Activité : Partition parfaite

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 paire.
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 impaire.
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.

Remarque 1.1

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.

2 Partition parfaite des entiers de 1 à n

Exercice 2.1

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

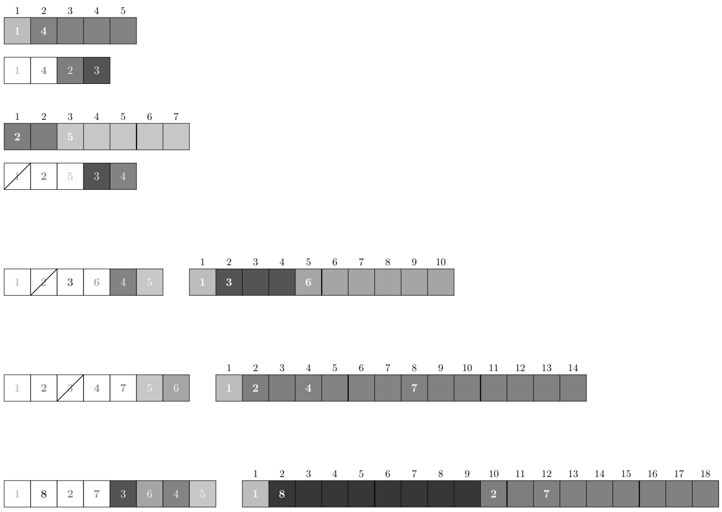

FIGURE 1 – Partition parfaite des entiers de 1 à n.

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 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 tant que le sac n'est pas rempli :
  - ranger le plus petit et le plus grand objet.

Exercice 2.3

Trouver une partition parfaite des entiers de 1 à 18.

Remarque 2.3

Toutes les paires d'objets formées par l'algorithme ont la même somme.

Exercice 2.4

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

— Déterminer la capacité du sac.
— Trier les objets par ordre décroissant.
— Répéter tant qu'il reste des objets :
  — ranger le plus grand objet dans le sac si sa capacité le permet ;
  — Sinon, retirer l'objet ;
  — Si le sac est rempli, arrêter.

Exercice 3.1

Appliquer l'algorithme glouton sur une instance de type 1.

#	Type	Taille	Entiers	Capacité
1	1	6	11, 8, 7, 5, 2, 1	17
2	1	6	16, 11, 10, 8, 4, 1	25
3	1	9	15, 13, 8, 7, 6, 5, 4, 2, 1	30
4	1	9	16, 12, 10, 9, 6, 5, 3, 2, 1	32
5	1	12	16, 15, 13, 12, 9, 8, 6, 5, 4, 3, 2, 1	47

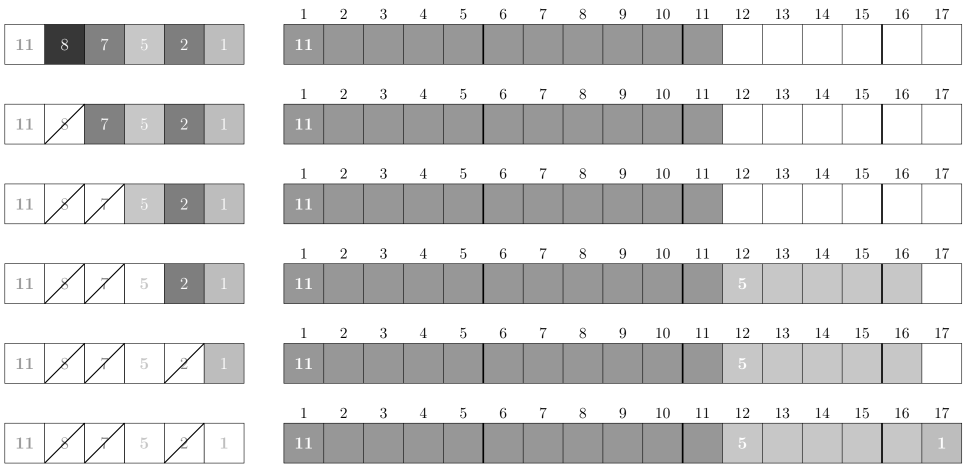

Figure 2 – Application de l'algorithme glouton sur l'instance #1 de type 1.

Exercice 3.2

Appliquer l'algorithme glouton sur une instance de type 2.

