teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun Mines-Ponts

Année : 2025

Filière : MP - PC - PSI

Épreuve : Sciences Industrielles pour l'Ingénieur

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles).

L'équipe UPSTI

Autour du sac à dos

Corrigé UPSTI

Question 1 Écrire une requête permettant de retourner la liste des identifiants de conteneurs et leur ratio tarification/longueur trié par ordre décroissant, partant de Marseille et devant aller à Barcelone, avec une mise à disposition avant le 01/01/2025.

Une requête sur une seule table est suffisante ici

```
SELECT idC, val/taille as ratio
FROM Conteneurs
WHERE portDepC="Marseille" and portDestC="Barcelone" and dateDisp<2025-01-01
ORDER by ratio DESC
```

Question 2 Écrire une requête permettant de retourner les identifiants de navire et leur nombre respectif de conteneurs attribués.

```
SELECT idN, COUNT(IdC)
FROM conteneurs
GROUP BY idN
```

Question 3 Écrire une fonction `profit(obj:[(int)],S:[int])->int` prenant en argument une liste `obj` et une liste `S`, et retournant la valeur du profit `P`.

```
def profit(obj,S) :
    somme = 0
    for i in range(len(S)) :
    # Il faut parcourir la liste S pour savoir si l'objet est pris
        somme += S[i]*obj[i][1] # Recuperation du profit de l'objet
    return somme
```

Question 4 Écrire une fonction `contrainte(obj:[(int)],S:[int],b:int)->bool` prenant en argument une liste `obj`, une liste `S` et un nombre `b`, et retournant le booléen `True` si la contrainte de ressources du problème du sac à dos est respectée, `False` dans le cas contraire.

```
def contrainte (obj,S,b) : # Meme structure que Q3
    somme = 0
    for i in range(len(S)) :
        somme +=S[i]*obj[i][0]
    return somme<=b
```

Question 5 Sur l'exemple de la figure 1 où n = 3, à quoi est égale la liste S pour les feuilles b et c ?

Pour la feuille b : gauche, gauche, droite soit la liste `[1,1,0]` tandis que pour la feuille c : gauche,droite, gauche soit `[1,0,1]`

Question 6 Donner le nombre de feuilles de l'arbre binaire en fonction du nombre d'objets n. Indiquer en la

justifiant la complexité temporelle en fonction du nombre d'objets n d'un algorithme de résolution du problème du sac à dos de type "force brute" qui envisagerait toutes les combinaisons possibles de sélection d'objets.

Chaque objet pouvant être pris ou non il y a 2 combinaisons. Soit pour n objets 2^n combinaisons.

Il faut évaluer pour chaque combinaison les fonctions `profit` et `contrainte`, qui sont de complexité linéaire, ce qui donne une complexité de la forme $O(n.2^n)$. La complexité est exponentielle, on ne peux donc pas traiter le problème par force brute.

Question 7 On prend le cas particulier où obj=[(2,3),(1,4),(4,4)] et b=5. Expliquer quelle liste S est construite par la stratégie gloutonne précédente.

1. Les ratios sont les suivants 1.5, 4,1 ;
2. Il faut en premier choisir l'objet 1, il reste 4 en taille ;
3. Puis l'objet 0, il reste 2 en taille ;
4. l'objet 2 ne peut-être choisi.

Dans ce cas la liste S construite est `[1,1,0]`

Question 8 Proposer le code Python complet des lignes 5, 12 et 15

Ligne 5 : Il s'agit de l'étape 1 pour construire la liste `Lq1` : `Lqi.append(obj[i][1]/obj[i][0])`

Ligne 12 :Il s'agit de l'étape 2 et de la modification simultanées des deux listes `Li[j] = Li[j-1]`

Ligne 15 : `Li[j] = i`

Question 9 Identifier le meilleur des cas et le pire des cas de la méthode de tri utilisée dans la fonction `construitLi` en précisant et justifiant leur complexité temporelle respective.

Il s'agit ici d'un tri par insertion.

Le meilleur des cas est celui où on ne rentre jamais dans la boucle while, c'est celui où la liste est déjà triée dans l'ordre décroissant. La complexité est dans ce cas o(n) car la boucle for des lignes 4 à 6 sont de complexité o(n) ainsi que le boucle for de la ligne 7 car la complexité de la boucle while est o(1).

