Kryptologia laboratorium 8.

Testy pierwszości

Tomasz Gzella
Instytut Matematyki Stosowanej

POLITECHNIKA GDAŃSKA

WYDZIAŁ FIZYKI TECHNICZNEJ
I MATEMATYKI STOSOWANEJ

Twierdzenie

Dla każdej liczby naturalnej n istnieje liczba pierwsza p większa od n.

Dowód: Liczba naturalna

$$m = n! + 1$$

jest większa od 1, więc ma dzielnik pierwszy p.
Ten dzielnik musi być większy od n, bo liczba m daje resztę 1 z dzielenia przez liczby nie przekraczające n.

Twierdzenie

Dla każdej liczby naturalnej n istnieje liczba pierwsza p większa od n.

Dowód: Liczba naturalna

$$m = n! + 1$$

jest większa od 1, więc ma dzielnik pierwszy p.
Ten dzielnik musi być większy od n, bo liczba m daje resztę 1 z dzielenia przez liczby nie przekraczające n.

Testy pierwszości to algorytmy sprawdzające, czy podana liczba jest pierwsza. Mają zastosowanie przy szyfrach, w których potrzebujemy dużych liczb pierwszych.

Testy pierwszości cz. 1

Standardowy test to sprawdzenie podzielności przez kolejne liczby naturalne:

Standardowy test to sprawdzenie podzielności przez kolejne liczby naturalne:

```
def pierwsza(a):
  p=1
  while p==1:
    for i=2,...,\sqrt{a}
      if (a mod i) == 0:
        p=0
        break
  wend
        return(p)
```

Standardowy test to sprawdzenie podzielności przez kolejne liczby naturalne:

```
def pierwsza(a):
  p=1
  while p==1:
    for i=2,...,\sqrt{a}
      if (a mod i) == 0:
        p=0
        break
  wend
        return(p)
```

Sprawdza się on dla niewielkich liczb. Niestety, dla liczby *n – cyfrowej* wykonujemy $k \cdot 10^{n/2}$ operacji.

Możemy więc użyć testu, który **z dużym prawdopodobieństwem** wskaże, że liczba nie jest złożona. Na początek użyjmy do tego małego twierdzenia Fermata:

MTF

Jeżeli p jest liczbą pierwszą, to dla dowolnej liczby całkowitej a:
$a^p - a \ (mod \ p) = 0$

Testy pierwszości cz. 2

Możemy więc użyć testu, który **z dużym prawdopodobieństwem** wskaże, że liczba nie jest złożona. Na początek użyjmy do tego małego twierdzenia Fermata:

MTF

Jeżeli p jest liczbą pierwszą, to dla dowolnej liczby całkowitej a:
$a^p - a \ (mod \ p) = 0$ (lub w postaci $a^{p-1} \ (mod \ p) = 1$)

Testy pierwszości cz. 2

Możemy więc użyć testu, który **z dużym prawdopodobieństwem** wskaże, że liczba nie jest złożona. Na początek użyjmy do tego małego twierdzenia Fermata:

MTF

Jeżeli p jest liczbą pierwszą, to dla dowolnej liczby całkowitej a:
$a^p - a \ (mod \ p) = 0$ (lub w postaci $a^{p-1} \ (mod \ p) = 1$)

```
def pierwszaF(a,s)
  p=1
  for i=1,...,s
    losuj t\in\{2,3,...,a-2\}
      if t^{a-1} mod a!=1
        p=0
        break
  return(p)
```

Lecz i ten test można ulepszyć, by był wydajniejszy.

Sprawdzamy test Millera-Rabina:

```
def pierwszaMR(a,s)
  p=1 #test pierwszosci
  a-1=2^ur  #wyliczyc u oraz r
  for i=1,...,s #co zrobic z duzym s?
    losuj t w {2,3,...,a-2} #usuwac z listy
    z=t^r mod a #binarnie
    if z!= 1 then
      j=0
      while z!=a-1 do
        z=z^2 mod a #binarnie
        j=j+1
        if z==1 or j==u then
          p=0, break
  return(p)
```

Dla dobrze dobranego s uzyskujemy mały błąd w tym teście.

Uwagi:

Dodatkowe warunki w teście pierwszości:

```
if a == 1 then
  p=0
elif a == 2 or a == 3 then
  p=1
elif a % 2 == 0 then
  p=0
else ...
```

W linii $a - 1 = 2^u r$ należy wyliczyć u oraz r

Uwagi:

Dodatkowe warunki w teście pierwszości:

```
if a == 1 then
  p=0
elif a == 2 or a == 3 then
  p=1
elif a % 2 == 0 then
  p=0
else ...
```

W linii $a - 1 = 2^u r$ należy wyliczyć u oraz r, używane w dalszej części programu:

```
def ur(a)
  u=0, r=a-1
  while r % 2 == 0 do
    r=r/2, u=u+1
  return(u,r)
```

Potęgowanie modulo - przykład

Przykładowo policzmy 5^{100} *mod* 22.
W poprzedniej funkcji **potega_m(a,b,n)** trzeba było wykonać aż 99 mnożeń modulo 22.

Potęgowanie modulo - przykład

Przykładowo policzmy 5^{100} *mod* 22.
W poprzedniej funkcji **potega_m(a,b,n)** trzeba było wykonać aż 99 mnożeń modulo 22.
Tym razem zapiszmy binarnie wykładnik potęgi $100_{10} = 0b1100100 = 1100100_2$.
Zatem $100 = 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6$,
czyli $100 = 2^2 + 2^5 + 2^6 = 4 + 32 + 64$,

Potęgowanie modulo - przykład

Przykładowo policzmy 5^{100} *mod* 22.
W poprzedniej funkcji **potega_m(a,b,n)** trzeba było wykonać aż 99 mnożeń modulo 22.
Tym razem zapiszmy binarnie wykładnik potęgi $100_{10} = 0b1100100 = 1100100_2$.
Zatem $100 = 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6$,
czyli $100 = 2^2 + 2^5 + 2^6 = 4 + 32 + 64$,
więc tylko te potęgi będą nam potrzebne

bin	5^{2^d}	*wynik*
0	$5^1 \ mod\ 22 = 5$	1
0	$5^2 \ mod\ 22 = 3$	1
1	$5^4 \ mod\ 22 = 9$	$1 \cdot 9 \ mod\ 22 = 9$
0	$5^8 \ mod\ 22 = 15$	9
0	$5^{16} \ mod\ 22 = 5$	9
1	$5^{32} \ mod\ 22 = 3$	$9 \cdot 3 \ mod\ 22 = 5$
1	$5^{64} \ mod\ 22 = 9$	$5 \cdot 9 \ mod\ 22 = 1$