Teoria dos números

Bruno Paiva de Oliveira

Esdras de Lima Chaves

Antonio Carlos Neto

Introdução

- Teoria dos números é o ramo da matemática pura que estuda propriedades dos números em geral, em particular dos números inteiros e a classe de problemas que surge no seu estudo.

- Grande parte dos estudos estão voltados para os números primos.

Maratona - Competitive Programming

- A Teoria dos Números é uma importante área a ser estudada, pois torna alguns problemas matemáticos em problemas fáceis de serem resolvidos (ou mais fáceis) se você sabe a Teoria por trás dos problemas.
- Caso contrário, podemos lidar com esses problemas utilizando força bruta, que provavelmente irá levar a um Tempo de Limite Excedido ou você não conseguirá trabalhar com um dado input sem um pré processamento.

Relembrando...

- **Números Primos -** Um número natural a partir do 2 é considerado primo se, e somente se, ele só é divisível por 1 e ele mesmo. Exemplo de números primos: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...}.

- Número primo é um tópico importante em Teoria dos Números e a fonte de diversos problemas de programação. A próxima seção destina-se a falar de alguns algoritmos envolvendo números primos.

É Primo?

- [2,N-1] : O(n);
- [2,√N] : O(√n);
- **2 && [3,√N] passo 2: O((√N)/2);**

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  int number, raiz_number, isPrime = 1;

  printf("Enter the number\n");
  scanf("%d",&number);

  raiz_number = sqrt(number);

  for(int i = 3; i <= raiz_number; i += 2)
    if (number%i == 0){
      isPrime = 0;
      break;
    }

  if (number == 1 || !isPrime || (number != 2 && number%2 == 0))
    printf("O numero %d, nao eh primo\n", number);
  else
    printf("O numero %d, eh primo\n", number);

  return 0;
}
```

É Primo?

- [2,N-1] : O(n);
- [2,√N] : O(√n);
- 2 && [3,√N] passo 2: O((√N)/2);
- **Intervalo de primos : O(√N/ln(√N)); ***

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  int number, qt_primos, k = 0, raiz_number, isPrime = 1;

  printf("Enter the number\n");
  scanf("%d",&number);

  raiz_number = sqrt(number);
  qt_primos = raiz_number - 1;

  int* primes = (int*) malloc((raiz_number+1)*sizeof(int));
  //Setando todos valores como falso
  for(int j = 0; j <= raiz_number; j++) primes[j] = 0;

  for(int i = 2; i <= raiz_number; i++)
    if(primes[i] == 0)
      for(int j = i*2; j <= raiz_number; j += i)
        if(primes[j] == 0){
          primes[j] = j;
          qt_primos--;
        }
```

É Primo?

- [2,N-1] : O(n);
- [2,√N] : O(√n);
- 2 && [3,√N] passo 2: O((√N)/2);
- **Intervalo de primos : O(√N/ln(√N)); ***

Sieve of Eratosthenes

```
27    int* lista_primos = malloc(qt_primos*sizeof(int));
28    for(int i = 2; i <= raiz_number; i++){
29      if (primes[i] == 0)
30        lista_primos[k++] = i;
31    }
32
33    for(int i = 0; i < qt_primos; i++)
34      if(number % lista_primos[i] == 0){
35        isPrime = 0;
36        break;
37      }
38
39    if (number == 1 || !isPrime)
40      printf("O numero %d, nao eh primo\n", number);
41    else
42      printf("O numero %d, eh primo\n", number);
43
44    return 0;
45  }
```

MDC - Máximo divisor comum
MMC - Mínimo divisor comum

Normalmente encontrar o MDC não é o principal objetivo de um problema relacionado com matemática, mas apenas uma parte do todo.