Le pire des cas est alors la liste triée dans l'ordre croissant. Dans ce cas la bouche while est de complexité o(n) ce qui donne une complexité en o(n**2)

Question 10 Proposer le code Python complet des lignes 4, 5, 6 et 7.

```
S = len(obj)*[0]
Li = construitLi(obj)
j = 0
while j<len(S) and b > 0 : # Il faut soit parcourir tous les objets
#soit le sac est plein
    if obj[Li[j]][0] <=b : # Il y a assez de place pour l'objet
        S[Li[j]] = 1 # On prend l'objet
        b = b - obj[Li[j]][0] # On diminue l'espace disponible
    j+=1
```

Question 11 On reprend le cas particulier où obj=[(2,3),(1,4),(4,4)] et b=5. Donner la solution optimale pour ce cas particulier. Conclure sur la pertinence d'une approche gloutonne.

En choisissant les objets 0 et 1, le profit est de 7.

Le profit aurait pu être de 8 en choisissant l'objet 1 et 2. L'approche gloutone donne un optimum mais qui

n'est pas forcement global.

Question 12 On reprend le cas particulier où obj=[(2,3),(1,4),(4,4)] et b=5. Donner sans justifications les valeurs du tableau `T` à l'issue de l'exécution des lignes 2 à 8. Puis, à la fin de chacune des trois itérations de la boucle `for` en ligne 9, donner la valeur de `T[i+1]`. Préciser ce que retourner la fonction `KPprogDynamique(obj,b)` et à quoi correspond cette valeur.

A la fin de la ligne 8 on a un tableau `T` qui ne contient que des 0 de dimension 6 (colonnes) par 4 lignes

- quand i = 0 : `T[1] = [0, 0, 3, 3, 3, 3]`
- quand i = 1 : `T[2] = [0, 4, 4, 7, 7, 7]`
- quand i = 2 : `T[3] = [0, 4, 4, 7, 7, 8]`

La valeur renvoyé est un entier qui représente le profit maximal.

Question 13 Déterminer en la justifiant la complexité asymptotique temporelle de la fonction `KPprogDynamique(obj`

Il faut évaluer chaque case du tableau (une fois à 0 et une fois avec sa valeur définitive). La complexité est alors de la forme O((n+1).(b+1))

Question 14 Donner les valeurs des variables `S`, `k` et `r` après chaque itération de la boucle `while` en ligne 8. Indiquer en justifiant si la complexité asymptotique temporelle de la fonction `KPprogDynamique(obj,b)` est ainsi modifiée.

A la fin de la ligne 8 on a un tableau `T` qui ne contient que des 0 de dimension 6 (colonne) par 4 lignes

- Initialement `S = [0,0,0]`, `k` = 2, `r` = 5
- Itération 1 : ligne 9 non vérifiée donc `S = [0,0,1]`, `k` = 1, `r = 1`
- Itération 2 : ligne 9 non vérifiée donc `S = [0,1,1]`, `k` = 0, `r` = 0
- Fin

Non car de complexité linéaire, dans le pire des cas k varie de n-1 à 0. O(n)

Question 15 Écrire une fonction `estFeuille(a:dict)->bool` qui prend en argument un arbre `a` et qui retourne le booléen `True` si `a` est une feuille, `False` dans la cas contraire.

Il suffit ici de regarder si un des deux fils est vide.

```
def estFeuille(a) :
    return a['g']=={}
```

Question 16 Écrire une fonction `possible(obj:[(int)],Sk:[int],b:int)->bool`

Il suffit ici de regarder si un des deux fils est vide.

```
def possible(obj,Sk,b) :
    S0 = Sk + [0]*(len(obj)-len(Sk)) # Nombre de 0 manquants
    S1 = Sk + [1]*(len(obj)-len(Sk)) # Nombre de 1 manquants
    return contrainte(obj,S0,b) and profit(obj,S1)>Pmin
```

Question 17 Le nom de la fonction `KPforceBrute(arbre:dict,obj:[(int)],b:int)` est modifié en `KPpse(arbre:`
Cette fonction prend en argument l'arbre des solutions arbre, la liste des objets `obj` et un nombre ressources `b`. Elle affecte aux variables globales `Pmin` et `Sol`respectivement la valeur du profit maximal et le vecteur solution correspondant, conformément à l'algorithme PSE. Réécrire le code Python à partir de la ligne 9 (autant de

lignes supplémentaires que nécessaire). Ne pas réécrire les autres lignes.

