

## AUTOUR DU SAC À DOS

On rappelle que le problème SUBSETSUM est défini comme suit :

**Instance :**  $x_0, \dots, x_{n-1} \in \mathbb{N}^*$  et  $S \in \mathbb{N}$

**Question :** existe-t-il  $I \subseteq [0 \dots n-1]$  tel que  $\sum_{i \in I} x_i = S$ ?

On appelle *objet* un couple  $(v, w)$  d'entiers naturels non nuls, où  $v$  est la *valeur* de l'objet et  $w$  son *poids*. On définit alors les deux problèmes de décision suivants :

■ 0/1-KNAPSACK

**Instance :**  $n$  objets  $(v_i, w_i)$  et deux entiers naturels  $V$  et  $W$

**Question :** existe-t-il  $I \subseteq [0 \dots n-1]$  tel que  $\sum_{i \in I} v_i \geq V$  et  $\sum_{i \in I} w_i \leq W$ ?

■ KNAPSACK

**Instance :**  $n$  objets  $(v_i, w_i)$  et deux entiers naturels  $V$  et  $W$

**Question :** existe-t-il  $x_0, \dots, x_{n-1} \in \mathbb{N}$  tels que  $\sum_{i=0}^{n-1} x_i v_i \geq V$  et  $\sum_{i=0}^{n-1} x_i w_i \leq W$ ?

### I Réductions

On admet que le problème SUBSETSUM est NP-complet.

► **Question 1** Montrer que les problèmes KNAPSACK et 0/1-KNAPSACK sont dans NP.

► **Question 2** En réduisant SUBSETSUM, montrer que 0/1-KNAPSACK est NP-complet.

On cherche désormais à montrer que KNAPSACK est NP-complet. Pour cela, on considère une instance  $F = (X, S)$ , avec  $X = x_0, \dots, x_{n-1}$  de SUBSETSUM et l'on définit :

■  $B = 1 + \max(x_0, \dots, x_{n-1}, S)$ ;

■  $y_i = (2^n + 2^i)nB$  pour  $0 \leq i < n$ ;

■  $y'_i = y_i + x_i$  pour  $0 \leq i < n$ ;

■  $S' = \left( n2^n + \sum_{i=0}^{n-1} 2^i \right) nB + S$ .

On considère alors l'instance  $G$  de KNAPSACK où :

■  $V = S'$

■  $W = S'$

■ les  $2n$  objets sont les couples  $(y_i, y_i)$  et  $(y'_i, y'_i)$  pour  $0 \leq i < n$ . Chaque objet a donc une valeur égale à son poids.

► **Question 3** On suppose dans cette question que  $F$  est une instance positive de SUBSETSUM. Montrer que  $G$  est une instance positive de KNAPSACK.

On suppose à présent que  $\sum_{i=0}^{n-1} \lambda_i y_i + \sum_{i=0}^{n-1} \lambda'_i y'_i = S'$ , avec les  $\lambda_i$  et  $\lambda'_i$  dans  $\mathbb{N}$ . On note  $N = \sum_{i=0}^{n-1} (\lambda_i + \lambda'_i)$ .

► **Question 4** Montrer que  $N \leq n$ .

► **Question 5** Montrer qu'on a les deux égalités suivantes :

■  $S = \sum_{i=0}^{n-1} \lambda'_i x_i$

■  $N2^n + \sum_{i=0}^{n-1} (\lambda_i + \lambda'_i) 2^i = n2^n + \sum_{i=0}^{n-1} 2^i$

On admet le résultat suivant : si  $\sum_{i=0}^{n-1} \alpha_i 2^i \equiv \sum_{i=0}^{n-1} 2^i \pmod{2^n}$  et si  $\sum_{i=0}^{n-1} \alpha_i \leq n$ , alors  $\alpha_0 = \dots = \alpha_{n-1} = 1$ .

► **Question 6** Montrer que KNAPSACK est NP-complet.

