

PROJET EULER
Problème 117

Table des matières

I	Résolution du problème	2
1	Présentation	2
2	Méthodes de réflexion	2
2.1	Première méthode (méthode récursive)	2
2.1.1	Algorithme de principe	2
2.1.2	Développement	2
2.2	Seconde méthode (méthode tetranacci)	2
2.2.1	Algorithme de principe	2
2.2.2	Développement	2
II	Optimisation	3
3	Objectifs	3
4	Méthode finale	3
4.1	Algorithme de principe	3
4.2	Développement	3
III	Comparaison des deux méthodes	4
5	Comparaison en temps	4
6	Complexité	4
IV	Annexes	5

Première partie

Résolution du problème

1 Présentation

2 Méthodes de réflexion

2.1 Première méthode (méthode récursive)

2.1.1 Algorithme de principe

```
1 Algorithme : Projet Euler, Problème 117, Méthode récursive
2 Type de Retour : Nombre
3 Paramètres : Nombre n
4
5 si n = 5:
6     retourner 15
7 si n = 4:
8     retourner 8
9 si n = 3:
10    retourner 4
11 si n = 2:
12    retourner 2
13
14 retourner Méthode récursive(n - 1) + Méthode récursive(n - 2) + Méthode récursive(n - 3) + Méthode récursive(n - 4)
```

2.1.2 Développement

2.2 Seconde méthode (méthode tetranacci)

2.2.1 Algorithme de principe

```
1 Algorithme : Projet Euler, Problème 117, Méthode tetranacci
2 Type de Retour : Nombre
3 Paramètres : Nombre n
4
5 liste := [0, 0, 0, 1]
6 Pour i allant de 4 à n + 4:
7     ajouter (liste[i-4] + liste[i-3] + liste[i-2] + liste[i-1]) à liste
8
9 retourner liste[n+3]
```

2.2.2 Développement

Deuxième partie

Optimisation

3 Objectifs

4 Méthode finale

4.1 Algorithme de principe

```
1 Algorithme : Projet Euler, Problème 117, Méthode finale
2 Type de Retour : Nombre
3 Paramètres : Nombre n
4
5 a := 0, b := 0, c := 0, d := 1
6 Pour i allant de 0 à n:
7     d := a + b + c + d, a := b, b := c, c := d
8
9 retourner d
```

4.2 Développement

Troisième partie

Comparaison des deux méthodes

5 Comparaison en temps

Afin de comparer l'efficacité de nos deux méthodes nous avons réalisé 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 avérée puisqu'elle met autant de temps à trouver le 30ème terme que la première pour trouver le 13ème.

Nous avons aussi réalisé 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 pouvons donc en conclure avec ce graphique que la complexité en temps des deux programmes est de l'ordre 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 Complexité

Quatrième partie

Annexes

Méthode récursive

```
1 import time
2
3 def recursive(n):
4     '''
5     :param n: nombre de cases à remplir
6     :return: nombre de possibilités pour remplir n-cases
7     '''
8     if n == 5:
9         return 15
10    if n == 4:
11        return 8
12    if n == 3:
13        return 4
14    if n == 2:
15        return 2
16    return recursive(n - 1) + recursive(n - 2) + recursive(n - 3) + recursive(n - 4)
17
18 nbTeste = int(input("Rentrez le nombre de case disponible : "))
19 begin = time.time()
20 print("Résultat au problème 117 avec", nbTeste, "comme entrée :", recursive(nbTeste))
21 print(f"Duration = {time.time() - begin} seconds to complete.")
```

Méthode tetranacci

```
1 import time
2
3 def tetranacci(n):
4     '''
5     :param n: nombre de cases à remplir
6     :return: nombre de possibilités pour remplir n-cases
7     '''
8     l = [0,0,0,1]
9     for i in range(4,n + 4):
10        l.append(l[i - 4] + l[i - 3] + l[i - 2] + l[i - 1])
11    return l[n + 3]
12
13 nbTeste = int(input("Rentrez le nombre de case disponible : "))
14 begin = time.time()
15 print("Résultat au problème 117 avec", nbTeste, "comme entrée :", tetranacci(nbTeste))
16 print(f"Duration = {time.time() - begin} seconds to complete.")
```

Méthode finale

```
1 import time
2
3 def finale(n):
4     '''
5     :param n: nombre de cases à remplir
6     :return: nombre de possibilités pour remplir n-cases
7     '''
8     a, b, c, d = 0, 0, 0, 1
9     for _ in range(n):
10         a, b, c, d = b, c, d, (a+b+c+d)
11     return d
12
13 nbTeste = int(input("Rentrez le nombre de case disponible : "))
14 begin = time.time()
15 print("Résultat au problème 117 avec", nbTeste, "comme entrée :", finale(nbTeste))
16 print(f"Duration = {time.time() - begin} seconds to complete.")
```