O MDC de mais de dois inteiros:

mdc(a,b,c) = mdc(a, mdc(b,c))

O MDC de dois inteiros ***a*** e ***b***, denotado por ***mdc(a,b)***, é o maior inteiro positivo ***d*** tal que:

- ***d*** | ***a*** (***d*** divide ***a***)
- ***d*** | ***b*** (***d*** divide ***b***)

Exemplos:
mdc(4,8) = 4
mdc(6,9) = 3

Um uso prático do MDC é para simplificar frações:
6/9 = (6 / mdc(6,9)) / (o /mdc(6,9)) = ⅔

É possível encontrar o ***mdc*** de dois inteiros através do algoritmo Divisão e Conquista de Euclides.

MDC - Máximo divisor comum

MMC - Mínimo divisor comum

O MMC de mais de dois inteiros:

mmc(a,b,c) = mmc(a, mmc(b,c))

O MMC de dois inteiros ***a*** e ***b***, denotado por ***mmc(a,b)***, é o menor inteiro positivo ***d*** tal que:

- ***a*** | ***d*** (***a*** divide ***d***)
- ***b*** | ***d*** (***b*** divide ***d***)

Exemplos:
mmc(4,8) = 8
mmc(6,9) = 18

A seguinte regra é válida:
mmc(a,b) = a * b / mdc(a,b)

Implementação:

```
int mmc(int a, int b) {
        return a * (b / mdc(a, b));
}
```

Algoritmo de Euclides

O algoritmo de Euclides é simples e eficiente para calcular o MDC. É um dos algoritmos mais antigos, por volta de 300 a.C.

O algoritmo é baseado no princípio de que o MDC não muda se o menor número for subtraído ao maior. Exemplo:

mdc(252,105) = 21

mdc(147,105) = 21

Como o maior dos dois números é reduzido, a repetição desse processo irá gerar sucessivamente números menores, até convergir em zero. Nesse momento, o **MDC é o outro número inteiro, maior que zero**.

Algoritmo de Euclides

Versão recursiva:

```
int euclidesMDC (int a, int b) {
        if (b == 0) return a;
        else return euclidesMDC(b, a % b);
}
```

Versão iterativa:

```
int euclidesMDC (int a, int b) {
        int resto;
        while (b != 0) {
                resto = a % b;
                a = b;
                b = resto;
        }
        return a;
}
```

Fatorial

Fatorial de N é representado por N!

N! = Nx(N-1)x...x2x1

Fatorial Iterativo

```c
#include <stdio.h>
int main()
{
    int n, i;
    unsigned long long factorial = 1;

    printf("Enter an integer: ");
    scanf("%d",&n);

    // show error if the user enters a negative integer
    if (n < 0)
        printf("Error! Factorial of a negative number doesn't exist.");

    else
    {
        for(i=1; i<=n; ++i)
        {
            factorial *= i;              // factorial = factorial*i;
        }
        printf("Factorial of %d = %llu", n, factorial);
    }

    return 0;
}
```

Fatoração em potências primas

N = N/PF

PF <= √N

Dividir em fatores menores economiza tempo!

```c
#include <stdio.h>
#include <math.h>

int main(int argc, char const *argv[]) {
  unsigned long long number;
  int potencia = 0;

  printf("Enter an integer: ");
  scanf("%llu",&number);

  printf("Prime Factors\n");
  while(number%2 == 0){
    number = number/2;
    potencia++;
  }
  if (potencia != 0)
    printf("2^%d\n", potencia);

  for (int i = 3; i <= number; i = i+2){
    potencia = 0;
    while (number%i == 0){
      number = number/i;
      potencia++;
    }
    if(potencia != 0)
      printf("%d^%d\n", i,potencia);
  }
  return 0;
}
```

Funções com Fatores Primos

- **numPF**
- numDiffPF
- sumPF
- numDiv
- sumDiv

```c
#include <stdio.h>
#include <math.h>

int main() {
  unsigned long long number;
  int numPF = 0;

  printf("Enter an integer: ");
  scanf("%llu",&number);

  for (int i = 2; i <= number; i = i+2){
    while (number%i == 0){
      number = number/i;
      numPF++;
    }
    if(i == 2)
      i--;
  }

  printf("Numero de Fatores Primos: %d\n",numPF );
  return 0;
}
```

