

# Séance à distance

Programmation dynamique : l'équation de Bellman & Premiers pas en SQL

**Déroulé conseillé :** Partie 1 (lire le cours + exemple,  $\approx 20$  min), exercice 1 ( $\approx 35$  min), Partie 2 (lecture des exemples,  $\approx 15$  min), exercice 2 sur machine ( $\approx 40$  min).

## 1 Programmation dynamique : l'équation de Bellman

La programmation dynamique s'applique quand un problème d'optimisation se décompose en **sous-problèmes qui se recoupent**. Elle repose sur une idée simple, due à Richard Bellman (1957) :

### Définition : principe d'optimalité et équation de Bellman

**Principe d'optimalité.** Une solution optimale d'un problème est formée de solutions optimales de ses sous-problèmes.

**Équation de Bellman.** C'est la relation de récurrence qui traduit ce principe : la valeur optimale  $V$  d'un état s'obtient en essayant tous les choix possibles, chacun menant à un état plus simple dont on connaît (récursivement) la valeur optimale :

$$V(\text{état}) = \min_{\text{choix } c} \left( \text{coût}(c) + V(\text{état obtenu après } c) \right)$$

(ou max s'il s'agit de maximiser un gain), complétée par un ou plusieurs **cas de base**.

L'équation de Bellman se programme directement sous la forme d'une **fonction récursive**. Mais les mêmes sous-problèmes reviennent très souvent : pour ne pas les recalculer, on enregistre chaque résultat dans un dictionnaire la première fois qu'il est calculé, et on le réutilise ensuite. C'est la **mémoïsation**.

### Exemple : le rendu de monnaie

On dispose de pièces de valeurs 1, 4 et 6 (en quantité illimitée). On veut rendre la somme 8 avec **le moins de pièces possible**.

*L'algorithme glouton* (prendre toujours la plus grosse pièce possible) donne  $6 + 1 + 1$ , soit 3 pièces. Or  $4 + 4$  n'en utilise que 2 : le glouton n'est pas optimal ici.

**1. Sous-problème.** Pour  $0 \leq s \leq 8$ , on note  $m(s)$  le nombre minimal de pièces pour rendre  $s$ .

**2. Équation de Bellman.** Pour rendre  $s$ , on choisit une première pièce  $p \leq s$ , puis on rend au mieux les  $s - p$  restants :

$$m(0) = 0 \quad \text{et} \quad m(s) = 1 + \min_{p \in \{1,4,6\}, p \leq s} m(s - p) \quad (s \geq 1).$$

### 3. Traduction récursive directe.

```
def rendu(pieces, s):  
    if s == 0:                                     # cas de base  
        return 0  
    meilleur = float('inf')  
    for p in pieces:  
        if p <= s:  
            meilleur = min(meilleur, 1 + rendu(pieces, s - p))    # Bellman  
    return meilleur
```

Cette fonction est correcte, mais elle recalcule sans cesse les mêmes valeurs : par exemple,  $m(2)$  est recalculé en venant de  $m(8)$  (pièce 6), de  $m(6)$  (pièce 4), de  $m(3)$  (pièce 1)... Le nombre d'appels explose quand la somme augmente.

**4. Version mémoïsée.** On ajoute un dictionnaire `memo` qui associe à chaque somme déjà traitée son résultat :

```
def rendu(pieces, s, memo):
    if s == 0:                                # cas de base
        return 0
    if s not in memo:                          # pas encore calculé
        meilleur = float('inf')
        for p in pieces:
            if p <= s:
                meilleur = min(meilleur, 1 + rendu(pieces, s - p, memo))
        memo[s] = meilleur                    # on enregistre
    return memo[s]                            # on réutilise

rendu([1, 4, 6], 8, {}) # renvoie 2
```

À la fin de l'appel, le dictionnaire `memo` contient :

| $s$    | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8        |
|--------|---|---|---|---|---|---|---|----------|
| $m(s)$ | 1 | 2 | 3 | 1 | 2 | 1 | 2 | <b>2</b> |

Chaque valeur n'a été calculée qu'une seule fois.

### Exercice 1 : le robot ramasseur

Un robot part de la case en haut à gauche d'une grille et doit atteindre la case en bas à droite. Il ne peut se déplacer que **vers la droite** ou **vers le bas**. Sur chaque case se trouvent des pièces, qu'il ramasse en passant (cases de départ et d'arrivée comprises). On veut **maximiser** le nombre de pièces ramassées.

|         | $j = 0$ | $j = 1$ | $j = 2$ | $j = 3$ |
|---------|---------|---------|---------|---------|
| $i = 0$ | 1       | 3       | 2       | 1       |
| $i = 1$ | 2       | 1       | 0       | 1       |
| $i = 2$ | 9       | 5       | 1       | 2       |

On note  $g[i][j]$  le nombre de pièces de la case  $(i, j)$ .

