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 S, 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 $n = 4, 5, 6, 7, 8$.

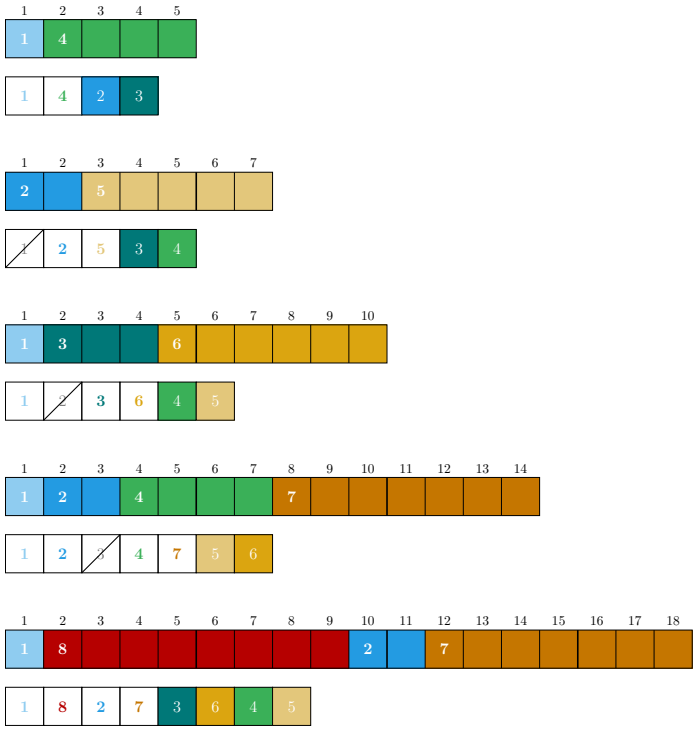


FIGURE 1 – Partition parfaite des entiers de 1 à n.

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 $n = 4 \times k + r$. ◁

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.

Type	Niveau	Sac	Capacité
1	Facile	11, 8, 7, 5, 2, 1	17
	Intermédiaire	16, 12, 10, 9, 6, 5, 3, 2, 1	32
	Difficile	16, 15, 13, 12, 9, 8, 6, 5, 4, 3, 2, 1	47
2	Facile	14, 13, 11, 7, 5, 3	26
	Intermédiaire	16, 15, 14, 13, 12, 9, 8, 6, 1	47
	Difficile	16, 15, 13, 12, 11, 10, 8, 7, 6, 4, 3, 2	53
3	Facile	13, 11, 9, 8, 6, 4	25
	Intermédiaire	16, 15, 14, 10, 9, 8, 6, 5, 3	43
	Difficile	16, 15, 14, 13, 12, 11, 10, 8, 7, 6, 4, 3	59
4	Facile	16, 15, 11, 4, 2, 1	24
	Intermédiaire	18, 15, 13, 10, 8, 5, 3, 2	37
	Difficile	18, 17, 16, 15, 14, 5, 2, 1	44

TABLE 1 – Instances du problème de partitionnement.

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

- Déterminer la capacité du sac : la somme des objets divisée par deux.
- Trier les objets par ordre décroissant.
- Répéter tant qu'il reste des objets et que le sac n'est pas rempli :
  - ranger le plus grand objet dans le sac si la capacité le permet.
  - Sinon éliminer l'objet.

Exercice 3.1

Appliquer l'algorithme glouton sur une instance de type 1 du tableau 1 page 5.

Exercice 3.2

Appliquer l'algorithme glouton sur une instance de type 2 du tableau 1 page 5.

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 répété sur une instance de type 2 du tableau 1 page 5.

