16.09.23

TD/TP : création des premiers algorithmes

LE GOURRIEREC Alan

3a Cyberlog

Enseignent : Marteau P.

Unité d'enseignement : Algorithmique et programmation impérative

Durée : ---

Établissement d'enseignement : ENSIBS (Vannes)

Table des matières

Introduction

Durant ce compte rendu, je vais présenter mes avancer dans les divers résultats en termes de programmation et surtout de réflexion à travers des divers exercices présenté. Ceci sera découpé en divers partie et dans ce compte rendu je vais vous présenter les résultats des 2 premiers exercices.

Exercice 1

Dans ce premier exercice, nous devons prouver, à l'aide d'un invariant de boucle, que le programme en pseudo code suivant s'arrête bien :

```
int i <- 1
float sum <- A[0]
while i < n:
   sum <- sum + A[i]
   i <- i + 1
return sum
```

Comme nous le voyons, i notre variable va s'incrémenter de 1 en 1 jusqu'à atteindre n. À ce moment-là, le programme arrêtera d'effectuer la boucle vue que la condition initiale "while i < n:" car il va dépasser cette valeur.

Notre programme va donc belle et bien effectuer la somme des valeurs comprises dans le tableau A[ ] comme demandé dans la consigne

Exercice 2

Durant ce second exercice nous allons observer un programme dans le but de calculer à une certaine puissance une valeur que nous noterons a. Nous allons aussi avoir une variable qui définit la puissance et/ou le nombre de fois que nous devrons réitérer cette opération. Pour se faire, nos variables doivent respecter certaines conditions :

- n doit être un entier naturel
- A doit être un réel

```
float Puissance(float a, int n):
float x <- 1.0
for i <- 1 to n
x <- x * a
return x
```

Nous allons aussi déterminer la Post-condition de nôtre programme qui n'es nul autre que :

$$\prod_{k=0}^{n} a$$

Dans un second temps, nous allons déterminer la terminaison de cet algorithme. Pour se faire, il faut réduire l'ambiguïté de la boucle for. Pour se faire, nous allons la décomposer en une boucle while :

```
while i < n
x<-x*a
i = i + 1
```

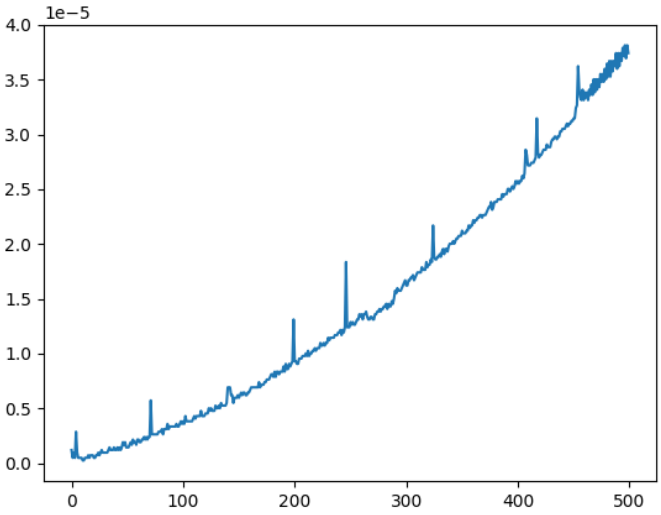
En faisant comme ceci, nous remarquons que, nous sortirons de la boucle lorsque i seras égale à n. De plus, nous allons observer la complexité de notre algorithme. Pour se faire, nous allons le programmer sur python pour pouvoir obtenir le temps à chaque itération de boucle :

```python
import time
import matplotlib.pyplot as plt

def Power(a,n):
    t = time.time()
    pwr,i = a,1
    if type(n)==float or n < 0:
        return "ERROR, entier naturels uniquement"
    for i in range(1,n):
        pwr = pwr * a
    return pwr, time.time()-t

max = 100
a = 2
labels = [x for x in range(max)]
values = [Power(a,x) for x in range(max)]
plt.plot(labels, values,)
plt.show()
```

De ce code, grâce à matplotlib, nous pouvons obtenir la courbe suivant qui nous donne le nombre d'itérations en fonction du temps.



Conclusion/Réflexions

Ne connaissant pas encore beaucoup la programmation récursive, je vais devoir apprendre ceci lors de la prochaine semaine afin de rattraper mon retard dans ce domaine précis et aussi réparer LaTeX car ce compte rendu est à mes yeux horrible et je m'en excuse, ceci sera changé pour la semaine prochaine