Activité : Partition

Table des matières

1 Description du problème

Définition 1.1 – Partition d'un ensemble

Un *multiensemble* (parfois appelé sac) est un ensemble dans lequel chaque élément peut apparaître plusieurs fois.
Soit un multiensemble S de n entiers naturels.

$$S = \{s_i \mid s_i > 0\}_{1 \leq i \leq n}.$$

Une *(bi)partition* de S est constituée de deux sous-multiensembles S_1 et S_2 tels que :
- S_1 et S_2 sont non vides : $S_1 \neq \emptyset$ et $S_2 \neq \emptyset$;
- S_1 et S_2 sont disjoints : $S_1 \cap S_2 = \emptyset$;
- S_1 et S_2 recouvrent S : $S_1 \cup S_2 = S$;

Exemple – Partition d'un ensemble

Soit un multiensemble $S = \{1, 2, 3, 4, 5\}$.

- Les multiensembles S_1 et S_2 forment une partition de S.
  - $S_1 = \{1\}$ et $S_2 = \{2, 3, 4, 5\}$
  - $S_1 = \{2, 4\}$ et $S_2 = \{1, 3, 5\}$
- Les multiensembles S_1 et S_2 ne forment pas une partition de S.
  - $S_1 = \{1, 2, 3, 4, 5\}$ et $S_2 = \emptyset$, car S_2 est vide.
  - $S_1 = \{1, 2, 3\}$ et $S_2 = \{3, 4, 5\}$, car leur intersection est non vide.
  - $S_2 = \{1, 2\}$ et $S_2 = \{4, 5\}$, car 3 est dans 4, mais n'appartient ni à S_1 ni à S_2.

Définition 1.2 – Partition parfaite d'un multiensemble pair

Un multiensemble d'entiers S est dit *pair* si la somme des entiers de S est pair.
Une *partition parfaite* d'un multiensemble pair est une partition telle que la valeur absolue de la différence entre la somme des entiers de S_1 et la somme des entiers de S_2 est 0.

Exemple – Partition parfaite d'un multiensemble pair

$S = \{1, 2, 3, 4\}$ est un multiensemble pair.

- $S_1 = \{1, 3\}$ et $S_2 = \{2, 4\}$ ne forment pas une partition parfaite.
- $S_1 = \{1, 4\}$ et $S_2 = \{2, 3\}$ forment une partition parfaite.

Définition 1.3 – Partition parfaite d'un multiensemble impair

Un multiensemble d'entiers S est dit *impair* si la somme des entiers de S est impair.
Une *partition parfaite* d'un multiensemble impair est une partition telle que la valeur absolue de la différence entre la somme des entiers de S_1 et la somme des entiers de S_2 est 1.

Exemple – Partition parfaite d'un multiensemble impair

$S = \{1, 2, 3, 4, 5\}$ est un multiensemble impair.

- $S_1 = \{2, 4\}$ et $S_2 = \{1, 3, 5\}$ ne forment pas une partition parfaite.
- $S_1 = \{1, 2, 5\}$ et $S_2 = \{3, 4\}$ forment une partition parfaite.

Définition 1.4 – Problème de partitionnement

En informatique, le problème de partitionnement consiste à déterminer si une partition parfaite d'un ensembles d'entiers existe. C'est un problème *NP-complet*. Cependant, il existe plusieurs algorithmes qui résolvent efficacement le problème que ce soit de manière approchée ou optimale. Pour ces raisons, il est réputé "le plus facile des problèmes difficiles" [1].

2 Partition parfaite des entiers de 1 à n

Exercice 2.1

Trouver une partition parfaite des entiers de 1 à n pour $n = 4, 5, 6, 7, 8$.

