Algo 25 - Programmation dynamique

Nous avons déjà vu :

→ la force brute _____

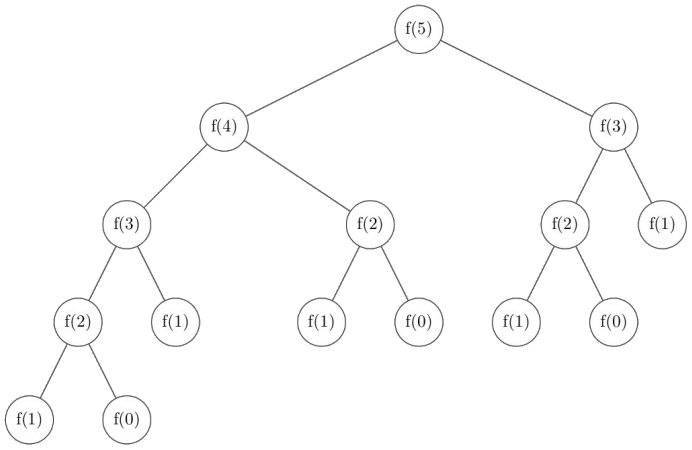
→ la stratégie gloutonne _____

→ diviser pour régner _____

I – Limite des techniques vues jusqu'à présent

Exercice de la file d'attente :

→ **`f(0) = 0`** pour la personne d'indice `0` dans la file.
→ **`f(1) = 1`** pour la personne d'indice 1 juste derrière
→ **`f(x) = f(n-1) + f(n-2)`** pour les suivants



Conclusion :

→ la force brute _____

```python
def f(n:int) -> int:
    '''Renvoie la bonne valeur de la suite de Fibonacci'''
    assert n >= 0

    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return f(n-1) + f(n-2)
```

→ la stratégie gloutonne _____

→ diviser pour régner _____

II – Programmation dynamique

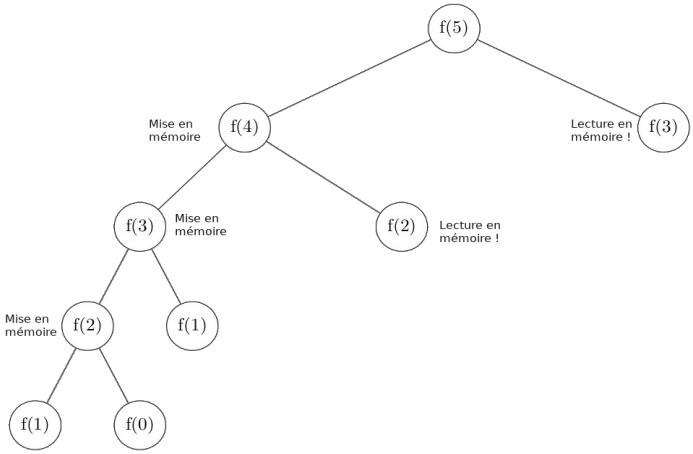
Principe général

La stratégie de la **programmation dynamique** consiste à **planifier l'ordre de résolution des différents résultats intermédiaires et les mémoriser** pour ne jamais refaire deux fois le même calcul.

On se demande quels sont les sous-problèmes à résoudre avant les autres.

On met en mémoire chaque sous-solution dès qu'on l'obtient. Il suffira ensuite d'aller relire le résultat à coût constant plutôt que de relancer le vrai calcul.

Sur notre exemple avec f(5), cela donnerait donc l'arbre d'appels suivant :



```python
def f(n:int) -> int:
    '''Renvoie la bonne valeur f(n) de la suite de Fibonacci'''

    assert n >= 0

    # création de la mémoire-cache
    m = [None for _ in range(n+2)]
    m[0] = 0
    m[1] = 1

    # remplissage de la mémoire-cache
    for k in range(2, n+1):
        m[k] = m[k-1] + m[k-2]

    # reponse
    return m[n]
```

Vocabulaire
Le mot ***Programmer*** s'entend ici sous le sens ***Planifier, Prévoir.*** Il n'y aucun lien avec le mot "coder".

Domaine d'utilisation
La stratégie de la **programmation dynamique** est utilisée lorsqu'on veut résoudre un problème

- dont les sous-problèmes sont **interdépendants** (par exemple f(9) et f(10) vont devoir faire intervenir f(5))
- dont la solution d'un sous-problème doit être utilisée pour résoudre le problème plus global (par exemple la solution de f(9) est utilisée pour trouver la solution de f(10))

III – Problème de la découpe

Une scierie récupère des troncs d'arbre de 10 mètres et plus pour en faire des planches. Voici le prix moyen des planches qu'elle peut vendre actuellement en fonction de la longueur de la planche :

Longueur (m)	1	2	3	4	5	6	7	8	9	10
Prix	1	5	8	9	10	17	17	20	24	30

```python
# Déclaration des constantes
LES_PRIX = [0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30]

# Déclaration des fonctions
def meilleure_vente(prix, taille):
    '''Renvoie le meilleur prix de vente pour une taille donnée'''

    assert len(prix) >= 2
    assert taille < len(prix)

    # Création du cache "meilleure vente"
    m = [valeur for valeur in prix]  # On copie prix pour considérer juste les planches entières au début

    # Meilleure vente pour une planche de 1m, puis 2m... jusqu'à taille
    for planche in range(1, taille+1):

        vente_max = prix[planche]  # 1 - meilleure vente associée à une planche pleine au départ

        # 2 - on cherche la meilleure vente avec découpe ou pas.
        for decoupe in range(1, planche):       # pour chaque découpe initiale possible
            prix_decoupe  = prix[decoupe]        # prix pour une telle decoupe entière
            vente_reste = m[planche - decoupe]  # meilleure vente avec le reste de la planche
            if (prix_decoupe + vente_reste) > vente_max:
                vente_max = prix_decoupe + vente_reste

        m[planche] = vente_max  # 3 - on mémoïse la vente optimale

    return m[taille]  # Meilleur vente pour une planche de la taille voulue

# Programme
print(meilleure_vente(LES_PRIX, 10))
```

IV – Mémoïsation [Hors programme ?]

Parfois, trouver l'ordre des calculs à réaliser n'est pas facile. Le plus simple est alors de faire de la **mémoïsation** :
→ on crée mémoire-cache (une structure mutable) dont on transmet l'adresse à chaque appel.
→ lorsqu'on doit faire un calcul, deux cas se présentent :
→ Si la mémoire contient déjà le calcul : on le lit à coût constant
→ Si la mémoire ne contient pas le calcul : on le mémorise et on le stocke dans la mémoire.