Suite de Fibonacci

Christophe Viroulaud

Terminale NSI

Suite de Fibonacci

Problématique

Mise en évidence du problème

Programmation dynamique

Approche top-down

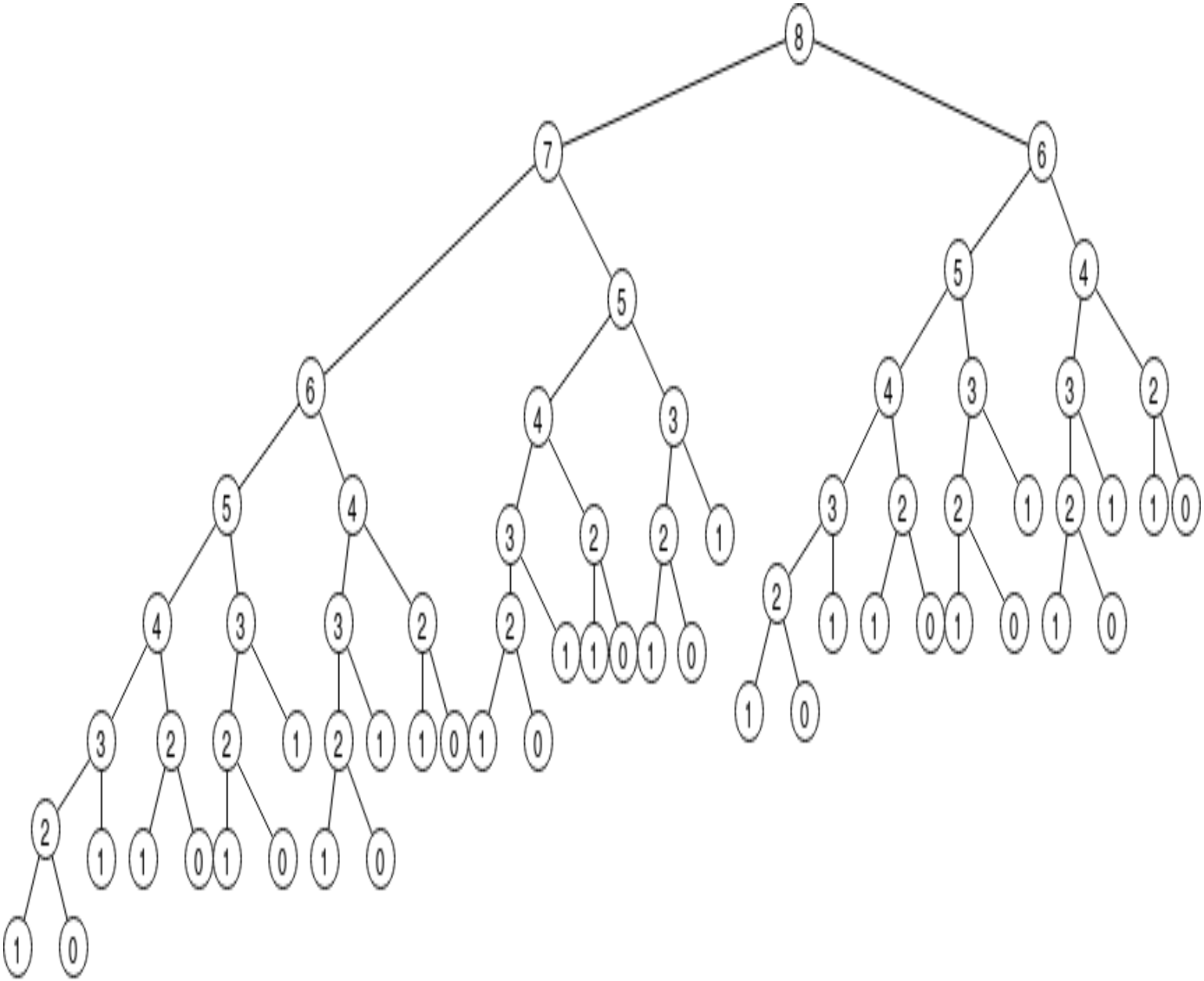
Bottom-up

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

Comment obtenir un calcul efficace des termes de la suite ?