#	Type	Taille	Entiers	Capacité
6	2	6	14, 13, 11, 7, 5, 3	26
7	2	6	13, 12, 11, 10, 7, 4	28
8	2	9	16, 15, 14, 13, 12, 9, 8, 6, 1	47
9	2	9	15, 14, 13, 12, 11, 10, 7, 4, 3	44
10	2	12	16, 15, 13, 12, 11, 10, 8, 7, 6, 4, 3, 2	53

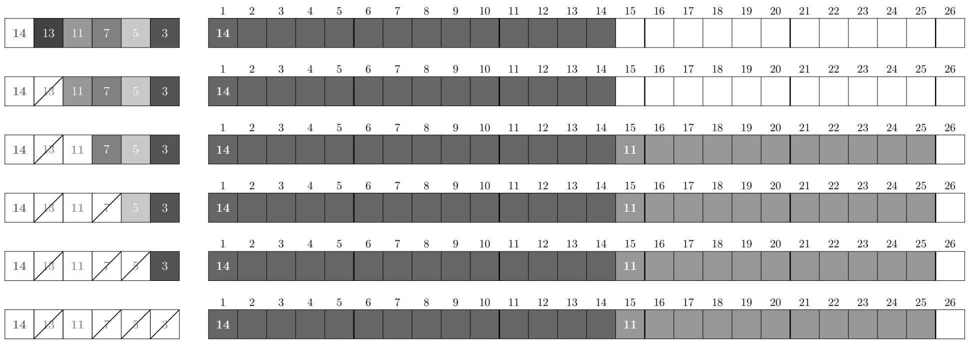

Figure 3 – Application de l'algorithme glouton sur l'instance #6 de type 2.

Remarque 3.1

- Trouver une autre solution en changeant l'ordre du glouton.
- Remarquer que plusieurs ordres donne la même solution : échanger 8 et 7 dans #1.
- Le glouton répété garantit de trouver des solutions distinctes.

Algorithme 3.2 – Algorithme glouton répété

Répéter jusqu'à ce que le sac contienne tous les objets :
- appliquer l'algorithme glouton ;
- Si le sac est rempli, arrêter ;
- Sinon éliminer le plus grand objet.

Exercice 3.3

Appliquer l'algorithme glouton répété sur une instance de type 2.

#	Type	Taille	Entiers	Capacité
6	2	6	14, 13, 11, 7, 5, 3	26
7	2	6	13, 12, 11, 10, 7, 4	28
8	2	9	16, 15, 14, 13, 12, 9, 8, 6, 1	47
9	2	9	15, 14, 13, 12, 11, 10, 7, 4, 3	44
10	2	12	16, 15, 13, 12, 11, 10, 8, 7, 6, 4, 3, 2	53

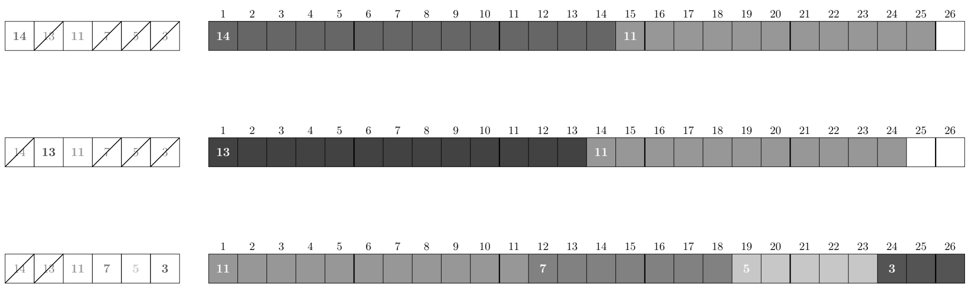

Figure 4 – Application de l'algorithme glouton répété sur l'instance #6 de type 2.

Exercice 3.4

Appliquer l'algorithme glouton répété sur une instance de type 3.

#	Type	Taille	Entiers	Capacité
11	3	6	13, 11, 9, 8, 6, 4	25
12	3	6	16, 13, 12, 11, 7, 3	31
13	3	9	16, 15, 14, 10, 9, 8, 6, 5, 3	43
14	3	9	16, 15, 13, 11, 9, 7, 5, 4, 3	41
15	3	12	16, 15, 14, 13, 12, 11, 10, 8, 7, 6, 4, 3	59