Il ne faut explorer la branche que si elle est possible.

```
        if possible(obj,arbre['g']['S'],b) :
            KPpse(arbre['g'],obj,b)
        if possible(obj,arbre['d']['S'],b) :
            KPpse(arbre['d'],obj,b)
```

Question 18 Proposer le code Python complet des lignes 4 et plus de la partie 2)c)i. de la fonction ci-dessus.

Il faut modifier les valeurs de T en leur appliquant le coefficient ρ

```
    for k in range(len(T)) :
        T[k] = rho*T[k]
```

Question 19 Proposer le code Python complet des lignes 7 et plus de la partie 2)c)ii.

Il s'agit ici d'une fonction de recherche d'un maximum. Il faut cependant faire attention aux noms de variables qui sont utilisés par la suite.

```
    Pmax,kmax = profit(obj,S[0]),0
    for k in range(len(S)) :
        if profit(obj,S[k]) > Pmax :
            Pmax,kmax = profit(obj,S[k]),k
```

Question 20 Proposer le code Python complet des lignes 14 et plus de la partie 2)c)iv.v.vi.

Il ne faut explorer la branche que si elle est possible.

```
    qte = 1 / (1 + PbestOfAll - Pmax) # Calcul de la quantite
    for elt in range(len(S[kmax])) :
        if S[kmax][elt]==1 : # Pour chaque objet de Smax choisi
            T[elt]+=qte # Ajout de la quantite
        if T[elt] < Tmin : # Respect des regles sur les taux
            T[elt] = Tmin
        if T[elt] > Tmax :
            T[elt] = Tmax
```

Question 21 On donne le nom de variable o0 (o zéro) pour l'indice de l'objet choisi. Proposer le code Python complet des lignes 15, 17 et 18, définies ci-dessus.

```
        o0 = randint(0,n-1) # Choix d'un objet aleatoire
        S[k][o0] = 1 # Choix de l'objet o0
        b2 -= obj[o0][1] # Retrait dans la place disponible
```

Question 22 Proposer le code Python complet des lignes 3, 5, 6, 9 et 10.

```
def construitProb(obj,candidats,b,T):
    m=len(candidats)
    prob= {} # Dictionnaire vide pour la structure de donnee
    for i in range(m):
```

```
        oi=candidats[i]
        prob[oi] = (T[oi]**alpha) * (b*obj[oi][1]/obj[oi][0])** beta
    s=sum(prob.values())
    for i in range(m):
        oi=candidats[i]
        prob[oi] = prob[oi]/s
    return prob
```

Question 23 Proposer le code Python complet des lignes 24 et plus, définies en page précédente.

```
while b2>0 and candidats!=[ ]:
    prob = construitProb(obj,candidats,b2,T) # Construire les probas
    o1 = choixCandidat(candidats,prob) # Choix du candidats
    S[k][o1] = 1 # Indique que l'objet o1 a ete choisi
    b2-= obj[o1][0] # Retrait dans la place disponible
    candidats.remove(o1) # Enlever o1 de la liste des candidats
    miseAjourCandidats(obj,candidats,b2) # Mettre a jour les candidats en fonc
```

Question 24 Commenter les résultats quant aux critères de performance (temps d'exécution et qualité de la solution proposée). Commenter les résultats obtenus par l'algorithme ACO en précisant sur quels paramètres de la méthode on peut jouer, et quelles sont les conséquences sur les performances.

Le profit maximal est 567 donné par la méthode force brute qui est exhaustive.

On peut voir que l'algorithme glouton est le plus efficace temporellement, mais le profit n'est pas maximal (-4%). C'est alors l'algorithme avec programmation dynamique qui semble le plus performant.

Pour l'ACO on peut jouer sur le nombre de fourmi (pas de test disponible faisant varier ce paramètre) ou alors le nombre d'itération. On remarque que le temps de calcul semble varier linéairement avec le nombre l'itération car le temps pour nbits = 50 est quasiment 10 fois plus grand que le temps pour nbits=5. De plus il faut un nombre d'itération minimale pour l'algorithme ACO pour que le principe de fonctionnement soit intéressant.