USTHB, Faculté d'Informatique,
Département IA & SD
M1, Master SII
Module : Design, Analyse et Complexité des Algorithmes

Bab-Ezzouar le 9 janvier 2023

Corrigé-type de l'Épreuve de Moyenne Durée

Problème du sac à dos

Considérer le problème du sac à dos énoncé comme suit :

« Un voyageur, avant son périple prépare un sac à transporter avec des objets possédants chacun un poids et une utilité. Le poids du sac est P et l'utilité minimum à atteindre est U. Quels objets faut-il mettre dans le sac de manière à respecter le poids P du sac et l'utilité U ? »

Le problème est décrit formellement comme suit :

Instance : *Un ensemble S contenant n objets* $S = \{o_1, o_2, \ldots, o_n\}$. *Chaque objet* $o_i \in S$ *a un poids* $p(o_i)$ *et une utilité* $u(o_i)$, *de type entier positif, P un entier positif et U un entier positif.*

Question : *Existe-il un sous ensemble* $S' \subseteq S$ *tel que* $\sum p(o_i) \leq P$ *et* $\sum u(o_i) \geq U$ *?*

Un exemple du problème du sac à dos est le suivant :
S = {laptop, chaussure, livre, montre} avec les poids et les valeurs indiqués dans le tableau suivant :

Objet	**Poids**	**Utilité**
Laptop	2 100 g	90 000 DA
Chaussure	350 g	4 000 DA
Livre	400 g	1 200 DA
Montre	50 g	210 000 DA

Le poids maximum P du sac est égal à 2 600 g et l'utilité minimale U est égale à 250 000 DA.

1) Définir des structures de données pour représenter tous les éléments du problème. (2 pts)

TVP : Tableau des valeurs des poids de S.

1	2	3	…	m	m+1		max
a_1	a_2	a_3	…	a_m	-1		m

TVU : Tableau des valeurs des utilités de S.

1	2	3	…	m	m+1		max
a_1	a_2	a_3	…	a_m	-1		m

VRPS : Vecteur Représentatif des poids des objets de S: **vecteur booléen**, VRP[i]=1 si TVP[i] = poids de l'objet o_i, 0 sinon

1	2	3	…	n	n+1		max
0/1	0/1	0/1	0/1	0/1	-1	…	n

VRUS : Vecteur représentatif des utilités des objets de S: **vecteur booléen**, VRU[i]=1 si TVU[i] = valeur de l'objet o_i, 0 sinon

1	2	3	…	n	n+1		max
0/1	0/1	0/1	0/1	0/1	-1	…	n

Les appliquer sur l'exemple ci-dessus. (1 pt)

TVP

1	2	3	4	5	6	7	8		…	max
2100 g	350 g	400 g	50 g	600g	80g	-1			…	6

TVU

1	2	3	4	5	6	7	8	9	…	max
90000 DA	4000 DA	1200 DA	4000 DA	210000 DA	500DA	20000DA	455DA	-1	…	8

VRPS : Vecteur Représentatif des poids des objets de S: **vecteur booléen**, VRPS[i]=1 si TVP[i] = poids de l'objet o_i, 0 sinon

1	2	3	4	5	6	7	8	9	…	max
1	1	1	1	0	0	0	0	0	…	4

VRUS : Vecteur représentatif des utilités des objets de S: **vecteur booléen**, VRUS[i]=1 si TVU[i] = valeur de l'objet o_i, 0 sinon

1	2	3	4	5	6	7	8	9	…	max
1	1	1	1	0	0	0	0	0	…	4

2) Écrire un algorithme itératif de résolution du problème basé sur la recherche en profondeur d'abord. (Indication : A chaque niveau de la recherche prévoir le cas où un objet est mis dans le sac et le cas où il n'est pas mis dans le sac.) (2 pts)

```
Algorithme SAD
entrée : TVP, TVU, VRPS, VRUS (* ensembles des poids et des utilités de S *), P, U,
         S : tabeau[1..n] d'objets (* S[1] = 'Laptop', … *)
sortie : échec / succès : VRPsac et VRUsac (* la solution sac *)

type élément : enregistrement sac : VRPsac, VRUsac; i :1..n+1 fin ;  (* sac et niveau de la recherche *)
var pile: tableau [1..k] d'éléments ;
    niveau, i :1..n+1 ;
    succès, échec : booléen ;
début
   succès := faux ;
   pour i :=1 à n faire début VRPsac[i] :=0 ; VRUsac[i] :=0 fin;        (* sac := {} ; *)
   niveau :=0 ;
   empiler(sac, niveau) ;
   tant que (non pile vide) et (non succès) faire
    début dépiler (sac, niveau) ;
          (p, u) := calc-poids-utilité (sac) ;
          si (p < P) et (u >U) alors succès := vrai
            sinon si (p <P) et (u < U) et (niveau < n) alors
                  début  niveau := niveau+1 ;
                         empiler (sac, niveau)
                         sac := sac ∪ S[niveau] ; (*inclure dans sac un autre objet de S *)
                         empiler (sac, niveau) ;
                  fin ;
    fin ;
    si pile vide alors échec := vrai ;
```