- Stratégie gloutonne.** À chaque étape, le robot va sur la case voisine (droite ou bas) qui contient le plus de pièces. Quel chemin suit-il ? Combien de pièces ramasse-t-il ?
- On note  $G(i, j)$  le nombre maximal de pièces que le robot peut avoir ramassées en *arrivant* sur la case  $(i, j)$ .
  - Que vaut  $G(0, 0)$  ? Exprimer  $G(0, j)$  (première ligne) et  $G(i, 0)$  (première colonne).
  - Par quelles cases le robot peut-il arriver en  $(i, j)$  si  $i \geq 1$  et  $j \geq 1$  ? En déduire l'**équation de Bellman** donnant  $G(i, j)$ .
- Calculer à la main toutes les valeurs  $G(i, j)$  (on pourra les présenter dans un tableau de même forme que la grille). Quel est le nombre maximal de pièces ? Comparer avec la question 1.
- Compléter la fonction récursive mémoïsée suivante, qui renvoie  $G(i, j)$  :

```
def robot(g, i, j, memo):
```

```

    if (i, j) in memo:                # déjà calculé
        return memo[(i, j)]
    if i == 0 and j == 0:             # case de départ
        res = ...
    elif i == 0:                      # première ligne
        res = ...
    elif j == 0:                      # première colonne
        res = ...
    else:                             # cas général
        res = ...
    memo[(i, j)] = res
    return res

grille = [[1, 3, 2, 1],
          [2, 1, 0, 1],
          [9, 5, 1, 2]]
print(robot(grille, 2, 3, {}))

```

5. (Bonus) À partir des valeurs  $G(i, j)$ , retrouver le chemin optimal en partant de l'arrivée et en « remontant ».

## 2 Introduction aux bases de données et au langage SQL

Une **base de données** est un ensemble organisé d'informations, géré par un logiciel appelé **SGBD** (système de gestion de bases de données) qui en assure le stockage, la cohérence et l'accès simultané par de nombreux utilisateurs. Dans le **modèle relationnel**, les données sont rangées dans des **tables** (ou relations) : chaque **colonne** est un attribut avec un type (entier, texte...), chaque **ligne** est un enregistrement, et une **clé primaire** identifie chaque ligne de façon unique. Pour interroger la base, on utilise le langage **SQL** (*Structured Query Language*) : on décrit *ce que l'on veut obtenir*, et c'est le SGBD qui décide *comment* le calculer.

### La table livres (extrait)

| id  | titre                    | auteur                   | annee | genre           | emprunts |
|-----|--------------------------|--------------------------|-------|-----------------|----------|
| 1   | Les Misérables           | Victor Hugo              | 1862  | roman           | 42       |
| 2   | Notre-Dame de Paris      | Victor Hugo              | 1831  | roman           | 27       |
| 7   | Le Comte de Monte-Cristo | Alexandre Dumas          | 1844  | aventure        | 38       |
| 13  | Le Petit Prince          | Antoine de Saint-Exupéry | 1943  | conte           | 56       |
| 17  | Frankenstein             | Mary Shelley             | 1818  | science-fiction | 26       |
| ... | ...                      | ...                      | ...   | ...             | ...      |

La table complète contient 18 livres. La clé primaire est **id**.

### La requête SELECT en exemples

Exemple 1 : choisir des colonnes avec **SELECT ... FROM ...**

```
SELECT titre, auteur FROM livres;
```

Résultat :

| titre               | auteur                   |
|---------------------|--------------------------|
| Les Misérables      | Victor Hugo              |
| Notre-Dame de Paris | Victor Hugo              |
| Le Rouge et le Noir | Stendhal                 |
| ...                 | ... (18 lignes au total) |

On écrit après **SELECT** les colonnes voulues, séparées par des virgules, et après **FROM** le nom de la table. **SELECT \* FROM livres;** affiche **toutes** les colonnes.

