BELLIK Nassim
NASS Michaël

Rapport de projet

Nombre de Schur

BELLIK Nassim
NASS Michaël

Sommaire

BELLIK Nassim
NASS Michaël

Introduction

Le 1er semestre à Efrei Paris en programmation nous a formé à utiliser un nouveau language : le Python. Dans ce projet, nous devions developer un programme calculant le nombre de Schur sur une durée de 3 semaines.

Le nombre de Schur est issu du Théorème de Schur, mis en place par Issai Schur. Le nombre de Schur consiste à créer des triplets de différentes couleurs sous la forme (a, b, z), où est égal à a+b, et z ne peut pas excéder le nombre maximal d'entrée choisi (par exemple pour 100, z va varier entre 2 et 100). Chacun des nombres de ces combinassions est ensuite colorié avec toutes les couleurs possibles. Pour connaitre le nombre de Schur, on regarde le premier ensemble de triplets qui ne contient aucun triplet monochromatique.

Dans ce rapport, nous allons expliquer comment nous avons créé le code permettant de calculer le nombre de Schur en Python, et ce qu'on en a appris. Ensuite, nous allons évoquer les difficultés que nous avons rencontré et comment on les a surmontées, comment on aurait pu améliorer notre code et les problèmes qui restent non résolus.

I. Le code

Avant de commencer le programme, on réfléchi à une résolution d'une première problématique qui est venue très rapidement : comment va-t-on lier les couleurs à un nombre ? Les codes couleurs nous étaient donnés par le sujet. Une approche que nous avons alors fréquemment observé consistait à créer une liste contenant les couleurs et à les utiliser manuellement avec l'index correspondant à la position de la couleur, à la main. Nous avons décidé de prendre une toute autre voie. Nous avons pour cela fait appel à une notion de Python qui n'a pas été abordée en cours : les classes. On a créé trois fichiers Python, un qui se nomme « main.py », un autre qui se nomme « echauffemement.py », et un dernier qui se nomme « classes.py ». Les deux premiers fichiers servent à séparer les exercices intermédiaires du programme final. Le fichier classes est le plus intéressant pour le démarrage, car c'est lui qui définit nos éléments pour arriver au nombre de Schur.

Commençons par la classe Color : elle contient toutes les couleurs, mais au lieu de faire appel à une liste avec un index, on peut l'appeler par son vrai nom.

```
class Color:
    def __init__(self):
        self.white = '\033[0m'
        self.red = '\033[31m'
        self.green = '\033[32m'
        self.orange = '\033[33m'
        self.blue = '\033[34m'
        self.purple = '\033[35m'
        self.colors = [self.white, self.red, self.green, self.orange, self.blue, self.purple]
```

Pour s'en servir, on fait appel à la classe qu'on initialise dans une variable :

colors = Color()

La variable colors contient alors toutes les couleurs ainsi qu'une liste « colors » dans les quelques cas où il est préférable de passer par une liste avec un index plutôt que des noms de variables. Ainsi, dans notre code, pour écrire en orange, on utilise tout simplement

print(colors.orange + ‘’du texte colorié”) au lieu de print(colors[2] + ‘’du texte colorié”)

Cette classe a donc comme but de simplifier la compréhension du code.

Ensuite, nous avons créé une classe Element, permettant d'avoir un nombre de couleur en tant qu'objet. Cela permet de simplifier grandement la gestion des couleurs sur nos nombres. Voici à quoi ressemble :

```
class Element:
    def __init__(self, a, b):
        self.value = a
        self.color = b

    def print(self):
        print(self.color, str(self.value))

    def printable(self):
        val = self.color + str(self.value)
        return val
```

Cette fois, pour initialiser un élément, comme on peut le voir avec la fonction __init__(), on doit lui fournir deux éléments d'entrée : une valeur et une couleur. En reprenant la variable colors précédente, pour créer le nombre 5 en vert, voila comment on procède :

value = Element(5, color.green)

La classe Element contient également deux fonctions, print() et printable() qui servent à remplacer la fonction print() standard de python qui ne connait pas notre classe Element et donc imprimera ‘’<type Element at 0xADDR>” si on exécute

print(value). Ainsi, pour imprimer notre 5 vert, on utilise value.print(). La fonction printable() sert à faire appel à notre 5 vert dans un print existant car elle retourne le 5 vert prêt à l'impression. On s'en sert comme cela :

print(''Le nombre à imprimer en couleur est : '' + value.printable())

Enfin, nous avons créé une classe Triplet qui permet elle aussi de simplifier la gestion des triplets.

```
class Triplet:
    def __init__(self, a, b, c):
        self.element = a
        self.element2 = b
        self.element3 = c

    def isSameColor(self):
        if self.element.color == self.element2.color and self.element.color == self.element3.color:
            return True
        else:
            return False

    def print(self):
        print("(" + self.element.printable() + ", " + self.element2.printable() + ", " + self.element3.printable() + ")")

    def printable(self):
        return "(" + self.element.printable() + ", " + self.element2.printable() + ", " + self.element3.printable() + ")"
```

