

Programmation dynamique

Rendu de monnaie

Christophe Viroulaud

Terminale - NSI

Algo 25

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Pour répondre à un problème on peut utiliser plusieurs stratégies algorithmiques selon l'objectif à atteindre :

- ▶ trouver une solution exacte même si cela prend un temps long,
- ▶ trouver une solution approchée mais plus rapidement.

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Peut-on trouver une solution optimale en un temps
raisonnable ?

1. Le rendu de monnaie

2. Approche gloutonne

3. Approche dynamique

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Définition

Le problème du rendu de monnaie consiste à minimiser le nombre de pièces à rendre pour une somme donnée. On dispose d'un système de monnaie (exemple : système européen).

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante (top-down)
Approche ascendante (bottom-up)

1. Le rendu de monnaie

2. Approche gloutonne

2.1 Principe

2.2 Algorithme

2.3 Optimalité

3. Approche dynamique

Le rendu de
monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

À retenir

La stratégie gloutonne est une méthode approchée de résolution. Elle consiste à faire un choix de résolution et ne pas revenir dessus.

Dans le problème du rendu de monnaie, le choix est fait de rendre d'abord **la plus grande pièce possible**.

Le rendu de monnaie

Approche gloutonne

Principe

Algorithme

Optimalité

Approche dynamique

Algorithme naïf

Approche descendante
(top-down)

Approche ascendante
(bottom-up)

1. Le rendu de monnaie

2. Approche gloutonne

2.1 Principe

2.2 Algorithme

2.3 Optimalité

3. Approche dynamique

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Activité 1 : Le système monétaire est donné dans l'ordre décroissant :

```
1 systeme = [10, 5, 2, 1]
```

1. Écrire la fonction itérative
`rendu_monnaie(somme: int, systeme: list)`
→ `list` qui renvoie la liste des pièces à rendre pour rembourser `somme`.
2. Écrire la fonction récursive équivalente.

```
1 def rendu_glouton(somme: int, systeme: list) -> list:
2     res = []
3     i_piece = 0
4     while somme > 0:
5         # si pièce est trop grande
6         if systeme[i_piece] > somme:
7             # on avance dans le système
8             i_piece += 1
9         else:
10            res.append(systeme[i_piece])
11            somme -= systeme[i_piece]
12
13     return res
```

```
1 >>> rendu_glouton(14, [10, 5, 2, 1])
2 [10, 2, 2]
```

Le rendu de
monnaie

Approche
gloutonne

Principe

Algorithme

Optimalité

Approche
dynamique

Algorithme naïf

Approche descendante
(top-down)

Approche ascendante
(bottom-up)

Le rendu de
monnaie

Approche
gloutonne

Principe

Algorithme

Optimalité

Approche
dynamique

Algorithme naïf

Approche descendante
(top-down)

Approche ascendante
(bottom-up)

```
1 def rendu_glouton_rec(somme: int, systeme: list,  
2   i_piece: int) -> list:  
3     if somme > 0:  
4       # si pièce est trop grande  
5       if systeme[i_piece] > somme:  
6         # on avance dans le système  
7         i_piece += 1  
8         return rendu_glouton_rec(somme, systeme,  
9   i_piece)  
10      else:  
11        # on rend la pièce  
12        somme -= systeme[i_piece]  
13        return [systeme[i_piece]] +  
14        rendu_glouton_rec(somme, systeme, i_piece)  
15  
16     else:  
17       # cas limite  
18       return []
```

```
1 >>> rendu_glouton_rec(14, [10, 5, 2, 1], 0)  
2 [10, 2, 2]
```

Le rendu de
monnaie

Approche
gloutonne

Principe

Algorithme

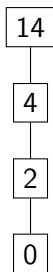
Optimalité

Approche
dynamique

Algorithme naïf

Approche descendante
(top-down)

Approche ascendante
(bottom-up)



1. Le rendu de monnaie

2. Approche gloutonne

2.1 Principe

2.2 Algorithme

2.3 Optimalité

3. Approche dynamique

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

```
1 systeme = [30, 24, 12, 6, 3, 1]
```

Code 1 – Système monétaire impérial britannique

Activité 2 : Dérouler à la main l'exécution de la fonction `rendu_monnaie` pour 48€ avec le système impérial.



FIGURE 1 – Solution donnée par l'algorithme glouton.

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante (top-down)
Approche ascendante (bottom-up)

Une meilleure solution

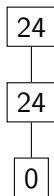


FIGURE 2 – Une meilleure solution.

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante (top-down)
Approche ascendante (bottom-up)

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Remarque

Le système monétaire européen garantit une solution optimale avec l'algorithme glouton. Ce système est dit **canonique**.