### Exemple 2 : filtrer les lignes avec WHERE

```
SELECT titre, annee FROM livres WHERE auteur = 'Jules Verne';
```

Résultat :

| titre                            | annee |
|----------------------------------|-------|
| Vingt mille lieues sous les mers | 1870  |
| Le Tour du monde en 80 jours     | 1872  |

Seules les lignes qui vérifient la condition sont gardées. Les textes s'écrivent entre apostrophes. On dispose des comparaisons =, <> (différent), <, <=, >, >=, que l'on peut combiner avec AND, OR et NOT :

```
SELECT titre, annee FROM livres WHERE genre = 'roman' AND annee > 1900;
```

Résultat :

| titre      | annee |
|------------|-------|
| L'Étranger | 1942  |
| La Peste   | 1947  |

### Exemple 3 : renommer une colonne avec AS

```
SELECT titre AS livre, emprunts AS nb_emprunts
FROM livres
WHERE auteur = 'Victor Hugo';
```

Résultat :

| livre               | nb_emprunts |
|---------------------|-------------|
| Les Misérables      | 42          |
| Notre-Dame de Paris | 27          |

AS ne modifie pas la table : il change seulement le nom de la colonne *dans le résultat*. Une requête peut s'écrire sur plusieurs lignes ; c'est le point-virgule qui marque la fin.

### Exemple 4 : compter les lignes avec COUNT(\*)

```
SELECT COUNT(*) AS nb_zola FROM livres WHERE auteur = 'Émile Zola';
```

Résultat :

| nb_zola |
|---------|
| 2       |

COUNT(\*) renvoie le **nombre de lignes** sélectionnées (ici, après le filtre WHERE). Sans WHERE, **SELECT COUNT(\*) FROM livres;** renvoie 18. On l'utilise pour l'instant comme une « formule

toute faite » : nous verrons plus tard comment cela fonctionne.

### Exemple 5 : trier avec ORDER BY

```
SELECT titre, annee FROM livres
WHERE auteur = 'Émile Zola' OR auteur = 'Jules Verne'
ORDER BY annee;
```

Résultat :

| titre                            | annee |
|----------------------------------|-------|
| Vingt mille lieues sous les mers | 1870  |
| Le Tour du monde en 80 jours     | 1872  |
| L'Assommoir                      | 1877  |
| Germinal                         | 1885  |

Le tri est croissant par défaut ; on écrit ORDER BY annee DESC pour un tri décroissant.

### Exemple 6 : limiter le résultat avec LIMIT et OFFSET

```
SELECT titre, emprunts FROM livres ORDER BY emprunts DESC LIMIT 3;
```

Résultat :

| titre           | emprunts |
|-----------------|----------|
| Le Petit Prince | 56       |
| 1984            | 51       |
| L'Étranger      | 47       |

LIMIT 3 ne garde que les 3 premières lignes : ce sont donc les 3 livres les plus empruntés. OFFSET permet de **sauter** des lignes avant de commencer :

```
SELECT titre, emprunts FROM livres ORDER BY emprunts DESC LIMIT 3 OFFSET 3;
```

Résultat :

| titre                            | emprunts |
|----------------------------------|----------|
| Les Misérables                   | 42       |
| Le Comte de Monte-Cristo         | 38       |
| Vingt mille lieues sous les mers | 35       |

On obtient les livres classés 4<sup>e</sup>, 5<sup>e</sup> et 6<sup>e</sup>.

### À retenir

Les clauses s'écrivent toujours dans cet ordre (seules SELECT et FROM sont obligatoires) :

```
SELECT ... FROM ... WHERE ... ORDER BY ... LIMIT ... OFFSET ...;
```

### 3 À vous de jouer sur machine

Où travailler ? Ouvrir l'activité SQL : <https://capytale2.ac-paris.fr/web/c/3503-12030913/mcer>.

#### Exercice 2 : interroger la bibliothèque

Pour chaque question, écrire la requête SQL, l'exécuter et noter le résultat obtenu.

1. Afficher tout le contenu de la table `livres`.
2. Afficher uniquement le titre et l'auteur de chaque livre.
3. Afficher les titres des livres de Victor Hugo.
4. Afficher les titres des livres parus avant 1850.
5. Afficher les titres et années des livres parus entre 1850 et 1900 (inclus).
6. Afficher le titre et le nombre d'emprunts des livres d'Albert Camus, en nommant les colonnes `ouvrage` et `popularite`.
7. Combien la bibliothèque possède-t-elle de livres ? Nommer la colonne `total`.
8. Combien la bibliothèque possède-t-elle de livres d'aventure ?
9. Afficher les titres et années de tous les livres, du plus récent au plus ancien.
10. Afficher les 5 livres les plus anciens.
11. Afficher le titre du livre le plus emprunté.
12. (*Bonus*) Afficher les livres classés 4<sup>e</sup>, 5<sup>e</sup> et 6<sup>e</sup> parmi les *moins* empruntés.