Para poder utilizar la criptografía RSA con seguridad es crucial:

1. Poder comprobar de manera eficiente si un número muy grande es primo o no. Es esto lo que nos permite encontrar los dos primos muy grandes p y q cuyo producto forma parte de la clave pública del usuario.

   En la práctica NO hace falta una demostración matemática completa de que los enteros p y q son primos, y basta con lo que llamamos *criterios probabilísticos de primalidad*, como el de Miller-Rabin.
2. Convencerse de que, en el estado actual de la tecnología, no es posible factorizar el entero n en un tiempo razonable. Todo entero n se puede factorizar probando posibles divisores hasta llegar a $\sqrt{n}$, pero para n muy grande el tiempo requerido sería tan grande que no tendría ningún sentido intentarlo.

1 Miller-Rabin (1980)

El teorema pequeño de Fermat afirma que si n es un primo entonces $a^{n-1} \equiv 1 \mod n$ para cada entero a primo con n. Si encontramos un a, primo con n, tal que el resto de dividir a^{n-1} entre n no es 1 podemos estar seguros de que n NO es primo.

Sin embargo, hay números que son compuestos pero verifican que $a^{n-1} \equiv 1 \mod n$ para cada entero a primo con n, es decir, que son compuestos pero se comportan como primos para el teorema pequeño de Fermat. Estos números se llaman *de Carmichael*.

EJERCICIO: Escribe código, lo más eficiente posible, que, dado un entero N, devuelva una lista de los enteros de Carmichael menores que N.

El test de Miller-Rabin está muy relacionado con el teorema pequeño de Fermat, pero es un "test probabilístico" que puede demostrar que un número compuesto lo es con una probabilidad tan grande como queramos, por ejemplo 0.999999999999. Si el test no puede probar que el número es compuesto podemos estar "casi seguros" de que es primo, y se usa en criptografía como si lo fuera. El test se basa en dos teoremas:

1. *Sea n un número primo, tal que $n-1=2^s d$ con d impar, y a un entero primo con n. Entonces, o bien $a^d \equiv 1 \mod n$, o bien hay un entero r en $\{0,1,2,...,s-1\}$ tal que $a^{2^r d} \equiv -1 \mod n$. Si encontramos un a primo con n tal que no se cumple ninguna de las dos condiciones, podemos estar seguros de que el número n es compuesto, y decimos que el entero a es un "testigo" de la no primalidad de n.*
2. *Si $n \geq 3$ es impar y compuesto entonces el conjunto $\{1,2,...,n-1\}$ contiene a lo más $(n-1)/4$ enteros que son primos con n y no son testigos de que n es compuesto.*

Supongamos que queremos "probar" que n es primo. Elegimos al azar un entero a en el conjunto $\{1,2,...,n-1\}$, si resulta que a no es un testigo se ha producido un hecho que tenía probabilidad menor que $1/4$. Si repetimos t veces la elección al azar de a y nunca encontramos un testigo se ha producido un hecho de probabilidad $(1/4)^t$ que podemos hacer tan pequeña como queramos tomando t suficientemente grande, y $1-(1/4)^t$ es la probabilidad de que si el número es compuesto Miller-Rabin lo detecte.

EJERCICIO

1. Escribe código para aplicar el test de Miller-Rabin a un entero n que detecte números compuestos con probabilidad p fijada. Debes, entonces, programar una función `miller-rabin(n,p)`, que dado un entero N, devuelva una lista de los enteros de Carmichael menores que N.
2. Escribe otra función que, dada la probabilidad p, busque enteros, dentro de un rango fijado *a priori*, que pasen el test de Miller-Rabin con probabilidad p, es decir, Miller-Rabin con esa p los declare primos, pero sean compuestos. Cuando p es muy próximo a 1, esta función devolverá probablemente una lista vacía.
3. Intenta aplicar el test de Miller-Rabin a un producto de dos primos muy grandes.
4. Usa Miller-Rabin para buscar el primo más grande que es menor que 2^{400}.

2 Pollard $p-1$ (1974)

El algoritmo $p-1$ de Pollard sirve para factorizar enteros n impares que tienen un factor primo p tal que $p-1$, que es compuesto, tiene factores primos no demasiado grandes. Necesita una cota *a priori*, B, del tamaño de los primos que aparecen en la factorización de $p-1$. Es decir, si n es compuesto y tiene un factor primo p tal que todos los factores primos de $p-1$ son menores que B el algoritmo puede encontrar un factor no trivial de n.

1. Se comienza eligiendo un entero M tal que $p-1$ sea con seguridad un divisor de M. Como no conocemos la factorización en primos de $p-1$ debemos tomar como M el producto de todos los primos menores que B con exponentes mayores que los exponentes con los que aparecen en la factorización de $p-1$. Por ejemplo, si $p-1=2^{k_1} \cdot 3^{k_2} ... p_i^{k_i} ...$ y como $p-1<n$, podemos afirmar con seguridad que $k_2 \cdot \log(2) < \log(n)$ y, en general $k_i < \log(n)/\log(p_i) = \log_{p_i}(n)$. Estas cotas, que son bastante más grandes de lo necesario, sólo dependen de n y los primos p_i y se verifica que $p-1$ divide a M.
2. Gracias al teorema pequeño de Fermat, podemos afirmar que $a^M \equiv 1 \mod p$ para todo a que sea primo con p. Elegimos entonces un a primo con n, y por tanto primo con p, y debe verificarse que p también divide a a^M-1 y, por tanto, p debe dividir al máximo común divisor de a^M-1 y n.
3. El máximo común divisor de a^M-1 y n es un divisor de n, que es lo que buscábamos, y nos sirve si es diferente de n. La ventaja enorme es que, como debemos saber, se puede calcular módulo n.
4. Si el algoritmo no encuentra un factor n, podemos probar con diferentes valores de a, por ejemplo haciendo un cierto número de elecciones aleatorias para a, y si todavía no encontramos un factor podemos incrementar B.

5. El problema es que si repetimos con diferentes valores de a o incrementamos la cota B el tiempo de ejecución del programa aumenta, y puede ocurrir que tarde tanto como lo que tardaría un programa que probara los posibles divisores menores que la raíz cuadrada de n (el método "bestia"). A pesar de sus limitaciones, el método es bastante eficiente y ha dado lugar a toda una serie de refinamientos que son los actualmente en uso para factorizar enteros grandes.
6. Por supuesto, antes de intentar factorizar n debemos estar convencidos de que es compuesto, por ejemplo aplicándole Miller-Rabin.

EJERCICIO:

1. Implementa el algoritmo de Pollard y comprueba su funcionamiento en un número razonable de casos.
2. El algoritmo $p-1$ funciona bien cuando n tiene un factor primo p tal que $p-1$ sólo tiene primos menores que un B y todos los demás factores de n son grandes respecto al correspondiente M. Trata de comprobar experimentalmente esta afirmación.