Problématique
Mise en évidence du problème
Programmation dynamique
Approche top-down
Bottom-up

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


8
7 6
6 5 5 4
5 4 4 3 4 3 3 2
4 3 3 2 3 2 2 1 3 2 2 1 2 1 1 0
3 2 2 1 2 1 1 0 2 1 1 0 1 0 2 1 1 0 1 0 1 0
2 1 1 0 1 0 1 0 1 0
1 0

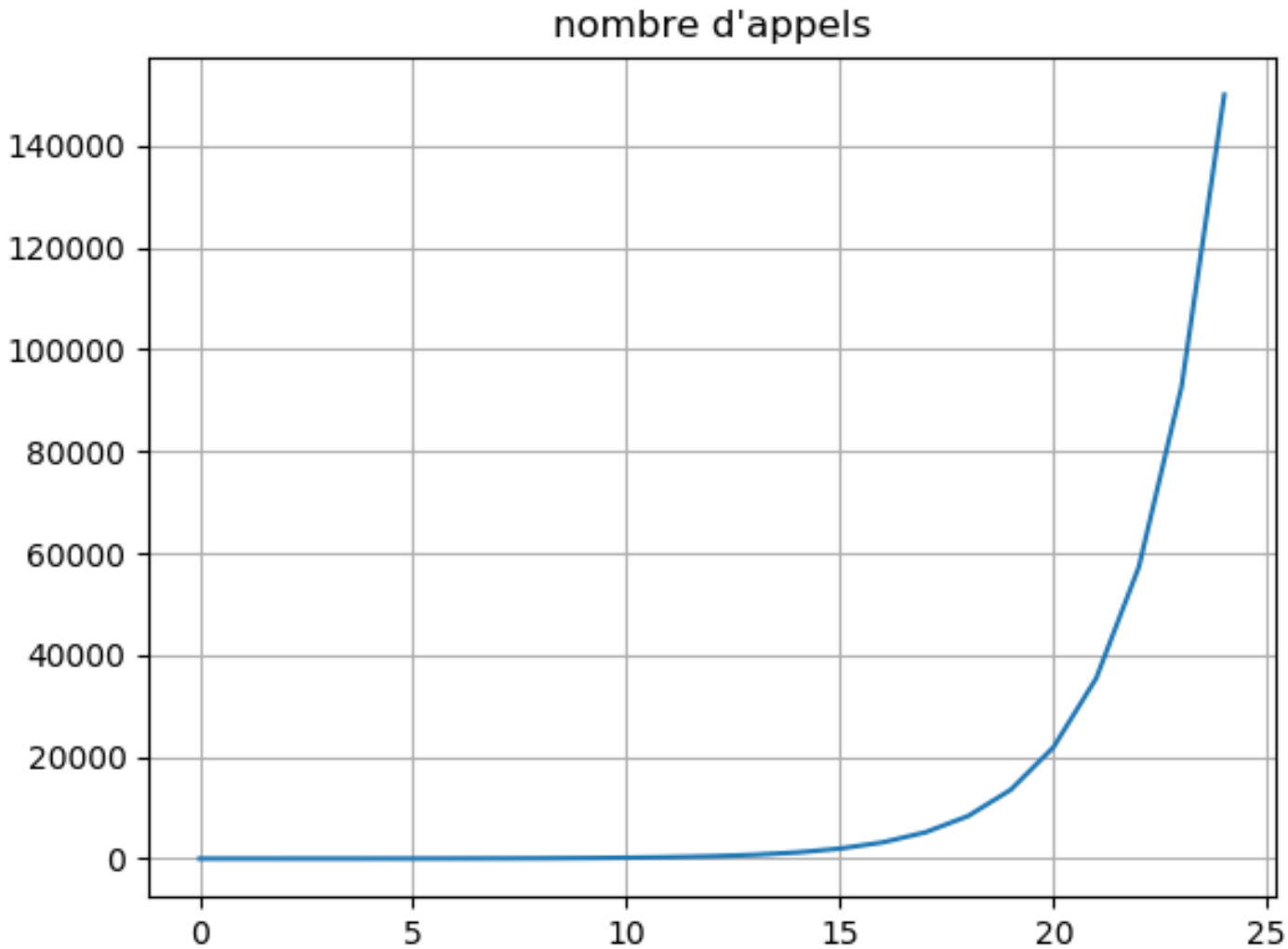
Activité 1 :

1. Tester la fonction *fibo* pour $n = 8$, $n = 10$.
2. À l'aide d'une variable globale (c'est mal) *compteur*, observer le nombre d'appels réalisés en fonction de n.

