Devoir Maison 2 – Algorithmique

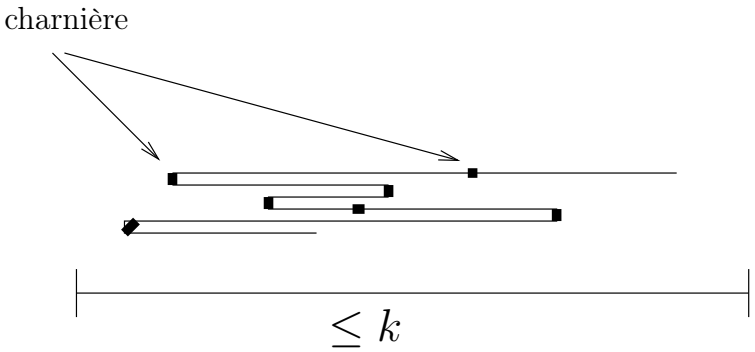
1 NP-complétude

1.1 Rappels

Le problème de 2-Partitions est le suivant. On dispose d'une liste d'entiers positifs $a_1, \dots, a_n$. On souhaite déterminer s'il existe une partition de $[\![1, n]\!] = I \sqcup J$ tel que $\sum_{i \in I} a_i = \sum_{j \in J} a_j$. On admettra que ce problème est NP-complet.

1.2 Un peu de charpenterie

Étant données n baguettes rigides de longueur entières $a_1, a_2, \dots, a_n$, pouvant être reliées dans cet ordre bout-à-bout par des charnières, et étant donné un entier k, ou souhaite assembler toutes les baguettes de manière qu'en repliant la chaîne obtenue la longueur totale ne dépasse pas k :



On nome ce problème "Charpenterie". Montrez que Charpenterie est NP-complet.

Indication : Une instance du problème 2-Partition est une liste d'entiers positifs $a_1, \dots, a_n$. On note $S = \sum_{i=1}^{n} a_i$ (si S est impair, on y ajoute 1) et on considère la liste $S, S/2, a_1, \dots, a_n, S/2, S$. On prendra ensuite $k = S$ pour obtenir un instance du problème Charpenterie.

2 Complexité amortie

2.1 Les tableaux

On dispose d'un tableau T de taille $|T|$ une puissance de 2. On peut accéder aux cases de celui-ci en temps 1, et modifier une case donnée en temps 1. On dispose de plus de deux opérations :

— Double() et double la taille de T et complète avec des 0. Si on note t_i les anciennes valeurs dans le tableau, on a donc :

$$T[i] = \begin{cases} t_i & \text{si } 0 \leq i < |T|/2 \\ 0 & \text{sinon} \end{cases}$$

Cette opération prends un temps $|T|$ (donc deux fois l'ancienne taille).

— Moitié() réduit de moitié la taille de T, en oubliant la première moitié du tableau. Si on note t_i les anciennes valeurs dans le tableau, on a donc :

$$T[i] = t_{i+|T|} \text{ si } 0 \leq i < |T|$$

Cette opération prends un temps $|T|$ (la moitié de l'ancienne taille).

On suppose le tableau comme étant initialement de taille $|T| = 1$.

2.2 Les files

On implémente une file à partir d'un tableau de la manière suivante :

Algorithm 1: Insertion

Entrées: Un élément x à insérer dans la file
Données: Le tableau T, et deux indices i, j représentant le début et la fin de la fille dans T
début
  $j \leftarrow j + 1$;
  si $j \geq |T|$ **alors**
    Double();
  $T[j] \leftarrow x$;

Algorithm 2: Extraction

Données: Le tableau T, et deux indices i, j représentant le début et la fin de la fille dans T
Sorties: L'élément x le plus ancien de la file, ou "File Vide"
début
  si $i > j$ **alors**
    retourner *"File Vide"*
  $x \leftarrow T[i]$;
  $T[i] \leftarrow 0$;
  $i \leftarrow i + 1$;
  si $i \geq |T|/2$ **alors**
    $i \leftarrow i - |T|/2$;
    $j \leftarrow j - |T|/2$;
    Moitié();
  retourner x

Algorithm 3: Multi-Extraction

Entrées: Le nombre d'extractions $k \geq 0$ à faire
Données: Le tableau T, et deux indices i, j représentant le début et la fin de la fille dans T
Sorties: La liste L constituée d'au plus k éléments, correspondant aux k extractions
début
  $L \leftarrow []$;
  pour ℓ ***de*** 1 ***à*** k **faire**
    $x \leftarrow$ Extraction();
    si $x =$ *"File Vide"* **alors**
      retourner L
    Ajouter x à la fin de L;
    `/* Ajouter un élément à la fin d'une liste se fait en temps 1 */`
  retourner L

2.3 Les questions

On suppose que T commence comme un tableau de taille 1, avec sa case initialisé à 0, et $i = 0$ et $j = -1$.

1. On n'autorise que des Insertions. Montrez que l'Insertion s'exécute en complexité amortie $O(1)$, en utilisant la méthode globale.
2. *On supposera pour simplifier que l'Insertion se fait en $O(1)$ (non-amorti, en toute situation).* On autorise les Insertions et les Extractions. Montrez que l'Extraction s'exécute en complexité amortie $O(1)$, en utilisant la méthode des acomptes.
3. *On supposera pour simplifier que l'Insertion et l'Extraction se font en $O(1)$ (non-amorti, en toute situation).* On autorise les trois opérations. Montrez que la multi-extraction se fait en complexité amortie $O(1)$ en utilisant la méthode du potentiel.
4. **(Bonus)** On autorise les trois opérations. Montrez qu'elles se font chacune en complexité amortie $O(1)$, en utilisant la méthode de votre choix.