Criptografía y Computación
grado en ingeniería informática

Práctica 1

Primalidad

Autor
Vladislav Nikolov Vasilev

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

Instrucciones de ejecución

Se ha adjuntado un *script* de Python que ejecuta cada uno de los ejercicios y muestra los resultados. Para ejecutarlo se necesita tener `python3` instalado en el equipo, y basta con situarse en el lugar donde se tenga el *script* y ejecutar:

```
$ python3 primalidad.py
```

La ejecución tarda poco más de 2 minutos debido a que en uno de los ejercicios se hace un cómputo bastante costoso (se calcula un primo fuerte de 500 bits).

Ejercicio 1

En este ejercicio se pide implementar una función que realice el test de Miller-Rabin dados un número impar n y un testigo a tal que $2 \leq a \leq n-2$. La función debe devolver verdadero en caso de que n sea probable primo y falso en caso contrario.

Por una parte, para realizar el test de Miller-Rabin necesitamos una función que calcule la descomposición de $n-1$ como $2^u \cdot s$, donde s es un número impar. Esta función se ha implementado de la siguiente forma:

```python
def descomposicion(n):
    # Inicializar u y s
    u = 0
    s = n

    while s % 2 == 0:
        u += 1
        s = s // 2

    return u, s
```

La función que realiza el test de Miller-Rabin para un n y un a dados es la siguiente:

```python
def miller_rabin(n, a):
    # 1. Descomponer n-1 como 2^u * s con s impar
    u, s = descomposicion(n-1)

    # 2. Calcular a = a^s mod n
    a = potencia_modular(a, s, n)

    # Si a == 1 o a == n-1, el numero es posible primo
    if a == 1 or a == n-1:
```

```python
            return True

    for i in range(1, u):
        a = potencia_modular(a, 2, n)

        # Si a == 1 sin haber pasado por n-1, el numero no es primo
        # ya que tiene mas de una solucion a x^2 - 1 = 0
        if a == 1:
            return False

        """
        Si a == n-1, el siguiente valor sera 1, por lo tanto,
        cumpliria el test de Fermat y tendria solo dos soluciones a
        la ecuacion x^2 - 1 = 0. Puede ser primo
        """
        if a == n-1:
            return True

    return False
```

Se ha probado la función anterior con $n = 1729$ y con dos testigos: $a_1 = 2$ y $a_2 = 10$. En el primero caso, la función ha determinado que n no es primo, mientras que en el segundo caso ha determinado que sí que lo es. Este comportamiento es el esperado, ya que sabemos que $1729 = 7 \cdot 247$ y que por tanto no es primo, y que $a = 10$ es un falso testigo.

Ejercicio 2

En este ejercicio se ha pedido que se implemente una función que realice el test de Miller-Rabin escogiendo m testigos aleatorios. La función es la siguiente:

```python
def test_primalidad(n, m):
    for i in range(m):
        # Escoger testigo tal que 2 <= a <= n-2
        a = random.randint(2, n-2)

        es_prob_primo = miller_rabin(n, a)

        if not es_prob_primo:
            return False

    return True
```

En el momento en el que el test de Miller-Rabin devuelva falso, se ha conseguido determinar que el número no es probable primo, y por tanto no se prueban más testigos.

Se ha probado esta función con tres números y con $m = 20$, ya que con dicho valor nos podemos asegurar que la probabilidad de que falle el test de primalidad sea menor a $\frac{1}{4^{20}}$. Los números probados han sido $n_1 = 341$, $n_2 = 1729$ y $n_3 = 2039568783564019774057658669290345772801939933143482630947726464532830627227012776329366160631440881733123728826771238795387094001583065673383282791544996983660719067664400370742171178056908727928481491120222863321448761833763265120835748216479339929612499173198362193042742802438031040150005637901 23$. De estos tres números, solo n_3 es primo.

Los resultados que ha ofrecido la función una vez que ha sido ejecutada han sido correctos, ya que ha dicho que los dos primeros números no son primos y que el tercero es probable primo.

Ejercicio 3

En este ejercicio se pide implementar una función que dado un número n calcule un número n' tal que $n \leq n'$ y n' sea probable primo.

