Prática em probabilidade: detecção de primos

Não se conhece ainda um algoritmo capaz de decidir em tempo polinomial se um número é primo. A maneira trivial de determinar se n é primo é através da "força bruta". Acompanhe N é primo ? [Swa91, devidamente corrigido..., págs. 16 a 20]

1ª Estratégia: Se N $\geq$ 4 é divisível por um fator entre 2 e (n-1) então, N é composto

```
1: lógico função EPRIMO1 (inteiro N)
2: lógico PRIMO
3: inteiro FATOR
4: PRIMO ← .V.
5: se N > 3
6:    FATOR ← 2
7:    enquanto PRIMO ∧ (FATOR ≤ N - 1)
8:       se N mod FATOR = 0
9:          PRIMO ← .F.
10:       senão
11:          FATOR++
12:       fim{se}
13:    fim{enquanto}
14: fim{se}
15: devolva PRIMO
16: fim função
```

2ª Estratégia: Se N for par (e diferente de 2) é composto, senão se for divisível por fator impar entre 3 e N-1 é composto, senão é primo. Se N < 4 é primo.

```
1: lógico função EPRIMO2 (inteiro N)
2: lógico PRIMO
3: inteiro FATOR
4: PRIMO ← ¬ ((N ≠ 2) ∧ (N mod 2 = 0))
5: se PRIMO ∧ (N > 4)
6:    FATOR ← 3
7:    enquanto PRIMO ∧ (FATOR ≤ N - 1)
8:       se N mod FATOR = 0
9:          PRIMO ← .F.
10:       senão
11:          FATOR ← FATOR + 2
12:       fim{se}
13:    fim{enquanto}
14: fim{se}
15: devolva PRIMO
16: fim função
```

3ª Estratégia: Só precisamos testar o divisor de N até a raiz quadrada de N...

```
1: lógico função EPRIMO3 (inteiro N)
2: lógico PRIMO
3: inteiro FATOR
4: PRIMO ← ¬ ((N ≠ 2) ∧ (N mod 2 = 0))
5: se PRIMO ∧ (N > 4)
6:    FATOR ← 3
7:    enquanto PRIMO ∧ (FATOR ≤ √N)
8:       se N mod FATOR = 0
9:          PRIMO ← .F.
10:       senão
11:          FATOR ← FATOR + 2
12:       fim{se}
13:    fim{enquanto}
14: fim{se}
15: devolva PRIMO
16: fim função
```

.

N	PRIMO1	PRIMO2	PRIMO3
10	10	5	3
100	100	50	10
1000	1000	500	32
10000	10000	5000	100

Testando estes 3 algoritmos para verificar se 10 números (100003, 199999, 300007, 400009, 500009, 599999, 700001, 799993 e 900001) são primos, tem-se os seguintes tempos: a soma de tempo dos 10 números é 186 seg. (alg. 1), 93 seg. (alg. 2) e 0,13 seg. (alg. 3)

Este último algoritmo parece bom, até tentarmos descobrir se o número $2^{400} - 593$ é ou não é primo. Para este número com o algoritmo acima serão necessárias 8.03469022^{59} operações. Em um computador capaz de fazer 10.000 comparações por segundo, serão necessários 2 547 000 000 000 000 000 000 000 000 000 000 000 000 000 000 séculos. Vamos combinar que não dá para esperar tanto.

Pequeno Teorema de Fermat

Dados p primo e a inteiro e $1 \leq a \leq p - 1$ então $(a^{p-1}) mod\ p = 1$

A dificuldade é que o teorema informa quando o número não é primo. Isso porque o teorema não funciona nas duas direções e o fato de que dois números p e a satisfaçam o a relação do teorema NÂO implica em que p seja primo. Trocando em miúdos: se p é primo, a relação do teorema é verdadeira. Se a relação é verdadeira para um determinado par p e a isso não necessariamente implica que p seja primo. Para concluir, o que o teorema efetivamente diz é que se $a^{p-1} mod\ p \neq 1$ então p não é primo.

Assim, uma primeira versão do algoritmo poderia ser

```
1: logico função EPRIMO (inteiro x)
2: inteiro a
3: a ← random(1..x-1)
4: se a^(x-1) mod x ≠ 1
5:    devolva FALSO
6: senão
7:    devolva VERDADEIRO ...mas poderia ser
      talvez
8: fim{se}
9: fim{função}
```

Perceba-se que este algoritmo é do tipo Montecarlo. Rápido, mas podendo responder errado. Devido aquelas dificuldades apontadas, este algoritmo pode devolver como primo um número composto.

