Teoria dos Números

Números primos

Prof. Edson Alves

Faculdade UnB Gama

Sumário

- 1. Números Primos
- 2. O crivo de Erastótenes

Números Primos

Números primos

Definição de número primo

Seja p um número inteiro positivo. Dizemos que p é **primo** se ele possui exatamente dois divisores positivos: o próprio p e o número 1.

Um número natural n > 1 que não é primo é denominado número **composto**.

Consequências da definição de primos

- O número 1 não é primo, pois possui um único divisor positivo
- O menor número primo, e o único que é par, é o número 2
- Os próximos números primos são, a saber, 3, 5, 7, 11, 13, 17, 19, 23, ...
- ullet Se p e q são primos e p divide q, então p=q
- $\bullet\,$ Se p é primo e p divide o produto ab, então p divide a ou p divide b

Identificação de números primos

- ullet Para se determinar se um inteiro positivo n é ou não primo pode-se recorrer diretamente à definição de primos
- $\bullet\,$ A verificação consiste em uma busca completa nos possíveis divisores de n
- \bullet Caso seja encontrado um divisor de n que seja diferente de 1 ou do próprio n, então n será composto

Identificação de primos com complexidade O(n)

```
7 bool is_prime(int n)
8 {
      if (n < 2)
          return false;
10
      for (int i = 2; i < n; ++i)
12
          if (n % i == 0)
              return false;
14
15
      return true;
16
17 }
```

Complexidade da identificação de números primos

- A rotina is_prime(), embora seja de fácil entendimento e codificação, tem complexidade ${\cal O}(n)$
- Há ainda o agravante que a principal operação realizada no laço é a divisão inteira, a qual é computacionalmente exigente
- A divisão contrasta com a adição e a multiplicação as quais podem, em geral, ser realizadas em um ou dois ciclos do processador
- Também são realizadas muitas operações desnecessárias
- ullet Por exemplo, se n for impar, qualquer tentativa de se encontrar um divisor par de n é infrutifera

Eliminação de operações desnecessárias

```
19 bool is_prime2(int n)
20 {
     if (n < 2)
          return false;
     if (n == 2)
24
          return true:
25
26
     if (n % 2 == 0)
          return false;
28
29
      for (int i = 3: i < n: i += 2)
          if (n % i == 0)
31
              return false;
32
33
      return true;
34
35 }
```

Redução na complexidade da identificação de primos

- Embora a rotina is_prime2() reduza a quantidade de operações em relação à rotina is_prime(), a complexidade não foi reduzida, permanecendo em O(n)
- Para reduzir a complexidade, é preciso observar que deve-se procurar por possíveis divisores d tais que $d \leq \sqrt{n}$
- Isto se deve ao fato de que se d divide n, então n=dk, e ou d ou k deve ser menor ou igual à raiz quadrada de n
- ullet Se ambos fossem maiores o produto dk seria maior do que n, uma contradição

Verificação de primalidade em $O(\sqrt{n})$

```
37 bool is_prime3(int n)
38 {
     if (n < 2)
          return false;
41
     if (n == 2)
42
          return true:
43
44
     if (n % 2 == 0)
45
          return false;
46
47
      for (int i = 3; i * i <= n; i += 2)
48
          if (n % i == 0)
              return false;
50
      return true;
52
53 }
```

Nova redução na complexidade da verificação de primalidade

- \bullet A rotina is_prime3() tem complexidade $O(\sqrt{n})$
- Observe que o teste do laço não utiliza a rotina sqrt(), para evitar erros de precisão e melhorar o tempo de execução
- É possível reduzir a complexidade uma vez mais, uma vez que os candidatos à divisores de is_prime3() são os ímpares entre 3 e \sqrt{n}
- ullet Suponha que a listagem P de todos os números primos seja conhecida
- Um algoritmo que utiliza apenas os elementos de P como candidatos a divisores tem complexidade $O(\pi(\sqrt{n}))$

Função $\pi(n)$

Definição da função $\pi(n)$

Seja n um inteiro positivo. A função $\pi(n)$ retorna o número de primos menores ou iguais a n.