## II Algorithmes d'approximation

Dans cette partie et la suivante, on s'intéresse à la variante « optimisation » des problèmes KNAPSACK et 0/1-KNAPSACK :

- 0/1-KNAPSACK<sub>0</sub>

**Instance :**  $n$  objets  $(v_i, w_i)$  et un entier  $W$

**Solution :** une partie  $I \subseteq [0 \dots n - 1]$  telle que  $\sum_{i \in I} w_i \leq W$

**But :** maximiser  $\sum_{i \in I} v_i$

- KNAPSACK<sub>0</sub>

**Instance :**  $n$  objets  $(v_i, w_i)$  et un entier  $W$

**Solution :**  $n$  entiers naturels  $\lambda_0, \dots, \lambda_{n-1}$  tels que  $\sum_{i=0}^{n-1} \lambda_i w_i \leq W$

**But :** maximiser  $\sum_{i=0}^{n-1} \lambda_i v_i$

Dans toute la suite, on suppose que tous les  $w_i$  sont inférieurs ou égaux à  $W$  (sans perte de généralité, puisque les objets ne vérifiant pas cette condition sont inutiles et peuvent facilement être éliminés).

On considère l'algorithme glouton suivant :

- on trie les objets par valeur décroissante de  $v_i/w_i$  ;
- on considère les objets dans cet ordre, et l'on prend chaque objet autant de fois que possible (étant donné la capacité qui reste disponible).

Par exemple, pour l'instance  $o_0 = (10, 4)$ ,  $o_1 = (20, 6)$ ,  $o_2 = (3, 3)$  et  $W = 17$  :

- on considère  $o_1$  en premier et l'on fixe  $\lambda_1 = 2$  ;
- on considère ensuite  $o_0$  et l'on fixe  $\lambda_0 = 1$  ;
- on considère finalement  $o_2$  et l'on fixe  $\lambda_2 = 0$ .

---

### Algorithme 1 Algorithme glouton pour KNAPSACK<sub>0</sub>

---

Trier les objets par  $v_i/w_i$  décroissant.

$\lambda \leftarrow (0, \dots, 0)$  (taille  $n$ )

$R \leftarrow W$

**pour**  $i = 0$  à  $n - 1$  **faire**

    Poser  $\lambda[i]$  maximal tel que  $\lambda[i] \cdot w_i \leq R$

$R \leftarrow R - \lambda[i] \cdot w_i$

**renvoyer**  $\lambda$

---

► **Question 7** On note  $v^*$  la valeur totale d'une solution optimale pour une instance  $(v_0, \dots, v_{n-1}), (w_0, \dots, w_{n-1}), W$  (que l'on suppose triée par  $v_i/w_i$  décroissants). Montrer que  $v^* \leq \frac{v_0}{w_0} W$ .

► **Question 8** Montrer que l'algorithme 1 fournit une  $\frac{1}{2}$ -approximation pour KNAPSACK.

On considère une instance de KNAPSACK donnée par deux tableaux  $v$  et  $w$  de même longueur  $n$  et un entier  $capacity$ .

► **Question 9** Écrire une fonction `by_ratio` qui prend en entrée les tableaux  $v$  et  $w$  et renvoie un tableau indices de taille  $n$  tel que :

- les indices sont exactement les entiers de  $0$  à  $n - 1$  ;
- si  $i < j$ , alors  $\frac{v.(indices.(i))}{w.(indices.(i))} \geq \frac{v.(indices.(j))}{w.(indices.(j))}$ .

On pourra utiliser la fonction `Array.sort`.

```
val by_ratio : int array -> int array -> int array
```

► **Question 10** Écrire une fonction OCaml `greedy` telle que l'appel `greedy v w capacity` renvoie le tableau  $\lambda$  obtenu par l'algorithme ci-dessus. Les tableaux  $v$  et  $w$  seront supposés de même longueur et correspondent respectivement aux valeurs et poids des objets ; l'entier `capacity` correspond à  $W$ .

```
val greedy : int array -> int array -> int -> int array
```

► **Question 11** Déterminer la complexité de `greedy` en fonction du nombre  $n$  d'objets.

► **Question 12** Montrer qu'en appliquant le même principe pour  $0/1\text{-KNAPSACK}_0$ , on n'obtient pas une  $\alpha$ -approximation (quel que soit  $\alpha > 0$ ).