Problématique
Mise en évidence du problème
Programmation dynamique
Approche top-down
Bottom-up

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

Nombre d'appels en fonction de n



Problématique

Mise en évidence du problème

Programmation dynamique

Approche top-down

Bottom-up

Une amélioration de l'approche *diviser pour régner*

Problématique
Mise en évidence du problème
Programmation dynamique
Approche top-down
Bottom-up

```
1  def fibo_top_down(n: int, track: list )->int:
2      """
3      calcule le terme de rang n
4      de la suite de Fibonacci
5      """
6      if track[n] > 0 :
7          return track[n]
8      if n == 0 :
9          track[0] = 0
10         return track[0]
11     elif n == 1 :
12         track[1] = 1
13         return track[1]
14     else :
15         track[n] = fibo_top_down(n-1, track) +
               fibo_top_down(n-2, track)
16         return track[n]
```

Appel de la fonction

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

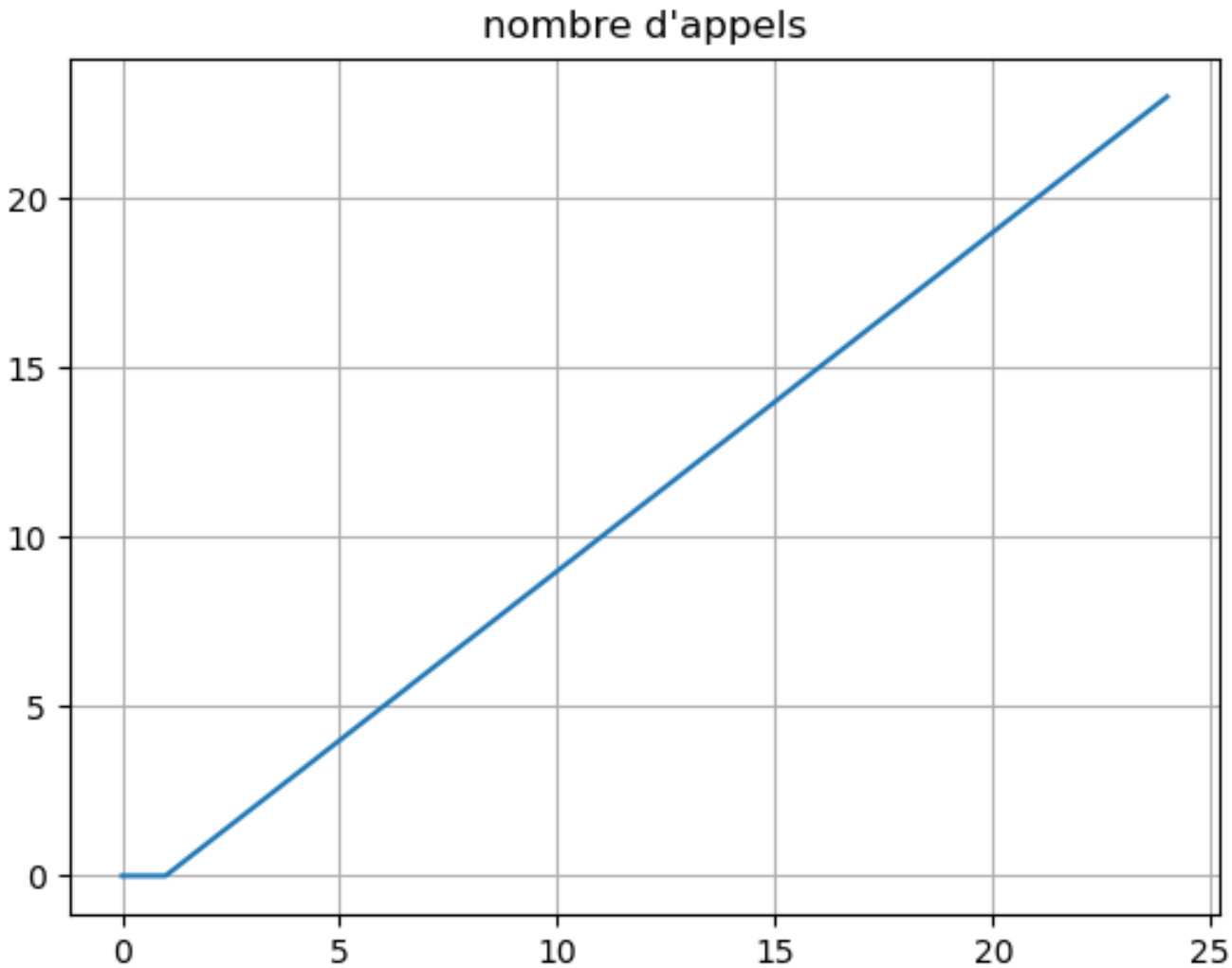
À retenir

La *mémoïsation* consiste à la *mise en cache* les valeurs déjà calculées pour pouvoir être réutilisées.

Problématique
Mise en évidence du problème
Programmation dynamique
Approche top-down
Bottom-up

Activité 2 : Tester la fonction en approche top-down. Compter le nombre d'appels.