Exercice 3.4

Appliquer l'algorithme glouton répété sur une instance de type 3 du tableau 1 page 5

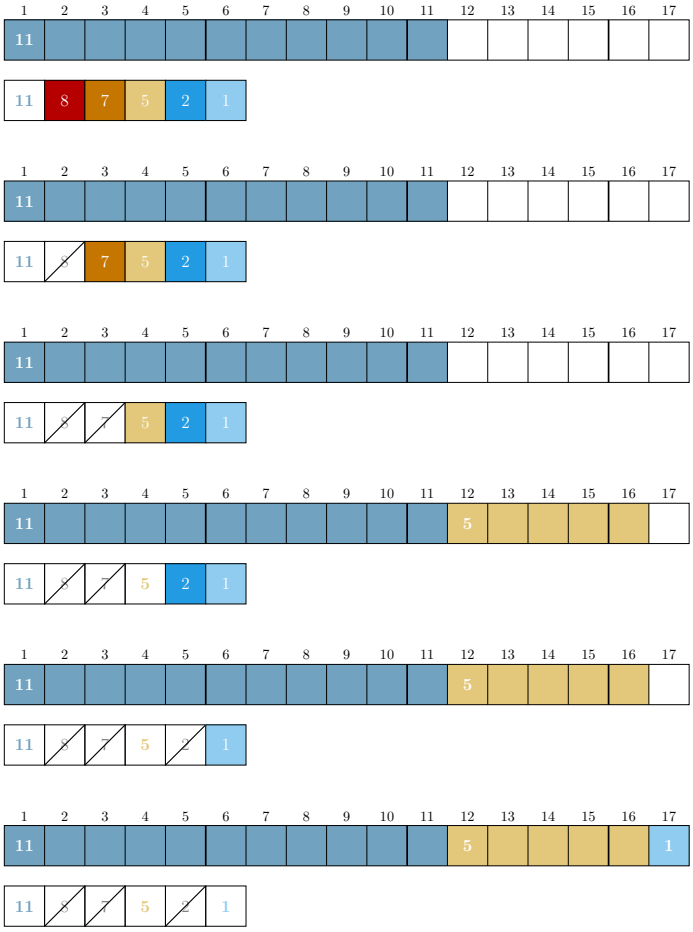


FIGURE 2 – Solution de l'instance facile de de type 1.

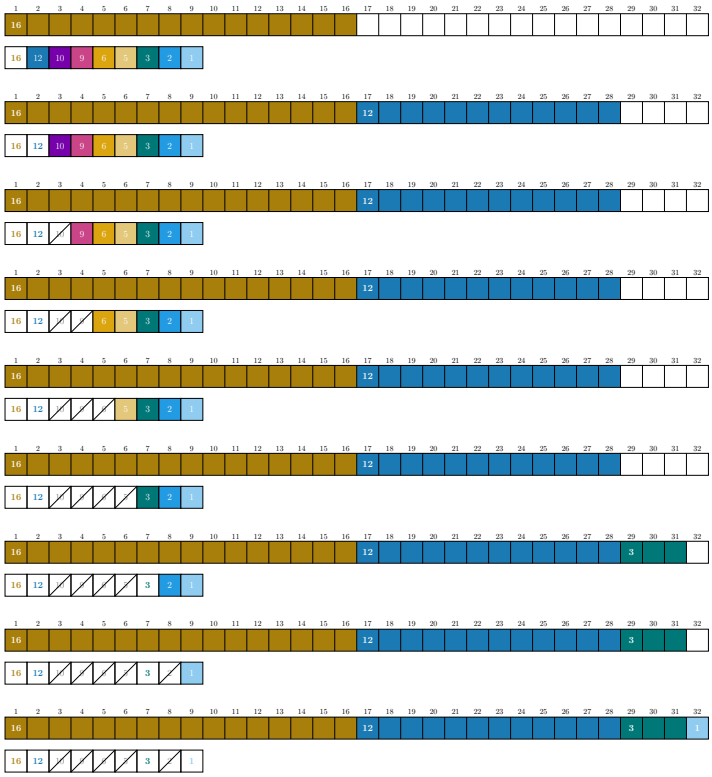


FIGURE 3 – Solution de l'instance intermédiaire de type 1.

