Mathématiques PT. 2025 - 2026

Mathématiques & Informatique - F. Gaunard
http://frederic.gaunard.com
Lycée Voltaire, Paris 11e.

Préparation à l'oral

Math II - Informatique
Semaine du 2 Juin

Sujet OB-INFO-2

1. Entrer dans l'éditeur de texte les lignes de code suivante, et les exécuter plusieurs fois.

```
import random
especes=(500,200,100,50,20,10,5,2,1)
pm=random.choices(especes,k=10)
print(pm)
```

Expliquer en une phrase ce que génèrent ces lignes de code.

2. Calculer la somme des termes de la liste `especes`.

3. Écrire une fonction `tirer` prenant en argument un entier naturel N non nul, qui simule le tirage aléatoire de N nombres entre 0 et 8 avec remise, puis renvoie une liste `L`$= [n_0, \dots, n_8]$ telle que n_i donne le nombre de fois où on a tiré le nombre i.

La liste `especes` représente l'ensemble des pièces et billets existant dans le système monétaire européen.

4. Implémenter une fonction `total` qui, prenant en entrée une liste `L`$= [n_0, \dots, n_8]$ donnant le nombre de chacune des pièces et de chacun des billets dont on dispose, et qui calcule la valeur totale correspondante à ces pièces et billets. Par exemple, `total([0,0,1,0,1,3,0,5,0])` renvoie `160`.

5. Implémenter une fonction `portefeuille` qui, partant d'une valeur totale donnée, renvoie une liste `L`$= [n_0, \dots, n_8]$ qui permet de décomposer cette valeur en pièces et billets, de sorte que l'on utilise le moins de pièces et billets possible. Par exemple, `portefeuille(160)` renvoie `[0,0,1,1,0,1,0,0,0]`. On pourra effectuer un grand nombre d'essais en utilisant la fonction `tirer`, puis choisir la meilleure solution trouvée.

6. Implémenter la fonction `portefeuille` en utilisant cette fois algorithme glouton, de sorte qu'à chaque étape on choisisse le plus grand billet (ou pièce) possible.

7. Discuter les performances des deux algorithmes.

Sujet OB-INFO-2 : Math II 2021, Solution

1. Ces lignes de code renvoient une liste de longueur 10, contenant des termes choisis au hasard dans la liste `especes`.

2.
```
S=0
for x in especes:
    S=S+x
print(S)
```

3.
```
def tirer(N):
    L=[0 for i in range(9)]
    nombres=[i for i in range(9)]
    T=random.choices(nombres,k=N)
    for i in range(N):
        L[T[i]]=L[T[i]]+1
    return L
```

4.
```
def total(L):
    S=0
    for k in range(9):
        S=S+L[k]*especes[k]
    return S
```

5. Le principe est le suivant : pour chaque valeur de N, on extrait un ensemble aléatoire de pièces et billets, et si le montant correspondant est v on peut renvoyer cet ensemble. Partant de $N = 1$, on fait 1000 essais à chaque fois, et si l'on n'a pas obtenu au cours de ces essais la valeur v alors on incrémente N et on recommence.

```
def portefeuille(v):
    S=0
    L=[]
    k=1
    N=1
    while S!=v :
        if k==1000:
            k=1
            N=N+1
        L=tirer(N)
        S=total(L)
        k=k+1
    return L
```

6. On commence par donner une fonction qui, étant donné une valeur `v`, renvoie le plus grand billet (ou pièce) dont le montant est inférieur à `v`.

```
def plus_grand_inf(v):
    for k in range(9):
        if especes[k]<=v:
            return k
    return 0


def portefeuille2(v):
    m=v
    L=[0 for i in range(9)]
    while m!=0:
        k=plus_grand_inf(m)
        L[k]=L[k]+1
        m=m-especes[k]
    return L
```

7. L'algorithme probabiliste a deux inconvénients antagonistes :

- ✗ si on effectue trop peu d'essais pour chaque valeur de N on risque de passer à côté d'une combinaison convenant, et donc de ne pas renvoyer une décomposition minimale;
- ✗ effectuer beaucoup d'essais pour chaque valeur de N ralentit l'exécution du code.

L'algorithme glouton ne présente pas ses inconvénients, mais il ne fonctionne pas dans toutes les situations : par exemple, si `especes=[6,4,1]`, alors `portefeuille2(8)` renverra `8=6+1+1` alors que `8=4+4` est une décomposition minimale. Néanmoins on peut démontrer que cet algorithme est optimal dans le système monétaire européen