I Polynôme et dictionnaire

Soit $P = \sum_{k=0}^{n} a_k X^k$ un polynôme. On représente P par un dictionnaire `p` tel que, pour tout $k \in [\![0, n]\!]$, si $a_k \neq 0$ alors `p[k]` vaut a_k (on ne stocke pas les coefficients nuls de P). Dit autrement, `p[k]` contient le coefficient de degré k de P.

1. Définir le dictionnaire représentant le polynôme $7 + 3X^2 - X^5$.

> Solution : `p = {0 : 7, 2 : 3, 5 : -1}`

2. Écrire une fonction `degre` renvoyant le degré d'un polynôme.

> Solution :

```python
def degre(p):
    maxi = 0
    for k in p:
        if k > maxi:
            maxi = k
    return maxi
```

> Ou, une solution un peu plus rapide à écrire :

```python
def degre(p):
    return max(p.keys())
```

3. Écrire une fonction `derive` qui renvoie la dérivée P' d'un polynôme P.

> Solution :

```python
def derive(p):
    dp = {}
    for k in p:
        if k != 0: # le terme constant disparaît
            dp[k - 1] = p[k]
    return dp
```

4. Définir une fonction `somme(p, q)` renvoyant un dictionnaire représentant la somme des deux polynômes `p` et `q`. Quelle est sa complexité en fonction des degrés n_1 et n_2 de `p` et `q` ?

> Solution : La complexité de la fonction suivante est $O(n_1 + n_2)$ (en considérant que les opération de dictionnaire sont en $O(1)$, ce qui est normalement le cas en moyenne seulement).

```python
def somme(p, q):
    r = {}
    for k in p:
        r[k] = p[k]
    for k in q:
        if k in r:
            r[k] += q[k]
        else:
            r[k] = q[k]
    return r
```

5. Faire de même avec une fonction `produit(p, q)`.

> Solution : Chaque terme $a_i X^i$ de P multiplié avec un terme $b_j X^j$ de Q va donner un terme $a_i b_j X^{i+j}$ dans le produit. D'où :

```python
def produit(p, q):
    r = {}
    for i in p:
        for j in q:
            if i + j in r:
                r[i + j] += p[i]*q[j]
            else:
                r[i + j] = p[i]*q[j]
    return r
```

La complexité est clairement $O(n_1 \times n_2)$.

6. Écrire une fonction `evalue(x, p)` renvoyant $P(x)$, si possible avec $O(n)$ multiplications, où $n = \deg(P)$.

Solution : On rappelle que le calcul de `x**n` demande $O(\log(n))$ multiplications (avec l'algorithme d'exponentiation rapide). Pour avoir une complexité $O(n)$, on peut utiliser la méthode de Horner. Une autre possibilité est de calculer et stocker toutes les puissances, pour éviter de les recalculer en totalité :

```python
def evalue(x, p):
    n = degre(p)
    puissances = [1]
    for i in range(n):
        puissances.append(puissances[-1]*x)
    res = 0
    for k in p:
        res += p[k]*puissances[k]
    return res
```