TD 2 : PGCD et inversion modulaire

Exercice 1. *Complexité binaire de l'algorithme d'Euclide*

Soit A et B deux entiers tels que $A \geq B$. On considère l'exécution de l'algorithme d'Euclide sur l'entrée (A, B). On note $R_0 = A$, $R_1 = B$ et $R_{i+2} = R_i \bmod R_{i+1}$ pour $i \geq 0$. On note ℓ l'indice du dernier R_i non nul.

1. Exprimer le nombre de divisions euclidiennes effectuées en fonction de ℓ.
2. Montrer que le nombre de divisions euclidiennes est borné par $1 + \log_\phi(A)$ et par $2 + \log_\phi(B)$. Quelle borne est la meilleure ?
3. Pour $i \geq 0$, on note n_i le nombre de chiffres (en une base β quelconque) de R_i.
   i. Exprimer la complexité du calcul de $R_i \bmod R_{i+1}$ en fonction de n_i et n_{i+1}.
   ii. En déduire la complexité binaire de l'algorithme d'Euclide, en fonction des n_i.
   iii. En déduire que la complexité binaire est bornée par $O(\log A \log B)$.

Exercice 2. *Algorithme de PGCD binaire*

1. Soit u et v deux entiers.
   i. Exprimer $\mathrm{pgcd}(2u, 2v)$ en fonction de $\mathrm{pgcd}(u, v)$.
   ii. Si v est impair, exprimer $\mathrm{pgcd}(2u, v)$ en fonction de $\mathrm{pgcd}(u, v)$.
   iii. Si u et v sont impairs, montrer que $\mathrm{pgcd}(u, v) = \mathrm{pgcd}(|u - v|/2, \min(u, v))$.
2. Déduire de la question précédente un algorithme de calcul de PGCD qui n'utilise comme opérations de base que la soustraction et la multiplication et la division par 2.
3. Prouver la correction de votre algorithme, et analyser sa complexité binaire pour deux entiers de n bits en entrée.
4. Qu'est-ce qui rend cet algorithme efficace en pratique ?

Exercice 3. *Inversion par le petit théorème de Fermat*

1. Montrer que le petit théorème de Fermat permet d'obtenir un algorithme d'inversion modulaire dans $\mathbb{Z}/p\mathbb{Z}$ (pour p premier), et analyser sa complexité.
2. Que se passe-t-il si on essaie d'utiliser l'algorithme dans $\mathbb{Z}/N\mathbb{Z}$, avec N non premier ?

Devoir à la maison – programmation.

L'objectif de ce devoir de programmation est d'implanter, en Python [1], certaines des fonctions vues en cours, en se basant sur les entiers Python (type `int`) qui sont des entiers multiprécision, et les fonctions sur les entiers qui ne sont pas explicitement interdites. *Remarque. Le quotient dans la division euclidienne de* `a` *par* `b` *s'obtient avec* `a // b`*, le reste avec* `a % b`*, et le couple (quotient, reste) avec* `divmod(a, b)`*.*
Il est impératif de tester chaque fonction !

3. i. Écrire l'algorithme d'Euclide étendu.
   ii. Écrire une version *itérative* de l'algorithme d'Euclide étendu.
   iii. Comparer les temps de calcul [2] des deux versions.
4. Écrire une fonction d'inversion modulaire : étant donné a et N, cette fonction calcule l'inverse de a *modulo* N. Si a n'est pas inversible *modulo* N, la fonction doit soulever une exception. *Par exemple, on peut utiliser :* `raise ValueError(f'{a} n'est pas inversible modulo {N}')` *pour cela.*
5. i. Écrire une fonction d'exponentiation rapide, qui calcule $a^b \bmod N$. *Il est interdit d'utiliser la fonction Python* `pow`*.*
   ii. Comparer le temps de calcul de cette fonction avec l'utilisation des opérateurs Python `a**b % N` et expliquer.

1. Utiliser Python 3, puisque Python 2 est officiellement devenu obsolète.
2. On pourra utiliser le module `timeit` : `https://docs.python.org/fr/3/library/timeit.html`.