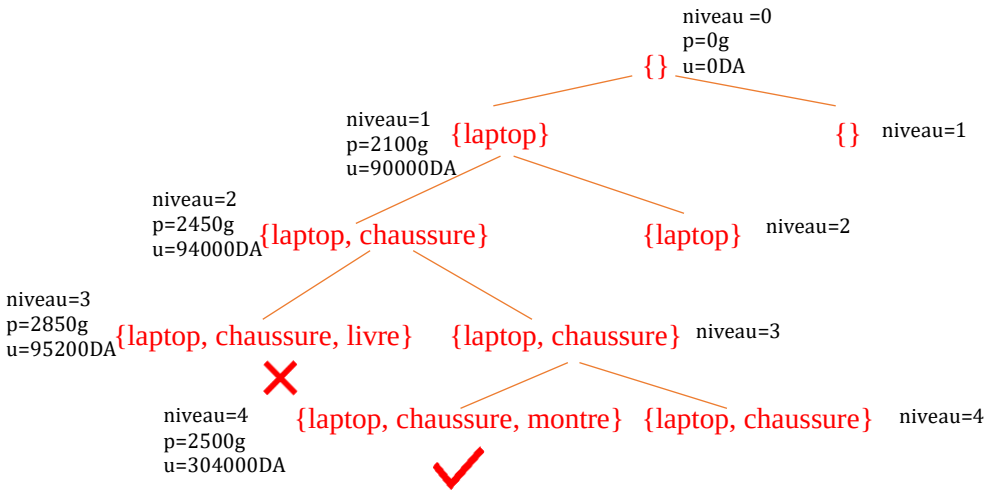
```
fin ;

Procédure calc-poids-utilité
entrée : TVP, TVU, VRPsac, VRUsac  (* sac *)
sortie : poids et utilité de sac
var p, u : entier ;
début
      p := 0 ; u := 0 ;
      pour chaque objet i dans sac faire
            p :=  p + TVP[VRPsac[i]] ; (* p+ poids(objet i) *)
            u :=  u + TVU[VRUsac[i]] ; (* u+ utilité(objet i) *)
      return (p, u) ;
fin ;
```

a) L'appliquer sur l'exemple donné (1 pt)

Une arborescence est construite où chaque niveau représente l'ajout ou non d'un objet. A chaque configuration du sac, le poids du sac (p) et l'utilité (u) sont calculés. 3 cas de figures peuvent se présenter :

- succès : le poids est inférieur au poids autorisé P (2600g) et l'utilité est supérieure à la valeur requise U (250000DA), la configuration est une solution. L'algorithme s'arrête.
- échec : toutes les configurations du sac sont explorées (la pile est vide) et les deux contraintes ne sont pas satisfaites.
- poursuite de l'exploration : le niveau de l'arborescence est inférieur ou égal à la cardinalité de S
    - Le poids est inférieur au poids autorisé et la valeur est inférieure à la valeur requise
    - Le poids dépasse le poids autorisé (2600g), la configuration est abandonnée quel que soit l'utilité et la configuration suivante est explorée.



ci-dessus se trouve l'exécution de la DFS sur l'exemple donné. La première solution trouvée est {laptop, chaussure, montre}, l'algorithme s'arrête. Toutes les branches qui se trouvent après constituent le contenu de la pile.

b) Calculer sa complexité. (1 pt)

Au pire des cas, l'arborescence est construite entièrement (cas d'échec).
A la racine : 1 nœud
Au premier niveau : 2 nœuds

Au second niveau : $2*2 = 2^2$ nœuds
…
Au dernier niveau : 2^n nœuds, n étant le nombre d'objets de S. Le nombre total de nœuds explorés est donc égal à : $1+2+ \dots + 2^n = 2^{n+1}-1$.
La complexité est donc $O(2^n)$.

3) Écrire un algorithme récursif de résolution du problème. (2 pts)

```
Algorithme SAD(n)
entrée : TVP, TVU, VRPS, VRUS (* S *), VRPsac, VRUsac (* sac *), P, V
         S : tabeau[1..n] d'objets (* S[1] = 'Laptop', … *)
sortie : échec / succès : VRPsac, VRUsac (* sac *)
var échec, succès : booléen ;

début
   si (n>=1) alors            (* S ≠ {} *)
    début
         VRPS[n] := 0 ; VRUS[n] :=0 ; (* S[n] := '',  S[n] est vide*)
         VRPsac[n] := 1 ; VRUsac[n] :=1 ; (* sac := sac ∪ S[n]*)
         (p, u) := Calc-poids-utilité (sac) ;
         si (p < P) et (u > U) alors succès := vrai ; stop ;
         si (p < P) et (u < U et n > 1) alors début n := n-1 ; SAD(n) ; fin
         si (p > P) et (n < 1) alors
                  début   VRPsac[n] := 0 ; VRUsac[n] :=0 ; (* sac := sac - S[n]*)
                          n := n-1 ; SAD(n) ;
                  fin
      fin sinon échec := vrai ; stop ;
fin ;

Programme principal
entrée : S avec n objet VRPS, VRUS
sortie : 'oui' si solution, 'non' sinon
var succès, échec : booléen ;
j : 1..n ;
sac : VRPsac, VRUsac (* sac étant un ensemble *)
début
         (* sac := {} ; *)
         pour j := 1 à n faire début VRPsac :=0 ; VRUsac := 0 ; fin ;
         SAD(n) ;
         si échec alors écrire 'non' ;
         si succès alors écrire 'oui' ;
fin
```

a) L'appliquer sur l'exemple donné. (1 pt)

