Numérique et Sciences Informatiques
Chapitre X - Programmation dynamique

Classe de terminale

La programmation dynamique est une méthode algorithmique pour résoudre des problèmes d'optimisation. Elle consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits au plus grands en stockant les résultats intermédiaires.

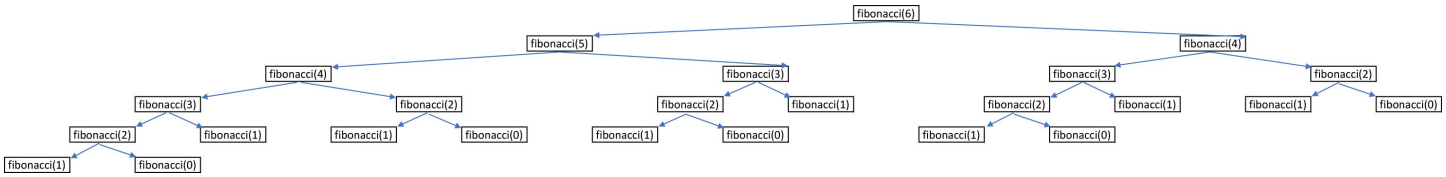
I. Un exemple classique pour bien comprendre : la suite de Fibonacci

On connaît l'algorithme récursif, ici implémenté en Python, de la suite de Fibonacci.

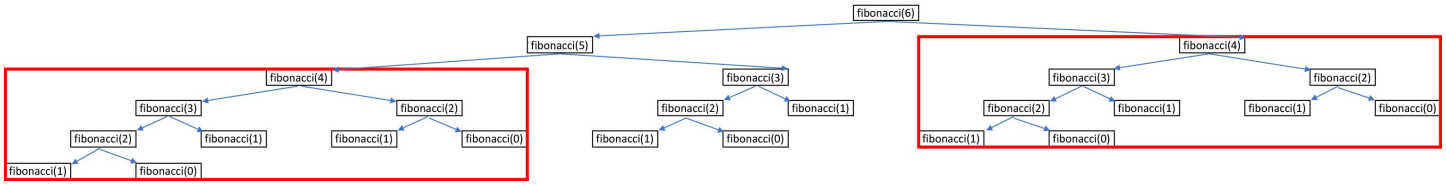
```python
def fibonacci(n):
    if n==0 or n==1:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)
```

Si on veut calculer `fibonacci(7)`, il faut calculer `fibonacci(6)` et `fibonacci(5)`.
Mais pour calculer `fibonacci(6)`, il faut calculer `fibonacci(5)` et `fibonacci(4)`.

On peut représenter cela sous forme de l'arbre suivant :



On s'aperçoit tout de suite que le sous-arbre de droite de `fibonacci(6)` est le même que le sous-arbre de gauche de `fibonacci(5)`.



Il serait donc judicieux de stocker les résultats précédents afin d'éviter des calculs redondants. C'est l'américain Richard Bellman qui a eu cette idée. Evidemment, cette technique n'est utile que si les sous-problèmes sont dépendants les uns des autres. On peut utiliser cette technique soit avec un algorithme récursif, soit avec un algorithme itératif :

- L'algorithme récursif, appelé **Top down** (du haut vers le bas) ou algorithme descendant est le suivant :

```python
def fibonacci(n, mem = None):
    if mem == None:
        mem = [1,1]
    if n >= len(mem):
        mem.append(fibonacci(n - 1, mem) + fibonacci(n - 2, mem))
    return mem[n]
```

  Dans cet algorithme descendant, on stocke les résultats des sous-problèmes dans une variable que l'on renvoie dans l'appel récursif : c'est ce que l'on appelle **mémoïsation**.

- L'algorithme itératif, appelé **Bottom up** (du bas vers le haut) ou algorithme ascendant est le suivant :

```python
def fibonacci2(n):
    mem = [1,1]
    for i in range(2, n + 1):
        mem.append(mem[i - 2] + mem[i - 1])
    return mem[n]
```

Dans cet algorithme ascendant, on commence par calculer des solutions des sous-problèmes les plus petits, puis de proche en proche, on calcule les solutions des sous-problèmes de plus en plus grand pour arriver à une solution du problème.

II. Généralités

La conception d'un algorithme dynamique se décompose en 4 étapes :

- Caractériser la structure d'une solution optimale (pour notre exemple de suite de Fibonacci, il s'agissait d'une liste)
- Définir (souvent de façon récursive) la valeur d'une solution optimale (`if n>= len(mem): mem.append()`)
- Calculer la valeur d'une solution optimale (`fibonacci(n - 2, mem) + fibonacci(n - 1, mem)`)
- Construire une solution optimale à partir des informations calculées (`return mem[n]`)

Comme on utilise les informations des sous-problèmes, le coût de ces algorithmes est en général moindre qu'un algorithme récursif classique.

Nous avons déjà utilisé, dans les chapitres précédents, des algorithmes dynamiques : lors de l'utilisation de la récursivité terminale par exemple.