ISM MPSI/PCSI, Informatique
Année 2024/2025

Corrigé du TP Informatique 26

Exercice 1

1. On saisit :

```
def fact1(n):
    """Calcul itératif de n!"""
    res=1
    for k in range(2,n+1):
        res*=k
    return res
```

2. On saisit :

```
def prod(a, b):
    """Produit récursif des éléments de range(a, b)"""
    if b <= a:
        return 1
    elif b == a+1:
        return a
    else:
        c = (a+b) // 2
        return prod(a, c) * prod(c, b)
```

3. On saisit :

```
def fact2(n):
    """Calcul récursif de n!"""
    return prod(1, n+1)
```

4. On teste :

```
>>> fact1(5)
120
>>> fact2(5)
120
```

4. Soit a, b entiers avec $b > a$. On note $n = b - a$ et

$$\mathscr{P}(n) : \qquad \mathrm{T}(a,b) = b - a - 1 = n - 1$$

• **Initialisation :** Si $n = 1$, on est sur le cas de base $b = a + 1$ et `prod(a,b)` renvoie `a` sans effectuer de multiplications d'où $\mathrm{T}(a,b) = 0 = n - 1$.

• **Hérédité :** Supposons $\mathscr{P}(k)$ vraie pour tout $k \in [\![1\,;\, n-1]\!]$ avec $n \geqslant 2$ et soient a, b entiers avec $b - a = n$. D'après l'écriture de `prod`, on a

$$\mathrm{T}(a,b) = \mathrm{T}(a,c) + \mathrm{T}(c,b) + 1 \quad \text{avec} \quad c = \left\lfloor \frac{a+b}{2} \right\rfloor$$

On a
$$c - a \leqslant \frac{a+b}{2} - a = \frac{b-a}{2} \leqslant \frac{n}{2} \leqslant n - 1$$

puis
$$b - c < b - \frac{a+b}{2} + 1 = \frac{b-a}{2} + 1 = \frac{n}{2} + 1 \leqslant n$$

Comme $b - c$ est entier et strictement inférieur à n, on en déduit $b - c \leqslant n - 1$. Ainsi, par hypothèse de récurrence, il vient

$$\mathrm{T}(a, b) = c - a - 1 + b - c - 1 + 1 = b - a - 1 = n - 1$$

ce qui clôt la récurrence. On a donc établi

$$\boxed{\forall (a, b) \in \mathbb{N}^2 \quad \text{avec} \quad b > a \qquad \mathrm{T}(a, b) = b - a - 1}$$

5. Le nombre de multiplications réalisées par `fact2(n)` est égal à $\mathrm{T}(1, n+1) = n+1-1-1 = n-1$ et celui de `fact1(n)` est le nombre de passages dans la boucle `for`. Ainsi

Les fonctions `fact1(n)` et `fact2(n)` réalisent toutes deux $n - 1$ multiplications.