Por exemplo, se $a = 2$ e $p = 341$, $2^{340} mod\ 341 = 1$ e 341 não é primo, pois é divisível por 11 e 31. Idem para $a = 3$ e $p = 91$, sendo que $91 = 7 \times 13$. Uma lista dos pares a e p que estragam o Pequeno Teorema de Fermat está em http://mathworld.wolfram.com / FermatsLittleTheorem.html.

A probabilidade de um número composto ser apontado como primo é de 0.5%. Se este número for inaceitável, diversos aleatórios podem ser criados e a primalidade apontada apenas se o número passar em todos os testes efetuados. O algoritmo poderia ficar assim

```
1: lógico função EPRIMO2 (inteiro x)
2: inteiro a,b
3: lógico resposta ← VERDADEIRO
4: enquanto b ≤ constante
5:    a ← random(1..x-1)
6:    se a^(x-1) mod x ≠ 1
7:       resposta ← FALSO
8:    fim{se}
9:    constante ← constante + 1
10: fim{enquanto}
11: devolva resposta
12: fim {função}
```

Como em diversos algoritmos de Montecarlo, pode-se aumentar a confiabilidade à custa de maior número de testes, de modo que a confiabilidade seja tão grande como se queira.

Números de Carmichael

Ainda que se faça uma quantidade grande de testes, há alguns números (conhecidos como Números de Carmichael) que desafortunadamente, embora compostos garantem a primalidade para qualquer a compreendidos entre 1 e o número - 1. Para estes números não adianta usar o pequeno teorema de Fermat, a resposta sempre sairá errada. Por sorte, estes números são raros. Na lista dos primeiros 100 bilhões de naturais há apenas 255 números de Carmichael. (os 3 primeiros são 561, 1105, 1729).

Assim, a nossa terceira versão de programa, deveria explicitamente excluir os números de carmichael como compostos, antes de responder.

Ps: Para descobrir os números de Carmichael, examine seus divisores e tente achar uma regra geral:

C	divisores	fatores
561	3, 11, 17, 33, 51, 187	$3 \times 11 \times 17$
1105	5, 13, 17, 65, 85, 221	$5 \times 13 \times 17$
1729	7, 13, 19, 91, 133, 247	$7 \times 13 \times 19$

Algoritmo de Rabin

Este algoritmo consegue dizer em poucos segundos se o número acima $2^{400} - 593$ é primo com probabilidade igual a

```
0.999999999999999999999999999999999999999999999999
  999999999999999999...999999999999999999999999999
                (400 noves)
  999999999999999999...999999999999999999999999999
```

este algoritmo cria a noção de "testemunha" da composição de um número n. Se uma única testemunha é encontrada, então n é composto. Mas, se um razoável esforço é gasto na busca de uma testemunha e nenhuma é encontrada, então n pode ser definido como primo. Rabin define uma testemunha da composição de n como um inteiro w satisfazendo uma de duas condições:

1. $w^{n-1} = 1\ mod\ n$ ou
2. para algum k, $m = \frac{n-1}{2^k}$ é inteiro e $1 < mdc(w^n - 1, n) < n$

Rabin estabeleceu o seguinte teorema: Se n é composto então mais da metade dos números do conjunto $2, 3, ...n - 1$ são testemunhas da composição de n. Agora fica fácil bolar o algoritmos

```
1: leia n
2: selecione m inteiros w1, w2, ..., wm
   randomicamente do conjunto 2,3,...n-1
3: para i=1, 2, ...  m
4:    teste se wi é testemunha
5:    se nenhum dos m inteiros é testemunha,
      responda SIM
6:    senão responda NÃO
7: fim{para}
```

Como se chega na probabilidade desejada

Usando o algoritmo de Rabin e o teorema acima, pode-se afirmar que $P_c \geq 0.5$ a cada execução, e daqui $P_p < 0.5$. Se usarmos dois aleatórios diferentes, e ambos relatarem primo fica-se com $P_p < 0.5 \times 0.5$ ou $P_p < 0.25$. Generalizando, ao usar i chamadas para o mesmo número, se todas reportarem primalidade, então este resultado é certo com $P_p < 0.5^i$, daí porque ao usar este algoritmo tantas vezes quantas se queira é possível garantir primalidade com a probabilidade desejada que no limite é assintótica a 1.

☞ Para você fazer

Use o método que quiser e descubra o maior fator próprio (fator próprio de um número inteiro é qualquer divisor diferente do número). Obviamente se n é primo, seu maior fator próprio é a unidade

1. 6662339023
2. 6281886451
3. 5414726651
4. 3966497287

1	2	3	4

Para seu governo, e para exemplificar o que estamos estudando, saiba que, nos 4 números acima, a $P(_{primo}) = 0.5$.