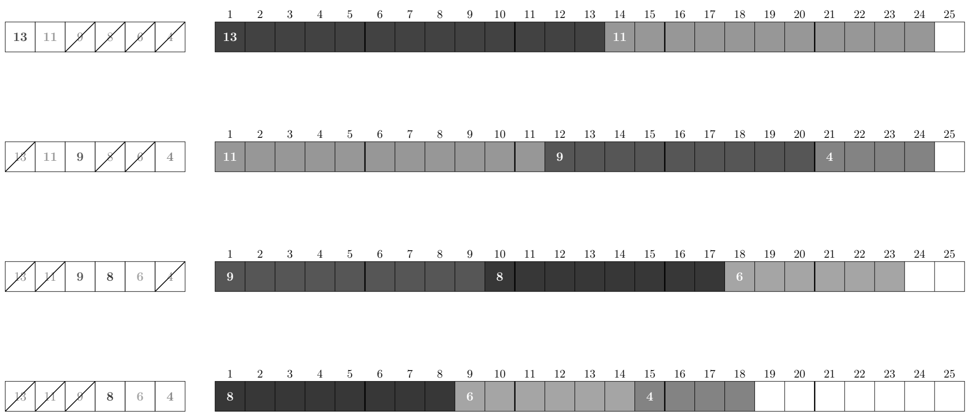

FIGURE 5 – Application de l'algorithme glouton répété sur l'instance #11 de type 3.

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 :
  - 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.
  - Si le sac est rempli (la dernière case est marquée), alors arrêter.

Algorithme 4.2 – Reconstruction du sac en programmation dynamique

Tant qu'il reste des marqueurs dans le sac :
- sélectionner le marqueur le plus à droite ;
- ranger l'objet à la place du marqueur en retirant des marqueurs si nécessaire.

Exercice 4.1

Appliquer la programmation dynamique sur une instance de type 3.

#	Type	Taille	Entiers	Capacité
11	3	6	13, 11, 9, 8, 6, 4	25
12	3	6	16, 13, 12, 11, 7, 3	31
13	3	9	16, 15, 14, 10, 9, 8, 6, 5, 3	43
14	3	9	16, 15, 13, 11, 9, 7, 5, 4, 3	41
15	3	12	16, 15, 14, 13, 12, 11, 10, 8, 7, 6, 4, 3	59

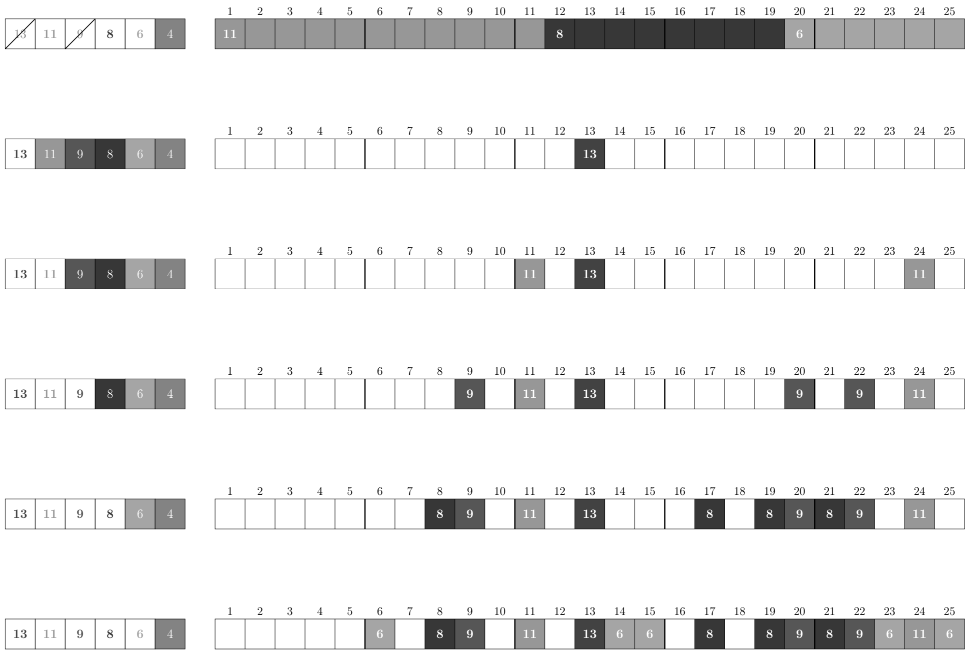

FIGURE 6 – Application de la programmation dynamique sur l'instance #11 de type 3.

Exercice 4.2

Appliquer la programmation dynamique sur une instance de type 4.

