera dans le dictionnaire avant de la renvoyer.

Q26. Donner l'instruction permettant de créer le dictionnaire vide `Memoire`. Pourrait-on utiliser comme clé du dictionnaire des listes `[i,w]` au lieu de tuples `(i,w)` ? Justifier.

Q27. Le DR fournit le code incomplet de la fonction `recur2` suivant le principe expliqué et permettant d'améliorer la fonction `recur`.
Compléter le code proposé.

Par exemple avec les données suivantes :

- `Poids = [5,3,3,3]`
- `Valeurs = [4,3,1,1]`
- `Pmax = 8`
- `Memoire`, un dictionnaire initialement vide,

et en exécutant `recur2(Poids,Valeurs,len(Poids),Pmax,Memoire)`, on obtient alors la valeur 7.

FIN

Numéro d'inscription

Numéro de table

Né(e) le

Nom :

Prénom :

Emplacement QR Code

Filières: TPC ET TSI

Session : 2023

Épreuve de : INFORMATIQUE

Consignes

- *Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer*
- *Rédiger avec un stylo non effaçable bleu ou noir*
- *Ne rien écrire dans les marges (gauche et droite)*
- *Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre*

DOCUMENT RÉPONSE

Ce Document Réponse doit être rendu dans son intégralité avec la copie.

Q10.

```
0 def optimise(Produits,Poids,Valeurs,Pmax):
1     Csol=[] # combinaison solution
2     Vsol=0  # valeur associee de la cargaison
3     LC=....
4     for C in LC:
5         val=teste_cargaison(Poids,Valeurs,Pmax,C)
6         if val .........
7             ............
8             ............
9     return Csol,Vsol
```

Q19.

```
0 def vmax(Poids,Valeurs,Pmax):
1     P2,V2=Tri2(Poids,Valeurs)
2     SP=0 # somme des poids
3     SV=0 # somme des valeurs
4     i=0
5     while  ............
6         SP=SP+P2[i]
7         SV=SV+V2[i]
8         i=i+1
9     return ............
```

NE RIEN ÉCRIRE DANS CE CADRE

Q24.

```
0 def recur(P,V,i,w):
1     """ P: liste des poids, V liste des valeurs"""
2     if i==0:
3         return  .......
4     if P[i-1]>w:
5         return recur(P,V,i-1,w)
6     else :
7         ..............
```

Q27.

```
0 def recur2(P,V,i,w,Memoire):
1     """ P: liste des poids, V liste des valeurs"""
2     if i==0:
3         return 0
4     if (i,w) in Memoire:
5         return  ..............
6     if P[i-1]>w:
7         Memoire[(i,w)]=recur2(P,V,i-1,w,Memoire)
8         return Memoire[(i,w)]
9     else:
10        if (i-1,w) not in Memoire:
11            Memoire[(i-1,w)]=.................
12        if ..................................................:
13            Memoire[(i-1,w-P[i-1])]=recur2(P,V,i-1,w-P[i-1],Memoire)
14        a=maxi(Memoire[(i-1,w)],V[i-1]+Memoire[(i-1,w-P[i-1])])
15        Memoire[(i,w)]=a
16        return  ..........
```