PROJET EULER
Problème 117

Table des matières

Première partie
Résolution du problème

1 Présentation

2 Méthodes de réflexion

2.1 Première méthode (méthode récursive)

2.1.1 Algorithme de principe

```
1
```

2.1.2 Développement

2.2 Seconde méthode (méthode tetranacci)

2.2.1 Algorithme de principe

```
2
```

2.2.2 Développement

Deuxième partie
Optimisation

3 Objectifs

4 Méthode finale

4.1 Algorithme de principe

```
3
```

4.2 Développement

Troisième partie

Comparaison des deux méthodes

5 Comparaison en temps

Afin de comparer l'efficacité de nos deux méthodes nous avons réaliser un tableau représentant le temps d'exécution de nos deux programmes en fonction de la valeur recherchée avec le même $vMax$ à chaque fois :

Ainsi, grâce à ce tableau, nous remarquons qu'à partir du 12ème terme, le temps d'exécution de la première méthode devient plus de 1000 fois supérieur à celui de la seconde. De plus, à partir du 16ème terme, la première méthode devient beaucoup trop long pour être mesuré. L'efficacité de la seconde méthode par rapport à la première est donc avéré puisqu'elle met autant de temps à trouver le 30ème terme que la première pour trouver le 13ème.

Nous avons aussi réaliser un graphique représentant l'évolution de ces temps d'exécution en fonction du terme recherché :

Ici, nous voyons bien que pour les deux méthodes, le temps d'exécution devient exponentiel à partir d'un certain point. Ce dernier est beaucoup plus grand pour la seconde méthode ce qui explique sa meilleure efficacité. Nous poyvons donc en conclure avec ce graphique que la complexité en temps des deux programmes est de l'odre de $O(e^n)$ où n est l'index du terme que l'on recherche. Nous n'avons pas réussi à montrer ceci grâce au calcul car la complexité en temps de nos programmes dépendent de plusieurs éléments à la fois : n, $vMax$, le nombre de chiffres du $n^{\text{ème}}$ terme.

6 Compléxité

Quatrième partie
Annexes

Méthode récursive

```
import time

def recursive(n):
    '''
    :param n: nombre de cases à remplir
    :return: nombre de possibilités pour remplir n-cases
    '''
    if n == 5:
        return 15
    if n == 4:
        return 8
    if n == 3:
        return 4
    if n == 2:
        return 2
    return recursive(n - 1) + recursive(n - 2) + recursive(n - 3) + recursive(n - 4)

nbTeste = int(input("Rentrez le nombre de case disponible : "))
begin = time.time()
print("Résultat au problème 117 avec", nbTeste, "comme entrée :", recursive(nbTeste))
print(f"Duration = {time.time() - begin} seconds to complete.")
```

Méthode tetranacci

```
import time

def tetranacci(n):
    '''
    :param n: nombre de cases à remplir
    :return: nombre de possibilités pour remplir n-cases
    '''
    l = [0,0,0,1]
    for i in range(4,n + 4):
        l.append(l[i - 4] + l[i - 3] + l[i - 2] + l[i - 1])
    return l[n + 3]

nbTeste = int(input("Rentrez le nombre de case disponible : "))
begin = time.time()
print("Résultat au problème 117 avec", nbTeste, "comme entrée :", tetranacci(nbTeste))
print(f"Duration = {time.time() - begin} seconds to complete.")
```

Méthode finale

```python
import time

def finale(n):
    '''
    :param n: nombre de cases à remplir
    :return: nombre de possibilités pour remplir n-cases
    '''
    a, b, c, d = 0, 0, 0, 1
    for _ in range(n):
        a, b, c, d = b, c, d, (a+b+c+d)
    return d

nbTeste = int(input("Rentrez le nombre de case disponible : "))
begin = time.time()
print("Résultat au problème 117 avec", nbTeste, "comme entrée :", finale(nbTeste))
print(f"Duration = {time.time() - begin} seconds to complete.")
```