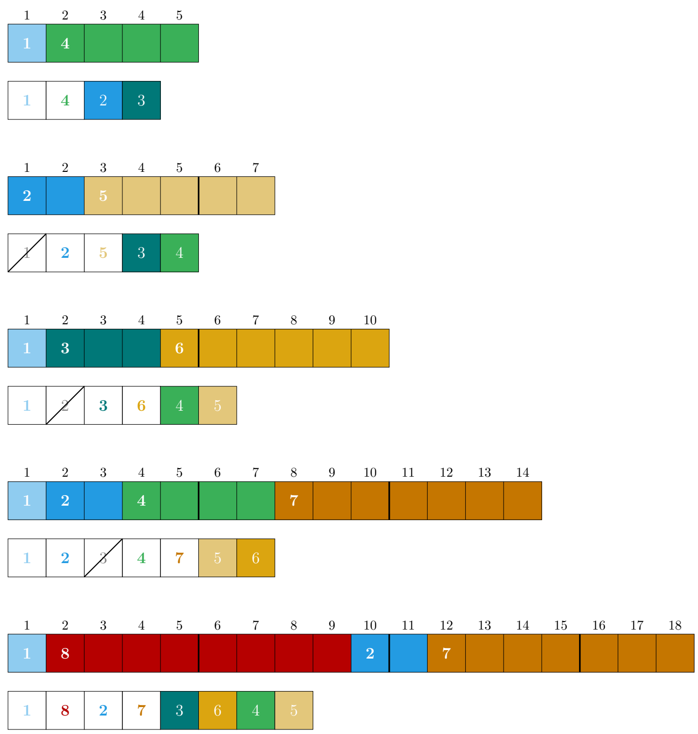

FIGURE 1 – Partition parfaite des entiers de 1 à n.

Définition 2.1 – Algorithme

Un *algorithme* répond à un problème. Il est composé d'un ensemble d'étapes simples nécessaires à la résolution, dont le nombre varie en fonction de la taille des données.

Remarque 2.1

Plusieurs algorithmes peuvent répondre à un même problème.

Remarque 2.2

Un algorithme peut répondre à plusieurs problèmes.

Exercice 2.2

Donner un algorithme pour trouver une partition parfaite des entiers de 1 à n.

Indice : Distinguer les cas en fonction du reste r de la division euclidienne de n par 4. C'est-à-dire qu'il existe $k \geq 0$ et $0 \leq r \leq 3$ tels quel $n = 4 \times k + r$. $\triangleleft$

Algorithme 2.1 – Partition parfaite des entiers de 1 à n.

— Soit $n = 4 \times k + r$ le quotient k et le reste r ($0 \leq r \leq 3$) de la division euclidienne de n par 4.
  — Si $r = 1$, alors éliminer l'objet 1.
  — Si $r = 2$, alors ranger l'objet 1 et éliminer l'objet 2.
  — Si $r = 3$, alors ranger les objets 1 et 2 et éliminer l'objet 3.
— Répéter $2 \times k$ fois l'action suivante
  (ou de manière équivalente, répéter tant que le sac n'est pas rempli) :
  — ranger le plus petit et le plus grand objet.

Remarque 2.3

Ce problème admet une symétrie évidente puisque l'on peut inverser la partition, c'est-à-dire échanger les ensembles S_1 et S_2. De manière générale, on peut toujours échanger des objets entre S_1 et S_2 si cela ne change pas leurs sommes.

Exercice 2.3

- Trouver une partition parfaite des entiers de 1 à 16.
- Remarquez que toutes les paires d'objets formées par l'algorithme ont la même somme. Trouvez d'autres partitions parfaites par échanges successifs.

3 Algorithmes gloutons

Définition 3.1 – Algorithme glouton

Un *algorithme glouton* est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local. Dans certains cas, cette approche aboutit à un optimum global, mais dans le cas général c'est une heuristique qui n'aboutit pas nécessairement à un optimum global.

Algorithme 3.1 – Algorithme glouton

- Calculer le somme des objets divisée par deux pour déterminer la capacité du sac.
- Trier les objets du sac par ordre décroissant.
- Sélectionner le premier objet.
- Répéter tant qu'il reste des objets et que le sac n'est pas rempli :
  - Ranger l'objet dans le sac si la capacité le permet, ou éliminer l'objet.
  - Sélectionner l'objet suivant.

Exercice 3.1Application de l'algorithme glouton

Appliquer l'algorithme glouton sur une instance.

Niveau	Sac	Capacité
Facile	11, 8, 7, 5, 2, 1	17
Intermédiaire	16, 12, 10, 9, 6, 5, 3, 2, 1	32
Difficile	?	?