1. Le rendu de monnaie
2. Approche gloutonne
3. **Approche dynamique**
 - 3.1 Algorithme naïf
 - 3.2 Approche descendante (top-down)
 - 3.3 Approche ascendante (bottom-up)

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante (top-down)
Approche ascendante (bottom-up)

À retenir

Pour être certain de trouver la solution optimale, il faut calculer toutes les possibilités.

Le principe d'optimalité de Bellman affirme qu'une solution optimale d'un problème s'obtient en combinant des solutions optimales à des sous-problèmes.

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf

Approche descendante
(top-down)

Approche ascendante
(bottom-up)

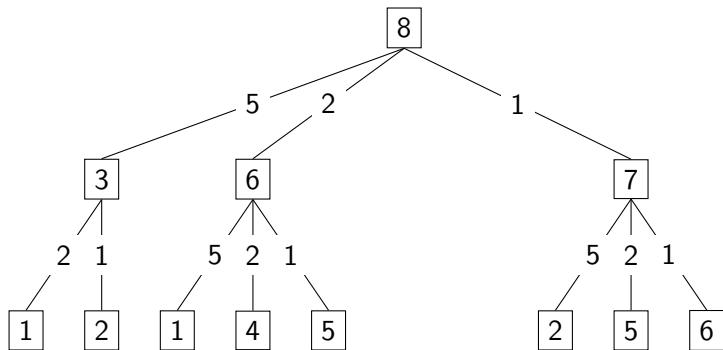


FIGURE 3 – Rendre 8€

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante (top-down)
Approche ascendante (bottom-up)

Le rendu de monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf

Approche descendante
(top-down)
Approche ascendante
(bottom-up)

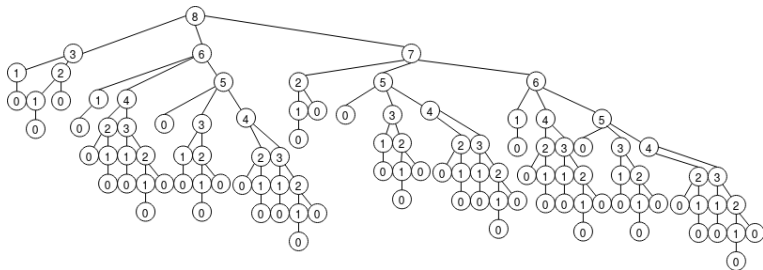


FIGURE 4 – Appels récursifs pour 8€

```

1 def rendu_naif(somme: int, systeme: list) -> list:
2     if somme > 0:
3         # somme = 1 + 1 + 1 ...
4         mini = [1 for _ in range(somme)]
5         # Teste tous les possibles pour chaque pièce
6         for piece in systeme:
7             if piece <= somme:
8                 pieces = [piece] + \
9                     rendu_naif(somme-piece, systeme)
10                if len(pieces) < len(mini):
11                    mini = pieces
12            # garde le rendu minimum
13            return mini
14 else:
15     return []

```

rendu de
monnaie

approche
bottom-up

principe
algorithme
optimalité

approche
dynamique

algorithme naïf

approche descendante
(top-down)

approche ascendante
(bottom-up)

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf

Approche descendante
(top-down)

Approche ascendante
(bottom-up)

Activité 3 :

1. Dérouler le début de l'algorithme à la main pour en comprendre le fonctionnement.
2. Ajouter une variable globale **COMPTEUR** qui compte le nombre d'appels récursifs.

```
1 def rendu_naif(somme: int, systeme: list) -> list:
2     global C
3     if somme > 0:
4         mini = [1 for _ in range(somme)]
5         for piece in systeme:
6             if piece <= somme:
7                 C += 1
8                 pieces = [piece] + \
9                     rendu_naif(somme-piece, systeme)
10                if len(pieces) < len(mini):
11                    mini = pieces
12            # garde le rendu minimum
13        return mini
14    else:
15        return []
```

Le rendu de
monnaie

approche
bottom-up

principe
algorithme
optimalité

approche
dynamique

algorithme naïf

approche descendante
(top-down)

approche ascendante
(bottom-up)

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf

Approche descendante
(top-down)
Approche ascendante
(bottom-up)

```
1 >>> C = 0
2 >>> rendu_naif(14, [10, 5, 2, 1])
3 [10, 2, 2]
4 >>> C
5 2651
```

1. Le rendu de monnaie
2. Approche gloutonne
3. **Approche dynamique**
 - 3.1 Algorithme naïf
 - 3.2 **Approche descendante (top-down)**
 - 3.3 Approche ascendante (bottom-up)

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante (top-down)
Approche ascendante (bottom-up)

Activité 4 :

1. Écrire la fonction `rendu_td(somme: int, systeme: list, track: list) → list` qui s'inspire de la version naïve mais maintient un tableau `track` des solutions optimales pour chaque somme à rendre. Le tableau sera initialisé par :

```
1 track = [[] for _ in range(somme+1)]
```