O cálculo de $\pi(n)$ não é trivial, mas este valor pode ser aproximado:

$$\pi(n) \approx \frac{n}{\log n}$$

O crivo de Erastótenes

Listagem dos N primeiros primos

- Na prática, para se verificar se um ou poucos números são primos, is_prime3() é suficiente
- Para verificar um conjunto de inteiros n, pode ser útil gerar uma lista de primos de antemão, a qual permitirá a identificação imediata de números presentes nesta listagem
- Uma maneira de se listar os N primeiros primos seria iterar sobre os inteiros do intervalo [1,N] e, para cada um deles, invocar a rotina is_prime3()
- A complexidade deste algoritmo seria $O(N \times \sqrt{N})$.

Listagem dos N primeiros primos em $O(N^{3/2})$

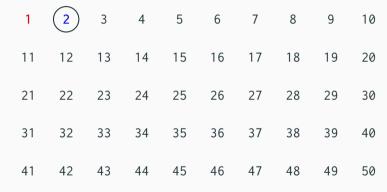
```
55 vector<int> primes(int N)
56 {
      vector<int> ps;
58
      for (int i = 2; i \le N; ++i)
59
          if (is_prime3(i))
60
               ps.push_back(i);
61
      return ps;
64 }
```

O Crivo de Erastótenes

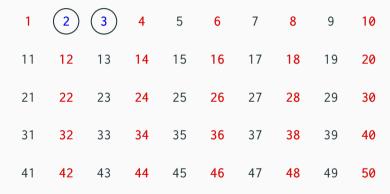
- Contudo, há uma forma mais eficiente de gerar esta lista: o Crivo de Erastótenes
- A ideia do crivo é eliminar os números compostos, os quais podem ser identificados imediatamente como múltiplos de um primo
- ullet Para isto, são listados os N primeiros naturais
- A cada iteração do crivo, é identificado o próximo número primo e todos seus múltiplos são eliminados da lista
- Ao final do algoritmo a lista conterá apenas números primos

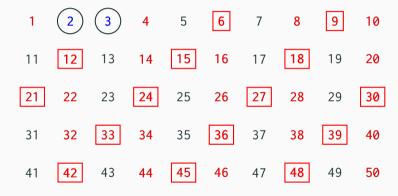
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

















- Como o próximo número primo, a saber 11, é maior do que a raiz quadrada de 50, o processo pode ser interrompido
- $\bullet\,$ Os números não crivados formam a relação de todos os primos menores ou iguais a N
- Uma implementação do crivo de Erastótenes em C++ pode o vetor de bits sieve para marcar os números
- Zero ou falso indica que o número não é primo

Implementação do crivo em C++

```
66 vector<int> primes2(int N) {
     vector<int> ps;
67
     bitset<MAX> sieve;
                        // MAX deve ser maior do que N
     sieve.set();
                                   // Todos são "potencialmente" primos
    sieve[1] = false:
                                 // 1 não é primo
71
     for (int i = 2; i \le N; ++i) {
72
         if (sieve[i]) {      // i é primo
             ps.push_back(i);
74
75
             for (int j = 2 * i; j \le N; j += i)
76
                 sieve[j] = false;
78
79
80
     return ps;
81
82 }
```

Aproximação para a complexidade do crivo

- ullet Na rotina primes2(), para cada i são crivados N/i números
- Portanto o número total T(N) de operações é aproximadamente N vezes o N-ésimo número harmônico H_N , isto é,

$$T(N) \approx N + \frac{N}{2} + \frac{N}{3} + \ldots + \frac{N}{N} = N \times H_N = N \left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N} \right) \le N \log N$$

• A desigualdade vale porque

$$\log N = \int_{1}^{N} \frac{1}{x} dx \ge \sum_{i=1}^{N} \frac{1}{i}$$

Complexidade do Crivo de Erastótenes

- Na aproximação para T(N) a variável i assume todos os naturais no intervalo [1,N], inclusive números compostos
- Porém na implementação i assume apenas valores primos
- Uma melhor aproximação seria, pelo Teorema de Merten,

$$\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \frac{N}{7} + \frac{N}{11} + \dots \le N \log \log N$$

• Logo a complexidade do crivo é $O(N\log\log N)$

Redução da constante de complexidade

- A implementação de primes2() é simples e direta
- É possível, contudo, diminuir a constante de complexidade e obter um melhor tempo de execução
- Primeiramente, os números pares podem ser tratados à parte: 2 é o único primo par, e os demais pares compostos não contribuem para o crivo
- O número 1 pode ser desprezado, uma vez que a saída da rotina é uma lista de primos

