DM #6 — Sac à dos binaire (avec solutions)

On rappelle le problème du *sac à dos binaire* (*subset sum*) : soit $\mathcal{S}$ un ensemble de n (supposé pair) entiers naturels et $t \in \mathbb{N}$, on cherche à décider s'il existe un sous-ensemble de $\mathcal{S}$ dont les éléments somment à t. Il serait par ailleurs facile d'adapter l'algorithme auquel on aboutira au problème de recherche sous-jacent, c'est à dire *trouver* un tel sous-ensemble s'il en existe un.
On propose d'étudier en détail la résolution de ce problème par une approche *rencontre au milieu*. Soient $\mathcal{S}_1$, $\mathcal{S}_2$ deux sous-ensembles de taille $n/2$ formant une partition de $\mathcal{S}$, on définit $\mathbb{T}_1 = \left\{\sum_{x\in\sigma} x, \sigma \in 2^{\mathcal{S}_1}\right\}$, $\mathbb{T}_2 = \left\{t - \sum_{x\in\sigma} x, \sigma \in 2^{\mathcal{S}_2}\right\}$.

1. Montrez qu'il existe $\sigma \in 2^{\mathcal{S}}$ tel que $\sum_{x\in\sigma} x = t$ ss.'il existe $\tau_1 \in \mathbb{T}_1$, $\tau_2 \in \mathbb{T}_2$ avec $\tau_1 = \tau_2$.

Soit un tel σ, on note $\sigma_1 = \sigma \cap \mathcal{S}_1$, $\sigma_2 = \sigma \cap \mathcal{S}_2$. Par le fait que $\mathcal{S}_1$ et $\mathcal{S}_2$ forment une partition de $\mathcal{S}$, l'on a $\left[\sum_{x\in\sigma_1} x\right] + \left[\sum_{x\in\sigma_2} x\right] = t$. On peut alors prendre $\tau_1 := \sum_{x\in\sigma_1} x \in \mathbb{T}_1$, $\tau_2 := t - \sum_{x\in\sigma_2} x \in \mathbb{T}_2$, et puisque $\tau_1 - \tau_2 = 0$ ils forment les éléments recherchés.
Réciproquement, soient de tels τ_1, τ_2 et σ_1, σ_2 les éléments de $\mathcal{S}_1$ & $\mathcal{S}_2$ à partir desquels ils sont construits, l'on a $\left[\sum_{x\in\sigma_1} x\right] + \left[\sum_{x\in\sigma_2} x\right] - t = 0$ et $\sigma_1 \cup \sigma_2 \in 2^{\mathcal{S}}$ donne l'élément recherché.

2. Donnez les tailles de $\mathbb{T}_1$ et $\mathbb{T}_2$.

On a $\#\mathcal{S}_1 = \#\mathcal{S}_2 = n/2$ donc $\#\mathbb{T}_1 = \#\mathbb{T}_2 = 2^{n/2}$, la taille de l'ensemble des parties de $\mathcal{S}_1$ ou $\mathcal{S}_2$.

On suppose que la taille de l'écriture en base 2 des éléments de $\mathbb{T}_1$ et $\mathbb{T}_2$ est du même ordre de grandeur que n (par exemple si $n = 256$, alors les éléments de $\mathbb{T}_1$ & $\mathbb{T}_2$ sont des entiers d'environ 256 bits). On suppose également que $\mathbb{T}_1$ et $\mathbb{T}_2$ sont représentés par des tableaux `t1` et `t2`.

3. Quel algorithme de tri choisiriez vous pour trier `t1` & `t2` ? Quel serait son coût ?

Un tri par base est ici le plus indiqué : pour une base constante, il permet de trier $\mathbb{T}_1$ & $\mathbb{T}_2$ en temps $O(2^{n/2} \times \log n)$; c'est mieux qu'un tri par comparaison optimal, dont le coût serait un $O(2^{n/2} \times n)$.

On suppose `t1` et `t2` triés par ordre de croissant.

4. Esquissez un algorithme pour décider s'il existe une collision entre `t1` et `t2` (c'est à dire une paire d'indices `i` et `j` tels que `t1[i]` est égal `t2[j]`).
Quel est son coût pire cas en fonction de n ?