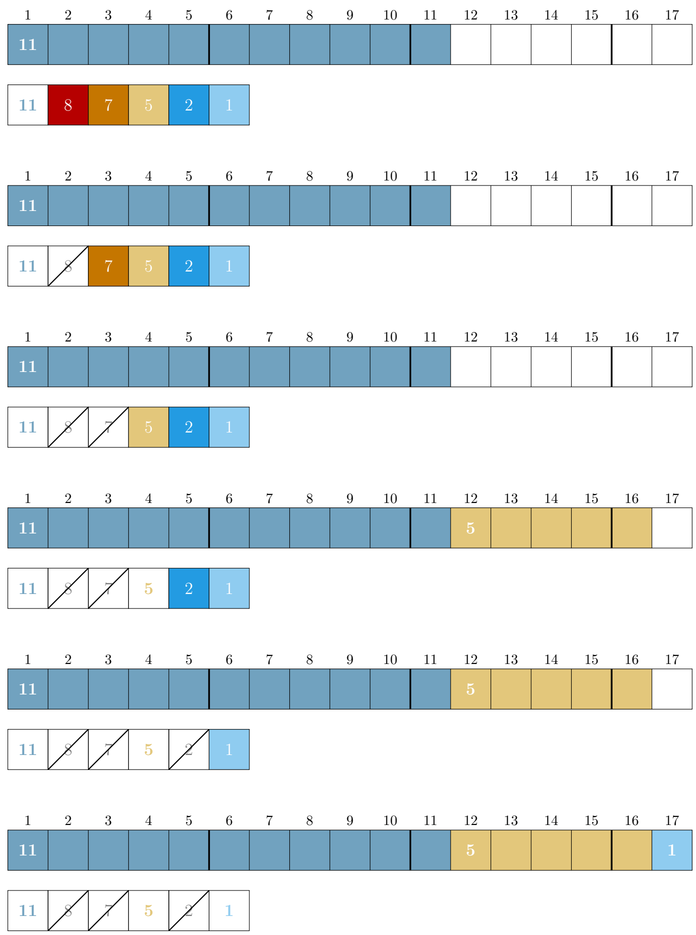

Figure 2 – Solution de l'instance facile de l'exercice 3

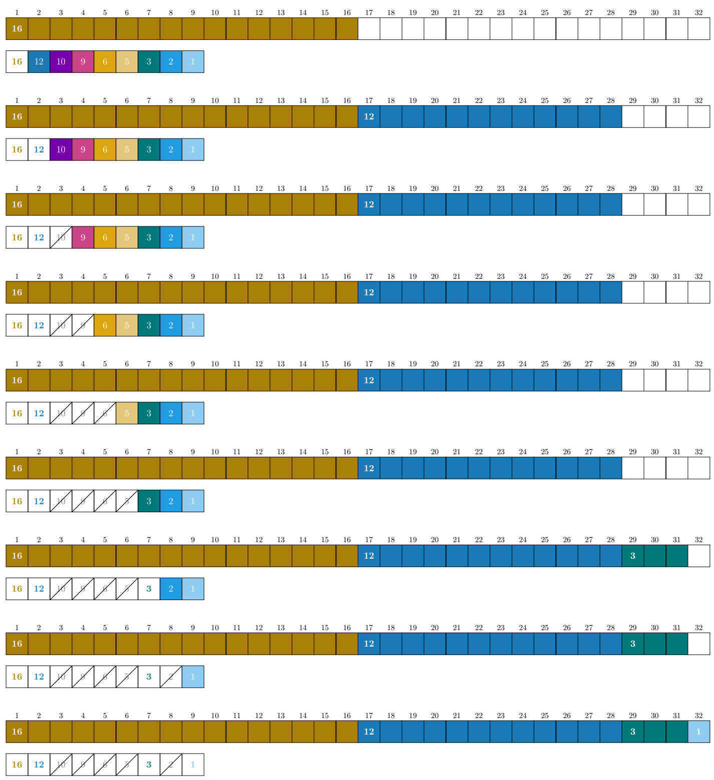

FIGURE 3 – Solution de l'instance intermédiaire de l'exercice 3

Exercice 3.2

Appliquer l'algorithme glouton sur une instance.

