Fundação CECIERJ - Vice Presidência de Educação Superior a Distância
Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação I
AD1 1^o semestre de 2009.

---

1 Questão Única

Escreva um programa que dado um número natural qualquer diga se ele é primo, perfeito, ou um primo de Merssene. Use três botões, um para cada tipo de teste.

A sua implementação deve aceitar inteiros longos e não ficar presa caso o número seja muito grande. O seu programa deve imprimir o tempo de execução em algum lugar, usando um componente adequado para exibição.

Algumas sugestões e requerimentos da implementação:

- Neste local há uma boa discusão de algoritmos para verificação de primalidade: http://en.wikipedia.org/wiki/Primality_test
- A definição de um número perfeito pode ser encontrada aqui: http://mathworld.wolfram.com/PerfectNumber.html
- Os 46 primos de Merssene conhecidos até hoje estão disponíveis neste local: http://www.mersenne.org/
- Supondo que o seu computador executa 1 gigaflop (10**9 flops) divisões, faça uma estimativa do tempo gasto para detectar que $2**61-1$ (2305843009213693951) é primo e compare-a com o tempo do seu algoritmo.

Gabarito da AD1 de Programação I – 2009/01

0.1 Números Primos

Um número primo (ou um primo) é um número natural com exatamente dois divisores naturais distintos: 1 e ele mesmo.

Somente divisores até $\lfloor\sqrt(n)\rfloor$ precisam ser testados. Isto é verdade porque se todos os inteiros menores do que este limite foram tentados, então $n/(\lfloor\sqrt(n)\rfloor + 1) < \sqrt(n)$. Em outras palavras, todos os fatores possíveis já tiveram os seus cofatores testados. Dado um fator de um número $n = ab$, o cofator de a é $b = n/a$.

Divisão por tentativa só consegue encontrar números primos pequenos. O Mathematica versão $>=$ 2.2 implementa o teste múltiplo de Rabin-Miller combinado com um teste de Lucas para encontrar pseudoprimos:

http://en.literateprograms.org/Miller-Rabin_primality_test_(C)

Sabendo-se que $2^{61} - 1$ (2305843009213693951) é primo, o algoritmo fará cerca de 2^{60} divisiões (se não for usado o limite $\lfloor\sqrt(n)\rfloor$). Considerando-se que um computador executa 10^9 divisiões por segundo (1 gigaflop), então o teste levará aproximadamente 36 anos:

- $1152921504606846976/(1000000000 * 31536000) = 36.558901085$

Usando o limite $\lfloor\sqrt(n)\rfloor$, o tempo cai para $1518500249/2 * 1000000000 = 0.759s$. Num Quadcore Q6600, o meu programa Pascal levou 17s. O mesmo programa escrito em C gastou também 17s.

Note-se que é necessário utilizar um computador com arquitetura de 64 bits, rodando um Lazarus compilado para 64 bits, para ser capaz de tratar um número desta magnitude. O mais simples é definir um tipo BigInteger para todos os inteiros do programa:

type BigInteger = Int64;

Para medir o tempo de execução do programa, pode ser usada a função *Time()*, para obter-se a diferença de tempo entre dois instantes do programa (início e fim do processamento).

Há vários prêmios para aqueles que encontrarem primos grandes, como o prêmio de \$250.000 dólares para o primeiro indivíduo ou grupo que encontrar o primeiro primo com pelo menos 1.000.000.000 de dígitos decimais.

0.2 Primos de Mersenne

Esta é a lista dos 46 primos conhecidos p, para os quais $M(p) = 2^p - 1$ é um primo de Mersenne. Pode haver outros entre o 39º e o 46º não encontrados ainda:

> mprimes = [2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593, 13466917, 20996011, 24036583, 25964951, 30402457, 32582657, 37156667, 43112609]

Euclides provou que sempre que $2^n - 1$ for primo, então $2^{(n-1)}(2^n - 1)$ é perfeito, e Euler mostrou que todos os números perfeitos ímpares são da forma, $2^{(n-1)}(2^n - 1)$. Assim, a lista acima permite gerar todos os 46 números perfeitos conhecidos, também.