On peut appliquer un algorithme similaire à celui de la fusion du tri fusion : on maintient deux compteurs `i` et `j` pour `t1` et `t2` et l'on incrémente celui correspondant à l'élément le plus petit, jusqu'à trouver une égalité ou avoir épuisé l'un des deux tableaux. Le coût pire cas est donné par la somme des tailles des deux tableaux, soit ici un $O(2^{n/2})$.

5. Montrez que l'on peut résoudre le problème du sac à dos binaire en temps et espace $\widetilde{O}(2^{n/2})$ (où la notation $\widetilde{O}$ indique que l'on « ignore » les facteurs (au plus) polynomiaux (ici, en n)).

Par la question 1, construire, trier et chercher une collision entre `t1` et `t2` permet de résoudre le problème. Il suffit alors de constater qu'aucune de ces étapes ne dépasse pas le coût cible :

- construire `t1` et `t2` (même de façon *relativement* naïve) prend un temps $\widetilde{O}(2^{n/2})$, et les stocker prend un espace $O(2^{n/2})$;
- les trier prend un temps $\widetilde{O}(2^{n/2})$ et peut se faire en espace supplémentaire constant ;
- l'étape de fusion prend un temps $O(2^{n/2})$ et peut se faire en espace supplémentaire constant.

Un algorithme dont le coût en espace est égal au coût en temps est en général peu attractif : il est par exemple bien plus accessible de réaliser 2^{40} opérations élémentaires (votre téléphone peut probablement le faire en temps raisonnable) que de stocker 2^{40} octets.

6. Esquissez une adaptation de l'algorithme de résolution de la question précédente qui permet de résoudre le problème du sac à dos binaire en temps T et espace $M \leq T$, sous la contrainte $T \times M = \widetilde{O}(2^n)$. (Par exemple, on doit pouvoir prendre $M \approx 2^{n/3}$ et $T \approx 2^{2n/3}$.)

On peut remarquer qu'au prix d'une stratégie légèrement moins efficace pour la détection de collision (par exemple basée sur un tableau associatif ou une recherche dichotomique), seul l'un des deux ensembles $\mathbb{T}_1$ et $\mathbb{T}_2$ a besoin d'être stocké. Il suffit alors de partitionner $\mathcal{S}$ en deux ensembles $\mathcal{S}_1$ et $\mathcal{S}_2$ de tailles déséquilibrées m et $n-m > m$, de stocker l'ensemble $\mathbb{T}_1$ de taille $\widetilde{O}(2^m)$ correspondant au plus petit, puis de chercher une collision avec les 2^{n-m} éléments de $\mathbb{T}_2$ générés *à la volée* (dans un ordre quelconque), ce qui prend un temps $\widetilde{O}(2^{n-m})$ et peut se faire en espace constant. L'algorithme ainsi adapté permet bien de résoudre le problème avec le *compromis temps-mémoire* $T \times M \approx 2^n$.

L'algorithme « de Schroeppel-Shamir » offre une solution plus attractive que celle de la question précédente pour diminuer le coût en espace de l'approche par rencontre au milieu. Il consiste à utiliser des représentations compressées de $\mathbb{T}_1$ et $\mathbb{T}_2$ de tailles seulement $O(2^{n/4})$, qui permettent néanmoins de toujours parcourir leurs éléments par ordre croissant pour un coût total $\widetilde{O}(2^{n/2})$. Ces représentations peuvent alors être substituées à `t1` et `t2` dans l'étape de recherche de collision ci-dessus, ce qui donne un algorithme en temps $\widetilde{O}(2^{n/2})$ et espace seulement $O(2^{n/4})$!

Dans ces représentations compressées, les éléments sont stockés dans des files de priorité (min) de taille au plus $2^{n/4}$.

7. Donnez une expression (fonction de n) du coût pire-cas des opérations d'insertion et d'extraction dans ces files de priorité, lorsqu'elles sont implémentées par des tas binaires.

Les opérations de file de priorité sur un tas binaire de capacité C sont de coût pire-cas $O(\log C)$. On a donc ici des coûts pire-cas en $O(n)$.

