AL04 – Programmation dynamique

1. Retour sur la suite de Fibonacci

Il est possible de coder de manière récursive les termes de la suite de Fibonacci:

Code Python

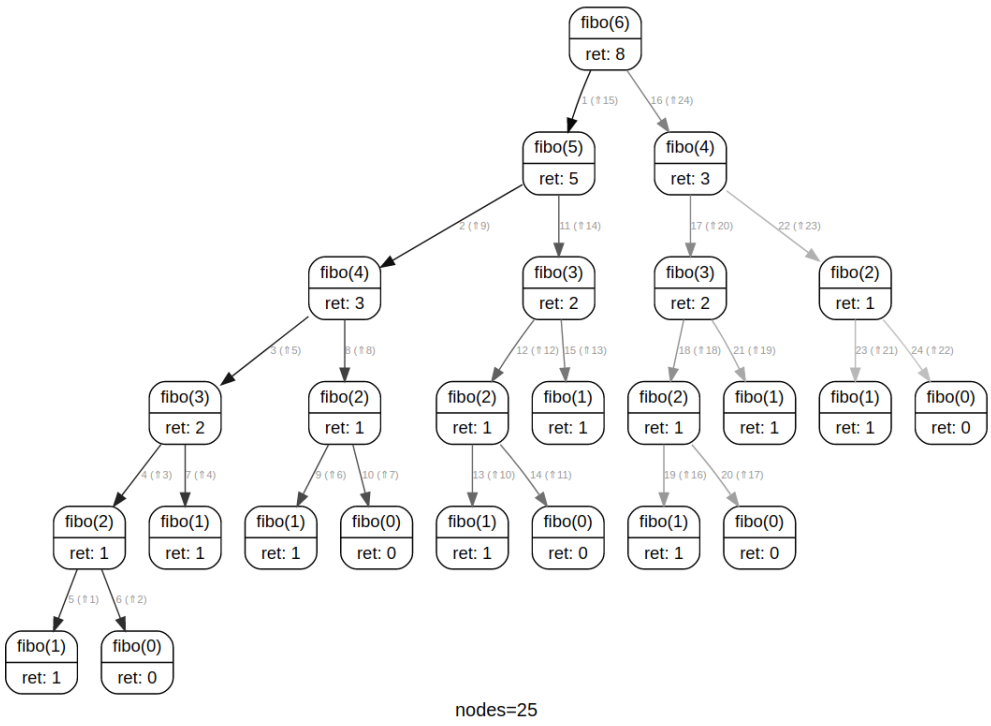
```python
def fibo_rec(n):
    if n == 0 or n == 1:
        return n
    return fibo_rec(n-1) + fibo_rec(n-2)
```

il est possible de mesurer le temps de calcul de `fibo(40)`:

Code Python

```python
import time
t0 = time.time()
fibo(40)
print(time.time() - t0)
```

La cause est la multitude d'appels récursifs qui nous conduit à refaire des calculs déjà effectués. On peut le voir sur l'arbre d'appels de `fibo(6)`:



Le calcul de `fibo(2)` se retrouve 5 fois dans l'arbre. Il est donc possible de pouvoir optimiser cet algorithme.

2. Programmation Top - Bottom

Se souvenir des calculs déjà effectués

On peut se souvenir des résultats par un phénomène de **mémorisation**.
L'idée est de stocker le résultats de chaque calcul, par exemple dans un dictionnaire.

- soit le calcul a déjà été effectué: on a donc juste à le lire dans le dictionnaire;
- soit le calcul n'a jamais été effectué: on l'effectue et on le stocke dans le dictionnaire.

Code Python

```python
def fibonacci_memo(n, memo={0: 0, 1: 1}):
    if n not in memo:
        memo[n] = fibonacci_memo(n - 1, memo) + fibonacci_memo(n - 2, memo)
    return memo[n]
```

Propriété 1: Approche Top - Down: Mémorisation

C'est une approche récursive où on commence par résoudre le problème principal en appelant des sous-problèmes, et on mémorise les résultats pour éviter de les recalculer.

- Utilise la récursion.
- Stocke les résultats intermédiaires dans un cache (dictionnaire par exemple).
- Peut souffrir d'un trop grand nombre d'appels récursifs.

Le principe est donc:

- On commence par `fibonacci(6)`, qui appelle `fibonacci(5)` et `fibonacci(4)`, etc.
- Dès qu'un calcul est fait, il est stocké pour éviter de le refaire.
- L'appel récursif descend jusqu'à `fibonacci(0)` et `fibonacci(1)`, puis remonte en réutilisant les valeurs mémorisées.

Exercice 1: Comparer le temps d'exécution de la méthode Top Bottom

3. Programmation Bottom - Up

Définition 1: Programmation Bottom - Up

C'est une approche itérative où on commence par résoudre les plus petits sous-problèmes en premier, puis on construit la solution du problème global à partir de ceux-ci.

- Utilise une boucle au lieu de la récursion.
- Évite la surcharge de la pile d'appels.
- Généralement plus efficace en termes de mémoire et de performance.

Le principe est donc:

- On commence par remplir un tableau en partant de `fib[0]` et `fib[1]`, puis on calcule tous les termes jusqu'à `fib[n]`.
- On évite la récursion, donc pas de problème de dépassement de pile.

Exercice 2: Fibonacci en Bottom - Up. Compléter et comparer le temps d'exécution

Code Python

```
def fibonacci_bottom_up(n):
    dict_fibo = {0:0, 1:1}
    for k in range(..... , ......):
        dict_fibo[k] = ..............................
    return ....................................
```

4. Méthode sans mémorisation

Pour calculer les valeurs de la suite de Fibonacci il suffit finalement de connaitre les deux précdentes. Il suffit donc de les sauvegarder dans une liste par exemple.

Code Python

```
def fibonacci_simple(n):
    if n <= 1:
        return n
    return fibonacci_simple_rec(n, 1, [0, 1])
def fibonacci_simple_rec(n, k, liste):
    if k == n:
        return ..................
    return fibonacci_simple_rec(.....................................
```

5. Bilan

Propriété 2: Bilan programmation dynamique

Lors d'un calcul effectué de manière récursive, il peut arriver que de multiples appels récursifs soient identiques. Pour éviter de recalculer plusieurs fois la même chose, on peut stocker les résultats intermédiaires. On appelle cette technique la mémoriation. Cette technique minimise le nombre d'opérations et accélère grandement l'exécution du programme. Le prix à payer est l'utilisation d'une structure de stockage des valeurs intermédiaires, et donc une augmentation de la mémoire utilisée par le programme.

Lors d'un calcul effectué de manière itérative, il est parfois plus simple de commencer par une petite version du problème pour progressivement remonter vers la solution du problème global.