Ejercicio Hoja de Problemas Probabilistas

Óscar Brizuela - 820773

EJERCICIO 11

Índice

1. Pruebas realizadas

1.1. Pruebas con *miTest.py*

El programa *miTest.py* se ha realizado basándose en el pequeño teorema de *Fermat* para la comprobación de primalidad. Por tanto, es seguro que todos aquellos números para los que el algoritmo devuelve el valor de verdad “falso” no son primos. Sin embargo, aquellos para los cuales el algoritmo devuelve el valor de verdad “verdadero” no tienen por qué ser primos, debido a la existencia de los falsos testigos de primalidad.

Para ello, se ha guardado en una lista todos los primos menores que 10000, procedente de Wikipedia. Una vez se ejecuta el algoritmo se compara la lista de “supuestos primos” que este devuelve con la lista real correspondiente. Aquellos que devuelva el algoritmo que no estén en la lista facilitada por Wikipedia no serán primos, a pesar de que el algoritmo los haya clasificado como que sí lo son.

Ahora, necesitamos saber con qué probabilidad nuestro algoritmo es capaz de clasificar erróneamente estos “no primos mal clasificados”. Debido a que, en nuestro algoritmo (función *fermat(n)* en *miTest.py*) estamos usando como base *a* un número aleatorio en el rango *[2 , n - 2]*, la probabilidad con la que nuestro algoritmo clasificará mal los falsos primos obtenidos será:

$$P(clasificar\ n\ como\ primo\ siendo\ n\ no\ primo) = número\ de\ falsos\ testigos\ de\ primalidad\ /\ número\ de\ “testigos\ totales”$$

o lo que es lo mismo:

$$P(clasificar\ n\ como\ primo\ siendo\ n\ no\ primo) = número\ de\ falsos\ testigos\ de\ primalidad\ /(n - 2)$$

Por ejemplo, el número 561, que cuenta con 318 falsos testigos de primalidad, tiene una probabilidad de ser erróneamente clasificado como primo por nuestro algoritmo (cuando en realidad no lo es) de:

$$318\ /\ (561 - 2) = 0.568 = 56.8\%$$

Hay que destacar que, debido a la base *a* utilizada, el *n* debe ser mayor o igual que 4.
Ahora que tenemos la lista de los “no primos mal clasificados” y las correspondientes probabilidades de que sean mal clasificados por nuestro algoritmo, necesitamos ejecutar varios experimentos para poder obtener el mayor número de “no primos mal clasificados” posible puesto que, al fin y al cabo, estamos eligiendo el testigo (base *a* utilizada en el pequeño teorema de *Fermat*) de manera aleatoria. Dicho con otras palabras, en un experimento pueden no salir algunos “no primos mal clasificados” que sí salen en otro experimento y viceversa.
Cabe destacar que estas repeticiones independientes del algoritmo no son para reducir la probabilidad de error al clasificar los números como primos o no primos, sino para obtener el mayor número de “no primos mal clasificados” posible.

1.1.1. ¿Cuántos números clasifica mal *miTest.py* con alta probabilidad? ¿Cuáles?

Para este apartado se han utilizado los resultados procedentes de 10 experimentos, con un umbral de probabilidad del 40%. Esto significa que los números escogidos son aquellos “no primos mal clasificados” por el algoritmo que tienen una probabilidad de ser mal clasificados por dicho algoritmo mayor al 40%. Estos números, con sus correspondientes probabilidades, son los siguientes:

Número	Probabilidad de ser mal clasificado como primo
561	0.5688729874776386
703	0.4593437945791726
1105	0.6944696282864914
1729	0.7492762015055009
1891	0.4753838009528851
2465	0.72675598863175
2701	0.4794368284549833
2821	0.7655196878325647
6601	0.7998181542657978
8911	0.7998653047480077

Tabla 1: falsos primos mal clasificados por miTest.py con sus probabilidades

Pruebas con *test_miller_rabin.py*

El programa *test_miller_rabin.py* se ha realizado siguiendo el algoritmo Miller-Rabin. Este algoritmo siempre devuelve el valor de verdad “verdadero” cuando *n* es primo. Sin embargo, devuelve “falso” con una probabilidad mayor o igual que 0.75 cuando *n* es un impar no primo, siendo así un algoritmo 0.75-correcto para comprobar la primalidad.

Debido a la existencia de los falsos testigos de primalidad en el sentido fuerte, cuyo número es mucho menor que el de falsos testigos de primalidad, la probabilidad media de elegir uno de estos falsos testigos en el sentido fuerte para un impar no primo menor que 1000 es de 0.01. Una de las razones es que más del 72% de los números menores de 1000 (entre 720 y 730 números) no admiten ningún falso testigo en el sentido fuerte. De esta manera, cada

llamada a la función *millerTest()* tiene como mucho una probabilidad de 0.25 de llegar a un falso testigo en el sentido fuerte, devolviendo verdadero de manera errónea. Por tanto, utilizamos el parámetro *k* para aumentar la precisión de nuestro algoritmo con el propósito de encontrar *k* falsos testigos en el sentido fuerte seguidos. De esta manera, cuanto mayor sea el valor de *k*, menor será el error en la clasificación. Estas repeticiones independientes del algoritmo hacen que pase a ser un algoritmo de Monte Carlo $1 - 4^{-k}$- correcto.

