

# Synthèse du cours 4 : programmation dynamique (1)

8 octobre 2024

François Laroussinie

**NB :** Ces synthèses ont pour but de compléter les notes prises en cours. Elles ne les remplacent pas ! En particulier, la plupart des preuves n'y figurent pas. Rappel : il faut programmer les algorithmes vus en cours.

## 0.1 Subset sum problem

Le problème est le suivant :

**Input :** Un ensemble  $E$  de  $n$  entiers  $x_1, \dots, x_n$  et un entier  $V$ .  
**Output :** existe-t-il un sous-ensemble  $E' \subseteq E$  de poids exactement  $V$ , ie tel que  $\sum_{x \in E'} x = V$ .

Pour résoudre ce problème, on peut utiliser un backtracking en énumérant tous les sous-ensembles de  $E$  et en s'arrêtant dès qu'un de ces sous-ensembles est de poids  $V$ . A faire...

Une autre approche consiste à utiliser une table de booléens  $K$  de taille  $n+1 \times V+1$  et de la construire de manière à vérifier :  $K[i, s] = \top$  ssi il existe un sous-ensemble de  $\{x_1, x_2, \dots, x_i\}$  de poids  $s$ . On a d'abord :  $K[0, s] = \perp$  pour  $s > 0$ , et  $K[i, 0] = \top$  pour tout  $0 \leq i \leq n$ . Ensuite, pour le cas général on a :

$$K[i, s] = \begin{cases} K[i-1, s] & \text{si } x_i > s \\ K[i-1, s] \vee K[i-1, s-x_i] & \text{sinon} \end{cases}$$

Cela donne un algorithme en  $O(n \cdot V)$  en temps et en espace :

**Procédure** Subset1 ( $V, E$ )

**begin**

$K$  : tableau de booléens de taille  $(n+1) \times (V+1)$   
**pour**  $s = 1 \dots V$  **faire**  $K[0, s] := \perp$   
**pour**  $i = 0 \dots n$  **faire**  $K[i, 0] := \top$   
**pour**  $i = 1 \dots n$  **faire**  
    **pour**  $s = 1 \dots V$  **faire**  
        **si**  $x_i \leq s$  **alors**  $K[i, s] := K[i-1, s] \vee K[i-1, s-x_i]$   
        **else**  $K[i, s] := K[i-1, s]$   
**return**  $K[n, V]$

La correction de l'algorithme s'énonce sous la forme d'un invariant :

**Correction :** Après la  $i$ -ème itération,  $K[i, s] = \top$  si et seulement si il existe un sous-ensemble de  $\{x_1, \dots, x_i\}$  de poids  $s$ .

On vérifie que c'est vrai pour la ligne 0 : on peut seulement réaliser un poids nul sans aucun élément. Supposons désormais que la ligne  $i$  est correcte, il reste à vérifier que c'est toujours vrai après le calcul de la ligne  $i+1$  : si  $K[i+1, s] = \top$ , alors c'est que on a soit  $K[i, s] = \top$  (et donc – car la ligne  $i$  est correcte – qu'un sous-ensemble de poids  $s$  est déjà

possible avec les éléments  $x_1, \dots, x_i$ ), soit  $K[i, s - x_{i+1}] = \top$  (et donc qu'un sous-ensemble des éléments  $\{x_1, \dots, x_i\}$  de poids  $s - x_{i+1}$  est possible, et on peut donc le compléter avec  $x_{i+1}$  pour arriver à  $s$ ), dans les deux cas il existe un sous-ensemble de poids  $s$ . Et inversement, si il existe un sous-ensemble de  $\{x_1, \dots, x_{i+1}\}$  de poids  $s$ , alors soit il contient  $x_{i+1}$  et c'est donc qu'il existe un sous-ensemble de  $\{x_1, \dots, x_i\}$  de poids  $s - x_{i+1}$  (et donc  $K[i, s - x_{i+1}] = \top$ ), soit il ne contient pas  $x_{i+1}$  et c'est donc qu'il existe un sous-ensemble de  $\{x_1, \dots, x_i\}$  de poids  $s$  (et donc  $K[i, s] = \top$ ), dans les deux cas on mettra  $K[i + 1, s]$  à  $\top$ .

On peut améliorer la complexité en espace en remarquant que seule la ligne  $i - 1$  est utilisée pour calculer la ligne  $i$ . En faisant attention à parcourir les valeurs dans le bon sens (décroissant) pour ne pas utiliser plusieurs fois un même élément  $x_i$ , on obtient l'algorithme `Subset2`.

```

Procédure Subset2 ( $V, E$ )
begin
  K : tableau de booléens de taille ( $V + 1$ )
  pour  $s = 1 \dots V$  faire  $K[s] := \perp$ 
   $K[0] := \top$  pour  $i = 1 \dots n$  faire
    pour  $s = V \dots x_i$  faire
       $K[s] := K[s] \vee K[s - x_i]$ 
  return  $K[V]$ 