Funções com Fatores Primos

- numPF
- **numDiffPF**
- sumPF
- numDiv
- sumDiv

```c
#include <stdio.h>
#include <math.h>

int main() {
  unsigned long long number;
  int potencia = 0, numPF = 0;

  printf("Enter an integer: ");
  scanf("%llu",&number);

  for (int i = 2; i <= number; i = i+2){
    potencia = 0;
    while (number%i == 0){
      number = number/i;
      potencia++;
    }
    if(potencia != 0)
      numPF++;
    if(i == 2)
      i--;
  }

  printf("Numero de Fatores Primos Distintos: %d\n",numPF );
  return 0;
}
```

Funções com Fatores Primos

- numPF
- numDiffPF
- **sumPF**
- numDiv
- sumDiv

```c
#include <stdio.h>
#include <math.h>

int main() {
  unsigned long long number;
  int sumPF = 0;

  printf("Enter an integer: ");
  scanf("%llu",&number);

  for (int i = 2; i <= number; i = i+2){
    while (number%i == 0){
      number = number/i;
      sumPF += i;
    }
    if(i == 2)
      i--;
  }

  printf("Soma dos Fatores Primos: %d\n",sumPF );
  return 0;
}
```

Funções com Fatores Primos

- numPF
- numDiffPF
- sumPF
- **numDiv**
- sumDiv

```c
#include <stdio.h>
#include <math.h>

int main() {
  unsigned long long number;
  int potencia = 0, power = 1;

  printf("Enter an integer: ");
  scanf("%llu",&number);

  for (int i = 2; i <= number; i = i+2){
    potencia = 0;
    while (number%i == 0){
      number = number/i;
      potencia++;
    }
    if(potencia != 0)
      power *= (potencia + 1);
    if(i == 2)
      i--;
  }

  printf("Numero de Divisores: %d\n",power);
  return 0;
}
```

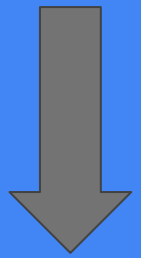
Funções com Fatores Primos

- numPF
- numDiffPF
- sumPF
- numDiv
- **sumDiv**

```c
#include <stdio.h>
#include <math.h>

int main() {
  unsigned long long number;
  int potencia = 0, power = 1, aux;
  printf("Enter an integer: ");
  scanf("%llu",&number);

  for (int i = 2; i <= number; i = i+2){
    potencia = 0;
    while (number%i == 0){
      number = number/i;
      potencia++;
    }
    if(potencia != 0){
      aux = i;
      for(int p = 0; p < potencia; p++)
        aux *= i;
      power *= ((aux - 1)/(i-1));
    }
    if(i == 2)
      i--;
  }

  printf("Soma de Divisores: %d\n",power);
  return 0;
}
```

Mudança de Base

Qualquer base

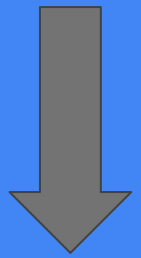


Decimal

```
int paraDecimal(char *str, int base)
{
    int tamanhoEntrada = strlen(str);
    int power = 1;
    int num = 0;
    int i;

    for (i = len - 1; i >= 0; i--)
    {
        if (val(str[i]) >= base)
        {
           printf("Invalid Number");
           return -1;
        }

        num += val(str[i]) * power;
        power = power * base;
    }
    return num;
}
```

Mudança de Base

Qualquer base

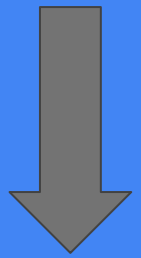


Decimal

```
int val(char c)
{
    if (c >= '0' && c <= '9')
        return (int)c - '0';
    else
        return (int)c - 'A' + 10;
}


int main()
{
    char str[] = "11A";
    int base = 16;
    printf("O equivalente de %d na base %d em decimal eh: %d\n", str, base, paraDecimal(str, base));
    return 0;
}
```

