Mise en évidence du problème

> Programmation dynamique

Principe

Approche top-down
Approche bottom-up

Christophe Viroulaud

Programmation dynamique suite de Fibonacci

Terminale - NSI

Algo 24

Mise en évidence du problème

dynamique

Principe

Approche bottom-up

$$F_n = \left\{ \begin{array}{ll} F_0 = 0 & \text{si } n = 0 \\ F_1 = 1 & \text{si } n = 1 \\ F_n = F_{n-1} + F_{n-2} & \text{si } n > 1 \end{array} \right.$$

Comment obtenir un calcul efficace des termes de la suite?

Programmation dynamique suite de Fibonacci

Mise en évidence du problème

Programmation dynamique

Principe

Approche top-down

ptimisation

1. Mise en évidence du problème

2. Programmation dynamique

Mise en évidence du problème

Programmation dynamique

Principe

Approche bottom-up

$$F_n = \begin{cases} F_0 = 0 & \text{si } n = 0 \\ F_1 = 1 & \text{si } n = 1 \\ F_n = F_{n-1} + F_{n-2} & \text{si } n > 1 \end{cases}$$

Activité 1: Écrire la fonction fibo(n: int) \to int qui calcule le terme de rang n de la suite.

Mise en évidence du problème

```
Programmation
dynamique
```

```
Principe
```

Approache bottom-ur

Optimisation

```
def fibo(n: int)->int:
 1
         11 11 11
 3
        calcule le terme de rang n
        de la suite de Fibonacci
 4
         11 11 11
 5
        if n == 0:
 6
 7
             return 0
        elif n == 1:
 8
 9
             return 1
        else:
10
             return fibo(n-1) + fibo(n-2)
11
```

Mise en évidence du problème

Programmation lynamique

Principe

Approche bottom-up

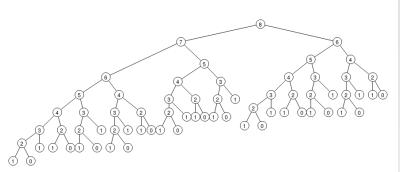


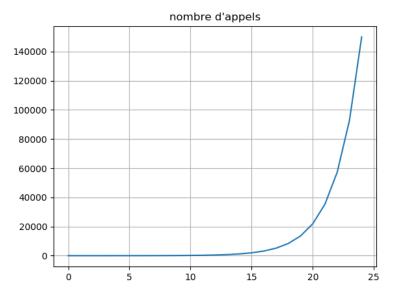
FIGURE 1 - Pour n = 8

n évidence blème

mmation ique

e top-down

bottom-up



 $\operatorname{Figure} 2$ – Nombre d'appels en fonction de n

Mise en évidence du problème

Programmation dynamique

Principe

Approche top-down Approche bottom-up

Remarque

La fonction calcule plusieurs fois la même valeur. Le nombre d'appels augmente de manière exponentielle.

Programmation dynamique suite de Fibonacci

Mise en évidence du problème

Programmation dynamique

Principe
Approche top-down
Approche bottom-up
Optimisation

- 1. Mise en évidence du problème
- 2. Programmation dynamique
- 2.1 Principe
- 2.2 Approche top-dowr
- 2.3 Approche bottom-up
- 2.4 Optimisation

Programmation dynamique - Principe

La programmation dynamique :

- s'appuie sur le principe de diviser pour régner,
- stocke les résultats intermédiaires pour éviter de les calculer à nouveau.

Programmation dynamique suite de Fibonacci

Mise en évidence du problème

Programmation dynamique

Principe

Approache top-down
Approache bottom-up
Optimisation

Mise en évidence du problème

Programmation dynamique

Principe

Approche top-down
Approche bottom-up
Optimisation

Remarque

Richard Bellman développe cette stratégie algorithmique dès les années 1950.

Mise en évidence du problème

dynamique

Principe

Approche top-down
Approche bottom-up
Optimisation

Remarque

En toute rigueur, on n'applique pas tout à fait le principe **diviser pour régner** dans la suite de Fibonacci : les problèmes ne sont pas indépendants.