Pour l'initialiser, on doit lui donner 3 variables de type Element. On aurait pu se passer de la variable c, mais pour pouvoir réutiliser Triplet dans d'éventuels autres contextes que (a, b, a+b), on a gardé la variable c. Les fonctions print() et printable() fonctionnent de la même manière que les fonctions de la classe Element, et elles font appel à Element.printable() pour pouvoir fonctionner. On a également ajouté une fonction supplémentaire dans Triplet : la fonction isSameColor() qui renvoie vrai ou faux selon si un triplet est monochromatique ou pas. Pour l'utiliser, on utilise par exemple :

if triplet.isSameColor():

    print(''Le triplet '' + triplet.printable() + '' est monochromatique'')

De là, on a nos fondations pour poursuivre la réalisation de notre programme.

Détails des exercices:

Exo1

on doit assigner une couleur donnée à une suite de chiffres demandée en fonction de la parité des chiffres : les nombres pairs seront affichés en rouge et les nombres impairs seront affichés en bleu.

Pour cela on a utilisé la variable « colors » correspondant à la classe « Color » initialisée, une boucle « FOR » pour parcourir l'ensemble des nombres voulues ainsi que le modulo 2 pour détecter la parité.

BELLIK Nassim
NASS Michaël

Exo2

On veut colorer les N premiers nombres entiers en n couleurs choisies aléatoirement auquel les n couleurs seront toutes présentes.

Pour cela, on utilise la bibliothèque(library) « random » pour gérer les sélections de liste de manière aléatoire.

On utilise "sample" pour choisir aléatoirement un ensemble de n couleurs.

Exo 3

Pour cet exercice, on doit vérifier si un triplet est monochromatique (VRAI) ou polychromatique (FAUX).

On a décidé de choisir nous même les valeurs contenues dans les triplets ainsi que leurs couleurs. En effet, si on avait fait en sorte que le programme procède de manière aléatoire à la création de triplets de couleurs, on aurait pris le risque de ne pas afficher la réponse en fonction du type de triplet.

On écrit simplemant "in" sans écrire "range" car on veut que le programme prenne en compte les triplets et non pas le nombre de triplet contenue dans la liste.

Exo 4

On crée le nombre coloré « Element » en blanc car « Element » doit posséder une couleur même si on ne s'en sert pas.

On va assigner à p l'entier donné par l'utilisateur.

On n'était donc pas obligé de passer par « Elelment ». Cependant la classe « Triplet » dont on se sert dans l'exercice doit être initialisée avec 3 « Element ».

On ne voulait pas créer une 2ème classe « Triplet », on a donc utilisé des « Element » avec du blanc.

On utilise aussi 2 boucles « FOR » imbriqués avec les compteurs i pour (a) et s pour (b) pour créer tous les triplets de forme (a, b, a+b) avec a+b<=p.

La 3ème boucle « FOR » servira a affiché la liste de tous les triplets.

Etant donné qu'on a utilisée la programmation orienté objet, nous n'avons pas eu besoin d'utiliser lse exercices suivants comme la conversion de nombre décimales en base (b) (Exo 7 et 8).

Ce sont plutot les exercices 5 et 6 qui nous ont servis d'exemple dans la réalisation du programme et particulièrement l'éxercice 9.

Nous ne prenons pas la peine d'expliquer ces exercices d'avantages car nous avons deja expliqué les bases de notre système de résolution (classes, fonctions,etc).

II. Les difficultés et problèmes

Nous avons rencontré de nombreuses difficultés pendant la réalisation de notre programme, dont une grande partie sont restées sans résolution à cause du très court délai ouvré à notre disposition. Parmi les problèmes qui restent sans résolution lors de la restitution du projet est le calcul de S(2). Toutes les étapes nécéssaires pour trouver S(n) sont mises en places mais il manque la dernière étape qui consiste à regrouper les informations pour trouver le nombre de Schur. Une tentative a été faite pour calculer S(n), mais nous avons mal appliqué le morceau de code responsable de fabriquer les triplets du nombre de Schur. Une autre piste d'amélioration aurait été d'utiliser les nombres binaires de base n (nombre de couleurs) et de longueur p pour générer les combinaisons de couleurs, plutôt que d'utiliser une génération manuelle. Enfin, nous aurions pu forcer l'affichage des couleurs dans nos fonctions print sur les virgules et les parenthèses pour avoir une interface plus esthétique.

Conclusion

Nous n'avons malheureusement pas eu le temps d'exploiter tout le potentiel de l'algorithme car il n'est pas intégralement terminé. Nous avons pu néanmoins découvrir le fonctionnement des classes dans Python, une thématique qui n'a pas été abordée en classe et qui a pu prouver son intérêt dans notre code. Nous sommes conscients que ce projet n'est pas complètement terminé mais nous savons comment le compléter et l'améliorer à partir de ce que nous avons accompli.