Mudança de Base

Decimal

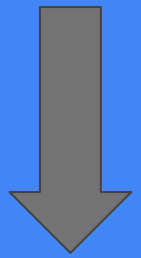


Qualquer base

```
char* doDecimal(char res[], int base, int
inputNum)
{
    int index = 0;

    while (inputNum > 0)
    {
        res[index++] = reVal(inputNum % base);
        inputNum /= base;
    }
    res[index] = '\0';

    reverseString(res);

    return res;
}
```

Mudança de Base

Decimal

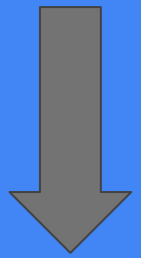


Qualquer base

```
char reVal(int num)
{
    if (num >= 0 && num <= 9)
        return (char)(num + '0');
    else
        return (char)(num - 10 + 'A');
}

void reverseString(char *str)
{
    int len = strlen(str);
    int i;
    for (i = 0; i < len/2; i++)
    {
        char temp = str[i];
        str[i] = str[len-i-1];
        str[len-i-1] = temp;
    }
}
```

Mudança de Base

Decimal



Qualquer base

```
int main()
{
    int inputNum = 282, base = 16;

    char res[100];

    printf("O valor decimal de %d na base %d eh: %d \n", inputNum, base, fromDeci(res, base, inputNum));

    return 0;
}
```

Módulo

Como visto anteriormente, existem diversos problemas que podem ser resolvidos utilizando o módulo.

Em alguns problemas, no entanto, só estamos interessados no resultado do módulo, tais que seu resultado intermediário e final caiba em um inteiro.

Exemplo: UVa 10176 - Ocean Deep! Make it Shallow!!

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

#define FOR(i, n) for (__typeof(n)i = 0; i < n; i++)

const int MOD = 131071;


int main() {
        char a[] = "cadeia_de_100_bits";
        int i = 0;
        int M = 0;
        FOR(i, strlen(a)) {
                M = (M << 1) +  a[i] - '0';
                M %= MOD ;
        }
}
```

Equação Diofantina

Dado a, b. Extend Euclidean calcular MDC(a, b) e os coeficientes x, y. tal que:

ax + by = MDC(a, b)

d = MDC(a, b)

Motivação: Suponha que você compra maçãs e laranjas e paga um total de R$ 8,39. Uma maçã custa R$ 0,25 e uma laranja R$ 0,18. Quantas frutas de cada você compra?

Podemos representar o problema pela equação:

25x + 18y = 839

```
int x;
int y;

void ExtendedEuclid(int a, int b) {
        if(b == 0) {
                x = 1;
                y = 0;
                d = a;
                return;
        }
        ExtendedEuclid(b, a%b);
        int x1 = y;
        int y1 = x - (a / b) * y;
        x = x1;
        y = y1;
}

void next(int *x, int *y, int a, int b) {
        // n é qual resultado da “lista” de possíveis resultados você quer
        *x = x0 + (b/d) * n;
        *y = y - (a/d) * n;
}
```

Função Totiente

Dado um número inteiro n, a função Totiente retorna a quantidade de números inteiros, no intervalo de 1 a n, que são coprimos de n. Formalmente:

O número de inteiros k no intervalo 1<= k <= n, tal que MDC(k, n) = 1

```
#include <stdio.h>

int phi(int n) {
  int result = n;
  for (int i = 2; i * i <= n; i++) {
     if(n % i == 0) {
        while(n % i == 0)
           n /= i;
        result -= result / i;
     }
  }
  if(n > 1)
     result -= result / n;
  return result;
}
```

Obrigado!

Referências:

-Halim, Steven. Competitive Programming, 3rd Edition;

-https://www.geeksforgeeks.org/convert-base-decimal-vice-versa/