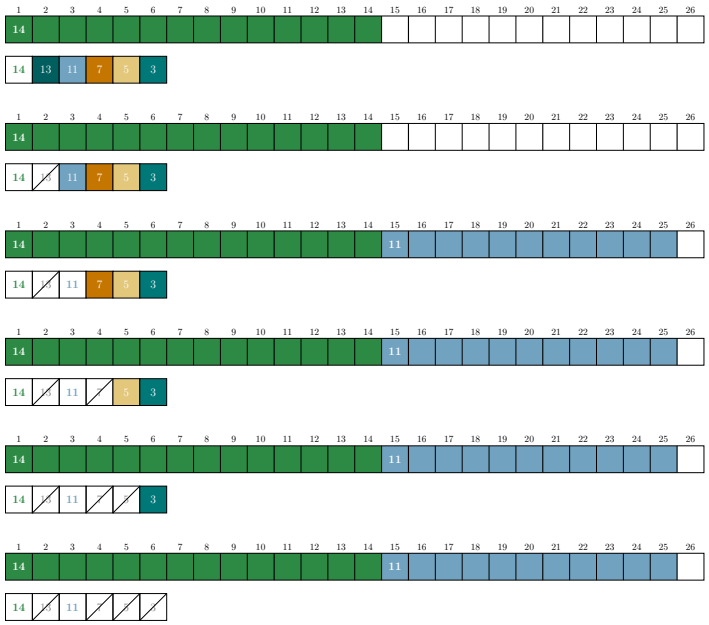


FIGURE 4 – Solution de l'instance facile de type 2.

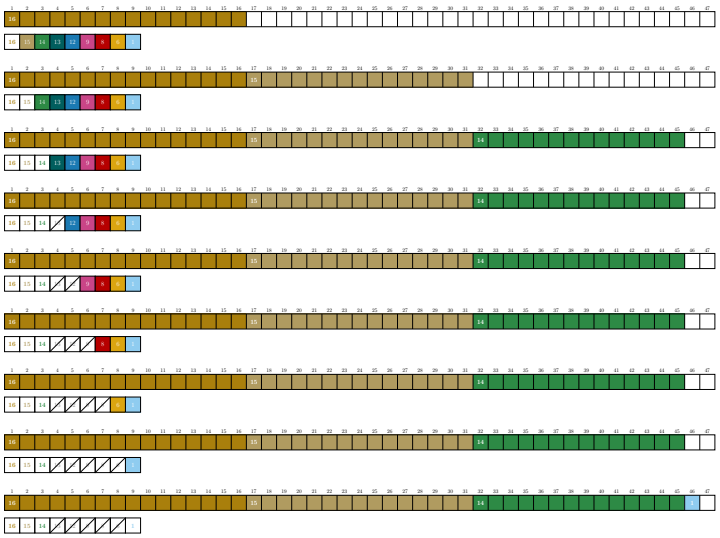


FIGURE 5 – Solution de l'instance intermédiaire de type 2.

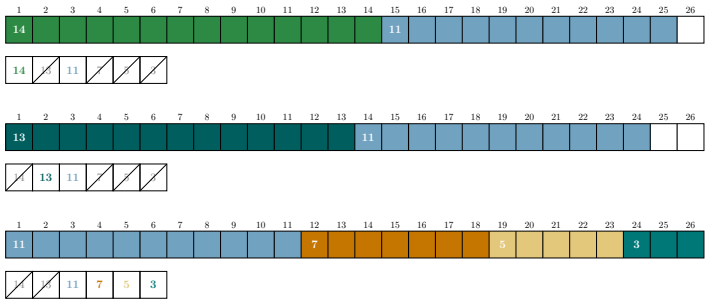


FIGURE 6 – Solution de l'instance facile de type 2.

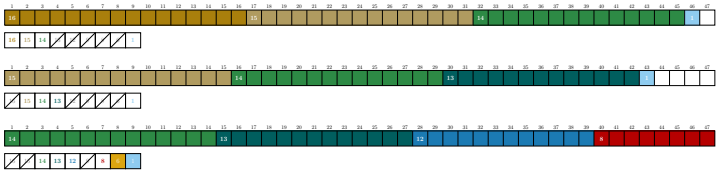


FIGURE 7 – Solution de l'instance intermédiaire de type 2.