Initialement
S = {laptop, chaussure, livre, montre}
sac = {}
SAD(4) : S = {chaussure, livre, montre}
sac = {laptop}
p = 2100g u = 90 000DA
n = 3

SAD(3) : S = {livre, montre}
sac = {laptop, chaussure}
p = 2450g u = 94 000DA
n = 2
SAD(2) : S = {montre}
sac = {laptop, chaussure, livre}
p = 2850g u = 95 200DA
(p > P et n < 1) alors
S = {montre}
sac = {laptop, chaussure}
n = 1
SAD (1) : S = {}
sac = {laptop, chaussure, montre}
p = 2500g u = 304 000DA
p < P et v > V alors succès.

b) Calculer sa complexité. (1 pt)

La procédure SAD(n) appelle 2 fois SAD(n-1). Au pire des cas, tous les appels sont exécutés, ce qui donne l'équation suivante :

$$\begin{cases} SAD(n) = 2 * SAD(n-1) \\ SAD(1) = 2 \end{cases}$$

$$SAD(n) = 2 * SAD(n-1) = 2 * 2 * SAD(n-2) = \cdots = 2 * 2 * \ldots 2 * SAD(1 = n - (n-1)) = 2^n$$

La complexité est donc $O(2^n)$.

4) Donner une solution positive ainsi qu'une solution négative pour l'exemple ci-dessus. (1 pt)

solution positive : sac = {laptop, montre} car p = 2150 < P et u = 300 000DA > U
solution négative : sac = S car p = 2900g > P

5) Écrire un algorithme non déterministe pour le problème du sac à dos en expliquant clairement les structures de données utilisées. (2 pts)

```
Algorithme générer-solution
entrée : TVP, TVU, VRPS, VRUS
sortie : VRPsac, VRUsac
var i : 1..n ;
    VRPsac, VRUsac : Tableau[1..max-1] de booléens ;

début
   pour i := 1 à n faire
        début   VRPsac[i] := random(0/1) ;
                VRUsac[i] := VRPsac[i] ;
        fin ;
fin
```

La complexité est O(n) car l'algorithme a une boucle 'pour' avec n itérations.

```
Algorithme vérification
entrée : VRPsac, VRUsac, P, U
sortie : positive / négative

var i :1..n ; p, u : entier ;

début p := 0 ;
        u := 0 ;
        pour i := 1 à n faire
          début    si VRPsac = 1 alors p := p + TVP[i] ;
                   si VRUsac = 1 alors u := u + TVU[i] ;
          fin ;
        si (p < P) et (u > U) alors écrire 'positive' sinon écrire 'négative'
fin
```

Quelle est sa complexité ? (1 pt)
La complexité de l'algorithme de vérification est O(n) étant donné que l'algorithme possède une boucle 'pour' à n itérations.

Que peut-on en déduire ? (1 pt)
Le problème du sac à dos appartient à la classe NP.

6) Construire une transformation polynomiale du problème de satisfiabilité (SAT) vers le problème du sac à dos. (3 pts)

Pour une instance de SAT à n variables, on fait correspondre S à n objets.
Pour une solution sol à n bits, sac contiendra les objets pour lesquels les variables sont à 1.
Chaque objet aura un poids égal à 1 de sorte que P = n car au maximum la solution aura un poids égal à n.
Chaque objet aura une utilité égale à 1. Il devient alors judicieux d'affecter la valeur 1 à U car au minimum le sac contiendra un seul objet qui correspond à une variable booléenne.
Exemple : considérer les 3 clauses et 5 variables booléennes suivantes :
$\overline{x}_1 + x_3 + x_5$
$\overline{x}_2 + x_3 + \overline{x}_5$
$\overline{x}_4 + x_5$
Le problème du sac à dos aura comme S un ensemble de 5 objets S= $\{o_1, o_2, o_3, o_4, o_5\}$ et le sac ne contiendra l'objet o_i que si x_i est mis à 1 et réciproquement. P = 5, U = 1 et tous les objets ont un poids égal à 1 et une utilité égale à 1.
La solution 00101 de l'instance SAT lui correspondra le sac = $\{o_3, o_5\}$. Son poids est égal à 2 <5 et son utilité est égale à 2 > 1.
La construction de la transformation est O(n) car la cardinalité de S est égale à n.

Que peut-on en déduire. (1 pt)

Le problème du sac à dos est NP-complet.