```

**Construction d'une solution.** A partir du tableau à deux dimensions, il est très facile de retrouver un ensemble solution (lorsqu'il en existe) :

```

Procédure Solution ( $V, E, K$ )
begin
  si  $K[n, V]$  alors
     $s := V$ 
    res :=  $\emptyset$ 
    pour  $i = n \dots 1$  faire
      si ( $s \geq x_i \wedge K[i - 1, s - x_i]$ ) alors
        res := res  $\cup \{x_i\}$ 
         $s := s - x_i$ 
      si ( $s == 0$ ) alors return res
  return  $\perp$ 

```

Mais on peut aussi coupler la construction de solutions avec la construction de la table  $K$  par l'algorithme « économe » en mémoire :

## 1 Rendre la monnaie

Ici l'objectif est de minimiser le nombre de pièces pour constituer une certaine somme  $S$  à partir d'un ensemble de  $n$  types de pièces de valeur  $p_1, \dots, p_n$ . Notons que l'on fait une hypothèse importante : on suppose que l'on dispose d'autant de pièces que nécessaire pour chaque catégorie  $p_1, \dots, p_n$ .

**Procédure Subset3** ( $V, E$ )

```

begin
  K : tableau de booléens de taille ( $V + 1$ )
  sol : tableau d'ensembles de taille ( $V + 1$ )
  pour  $s = 1 \dots V$  faire
    | K[s] :=  $\perp$ ,   sol[s] := undef
  K[0] :=  $\top$ ,   sol[0] :=  $\emptyset$ 
  pour  $i = 1 \dots n$  faire
    | pour  $s = V \dots x_i$  faire
      | | si K[s -  $x_i$ ] alors
        | | | sol[s] := sol[s -  $x_i$ ]  $\cup$  { $x_i$ }
        | | | K[s] :=  $\top$ 
  return (K[V], sol[V])

```

On définit alors  $T[i, s]$  le nombre minimal de pièces de valeurs  $p_1, \dots, p_i$  pour constituer la somme  $s$ . On a alors  $T[i, 0] = 0$  pour tout  $i$ , et par convention on prend  $T[0, s] = \infty$  pour  $s > 0$ . On a alors :

$$T[i, s] = \begin{cases} \min(T[i-1, s], 1 + T[i, s - p_i]) & \text{si } s \geq p_i \\ T[i-1, s] & \text{sinon} \end{cases}$$

Le résultat de notre problème est alors  $T[n, V]$  (et si  $T[n, V] = \infty$ , c'est qu'il n'est pas possible de réaliser la somme  $V$  avec ces pièces!).

**Exemple.** On reprend le cas  $S = 8$  et  $p_1 = 1, p_2 = 4$  et  $p_3 = 6$ .

$S \rightarrow$	0	1	2	3	4	5	6	7	8
	0	$\infty$							
$p_1 = 1$	0	1	2	3	4	5	6	7	8
$p_1 = 1$	0	1	2	3	4	5	6	7	8
$p_2 = 4$	0	1	2	3	1	2	3	4	2
$p_3 = 6$	0	1	2	3	1	2	1	2	2

D'où l'algorithme suivant où on n'utilise pas les valeurs  $T[s, 0]$  :

$T[1 \dots n, 0 \dots S]$  : tableau d'entiers  
 $T[i, 0] = 0 \quad \forall i$   
 Pour  $i = 1, \dots, n$  :  
 ... Pour  $s = 1, \dots, S$  :  
 ... ..  $T[i, s] = \begin{cases} \infty & \text{si } i = 1 \wedge s < p_i \\ 1 + T[1, s - p_1] & \text{si } i = 1 \wedge s \geq p_i \\ T[i-1, s] & \text{si } i > 1 \wedge s < p_i \\ \min(T[i-1, s], 1 + T[i, s - p_i]) & \text{sinon} \end{cases}$

Pour reconstituer un ensemble solution, on applique le schéma suivant :

— Si  $T[i, s] = T[i-1, s]$ , alors la pièce  $p_i$  n'est pas utile et la solution est celle pour  $T[i-1, s]$ ,

- Si  $T[i, s] = 1 + T[i, s - p_i]$ , alors il faut une pièce de  $p_i$  ajoutée à une solution pour  $T[i, s - p_i]$ ,
- (Et si  $T[i - 1, s] = 1 + T[i, s - p_i]$ , alors les deux choix sont possibles!)

La complexité de la construction de  $T[.,.]$  est  $n \cdot (S + 1)$ , donc en  $O(n \cdot S)$ . Retourner un ensemble solution se fait en  $O(n + T[n, S])$  : le  $n$  vient du nombre maximal de « saut » de pièces (1 pour chaque catégorie) et  $T[n, S]$  vient du nombre maximal d'opérations pour chaque pièce.

On peut là encore améliorer l'algorithme en ne prenant qu'un tableau à une dimension :

$T[0 \dots S]$  : tableau d'entiers  
 $T[0] = 0 \quad \forall i$   
 Pour  $i = 1, \dots, n$  :  
 ... Pour  $s = 1, \dots, S$  :  
 ... ..  $T[s] = \begin{cases} \infty & \text{si } i = 1 \wedge s < p_i \\ 1 + T[s - p_1] & \text{si } i = 1 \wedge s \geq p_i \\ T[s] & \text{si } i > 1 \wedge s < p_i \\ \min(T[s], 1 + T[s - p_i]) & \text{sinon} \end{cases}$