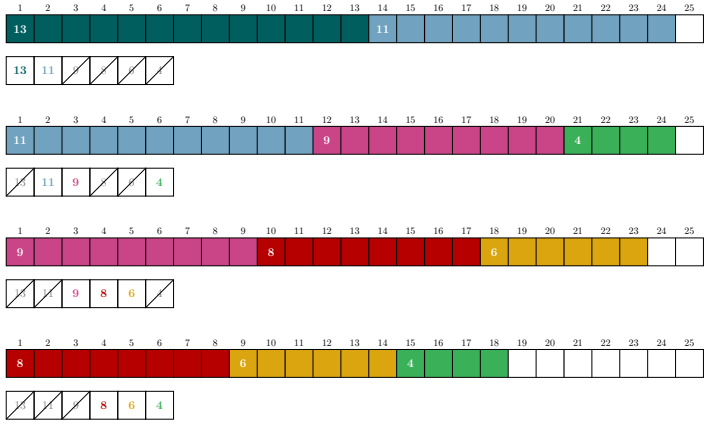


FIGURE 8 – Solution de l'instance facile de type 3.

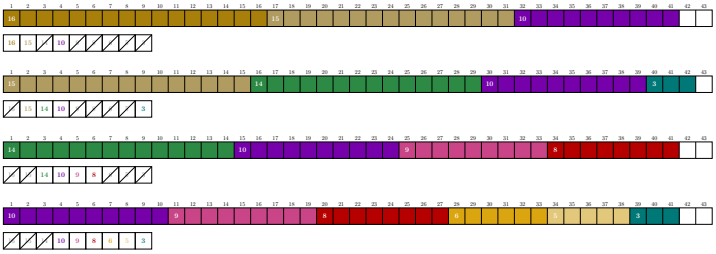


FIGURE 9 – Solution de l'instance intermédiaire de type 3.

4 Programmation dynamique

Algorithme 4.1 – Algorithme de programmation dynamique

- Déterminer la capacité du sac.
- Trier les objets par ordre décroissant.
- Répéter tant qu'il reste des objets et que le sac n'est pas rempli (la dernière case n'est pas marquée) :
  - répéter pour chaque case marquée en partant de la dernière :
    - déterminer la case atteinte en rangeant l'objet immédiatement après la case marquée (utiliser l'objet comme règle).
    - Si la case atteinte n'est pas marquée, placer un marqueur de l'objet.

Exercice 4.1

Appliquer la programmation dynamique sur une instance de type 3 du tableau 1 page 5.

Inf	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Inf	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	2	0
Inf	0	0	0	0	0	0	0	0	3	0	2	0	1	0	0	0	0	0	0	3	0	3	0	2	0
Inf	0	0	0	0	0	0	0	4	3	0	2	0	1	0	0	0	4	0	4	3	4	3	0	2	0
Inf	0	0	0	0	0	5	0	4	3	0	2	0	1	5	5	0	4	0	4	3	4	3	5	2	5

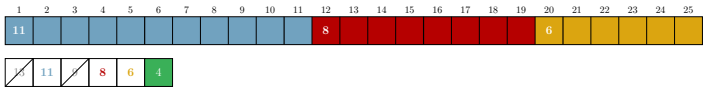


FIGURE 10 – Solution de l'instance facile de type 3.

```
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
[20]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
[39]   0   0   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   1   0   0
[20]   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0
[39]   0   0   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   2   1   0   0
[20]   0   0   0   0   0   0   0   0   0   0   3   3   2   0   0   0   0   0   0
[39]   0   0   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   0   4   0   0   0   3   2   1   0   0
[20]   0   0   0   0   0   4   4   4   0   0   3   3   2   0   0   0   0   0   0
[39]   0   4   4   4   0   0
 [1] Inf   0   0   0   0   0   0   0   0   5   4   0   0   0   3   2   1   0   0
[20]   5   0   0   0   5   4   4   4   0   0   3   3   2   0   5   5   5   0   0
[39]   5   4   4   4   0   0
 [1] Inf   0   0   0   0   0   0   0   6   5   4   0   0   0   3   2   1   6   6
[20]   5   0   0   6   5   4   4   4   6   0   3   3   2   6   5   5   5   0   6
[39]   5   4   4   4   6   6
 [1] Inf   0   0   0   0   0   0   0   6   5   4   0   0   0   3   2   1   6   6
[20]   5   0   0   6   5   4   4   4   6   0   3   3   2   6   5   5   5   0   6
[39]   5   4   4   4   6   6
```

FIGURE 11 – Solution de l'instance intermédiaire de type 3.