6. On obtient :

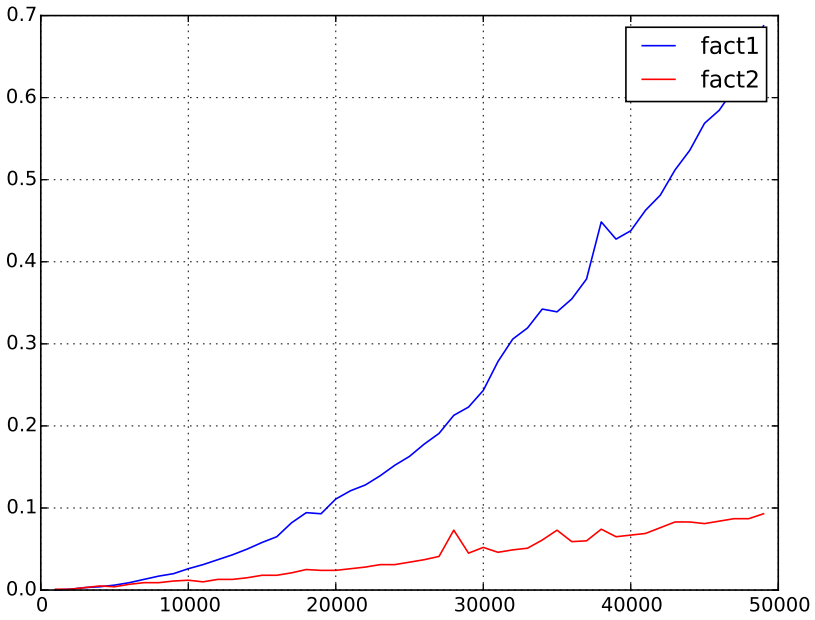

FIGURE 1 – Comparaison `fact1` *versus* `fact2`

7. On observe des écarts de performance conséquents alors que le nombre de multiplications réalisées par `fact1` et `fact2` sont identiques. Dans le calcul itératif, l'opération `res*=k` réalise une multiplication de `k` avec un nombre `res` de plus en plus grand. Par exemple, pour le calcul de 5000!, le programme effectue les calculs intermédiaires suivants :

$$\ldots \quad 4000! \times 4001, \qquad 4001! \times 4002, \quad \ldots \quad 4999! \times 5000$$

Or, la multiplication d'un entier avec un grand entier requiert plus de temps de calcul qu'une multiplication entre deux petits nombres. La technique récursive qui consiste à découper chaque produit en deux sous-produits fait qu'en arrivant au bout d'une branche d'appels récursifs, on réalise majoritairement des multiplications avec des petits nombres. Ainsi, détaillant les

multiplications faites en fin de récursion, le programme `fact2` effectue les calculs intermédiaires suivants :

$$((\dots((1\times 2)\times(3\times 4))\times\dots)\times(\dots\times((4997\times 4998)\times(4999\times 5000))\dots))$$

Il y a autant de multiplications qu'avec `fact1` mais la majorité des multiplications est faites sur des petits nombres d'où l'écart de performance.

Exercice 2

1. On saisit :

```
def compter(L,x):
    res=0
    for a in L:
        res+=a==x
    return res
```

2. Pour x élément de `L`, on note $o(x)$ son nombre d'occurrence. Soient x et y strictement majoritaires avec $x\neq y$. Alors, on a $o(x)+o(y)>n$ ce qui est absurde d'où

S'il existe un élément strictement majoritaire, alors celui-ci est unique.

Soit x l'élément strictement majoritaire de `L`. Notons $o_1(x)$ le nombre d'occurrences de x dans `L[:n//2]`, liste de cardinal $\lfloor n/2\rfloor$ et $o_2(x)$ le nombre d'occurrences de x dans `L[n//2:]`, liste de cardinal $n-\lfloor n/2\rfloor$. On a

$$o_1(x)\leqslant\frac{\lfloor n/2\rfloor}{2}\quad\text{et}\quad o_2(x)\leqslant\frac{n-\lfloor n/2\rfloor}{2}\quad\Longrightarrow\quad o(x)=o_1(x)+o_2(x)\leqslant\frac{n}{2}$$

Par contraposée, on en déduit que

L'élément strictement majoritaire de `L` l'est aussi pour une des sous-listes considérées.

3. On saisit :

```
def majo1(L):
    n=len(L)
    if n==1:
        return L[0],1
    else:
        m1,c1=majo1(L[:n//2])
        if c1>0:
            c1=c1+compter(L[n//2:],m1)
        m2,c2=majo1(L[n//2:])
        if c2>0:
            c2=c2+compter(L[:n//2],m2)
        if c1>n/2:
            return [m1,c1]
        elif c2>n/2:
            return [m2,c2]
        else:
            return [0,-1]
```

4. On peut améliorer notablement la version précédente en faisant en sorte qu'il n'y ait plus, en moyenne, qu'un seul appel récursif :

```
def majo2(L):
    n=len(L)
    if n==1:
        return L[0],1
    else:
        m1,c1=majo2(L[:n//2])
        if c1>0:
            c1=c1+compter(L[n//2:],m1)
            if c1>n/2:
                return [m1,c1]
        if c1<=n/2:
            m2,c2=majo2(L[n//2:])
            if c2>0:
                c2=c2+compter(L[:n//2],m2)
                if c2>n/2:
                    return [m2,c2]
            if c2<=n/2:
                return [0,-1]
```