Niveau	Sac	Capacité
Facile	14, 13, 11, 7, 5, 3	26
Intermédiaire	16, 15, 14, 13, 12, 9, 8, 6, 1	47
Difficile	?	?

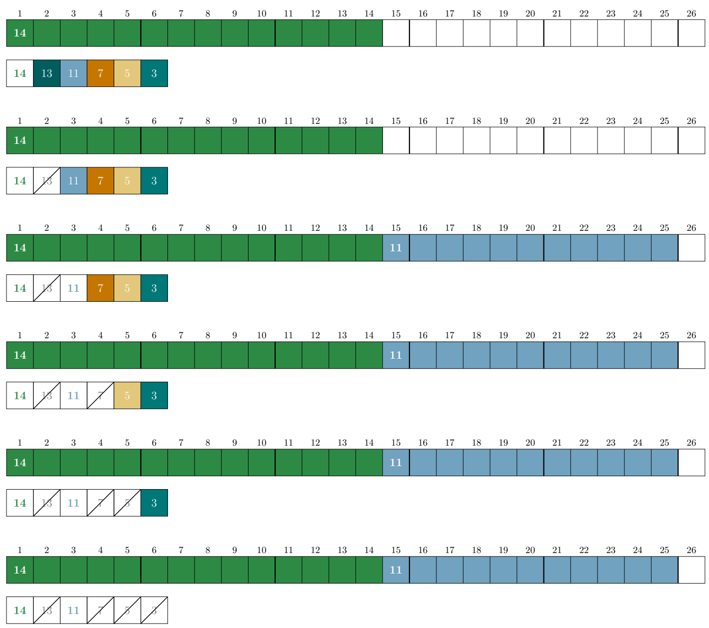

FIGURE 4 – Solution de l'instance facile de l'exercice 3

Algorithme 3.2 – Algorithme glouton répété

Répéter jusqu'à ce que le sac soit rempli ou contienne tous les objets :
- — Appliquer l'algorithme glouton.
- — Éliminer le plus grand objet

Exercice 3.3

Appliquer l'algorithme glouton sur une instance de l'exercice 3.

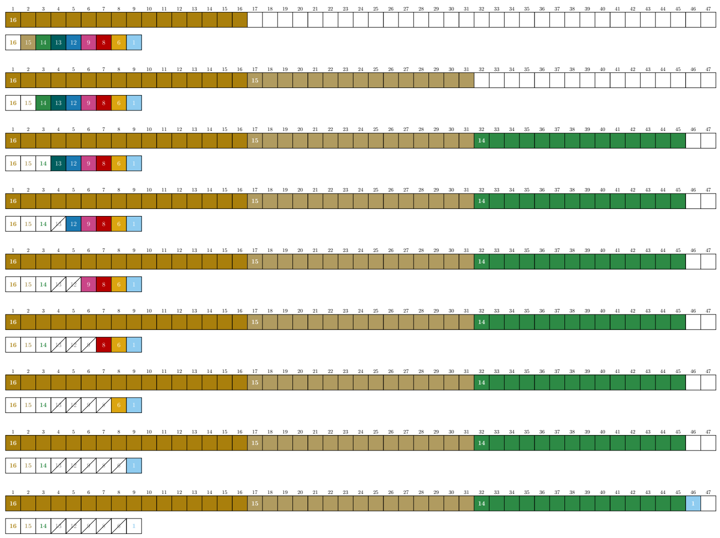

Figure 5 – Solution de l'instance intermédiaire de l'exercice 3

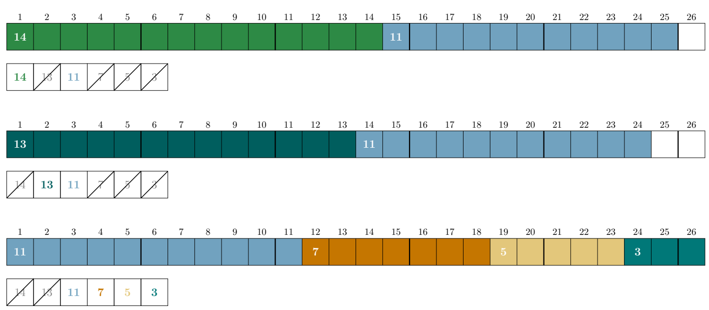

Figure 6 – Solution de l'instance facile de l'exercice 4

Exercice 3.4

Appliquer l'algorithme glouton répété sur une instance.