Para determinar dicho número nos podemos ayudar de la función anterior. Podemos ir recorriendo los números a partir de n y hacerles un test de primalidad, y en el momento en el que nos encontremos con un probable primo, devolverlo. Dicha funcionalidad se ha implementado de la siguiente forma:

```python
def siguiente_primo(n, m):
    es_posible_primo = False

    while not es_posible_primo:
        es_posible_primo = test_primalidad(n, m)

        if es_posible_primo:
            posible_primo = n

        n += 1

    return posible_primo
```

Para ver si la función es correcta se ha probado con $n_1 = 14$ y $n_2 = 1729$. En el primer caso se ha obtenido $n'_1 = 17$, mientras que en el segundo se ha obtenido $n'_2 = 1733$. Ambos números son primos (aparecen en cualquier lista de números primos que se pueda encontrar por internet), y por tanto, el funcionamiento parece correcto.

Ejercicio 4

En este ejercicio se pide implementar una función que dado un número n encuentre el primer probable primo fuerte n' tal que $n \leq n'$. El número n' es primo fuerte si tanto él como $\frac{n'-1}{2}$ son primos.

Para hacerlo, se han implementado las siguientes funciones:

```
def test_primo_fuerte(n, m):
    return test_primalidad((n - 1) // 2, m)


def siguiente_primo_fuerte(n, m):
    es_primo_fuerte = False

    while not es_primo_fuerte:
        n = siguiente_primo(n, m)
        es_primo_fuerte = test_primo_fuerte(n, m)

        if es_primo_fuerte:
            primo_fuerte = n

        n += 1

    return primo_fuerte
```

La función va buscando primos probables y cada vez que se topa con uno intenta determinar si es un pirmo fuerte. En caso de que lo sea, lo devuelve, y en caso contrario, continua con la búsqueda.

Se ha probado la función con $n_1 = 12$ y con $n_2 = 1729$. En el primer caso, el primer primo fuerte encontrado ha sido 23, mientras que en el segundo ha sido 1823. Con la ayuda de una tabla de primos que se puede encontrar en internet se han comprobado los resultados y se ha visto que son correctos. En el primer caso esto es así porque tanto 23 como 11 son primos, mientras que en el segundo porque tanto 1823 como 911 son primos. Por tanto, la función parece tener el comportamiento esperado.

Ejercicio 5

En este apartado se ha pedido implementar una función que calcule el primer probable primo fuerte de n bits. Este primo, p, al tener n bits, tendrá que tener su valor en el rango $2^{n-1} \leq p \leq 2^n - 1$.

La función que se ha implementado es la siguiente:

```python
def primo_fuerte_n_bits(n, m):
    return siguiente_primo_fuerte(2 ** (n-1), m)
```

Se ha probado la función anterior con una serie de valores de n y se han obtenido los siguientes resultados:

- Con $n = 10$ bits se ha obtenido que el primer primo fuerte es 563.
- Con $n = 25$ bits se ha obtenido que el primer primo fuerte es 16777907.
- Con $n = 50$ bits se ha obtenido que el primer primo fuerte es 562949953422839.
- Con $n = 100$ bits se ha obtenido que el primer primo fuerte es 63382530011 4114700748351612867.
- Con $n = 500$ bits se ha obtenido que el primer primo fuerte es 16366953039480 7093500659484841379957610832102302153239474164568404806689820233727 7441635046162952078575443342063780035504608628272942696526664264070 799.

Cada resultado se ha pasado por un factorizador en línea y se ha comprobado que todos cumplen las condiciones para ser primos fuertes. Por tanto, la función tiene el comportamiento esperado.

Ejercicio 6

En este ejercicio se deben escoger tres números compuestos n_1, n_2 y n_3, los cuáles deben cumplir una serie de restricciones. En el caso de n_1 se tienen que obtener todos los falsos testigos, mientras que para n_2 y n_3 se deben probar 200 testigos aleatorios y determinar cuántos de ellos han sido falsos testigos.

Para calcular todos los falsos testigos de un número se ha utlizado la siguiente función:

```python
def calcular_todos_falsos_testigos(n):
    falsos_testigos = []

    for a in range(2, n - 1):
        if miller_rabin(n, a):
            falsos_testigos.append(a)

    return falsos_testigos
```

Para probar una serie de m testigos aleatorios y determinar aquellos que resulten ser falsos testigos se ha utilizado la siguiente función:

```
def calcular_m_falsos_testigos(n, m):
    falsos_testigos = []

    for _ in range(m):
        a = random.randint(2, n - 2)

        if miller_rabin(n, a):
            falsos_testigos.append(a)

    return falsos_testigos
```

Finalmente, para calcular la proporción de falsos testigos se ha utilizado la siguiente función:

```
def calcular_proporcion_falsos_testigos(falsos_testigos, m):
    return len(falsos_testigos) / m
```

Una vez que se han visto las funciones a utilizar, vamos a pasar a ver qué números compuestos se han escogido y los resultados que se han obtenido.