Crivo com tratamento diferente para pares

```
84 vector<int> primes3(int N)
85 {
     bitset<MAX> sieve;
                                // MAX deve ser maior do que N
                                         // Os pares são tratados à parte
     vector<int> ps { 2 };
     sieve.set();
                                         // Todos são "potencialmente" primos
88
89
     for (int i = 3; i \le N; i += 2) { // Apenas impares são verificados agora
90
         if (sieve[i]) {
                          // i é primo
91
             ps.push_back(i);
93
             for (int j = 2 * i; j \le N; j += i)
                 sieve[i] = false:
96
97
98
     return ps;
100 }
```

Nova redução da constante de complexidade

- Embora o laço externo de primes3() só considere números ímpares, o laço interno itera por pares desnecessariamente
- Outra observação importante: o crivo deve começar no quadrado de i, pois quaisquer múltiplos de i menores do que i^2 já foram crivados
- Estas duas observações reduzem novamente a constante de complexidade
- Para evitar problemas de overflow na condição do laço interno, o tipo de dado foi alterado de int para long long

Crivo sem pares nos laços

```
vector<long long> primes4(long long N)
103 {
      bitset<MAX> sieve;
                              // MAX deve ser maior do que N
104
      vector<long long> ps { 2 };  // Os pares são tratados à parte
105
     sieve.set();
                                          // Todos são "potencialmente" primos
106
107
      for (long long i = 3; i \le N; i += 2) { // Apenas impares são verificados agora
108
          if (sieve[i]) {
                                        // i é primo
              ps.push_back(i);
110
111
              for (long long j = i * i; j \le N; j += 2*i) // Múltiplos ímpares >= i*i
112
                  sieve[i] = false:
113
114
115
116
      return ps;
117
118 }
```

Tratamento de múltiplos de 3 à parte

- Assim como foi feito para os pares, os múltiplos de 3 também podem ser tratados à parte
- A inclusão prévia do 3 na lista de primos é a parte trivial: difícil é evitar os múltiplos de 3 no laço externo
- Isto pode ser feito observando que, seguindo a sequência dos ímpares a partir de
 3, primeiro há um múltiplo de 3, depois um número cujo resto da divisão por 3 é
 2, por fim um número cuja divisão por 3 é 1, e o ciclo se reinicia
- Os múltiplos de 3, portanto, podem ser ignorados

Crivo com múltiplos de 2 e 3 tratados à parte

```
vector<long long> primes5(long long N)
121 {
     bitset<MAX> sieve: // MAX deve ser maior do que N
     vector<long long> ps { 2, 3 }; // Pares e múltiplos de 3 são tratados à parte
123
                                 // Todos são "potencialmente" primos
     sieve.set():
124
125
     // O incremento alterna entre saltos de 2 ou 4, evitando os múltiplos de 3
126
     for (long long i = 5, step = 2; i \le N; i += step, step = 6 - step) {
127
          if (sieve[i]) {
                                                            // i é primo
128
              ps.push_back(i);
30
              for (long long i = i * i: i \le N: i += 2*i) // Múltiplos impares >= i*i
131
                  sieve[j] = false;
132
133
134
     return ps;
35
36 }
```

Verificação das implementações do crivo

Uma maneira de verificar rapidamente se o crivo está produzindo os primos corretamente é checar o número de primos gerados, segundo a tabela abaixo:

$\pi(n)$
4
25
168
1229
9592
78498
664579

Possível saída para as rotinas de teste de primalidade

```
==== Testes de primalidade:
is_prime(999983) = 1 (0.010074394000000 ms)
is_prime2(999983) = 1 (0.005721907000000 ms)
is_prime3(999983) = 1 (0.000006486000000 ms)
==== Geração de primos até N:
primes(10000000) = 664579 (2.428975496000000 ms)
primes2(10000000) = 664579 (0.172493493000000 ms)
primes3(10000000) = 664579 (0.136014180000000 ms)
primes4(10000000) = 664579 (0.067260405000000 ms)
primes5(10000000) = 664579 (0.059135050000000 ms)
```

Problemas propostos

- 1. AtCoder Beginner Contest 096D Five, Five Everywhere
- 2. AtCoder Beginner Contest 149C Next Prime
- 3. Codeforces 327B Hungry Sequence
- 4. OJ 543 Goldbach
- 5. OJ 11001 Necklace

Referências

- 1. The PrimesPage. How Many Primes Are There?. Acesso em 08/11/2017.
- 2. Wikipédia. Harmonic Number. Acesso em 08/11/2017.
- 3. Wikipédia. Mertens' theorems. Acesso em 08/11/2017.