Niveau	Sac	Capacité
Facile	13, 11, 9, 8, 6, 4	25
Intermédiaire	16, 15, 14, 10, 9, 8, 6, 5, 3	43
Difficile	?	?

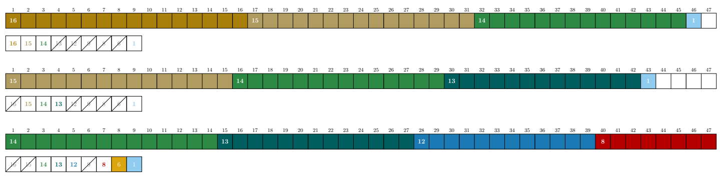

Figure 7 – Solution de l'instance intermédiaire de l'exercice 4

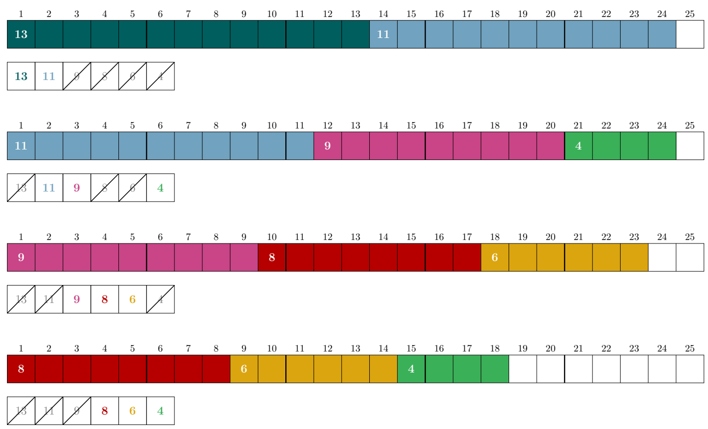

Figure 8 – Solution de l'instance facile de l'exercice 4

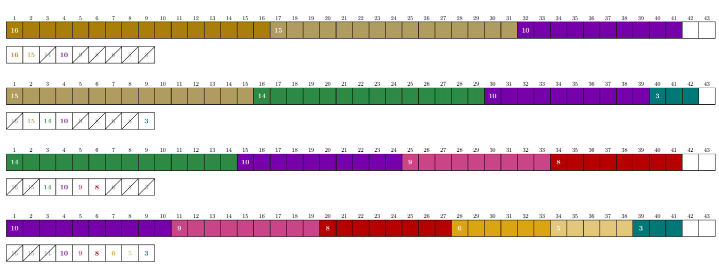

Figure 9 – Solution de l'instance intermédiaire de l'exercice 4

4 Méthodes exactes

Exercice 4.1

Appliquer la programmation dynamique sur une instance de l'exercice 4.

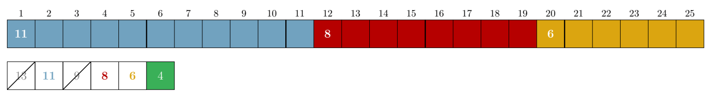

Figure 10 – Solution de l'instance facile de l'exercice 4

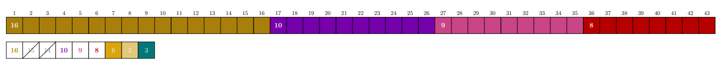

Figure 11 – Solution de l'instance intermédiaire de l'exercice 4

Exercice 4.2

Appliquer l'algorithme glouton répété sur une instance.

Niveau	Sac	Capacité
Facile	16, 15, 11, 4, 2, 1	24
Intermédiaire	18, 15, 13, 10, 8, 5, 3, 2	37
Difficile	?	?