Exercice 4.2

Appliquer la programmation dynamique sur une instance de type 4 du tableau 1 page 5

```
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
[20]   0   0   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   1   0   0
[20]   0   0   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   2   1   0   0
[20]   0   0   0   0   0   0
 [1] Inf   0   0   0   4   0   0   0   0   0   0   0   3   0   0   0   2   1   0   0
[20]   4   4   0   0   0   0
```

```
 [1] Inf   0   5   0   4   0   5   0   0   0   0   3   0   5   0   2   1   5   5
[20]   4   4   5   5   0   0
 [1] Inf   6   5   6   4   6   5   6   0   0   0   3   6   5   6   2   1   5   5
[20]   4   4   5   5   6   0
 [1] Inf   6   5   6   4   6   5   6   0   0   0   3   6   5   6   2   1   5   5
[20]   4   4   5   5   6   0
```

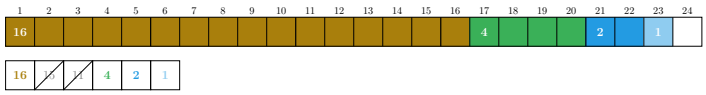


FIGURE 12 – Solution de l'instance facile de type 4.

```
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
[20]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   1
[20]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   0   0   0   3   0   2   0   0   1
[20]   0   0   0   0   0   0   0   0   0   3   0   0   3   0   2   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   0   0   4   0   0   3   0   2   0   0   1
[20]   0   0   0   0   4   0   4   0   0   3   0   0   3   0   2   0   0   0   0
 [1] Inf   0   0   0   0   0   0   0   5   0   4   0   0   3   0   2   0   0   1
[20]   0   0   5   0   4   0   4   5   0   3   0   0   3   0   2   0   0   5   0
 [1] Inf   0   0   0   0   6   0   0   5   0   4   0   0   3   0   2   0   0   1
[20]   0   6   5   0   4   0   4   5   0   3   0   6   3   0   2   0   0   5   0
 [1] Inf   0   0   7   0   6   0   0   5   0   4   7   0   3   0   2   7   0   1
[20]   0   6   5   0   4   7   4   5   0   3   7   6   3   0   2   7   0   5   0
 [1] Inf   0   8   7   0   6   0   8   5   0   4   7   8   3   0   2   7   8   1
[20]   0   6   5   8   4   7   4   5   8   3   7   6   3   8   2   7   8   5   0
 [1] Inf   0   8   7   0   6   0   8   5   0   4   7   8   3   0   2   7   8   1
[20]   0   6   5   8   4   7   4   5   8   3   7   6   3   8   2   7   8   5   0
```

FIGURE 13 – Solution de l'instance intermédiaire de type 4.

5 Tester et Générer

Algorithme 5.1 – Algorithme Tester-et-Générer

- Déterminer la capacité.
- Trier les objets par ordre décroissant.
- Descente : Répéter tant que le dernier objet n'est pas dans le sac :
  - Répéter pour chaque objet :
    - ranger l'objet dans le sac si la capacité le permet.
  - Si le sac est rempli, arrêter l'algorithme.
  - Sinon, effectuer un retour arrière simple : retirer le plus petit objet du sac.
- Retour arrière sautée quand le plus petit objet du deck est dans le sac.
  - Retirer les objets du sac trouver jusqu'à ce que vous trouviez le plus petit objet que vous n'avez pas pu faire rentrer.
  - Sinon éliminer l'objet.

Exercice 5.1

Appliquer Tester-et-Générer sur une instance de type 3 du tableau 1 page 5.

Exercice 5.2

Appliquer Tester-et-Générer sur une instance de type 4 du tableau 1 page 5.