Se ha escogido que n_1 sea $n_1 = 11^2 = 121$. Los falsos testigos que se han encontrado han sido 3, 9, 27, 40, 81, 94, 112 y 118, lo cuál representa una proporción de aproximadamente 0.06 de todos los posibles testigos.

Para n_2 se ha escogido que su valor sea $n_2 = 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 = 1062347$. En este caso no se ha obtenido ningún falso testigo.

Para n_3 se han escogido un primo fuerte mayor que 10000000 y otro fuerte de 25 bits y se han multiplicado. El valor de n_3 obtenido ha sido $n_3 = 10000223 \cdot 16777907 = 167782811473261$. Nuevamente tampoco se han encontrado falsos testigos.

Ejercicio 7

En este ejercicio se pedía que se probasen 200 testigos aleatorios y que se determinasen cuáles de ellos habían sido falsos testigos para el número $n = 3215031751$.

Los falsos testigos encontrados han sido los siguientes:

2989642428, 2543428172, 614042734, 1424011350, 300843473, 3183414300, 2410287500, 2962532651, 419843896, 1715915166, 2241650903, 1889534559, 3105476637, 516122673, 2109566925, 901700272, 1197571907, 2231960901, 3128587513, 3148414053, 3034801065, 1955701261, 327244541, 2216210500, 1845860340, 1084176316, 2012890817, 313758577, 515904415, 959186557, 2272308219, 1423678299, 3202777707, 750340098, 1310849420, 508705234, 2108513998, 1171046710, 2130543965, 2324710841, 2050129978, 666132191

De los 200 testigos probados, un total de 42 de ellos han resultado ser falsos testigos, lo cuál representa una proporción de 0.21.

Ejercicio 8

En este ejercicio se ha pedido que se escogiesen 100 testigos aleatorios para el número $n = 2199733160881$ y se determinase cuáles y cuántos de ellos resultaban ser falsos testigos según el test de Fermat y el de Miller-Rabin.

La función que se ha implementado para hacer esto se puede ver a continuación:

```python
def falsos_testigos_fermat_miller_rabin(n, m):
    falsos_testigos_fermat = []
    falsos_testigos_miller_rabin = []

    for _ in range(m):
        a = random.randint(2, n - 2)

        if test_fermat(n, a):
            falsos_testigos_fermat.append(a)

        if miller_rabin(n, a):
            falsos_testigos_miller_rabin.append(a)

    return falsos_testigos_fermat, falsos_testigos_miller_rabin
```

Los falsos testigos que se han encontrado mediante el test de Fermat han sido los siguientes:

541294428553, 427414670640, 856840996708, 869459582987, 1386150924507, 2021284951395, 1049180344086, 1436874409098, 2080622412228, 1895446961219, 607589646504, 95738089244, 500152881760, 273343767955, 816294840698, 2137304862156, 1686099093795, 105112798591, 76108696568, 2084939446410, 288639221475, 2175697704600, 1521166120712, 1208550220361, 734596648309, 1775915927632, 1444025072928, 1539558933563, 317159111698, 153191386359, 107087172632, 626790072879, 1721765538668, 1339295283967, 309820597244, 1499387178030, 645650661512, 251351171707, 125102898679, 670739616035, 1664579380, 1166105943589, 1623944880881, 1439812021977, 1291686686494, 1254049709277, 973102251046, 697908434478, 1154053174720, 1931057820368, 645605063013, 348148721167, 1467216861092, 2100887387037, 1196824929484, 506736084768, 382854241647, 1083675018117, 157094190622, 770574992, 611934573350, 1723351143188, 1748441690699, 362412962126, 443665136244, 1518280794907, 1494524424336, 278756486397, 2059386093625, 545388027793, 851959036442, 1992276265019, 2195305341090, 399547067632, 89834049489, 696316386402, 589834218012, 557877780581, 62877284258, 2071701656149, 1813750085866, 199765571834, 696380393864, 873234754041, 10738916652, 925428820553, 1057060786293, 2009802064885, 257038689587, 277003366590, 531869096808, 1194918230399, 581254591237, 337015700018, 1190017832155, 1329585096763, 1955831589860, 137053152985, 1154693933317, 363634170708

Los falsos testigos encontrados mediante el test de Miller-Rabin han sido los siguientes:

1686099093795, 105112798591, 697908434478, 1518280794907, 589834218012, 199765571834, 277003366590

En este caso particular, todos los testigos probados han resultado ser falsos testigos utilizando el test de Fermat, lo cuál es una proporción de 1, mientras que solo 7 de ellos han resultado serlo con el de Miller-Rabin, lo cuál es una proporción de 0.07.