TP #5 — Produit de polynômes

Exercice 1. *Fonctions préliminaires*

1. Écrivez des fonctions :
   ```
   mul257(x:int, y:int) -> int
   add257(x:int, y:int) -> int
   sub257(x:int, y:int) -> int
   ```
   qui calculent respectivement le produit, l'addition et la soustraction dans $\mathbb{Z}/257\mathbb{Z}$.
2. Écrivez une fonction :
   ```
   fma257(a:int, x:int, y:int) -> int
   ```
   qui calcule dans $\mathbb{Z}/257\mathbb{Z}$ (sans utiliser vos fonctions de la question précédente) l'opération « *fused multiply and add* », définie comme l'addition de a au produit xy.
3. Écrivez une fonction :
   ```
   trim(P:list[int]) -> list[int]:
   ```
   qui supprime tous les éventuels `0` « de queue » de son argument P, et renvoie celui-ci. Dans le cas particulier où P ne contient *que* des `0`, *trim* renvoie la liste `[0]`.
   Par exemple, `trim([0,1,0,2,0,0])` renvoie `[0,1,0,2]` ; `trim([0,0,1])` renvoie `[0,0,1]` ; `trim([0,0,0])` renvoie `[0]`.

Exercice 2. *Polynômes sur $\mathbb{N}$, entiers naturels… c'est (presque) pareil !*

Dans cet exercice on considère des polynômes à coefficients dans $\mathbb{N}$ (mais une généralisation à, par exemple, des coefficients dans $\mathbb{Z}/p\mathbb{Z}$ est possible).
Soit un polynôme $P \in \mathbb{N}[X]$, on définit son évaluation en la base $B \in \mathbb{N}$ (de la façon habituelle) comme $p := \sum_{i=0}^{\infty} P_i B^i$; soit $p \in \mathbb{N}$ un entier et $p^B = (p_i^B)$ son écriture en base B, on définit son « interpolation » comme le polynôme $P := \sum_{i=0}^{\infty} p_i^B X^i$.
On peut alors remarquer que *dans certains cas*, le produit de deux polynômes P, Q peut être calculé en évaluant P et Q en p, q en une certaine base, calculant le produit $pq \in \mathbb{N}$, et interpolant le résultat.
Par exemple : soit $P = 3X^2 + X$, $Q = 4X^2 + X + 2$, les évaluations de P et Q en $B = 100$ sont respectivement `30100` et `40102`, de produit `1207070200`, et l'on a $PQ = 12X^4 + 7X^3 + 7X^2 + 2X$.

1. Écrivez une fonction :
   ```
   ksmul(P:list[int], Q:list[int], B:int) -> list[int]
   ```
   qui utilise l'approche esquissée ci-dessus pour calculer et renvoyer le produit de deux polynômes à coefficients dans $\mathbb{N}$.

2. Quel contrainte sur B pouvez-vous anticiper, afin que le calcul effectué par *ksmul* soit correct ?
3. Testez votre fonction sur de petites entrées variées avec des bases variées, et vérifiez votre hypothèse de la question précédente.
4. Supposez que vous puissiez évaluer un polynôme à coefficients dans $\mathbb{N}$ « en boîte noire », et que vous connaissiez (un majorant de) son degré. Donnez un algorithme utilisant au plus deux évaluations boîte noire et permettant de déterminer ses coefficients.
5. Implémentez et testez cet algorithme (avec des polynômes « connus », mais en respectant les contraintes).

Exercice 3. *Produit naïf*

1. Écrivez une fonction :

   ```
   pmul(P:list[int], Q:list[int]) -> list[int]
   ```

   qui calcule « naïvement » (de façon habituelle) et renvoie le produit dans $\mathbb{Z}/257\mathbb{Z}[X]$ des polynômes représentés par les listes P et Q comme $\sum_{i=0}^{\ell_P-1} P[i]X^i$ et $\sum_{i=0}^{\ell_Q-1} Q[i]X^i$. (Autrement dit, `P[i]` représente le coefficient de X^i dans P, et de même pour `Q`.)
2. Écrivez une fonction :

   ```
   pmul_perf(d:int)
   ```

   qui génère aléatoirement et uniformément deux polynômes de degré (au plus) d de $\mathbb{Z}/257\mathbb{Z}[X]$ représentés comme à la question précédente, et utilise *pmul* pour calculer leur produit. Cette fonction ne doit rien renvoyer.
3. Utilisez `timeit` (par exemple avec :

   ```
   timeit('pmul_perf(1000)', number=1, globals=globals())
   ```

   pour vérifier expérimentalement que le coût asymptotique de *pmul* est un $O(d^2)$, avec d le degré maximum de ses arguments.
4. *(Optionnel.)* En modifiant légèrement *pmul_perf* et vos mesures, vérifiez expérimentalement que le coût asymptotique de *pmul* est plus précisément un $O(d_P d_Q)$, avec d_P et d_Q les degrés de ses deux arguments.

