

Álgebra A

Trabalho Prático 2: Primalidade

Questão 1. Implemente uma função que verifica se um número n passa no teste de Miller com base a . Para facilitar, sua função irá receber alguns valores redundantes que simplificam as contas (ver tabela abaixo).

Valor de retorno 0 se o número é definitivamente composto,
 1 se o número é primo ou pseudoprimo forte
 para a base a .

Assinatura `int talvez_primo(const mpz_t a,`
 `const mpz_t n,`
 `const mpz_t n1,`
 `unsigned int t,`
 `const mpz_t q)`

	Nome	Tipo	Descrição
Entrada	a	<code>mpz_t</code>	Base do teste de primalidade
	n	<code>mpz_t</code>	Número cuja primalidade está sendo testada
	$n1$	<code>mpz_t</code>	Variável auxiliar: $n1 = n - 1$
	t	<code>unsigned int</code>	Variável auxiliar: $n - 1 = 2^t q$.
	q	<code>mpz_t</code>	Variável auxiliar: $n - 1 = 2^t q$.

Como testar: Além dos casos usuais, não esqueça de testar os casos $a = n$ e $n \mid a$. Ao contrário da simplificação vista em sala, a não é necessariamente menor que n .

Questão 2. Implemente a função `provavelmente_primo`, com o teste de primalidade de Miller–Rabin: Dado um número de iterações `iter` e um número `n`, seu programa deve, por `iter` vezes, gerar uma base aleatória `a` entre 2 e $n - 1$ e usar a função `talvez_primo` para verificar a primalidade de `n`. Como mencionado em sala, a probabilidade dessa função errar é no máximo 4^{-iter} .

Recomenda-se ler o Apêndice A, que fala de números aleatórios e contém a função `void numero_aleatorio(mpz_t r, const mpz_t n)`, cujo resultado é um número aleatório `r` entre 1 e `n`.

Valor de retorno 0 se o número é definitivamente composto,
1 se o número é provavelmente primo.

Assinatura `int provavelmente_primo(const mpz_t n,`
`unsigned int iter,`
`gmp_randstate_t rnd)`

	Nome	Tipo	Descrição
Entrada	<code>n</code>	<code>mpz_t</code>	Número a testar primalidade.
	<code>iter</code>	<code>unsigned int</code>	Número de iterações do teste de Miller.
E/S	<code>rnd</code>	<code>gmp_randstate_t</code>	O estado do gerador aleatório.

Como testar: Você pode comparar sua resposta com o resultado da função do GMP `mpz_probab_prime_p`, que tem a mesma assinatura. A função do GMP retorna um valor não-nulo (não necessariamente 1) se o número for provavelmente primo.

Questão 3. Implemente uma função que, dado $b \geq 1$, retorna um primo aleatório no intervalo $[2, 2^b)$. A função `mpz_urandomb(r, rnd, b)` gera um número aleatório com até `b` bits e colocá-lo na variável `r` (ou seja, $0 \leq r < 2^b$). O teste de primalidade que você usar deve ter probabilidade no máximo 4^{-20} de dizer que um número é primo erroneamente.

Valor de retorno Não há.

Assinatura `void primo_aleatorio(mpz_t r,`
`unsigned int b,`
`gmp_randstate_t rnd)`

	Nome	Tipo	Descrição
Entrada	<code>b</code>	<code>unsigned int</code>	
Saída	<code>r</code>	<code>mpz_t</code>	Um número primo aleatório entre 2 e 2^b .
E/S	<code>rnd</code>	<code>gmp_randstate_t</code>	O estado do gerador aleatório.

Dica: Tome cuidado para não enviesar sua escolha de primos. Um primo `p` tal que `p - 2` também é primo deve ser gerado com mesma probabilidade que um número `p` que é antecedido de muitos números compostos.

A Números aleatórios

No GMP, o estado do gerador de números aleatório é explícito. Você deve criar **uma única variável** do tipo `gmp_randstate_t`, inicializá-la e passá-la para todas as funções que podem precisar de números aleatórios.

O gerador de números aleatórios requer um *seed* para inicializar o processo, e irá gerar sempre os mesmos números se o *seed* for fixo. Isso parece ruim, mas é ótimo para debugar. Não esqueça de escolher seu próprio *seed* (ao invés de 12394781 no código abaixo) na hora de gerar suas chaves!

O seguinte programa lê um número n da entrada e imprime um número aleatório no intervalo $[1, n]$. Você pode usá-lo como exemplo.

```
#include <stdio.h>
#include <gmp.h>

void numero_aleatorio(mpz_t r, const mpz_t n, gmp_randstate_t rnd) {
    mp_bitcnt_t num_bits = mpz_sizeinbase(n, 2);
    do {
        mpz_urandomb(r, rnd, num_bits);
    } while (!(mpz_cmp_ui(r, 1) >= 0 && mpz_cmp(r, n) <= 0));
}

int main(int argc, char **argv) {
    gmp_randstate_t rnd;
    gmp_randinit_default(rnd);
    gmp_randseed_ui(rnd, 12394781);

    mpz_t n, aleatorio;
    mpz_init(n);
    mpz_init(aleatorio);

    gmp_scanf("%Zd", n);
    numero_aleatorio(aleatorio, n, rnd);
    gmp_printf("%Zd\n", aleatorio);

    mpz_clear(aleatorio);
    mpz_clear(n);
}
```