#	Type	Taille	Entiers	Capacité
16	4	6	16, 15, 11, 4, 2, 1	24
17	4	6	15, 14, 9, 8, 6, 1	26
18	4	8	18, 15, 13, 10, 8, 5, 3, 2	37
19	4	8	18, 15, 13, 10, 8, 5, 3, 2	37
20	4	8	18, 17, 16, 15, 14, 5, 2, 1	44

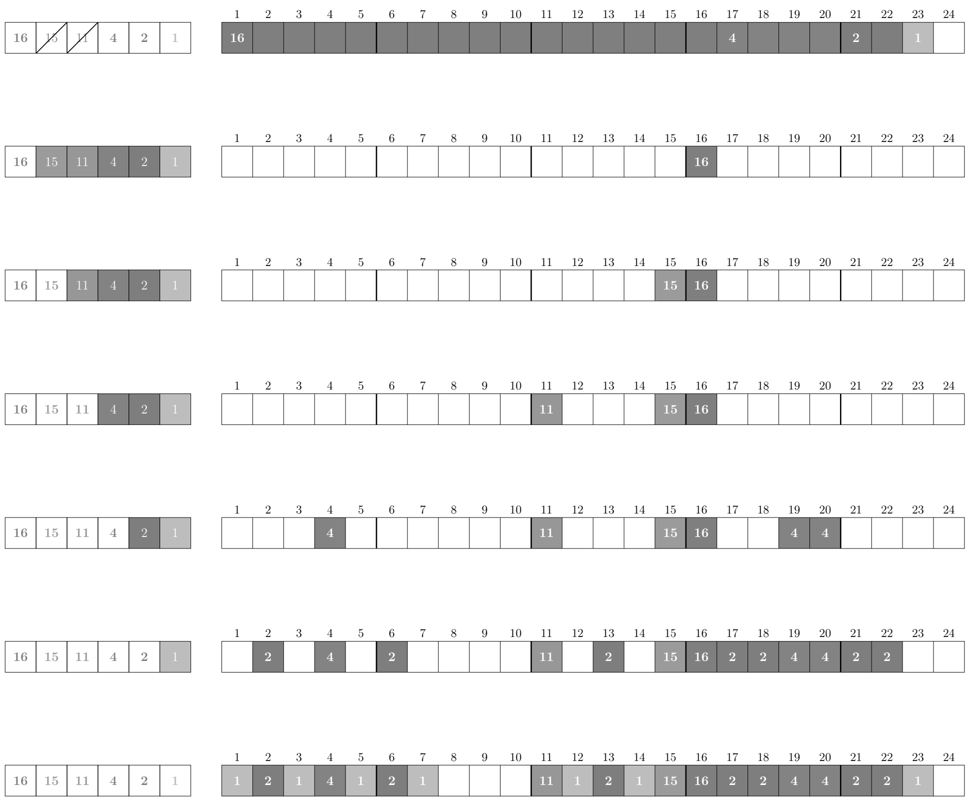

Figure 7 – Application de la programmation dynamique sur l'instance #16 de type 4.

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.

6 Problèmes connexes

#	Type	Taille	Entiers	Capacité
1	1	6	11, 8, 7, 5, 2, 1	17
2	1	6	16, 11, 10, 8, 4, 1	25
3	1	9	15, 13, 8, 7, 6, 5, 4, 2, 1	30
4	1	9	16, 12, 10, 9, 6, 5, 3, 2, 1	32
5	1	12	16, 15, 13, 12, 9, 8, 6, 5, 4, 3, 2, 1	47
6	2	6	14, 13, 11, 7, 5, 3	26
7	2	6	13, 12, 11, 10, 7, 4	28
8	2	9	16, 15, 14, 13, 12, 9, 8, 6, 1	47
9	2	9	15, 14, 13, 12, 11, 10, 7, 4, 3	44
10	2	12	16, 15, 13, 12, 11, 10, 8, 7, 6, 4, 3, 2	53
11	3	6	13, 11, 9, 8, 6, 4	25
12	3	6	16, 13, 12, 11, 7, 3	31
13	3	9	16, 15, 14, 10, 9, 8, 6, 5, 3	43
14	3	9	16, 15, 13, 11, 9, 7, 5, 4, 3	41
15	3	12	16, 15, 14, 13, 12, 11, 10, 8, 7, 6, 4, 3	59
16	4	6	16, 15, 11, 4, 2, 1	24
17	4	6	15, 14, 9, 8, 6, 1	26
18	4	8	18, 15, 13, 10, 8, 5, 3, 2	37
19	4	8	18, 15, 13, 10, 8, 5, 3, 2	37
20	4	8	18, 17, 16, 15, 14, 5, 2, 1	44

Allons plus loin

test

Références