Dans tout ce qui suit, on suppose sans perte de généralité que n est divisible par 4.
La représentation compressée de $\mathbb{T}_1$ fonctionne de la façon suivante :

— on partitionne $\mathcal{S}_1$ en deux sous-ensemble $\mathcal{S}_{11}$ et $\mathcal{S}_{12}$ de tailles $n/4$, et l'on représente les $2^{n/4}$ éléments de $\mathbb{T}_{11} := \left\{\sum_{x\in\sigma} x, \sigma \in 2^{\mathcal{S}_{11}}\right\}$ & $\mathbb{T}_{12} := \left\{\sum_{x\in\sigma} x, \sigma \in 2^{\mathcal{S}_{12}}\right\}$ sous forme de tableaux triés par ordre croissant `t11` et `t12` en réalité, `t12` n'a pas besoin d'être trié ;
— on initialise une file de priorité min `pq1` avec les $2^{n/4}$ couples *(priorité, donnée)* : `(t11.(0) + t12.(j), (0, j))`, pour $j \in [\![2^{n/4}]\!]$;
— pour parcourir les éléments de $\mathbb{T}_1$ par ordre croissant, on itère la séquence suivante (donnée avec une syntaxe OCaml) tant que possible :

```
let x, (i, j) = pop pq1 in
let ip1 = i + 1 in
next := x ;
if ip1 < tpn4 then
    push pq1 (t11.(ip1) + t12.(j), (ip1, j))
```

où `tpn4` contient la taille des tableaux `t11` & `t12` et la référence `next` est modifiée avec la valeur de $\mathbb{T}_1$ à considérer.

Les questions suivantes visent à montrer que la suite des valeurs prises par `!next` énumère bien les éléments de $\mathbb{T}_1$ par ordre croissant.

On note $\mathbb{V}_1$ l'ensemble des couples (i, j) qui ont été extraits de `pq1` au terme d'un certain nombre (éventuellement nul) d'itérations de la séquence ci-dessus.

8. Montrez que la boucle itérant la séquence ci-dessus satisfait l'invariant qu'il existe une suite finie d'entiers (ℓ_k) telle que pour tout $j \in [\![2^{n/4}]\!]$:

— $\mathbb{V}_1$ contient l'ensemble des couples $(i < \ell_j, j)$;
— et `pq1` contient au plus une donnée (i, j), qui si elle existe satisfait $i = \ell_j$.

L'initialisation est immédiate en prenant $\ell_j = 0$ pour tout j, puisque $\mathbb{V}_1 = \varnothing$ et `pq1` contient exactement les couples $(0, j)$.
On suppose la propriété vraie au début d'une itération et l'on va montrer sa conservation. Soit (i, j) l'unique couple extrait de `pq1` à cette itération, par hypothèse l'on a $i = \ell_j$ et $\mathbb{V}_1 \supseteq \{(i, j), i < \ell_j\}$. La première partie de la propriété invariante est donc conservée à la fin de l'itération si l'on prend des nouvelles valeurs (ℓ'_k) données par $\ell'_k = \ell_k$ pour $k \neq j$, et $\ell'_j = \ell_j + 1$. On distingue ensuite deux cas en fonction du résultat du test de l'avant-dernière ligne : s'il s'évalue à faux aucun élément n'est ajouté à `pq1`, qui ne contient donc plus de couple de la forme $(\_, j)$; sinon il contient un unique couple $(i + 1 = \ell'_j, j)$; dans les deux cas la propriété est conservée.

Par construction, les éléments de $\mathbb{T}_1$ sont donnés par les $2^{n/2}$ sommes `t11.(i) + t12.(j)` pour $(i,j) \in [\![2^{n/4}]\!] \times [\![2^{n/4}]\!]$. Pour tout $\mathbb{X} \subseteq [\![2^{n/4}]\!] \times [\![2^{n/4}]\!]$, on note $\mathbb{T}_1^{\mathbb{X}}$ l'ensemble $\{$`t11.(i) + t12.(j)`$, (i,j) \in \mathbb{X}\}$.

9. Montrez que pour la même suite (ℓ_k) qu'à la question précédente, l'on a l'invariant de boucle que pour tout $j \in [\![2^{n/4}]\!]$, ou bien $\mathbb{V}_1 \supseteq [\![2^{n/4}]\!] \times \{j\}$, ou bien :

$$\min \mathbb{T}_1^{[\![2^{n/4}]\!]\times\{j\}} \setminus \mathbb{T}_1^{\mathbb{V}_1} = \mathbb{T}_1^{\{\ell_j, j\}}$$

(C'est à dire que l'on a ou bien extrait tous les éléments de la forme $(\_, j)$ de `pq1`, ou bien l'élément de $\mathbb{T}_1$ minimum (pour j fixé) qui peut s'écrire comme une somme `t11.(i) + t12.(j)` avec $(i,j) \notin \mathbb{V}_1$ est obtenu pour $i = \ell_j$.)