6 Problèmes connexes

Algorithme 6.1

test

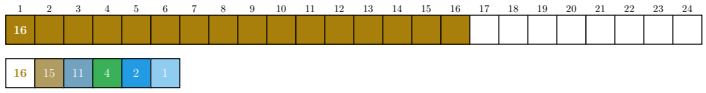
Allons plus loin

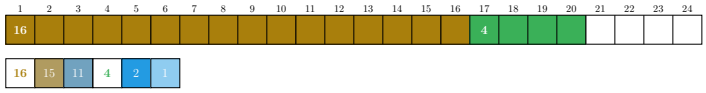
test

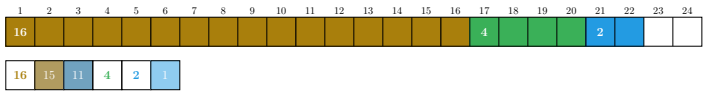
Références

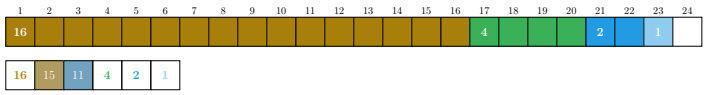
[1] Stephan Mertens. The easiest hard problem : Number partitioning, 2003.

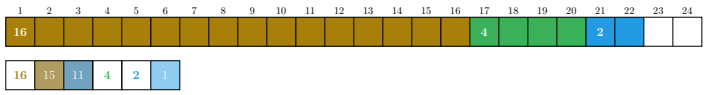

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 15 11 4 2 1

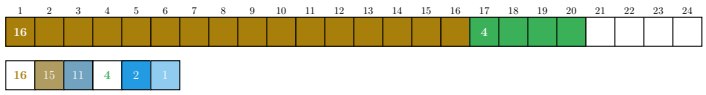

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16
16 15 11 4 2 1

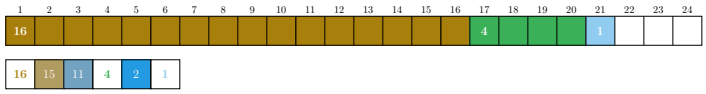

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4
16 15 11 4 2 1

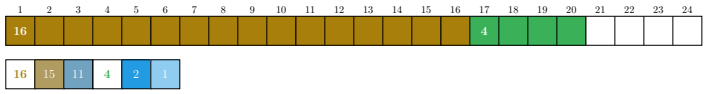

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4 2
16 15 11 4 2 1

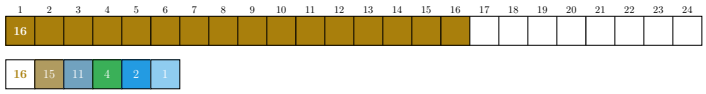

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4 2 1
16 15 11 4 2 1

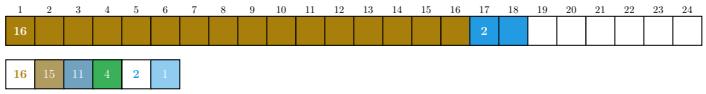

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4 2
16 15 11 4 2 1

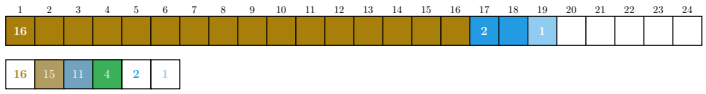

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4
16 15 11 4 2 1

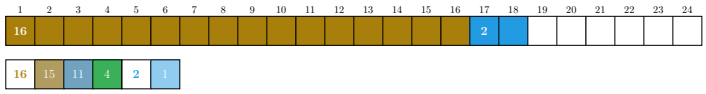

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4 1
16 15 11 4 2 1

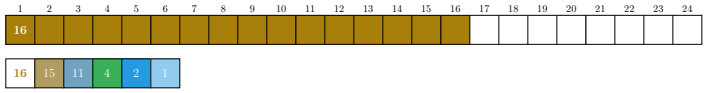

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16
16 15 11 4 2 1

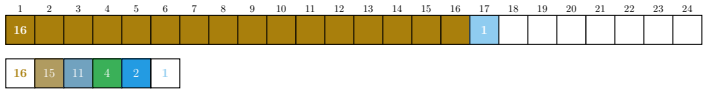

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4
16 15 11 4 2 1


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4 1
16 15 11 4 2 1


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 4
16 15 11 4 2 1


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16
16 15 11 4 2 1


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 2
16 15 11 4 2 1


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 2 1
16 15 11 4 2 1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 2
16 15 11 4 2 1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16
16 15 11 4 2 1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
16 1
16 15 11 4 2 1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

FIGURE 15 – Solution de l'instance intermédiaire de l'exercice 3