2. Ajouter une variable globale `COMPTEUR` qui compte le nombre d'appels récursifs.

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

```

1 def rendu_td(somme: int, systeme: list, track: list) ->
  list:
2     # le choix de pièces a déjà été calculé
3     if len(track[somme]) > 0:
4         return track[somme]
5     elif somme > 0:
6         mini = [1 for _ in range(somme)]
7         # Teste tous les possibles pour chaque pièce
8         for piece in systeme:
9             if piece <= somme:
10                pieces = [piece] + \
11                    rendu_td(somme-piece, systeme, track)
12                if len(pieces) < len(mini):
13                    mini = pieces
14            # garde le rendu minimum
15            track[somme] = mini
16            return mini
17     else:
18         return []

```

endu de
naie

roche
tonne

ipe
ithme
nalité

roche
mique

ithme naïf
oche descendante
(down)
oche ascendante
(om-up)

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf

**Approche descendante
(top-down)**

Approche ascendante
(bottom-up)

```
1 >>> track = [[] for _ in range(s+1)]
2 >>> rendu_td(s, [10, 5, 2, 1], track)
3 [10, 2, 2]
```

Code 2 – Appel

```
1 >>> C = 0
2 >>> track = [[] for _ in range(s+1)]
3 >>> rendu_td(s, [10, 5, 2, 1], track)
4 [10, 2, 2]
5 >>> C
6 42
```

Code 3 – Nombre d'appels récursifs

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf

Approche descendante
(top-down)

Approche ascendante
(bottom-up)

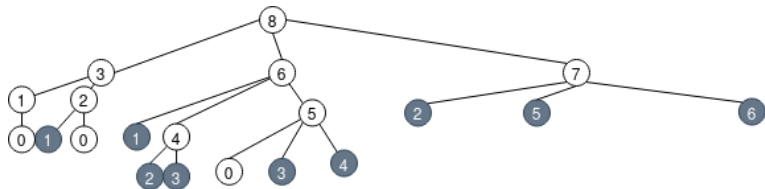


FIGURE 5 – Approche dynamique pour 8€

1. Le rendu de monnaie
2. Approche gloutonne
3. **Approche dynamique**
 - 3.1 Algorithme naïf
 - 3.2 Approche descendante (top-down)
 - 3.3 **Approche ascendante (bottom-up)**

Le rendu de monnaie

Approche gloutonne

Principe
Algorithme
Optimalité

Approche dynamique

Algorithme naïf
Approche descendante (top-down)
Approche ascendante (bottom-up)

Activité 5 :

1. Écrire la fonction `rendu_bu(somme: int, systeme: list) → list` qui complète un tableau `track` des solutions optimales, en le remplissant d'abord par les plus petites sommes à rendre. Le tableau sera initialisé par :

```
1 track = [[] for _ in range(somme+1)]
```

2. Ajouter une variable globale `COMPTEUR` qui compte le nombre d'appels récursifs.

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

```
1 def rendu_bu(somme: int, systeme: list) -> list:
2     track = [[] for _ in range(somme+1)]
3     # pour chaque pièce de track on cherche le
    nombre minimum de pièces à rendre
4     for x in range(1, somme+1):
5         # on ne rend que des pièces de 1
6         mini = [1 for _ in range(x)]
7         for piece in systeme:
8             if piece <= x:
9                 # prend la solution optimale pour 'x
    -piece'
10                pieces = [piece] + track[x-piece]
11                if len(pieces) < len(mini):
12                    mini = pieces
13            track[x] = mini
14
15     return track[somme]
```

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
Approche ascendante
(bottom-up)

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)
**Approche ascendante
(bottom-up)**

```
1 >>> rendu_bu(s, [10, 5, 2, 1])  
2 [10, 2, 2]
```

Code 4 – Appel

```
1 track = [[], [1], [2], [2, 1], [2, 2], [5], [5, 1],
           [5, 2], [], [], [], [], [], [], []]
```

Code 5 – État de `track` au début de l'itération `x = 8`

```
1 for piece in systeme:
2     if piece <= x:
3         # prend la solution optimale pour 'x-piece'
4         pieces = [piece] + track[x-piece]
5         if len(pieces) < len(mini):
6             mini = pieces
```

Code 6 – Pour `piece = 5`, la variable `pieces = [5] + [2, 1]`

Le rendu de
monnaie

Approche
gloutonne

Principe
Algorithme
Optimalité

Approche
dynamique

Algorithme naïf
Approche descendante
(top-down)

**Approche ascendante
(bottom-up)**

```
1 >>> C = 0
2 >>> rendu_bu(s, [10, 5, 2, 1])
3 [10, 2, 2]
4 >>> C
5 42
```

Code 7 – Nombre d'appels récurifs