4.1 Programmation dynamique

4.2 Tester et Générer

A Problèmes connexes

Algorithme A.1

test

Allons plus loin

test

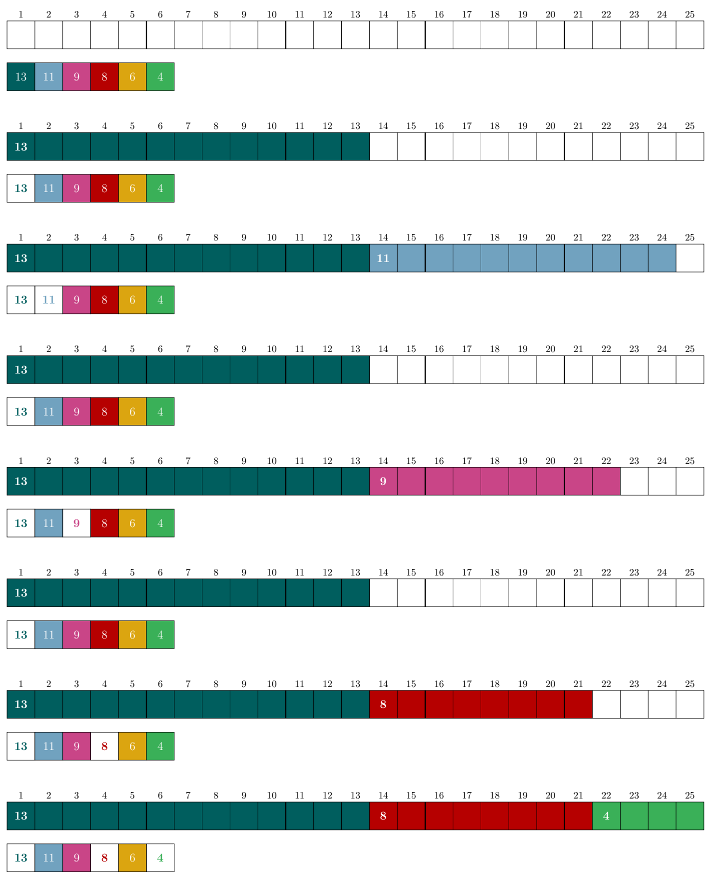

FIGURE 12 – Solution de l'instance facile de l'exercice 4

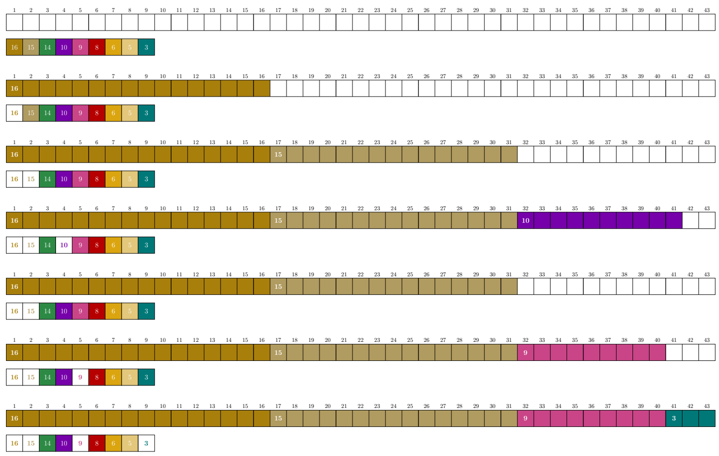

FIGURE 13 – Solution de l'instance intermédiaire de l'exercice 4

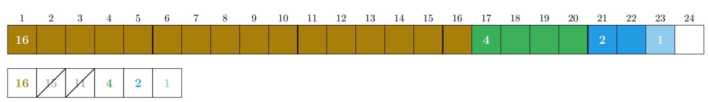

FIGURE 14 – Solution de l'instance facile de l'exercice 4

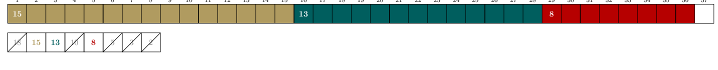

FIGURE 15 – Solution de l'instance intermédiaire de l'exercice 4

Références

[1] Stephan Mertens. The easiest hard problem : Number partitioning, 2003.