Exercice 4. *Algorithme « de Karatsuba »*

On souhaite maintenant implémenter un algorithme plus efficace, qui se base sur l'observation suivante : soit P, Q deux polynômes et $n, P_{hi}, P_{lo}, Q_{hi}, Q_{lo}$ un entier et quatre polynômes tels que l'on a $P = P_{hi}X^n + P_{lo}$, $Q = Q_{hi}X^n + Q_{lo}$, le produit PQ est égal à :

$$P_{hi}Q_{hi}X^{2n} + (P_{hi}Q_{lo} + P_{lo}Q_{hi})X^n + P_{lo}Q_{lo}$$

ce qui peut se calculer *via* le calcul des **trois** produits :

— $PQ_{hi} := P_{hi}Q_{hi}$

— $PQ_{lo} := P_{lo}Q_{lo}$
— $PQ_{mid} := (P_{hi} + P_{lo})(Q_{hi} + Q_{lo})$

comme :

$$PQ_{hi}X^{2n} + (PQ_{mid} - PQ_{hi} - PQ_{lo})X^n + PQ_{lo}$$

On peut prendre ci-dessus P_{hi} et P_{lo} respectivement le quotient et le reste dans la division de P par X^n (et de même pour Q), mais : 1) ce n'est pas obligatoire ; 2) il ne faut *surtout pas* les calculer en utilisant une implémentation générique de division.

1. Écrivez une fonction :

```
padd(P:list[int], Q:list[int]) -> list[int]
```

qui calcule et renvoie la somme de deux polynômes $P, Q \in \mathbb{Z}/257\mathbb{Z}$, représentés comme à l'exercice précédent.

2. Écrivez une fonction :

```
psub(P:list[int], Q:list[int]) -> list[int]
```

qui calcule et renvoie la différence $P - Q$ de deux polynômes $P, Q \in \mathbb{Z}/257\mathbb{Z}$, représentés comme à la question précédente.

3. Écrivez une fonction :

```
kmul(P:list[int], Q:list[int], t:int) -> list[int]
```

qui calcule et renvoie le produit de deux polynômes $P, Q \in \mathbb{Z}/257\mathbb{Z}$, représentés comme à la question précédente. Cette fonction doit utiliser l'algorithme récursif suivant :

— Cas de base : si le maximum des degrés de P, Q est inférieur à t, on calcule PQ naïvement en utilisant *pmul*
— Cas récursif :
  — On « pad » les listes représentant P et Q en leur ajoutant au besoin des `0` de façon à ce qu'elles soient toutes deux de même longueur paire ℓ.
  — On calcule récursivement les trois produits PQ_{hi}, PQ_{mid}, PQ_{lo} pour une valeur de n égale à $\ell/2 - 1$. (Autrement dit, les arguments des appels récursifs sont les $\ell/2$ premières ou dernières entrées de P ou Q.)
  — On combine les résultats des appels récursifs pour construire le résultat PQ.

4. Écrivez une fonction :

```
pkmul_test(d:int, t:int, hn:int) -> bool
```

qui génère aléatoirement et uniformément jusqu'à *hn* paires de polynômes P, Q de degré (au plus) d, et renvoie `True` si tous leurs produits tels que calculs par *pmul* et *kmul* sont égaux, et `False` sinon. (Pensez au besoin à utiliser *trim*.)

5. Écrivez une fonction

   ```
   kmul_perf(d:int)
   ```

   qui génère aléatoirement et uniformément deux polynômes de degré (au plus) d de $\mathbb{Z}/257\mathbb{Z}[X]$ et utilise *kmul* pour calculer leur produit. Cette fonction ne doit rien renvoyer.
6. Utilisez `timeit` pour vérifier expérimentalement que le coût asymptotique de *kmul* est un $O(d^{\log_2(3)})$, avec d le degré maximum de ses arguments (concrètement, cela veut dire que le coût triple quand les degrés doublent).