Par le fait que `t11` est trié par ordre croissant, l'élément minimum de $\mathbb{T}_1$ de la forme `t11.(i) + t12.(j)` pour j fixé est obtenu en prenant $i = 0$. La propriété est donc bien initialisée avant la première itération quand $\mathbb{V}_1 = \emptyset$ et pour les ℓ_k initialisés tout à zéro.
Pour la conservation, si $\mathbb{V}_1 \supseteq [\![2^{n/4}]\!] \times \{j\}$ alors il n'y a rien à montrer. Sinon par la question précédente $\mathbb{V}_1 \supseteq \{(i,j), i < \ell_j\}$ et la propriété est à nouveau satisfaite par le fait que `t11` est trié.

10. Conclure que la suite des valeurs prises par `!next` en itérant la séquence ci-dessus est bien celle des éléments de $\mathbb{T}_1$ par ordre croissant.

On montre l'invariant de boucle que si `pq1` n'est pas vide, alors elle contient forcément le couple `t11.(i) + t12.(j), (i, j)` avec `t11.(i) + t12.(j)` l'élément minimum de $\mathbb{T}_1 \setminus \mathbb{T}_1^{\mathbb{V}_1}$ (la définition de $\mathbb{V}_1$, les spécifications d'une file de priorité min et les première & troisième lignes feront le reste).
Par construction $\mathbb{T}_1 = \bigcup_{j \in [\![2^{n/4}]\!]} \mathbb{T}^{[\![2^{n/4}]\!]\times\{j\}}$, et donc :

$$\min \mathbb{T}_1 \setminus \mathbb{T}_1^{\mathbb{V}_1} = \min_{j \in [\![2^{n/4}]\!]} \min \mathbb{T}_1^{[\![2^{n/4}]\!]\times\{j\}} \setminus \mathbb{T}_1^{\mathbb{V}_1}$$

ce qui par la question précédente vaut :

$$\min_{j \in [\![2^{n/4}]\!]} \mathbb{T}_1^{\{\ell_j, j\}}$$

(en considérant ici uniquement les j tels que $\ell_j < 2^{n/4}$). Par la question 8 l'on a l'invariant que `pq1` contient toutes les données (i,j) avec $i = \ell_j$, qui par la structure du code sont associées aux priorités `t11.(i) + t12.(j)`, ce qui permet de conclure.

11. Discutez brièvement du coût de mise en place et d'opération de cette représentation compressée de $\mathbb{T}_1$.

Les constructions de `t11` et `t12` se font en temps et espace $\widetilde{O}(2^{n/4})$, de même que l'initialisation de `pq1`. Chacune des itérations de la séquence ci-dessus se fait avec un nombre constant d'opérations sur une file de priorité contenant au plus $O(2^{n/4})$ éléments, et donc en temps $O(n)$ et espace constant. On peut ainsi énumérer les éléments de $\mathbb{T}_1$ par ordre croissant en espace $\widetilde{O}(2^{n/4})$ et en temps $O(n)$ par élément, soit en temps $\widetilde{O}(2^{n/2})$ au total.

12. Expliquez comment construire et opérer de façon similaire une file de priorité permettant d'énumérer (avec le même coût en temps et espace) les éléments de $\mathbb{T}_2$ par ordre croissant.

La seule subtilité est que les éléments de $\mathbb{T}_2$ sont donnés par la *soustraction à t* des sommes de toutes les parties de $\mathcal{S}_2$. L'élément minimum de $\mathbb{T}_2$ est donc donné par la soustraction à t de la somme *maximale*, ce qui nécessite une petite adaptation. En reprenant les mêmes notations, on peut par exemple trier `t21` par ordre décroissant, initialiser `pq2` à `t - (t21.(0) + t22.(j)), (0, j)` pour tout j, et similairement remplacer la dernière ligne de la séquence par :

```
push pq2 (t - (t21.(ip1) + t22.(j)), (ip1, j))
```