Nombre d'appels en fonction de n



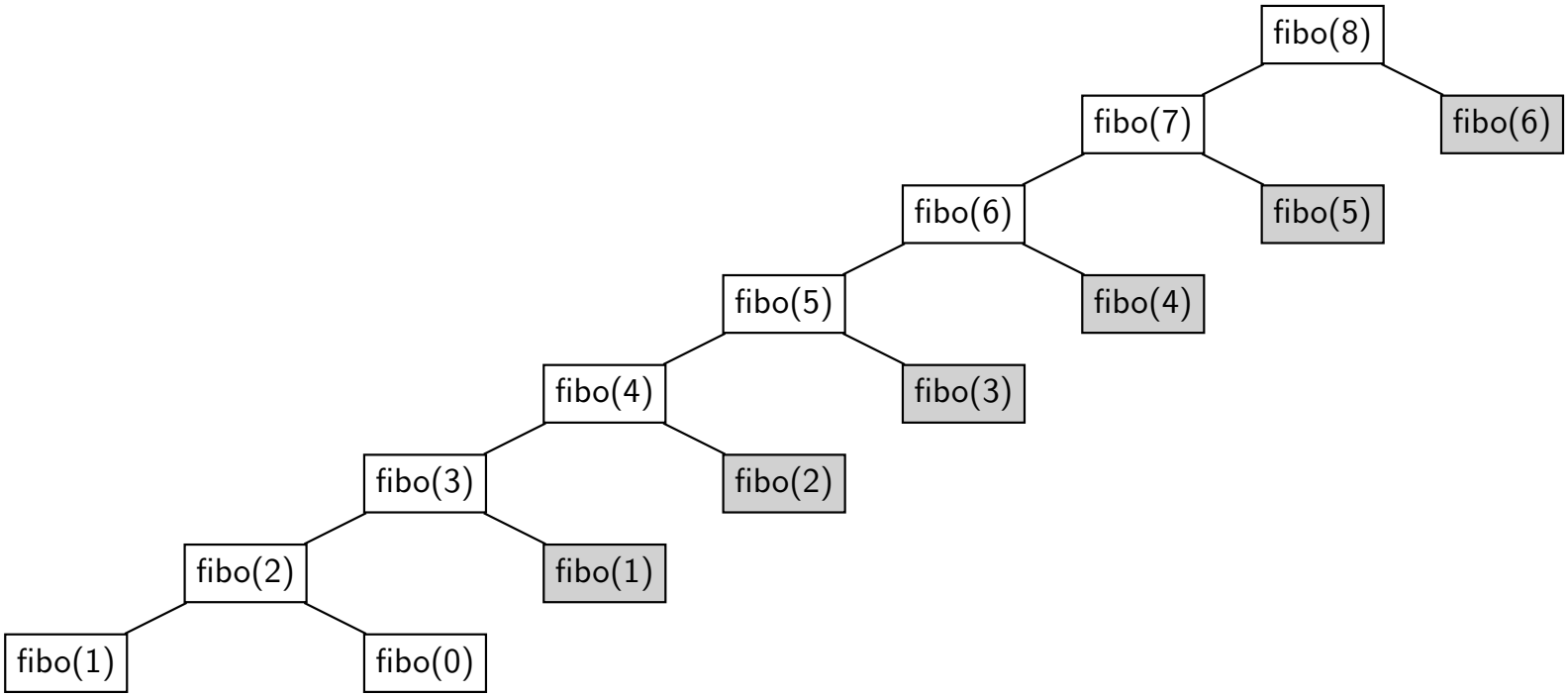


FIGURE – Appels récursifs pour $n = 8$

Problématique

Mise en évidence du problème

Programmation dynamique

Approche top-down

Bottom-up

Approche *itérative* qui résout d'abord les sous-problèmes.

```
1  def fibo_bottom_up(n: int)->int:
2      track = [0 for _ in range(n+1)]
3      track[1] = 1
4      for i in range(2, n+1):
5          track[i] = track[i-1] + track[i-2]
6      return track[n]
```

Code 1 – Approche bottom-up

Problématique
Mise en évidence du problème
Programmation dynamique
Approche top-down
Bottom-up

Activité 3 : Combien d'itérations effectue-t-on ?

Top-down ou Bottom-up ?

- Complexité en temps souvent équivalente,

Top-down ou Bottom-up ?

- Complexité en temps souvent équivalente,
- Complexité en espace peut être optimisée : s'il est simple de déterminer quels résultats vont être nécessaires, l'approche bottom-up est intéressante.

Suite de Fibonacci

Problématique

Mise en évidence du problème

Programmation dynamique

Approche top-down

Bottom-up

Approche bottom-up optimisée en espace

```
1  def fibo_bottom_up2(n: int)->int:
2      track0 = 0
3      track1 = 1
4      for i in range(2, n+1):
5          track0, track1 = track1, track0 + track1
6      return track1
```