Programmation dynamique suite de Fibonacci

1. Mise en évidence du problème

- 2. Programmation dynamique
- 2.1 Principe
- 2.2 Approche top-down
- 2.3 Approche bottom-up
- 2.4 Optimisation

Mise en évidence du problème

Programmation dynamique Principe

Approche top-down
Approche bottom-up

À retenir

Dans l'approche top-down (ou descendante), on :

- applique une méthode récursive,
- stocke les résultats des appels récursifs (mémoïsation).

Mise en évidence du problème

rogrammation ynamique

Approche top-down
Approche bottom-up
Optimisation

```
Programmation dynamique
Principe
Approche top-down
Approche bottom-up
Ontimisation
```

```
def fibo_td(n: int, track: list) -> int:
    # déjà calculé
    if track[n] > 0:
        return track[n]
    if n == 0:
        track[0] = 0
        return track[0]
    elif n == 1:
        track[1] = 1
        return track[1]
    else:
        track[n] = fibo_td(n-1, track) + \
                    fibo td(n-2, track)
        return track[n]
```

6

.0

.3

Code 1 – track stocke les résultats déjà calculés.

```
Programmation
   dynamique
suite de Fibonacci
```

```
Approche top-down
```

```
n = 20
track = [-1 for _ in range(n+1)]
fibo_td(n, track)
```

Code 2 - Appel de la fonction

Approche top-down

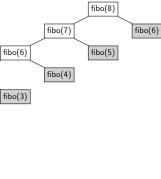


FIGURE 3 – Appels récursifs pour n = 8

fibo(5)

fibo(2)

fibo(4)

fibo(1)

fibo(3)

fibo(0)

fibo(2)

fibo(1)

fibo(6)

nombre d'appels 20 -15 -10 5 0 10 15 20

 $\ensuremath{\mathrm{Figure}}$ 4 – Nombre d'appels en fonction de n

Programmation dynamique suite de Fibonacci

Mise en évidence

rogrammation ynamique

Approache bottom up

Optimisation

Programmation dynamique suite de Fibonacci

Approche bottom-up

- 2. Programmation dynamique

- 2.3 Approche bottom-up
- 2.4 Optimisation

Approche bottom-up

À retenir

Dans l'approche bottom-up (ou ascendante), on :

- applique une méthode itérative,
- résout d'abord les petits problèmes.

```
du problème
Programmation
dynamique
Principe
Approche top-down
Approche bottom-up
```

```
def fibo_bu(n: int) -> int:
    # résultats déjà calculés
    track = [0 for _ in range(n+1)]
    track[1] = 1

# calcule de proche en proche
for i in range(2, n+1):
    track[i] = track[i-1] + track[i-2]

return track[n]
```

1

6

0

Code 3 - track stocke les résultats.

Mise en évidence du problème

rogrammation ynamique Principe

Approche bottom-up
Optimisation

À retenir

La complexité en temps de cette approche est équivalente à celle précédente : on ne calcule chaque sous-problème qu'une seule fois.

Programmation dynamique suite de Fibonacci

Mise en évidence du problème

dynamique

Principe

Approche bottom-up

Optimisation

- 1. Mise en évidence du problème
- 2. Programmation dynamique
- 2.1 Principe
- 2.2 Approche top-down
- 2.3 Approche bottom-up
- 2.4 Optimisation

Optimisation

Remarque

La complexité en espace dépend de la taille du tableau track. Cependant, si seul le résultat final nous intéresse, il est possible d'optimiser l'approche ascendante.

Programmation dynamique suite de Fibonacci

Mise en évidence du problème

Programmation dynamique

Principe

Approche top-down Approche bottom-up

Optimisation

```
du problème
```

```
Principe
Approche top-down
Approche bottom-up
```

```
Optimisation
```

```
def fibo_bu(n: int) -> int:
    track0 = 0
    track1 = 1

for i in range(2, n+1):
    # calcule de proche en proche
    track0, track1 = track1, track0 + track1

return track1
```

4

6

9

Code 4 - On ne garde en mémoire que les deux dernières valeurs.