Feuille d'exercices n°11 de l'option d'informatique.

Autour des nombres premiers.

1. Écrire une fonction **trouve_diviseur** qui retourne le plus petit diviseur d'un entier n en testant successivement les entiers à partir de 2. En déduire une fonction qui détermine si un nombre est premier ou non. Améliorer la fonction **trouve_diviseur** en créant une fonction **suivant** qui calcule le diviseur suivant à utiliser sachant qu'il n'y a pas besoin de vérifier les nombres pairs si 2 ne convient pas. En comptant le nombre d'appels à la fonction **mod**, quelle est la complexité de la fonction **trouve_diviseur** dans le pire des cas?
2. Écrire une fonction **est_premier** qui teste si un nombre n est premier en regardant s'il est divisible par l'un des entiers de 2 à $n - 1$ en utilisant la fonction **mod**.
En comptant le nombre d'appels à la fonction **mod**, quelle est la complexité de la fonction **est_premier** en fonction de n dans le pire des cas?
3. En remarquant qu'il suffit de tester les diviseurs potentiels inférieur à $\sqrt{n}$, améliorer la fonction précédente. Que peut-on dire alors de la complexité dans le pire des cas de la fonction obtenue?
4. On veut cette fois écrire un test permettant de savoir si un nombre est premier ou non de manière probabiliste, c'est à dire que si la procédure nous dit que le nombre est premier il y a de fortes chances pour qu'il le soit et si la procédure nous dit qu'un nombre n'est pas premier, il est sûr qu'il ne l'est pas. L'algorithme repose sur le résultat suivant connu sous le nom de petit théorème de Fermat qui nous dit :

Si un nombre n est premier et a un entier positif inférieur à n , alors a élevé à la puissance n est congru à a modulo n ($a^n \equiv a[n]$).

- Écrire une fonction **expmod b e n** qui élève le nombre b à la puissance e modulo l'entier n (on s'efforcera d'écrire une exponentielle rapide (en $O(\ln n)$) en exploitant le résultat suivant sur les congruences ($a_1 \equiv b_1[n]$ et $a_2 \equiv b_2[n] \Rightarrow a_1.a_2 \equiv b_1.b_2[n]$).
Le résultat est un entier entre 0 et $n - 1$.
- Écrire une fonction **test_fermat n** qui prend un entier a au hasard entre 2 et $n - 1$ et teste si $a^n \equiv a[n]$.
- Écrire une fonction **estpremier_rapide k n** qui effectue k fois le test de Fermat sur n et retourne **true** s'ils sont tous vrais, sinon il s'arrête au premier test faux et retourne **false**.

Il existe malheureusement des nombres qui ne sont pas premiers et qui passe avec succès tous les tests de Fermat, c'est à dire $\forall a \in \llbracket 0, n - 1 \rrbracket, a^n \equiv a[n]$. Ces nombres sont appelés des nombres de Carmichael. Il y a 16 nombres de Carmichael inférieurs à 100000, les plus petits étant 561, 1105, 1729, 2465, 2821 et 6601.

5. Une des variantes du test de Fermat, qui ne peut pas être mise en défaut, a été trouvée par Solovay et Strassen (1977). Pour savoir si un nombre *impair* n est un nombre premier, elle consiste à choisir au hasard un nombre $a < n$, puis à tester si $\text{PGCD}(a, n) = 1$ et à calculer une quantité appelée symbole de Jacobi: $J(a, n)$, égale à ± 1 . Si n est premier alors $J(a, n)$ est toujours congru modulo n à $a^{(n-1)/2}$ pour tout a tel que $\text{PGCD}(a, n) = 1$. Sinon il peut être prouvé que cette relation n'est pas vérifiée pour au moins la moitié des nombres $a < n$. Ainsi, lorsque nous trouvons que la relation est vérifiée pour un nombre a choisi au hasard, nous pouvons affirmer qu'il y a plus d'une chance sur deux pour que n soit premier. Le symbole de Jacobi peut être calculé en utilisant les réductions suivantes :

$$J(a, n) = \begin{cases} 1 & \text{si } a = 1 \\ J(a/2, n) \cdot (-1)^{(n^2-1)/8} & \text{si } a \text{ est pair} \\ J(\text{reste}(n, a), a) \cdot (-1)^{(a-1)(n-1)/4} & \text{autrement} \end{cases}$$

- Mettre en œuvre par une procédure en temps $O(\log n)$ le test de Solovay-Strassen. (attention la fonction **mod** ne marche pas avec des nombres négatifs).
- Écrire une fonction **estpremier_rapide_bis k n** qui effectue k fois le test de Solovay-Strassen sur n et retourne **true** s'ils sont tous vrais, sinon il s'arrête au premier test faux et retourne **false**.

Référence bibliographique : «structure et interprétation des programmes informatiques» Abelson et Sussman. InterEditions.