Suponiendo que un "no primo mal clasificado" tiene un número *d* de falsos testigos en el sentido fuerte, la probabilidad de *Miller-Rabin* de clasificar erróneamente este número como primo es 0.25^{d}. Por tanto, al realizar *k* iteraciones, esta probabilidad se ve reducida a $0.25^{d^{k}}$
.

Nota:

Debido a que el código implementado para el algoritmo Miller-Rabin es ciertamente diferente al código en los apuntes de clase (no es una traducción directa de ellos puesto que, a pesar de que el parámetro d en el código corresponde al parámetro t en los apuntes, no encuentro la misma equivalencia con el entero s ni con el booleano parar), no he podido obtener el número de falsos testigos de primalidad en el sentido fuerte de un número, con el objetivo de calcular, a raíz de este, la probabilidad de que este sea erróneamente clasificado como primo por el algoritmo Miller-Rabin.

Sin embargo, para calcular este número de falsos testigos de primalidad en el sentido fuerte para un número n según el código de los apuntes, podríamos invocar a una función similar a la siguiente con un número noPrimo que no esté en la lista de 10000 primos reales antes mencionada:

```
def numero_de_falsos_testigos_sentido_fuerte(noPrimo):
     t = 1
     s = 0
     a = 2
     nFalsosTestigos = 0
     while t < n:
          if (a^2^s) * t % noPrimo == noPrimo - 1
               nFalsosTestigos += 1     // a es falso testigo
          s += 1
          a += 1
          t += 2
     return nFalsosTestigos
```

Al ejecutar nuestro algoritmo *Miller-Rabin*, por tanto, ciertos números son clasificados como primos cuando no lo son. En la tabla 2 se encuentran varios resultados en función del número de iteraciones (*k*). Es lógico que cuanto mayor sea *k* menor sea el número de "no primos mal clasificados", puesto que la precisión del algoritmo aumenta debido a las razones explicadas anteriormente.

Cabe recordar que estos “no primos mal clasificados” no van a ser siempre los mismos para una misma *k*, debido que la base *a* es, al fin y al cabo un número aleatorio (en realidad, pseudo-aleatorio).

Valor del número de repeticiones del algoritmo (*k*)	**Números no primos mal clasificados como primos**
1	{49, 91, 169, 276, 286, 325, 451, 671, 806, 976, 1300, 1729, 3091, 3721, 4382, 4795, 5713, 6903, 7201, 7525, 8029, 8695, 9100}
2	{1541, 2501}
3	{}

Tabla 2: falsos primos mal clasificados por test_miller_rabin.py respecto al número de repeticiones del algoritmo

Conclusiones

Hemos comparado el rendimiento de dos algoritmos probabilistas: uno que no es correcto, implementado en el fichero *miTest.py* y basado en el pequeño teorema de *Fermat* y los falsos testigos de primalidad, y otro que es 0.75-correcto (sin tener en cuenta las repeticiones del algoritmo para mejorar su precisión), que es una implementación del algoritmo *Miller-Rabin*, basado en una extensión del pequeño teorema de Fermat y en los falsos testigos de primalidad en el sentido fuerte.
Hemos comprobado que, debido a que el algoritmo de *miTest.py* no es *p-correcto* para ningún $p > 0$, la probabilidad de error no puede disminuirse mediante repeticiones independientes del algoritmo, puesto que cada “no primo mal clasificado” cuenta con una probabilidad de que el algoritmo lo clasifique erróneamente que es independiente del número de repeticiones de dicho algoritmo (es una relación entre el número de testigos totales y el número de falsos testigos de primalidad). Sin embargo, los números “no primos mal clasificados” como primos por el algoritmo *Miller-Rabin*, implementado en el fichero *test_miller_rabin.py,* tienen una probabilidad que sí se puede reducir realizando varias repeticiones del algoritmo, debido a que este número de repeticiones reduce la posibilidad de encontrar varios falsos testigos de primalidad en el sentido fuerte seguidos, caso en el que dicho algoritmo clasificará erróneamente un número no primo como primo.

Bibliografía

Implementación de *Miller-Rabin*:
https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/

Lista de primos menores que 10000:
https://es.wikipedia.org/wiki/Anexo:N%C3%BAmeros_primos

Precisión del algoritmo *Miller-Rabin*:
http://funes.uniandes.edu.co/20158/1/Martinez2010El.pdf