On considère la *relaxation*  $\text{FRACTIONAL-}0/1\text{-KNAPSACK}_0$  du problème  $0/1\text{-KNAPSACK}_0$  dans laquelle on s'autorise à prendre une quantité fractionnelle de chaque objet :

**Instance :**  $n$  objets  $(v_i, w_i)$  et un entier  $W$

**Solution :**  $n$  rationnels  $\lambda_0, \dots, \lambda_{n-1} \in [0, 1]$  tels que  $\sum_{i=0}^{n-1} \lambda_i w_i \leq W$

**But :** maximiser  $\sum_{i=0}^{n-1} \lambda_i v_i$

► **Question 13** On considère un ensemble  $(v_i, w_i)_{0 \leq i < n}$  d'objets, et l'on note  $V^*$  la valeur optimale associée pour  $0/1\text{-KNAPSACK}_0$  et  $V_f^*$  la valeur optimale associée pour  $\text{FRACTIONAL-}0/1\text{-KNAPSACK}_0$ . Montrer que  $V^* \leq V_f^*$ .

► **Question 14** Proposer un algorithme glouton très simple résolvant  $\text{FRACTIONAL-}0/1\text{-KNAPSACK}_0$  de manière optimale (ce que l'on justifiera très rapidement), en temps  $O(n \log n)$ .

► **Question 15** On considère l'algorithme suivant pour  $0/1\text{-KNAPSACK}_0$  :

---

**Algorithme 2** Algorithme d'approximation pour  $0/1\text{-KNAPSACK}_0$

---

Trier les objets par  $v_i/w_i$  décroissant.

$I \leftarrow \emptyset$

$R \leftarrow W$

$V \leftarrow 0$

$j \leftarrow -1$

**pour**  $i = 0$  à  $n - 1$  **faire**

**si**  $w_i \leq R$  **alors**

$I \leftarrow I \cup \{i\}$

$R \leftarrow R - w_i$

$V \leftarrow V + v_i$

**sinon si**  $j = -1$  **alors**

$j \leftarrow i$

**si**  $j \neq -1$  et  $v_j > V$  **alors**

**renvoyer**  $\{j\}$

**sinon**

**renvoyer**  $I$

---

Montrer que cet algorithme fournit une  $1/2$ -approximation pour  $0/1\text{-KNAPSACK}_0$ .

*Le sujet continue page suivante.*

### III Résolution exacte

On souhaite résoudre de manière exacte le problème 0/1-KNAPSACK<sub>o</sub> en utilisant une méthode de type *Branch and Bound*. L'instance est donnée par deux tableaux  $v$  et  $w$  de même longueur  $n$  et un entier  $capacity$ . On suppose les objets déjà triés par ordre décroissant de  $v/w$ .

```
(* La première composante correspond à v, la deuxième à w *)
type instance = int array * int array * int
```

Une solution partielle de taille  $k$  correspond au choix de prendre ou non chacun des  $k$  premiers objets  $o_0, \dots, o_{k-1}$ . Elle sera représentée par :

- l'entier  $k$ ;
- un tableau  $sol$  de  $n$  booléens, tel que pour  $0 \leq i < k$   $sol.(i)$  soit vrai si et seulement si on a choisi de prendre l'objet  $i$ . Les valeurs de  $sol.(i)$  pour  $i \geq k$  n'ont pas d'importance.

► **Question 16** Écrire une fonction `relax` telle que l'appel `relax (v, w, capacity) k` renvoie la valeur optimale pour l'instance  $(v_k, \dots, v_{n-1}), (w_k, \dots, w_{n-1}), capacity$  de la relaxation FRACTIONAL-0/1-KNAPSACK<sub>o</sub>.

```
val relax : instance -> int -> float
```

On demande une complexité linéaire en  $n = |v|$ .

► **Question 17** Écrire une fonction `solve` prenant en entrée une instance de 0/1-KNAPSACK<sub>o</sub> et renvoyant un couple `opt_sol, opt_v` telle que `opt_v` code une solution optimale et que `opt_v` soit la valeur associée. On procédera à un élagage par la technique dite *Branch and Bound*, dont on justifiera qu'il est correct (*i.e.* qu'on ne risque pas de « rater » la solution optimale).

```
val solve : instance -> (bool array * int)
```