Taller ProgComp: Track Matemática

Números Primos

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María,
Università di Pisa

April 8, 2024

Part I

Criba de Eratosthenes

Criba de Eratosthenes

Problema

Dado un número entero n, encontrar todos los números primos entre $[1, n]$.

CRIBA DE ERATOSTHENES

Problema

Dado un número entero n, encontrar todos los números primos entre $[1, n]$.

¿Cómo encontrarlos de manera eficiente?

ALGORITMO

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

ALGORITMO

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~

ALGORITMO

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~

ALGORITMO

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~

Algoritmo

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~

Algoritmo

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~

ALGORITMO

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~

CÓDIGO

```cpp
vector< bool > criba(int n) {
  vector< bool > is_prime(n+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int i = 2; i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
      for (int j = i * i; j <= n; j += i)
          is_prime[j] = false;
    }
  }
  return is_prime;
}
```

Código

```
vector< bool > criba(int n) {
  vector< bool > is_prime(n+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int i = 2; i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
      for (int j = i * i; j <= n; j += i)
          is_prime[j] = false;
    }
  }
  return is_prime;
}
```

¿Cuál es la complejidad?

CÓDIGO

```
vector< bool > criba(int n) {
  vector< bool > is_prime(n+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int i = 2; i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
      for (int j = i * i; j <= n; j += i)
          is_prime[j] = false;
    }
  }
  return is_prime;
}
```

¿Cuál es la complejidad? $O(n \log \log n)$

Un poco de análisis

¿Se podrá mejorar? Mmmm...

Un poco de análisis

¿Se podrá mejorar? Mmmm...
Se itera de todos los números de 2 hasta n, pero ¿es realmente necesario?

UN POCO DE ANÁLISIS

¿Se podrá mejorar? Mmmm...
Se itera de todos los números de 2 hasta n, pero ¿es realmente necesario? No!

Un poco de análisis

¿Se podrá mejorar? Mmmm...
Se itera de todos los números de 2 hasta *n*, pero ¿es realmente necesario? No!
Solo es necesario chechear todos lo números que no sobrepasen la raiz. :o

```
vector< bool > criba(int n) {
  vector< bool > is_prime(n+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int i = 2; i * i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
      for (int j = i * i; j <= n; j += i)
          is_prime[j] = false;
    }
  }
  return is_prime;
}
```

¿Cuál es la nueva complejidad?

UN POCO DE ANÁLISIS

¿Se podrá mejorar? Mmmm...
Se itera de todos los números de 2 hasta n, pero ¿es realmente necesario? No!
Solo es necesario chechear todos lo números que no sobrepasen la raiz. :o

```
vector< bool > criba(int n) {
  vector< bool > is_prime(n+1, true);
  is_prime[0] = is_prime[1] = false;
  for (int i = 2; i * i <= n; i++) {
    if (is_prime[i] && (long long)i * i <= n) {
      for (int j = i * i; j <= n; j += i)
          is_prime[j] = false;
    }
  }
  return is_prime;
}
```

¿Cuál es la nueva complejidad? $O(n \log \log \sqrt{n})$

Part II

Primality Tests

Primo o no primo

Problema

Dado un entero n, queremos saber si ese número es primo o no.

Ejemplo

- $n = 123$ no es primo
- $n = 5$ es primo

Primera idea

Podriamos pasar por todos los números de 1 a n y ver si alguno lo divide.

```
bool is_prime(int n) {
  for (int d = 2; d <= n; d++) {
    if(n % d == 0)
      return false;
  }
  return n >= 2;
}
```

¿Cuál es la complejidad?

Primera idea

Podriamos pasar por todos los números de 1 a n y ver si alguno lo divide.

```
bool is_prime(int n) {
  for (int d = 2; d <= n; d++) {
    if(n % d == 0)
      return false;
  }
  return n >= 2;
}
```

¿Cuál es la complejidad? $O(n)$
¿Será eficiente?

Primera idea

Podriamos pasar por todos los números de 1 a n y ver si alguno lo divide.

```
bool is_prime(int n) {
  for (int d = 2; d <= n; d++) {
    if(n % d == 0)
      return false;
  }
  return n >= 2;
}
```

¿Cuál es la complejidad? $O(n)$
¿Será eficiente?
Bueno, si:

- $n = 10^5$, no es terrible.
- $n = 10^{16}$, estamos perdidos.

UNA MEJORA

Pregunta

¿Será necesario pasar por todos los números de 1 a n?

Una mejora

Pregunta

¿Será necesario pasar por todos los números de 1 a n?

En efecto, no es necesario :o.

Optimización

Digamos que d divide a n. Entonces, existe $\frac{n}{d}$.

- Si $d \leq \sqrt{n} \Rightarrow \frac{n}{d} \geq \sqrt{n}$
- Si $d \geq \sqrt{n} \Rightarrow \frac{n}{d} \leq \sqrt{n}$

Entonces, si encontramos un divisor menor a $\sqrt{n}$, si o si existe un divisor mayor a $\sqrt{n}$.
Por lo tanto basta con ver los números hasta la raiz de n!

Una mejora

Pregunta

¿Será necesario pasar por todos los números de 1 a n?

En efecto, no es necesario :o.

Optimización

Digamos que d divide a n. Entonces, existe $\frac{n}{d}$.

- Si $d \leq \sqrt{n} \Rightarrow \frac{n}{d} \geq \sqrt{n}$
- Si $d \geq \sqrt{n} \Rightarrow \frac{n}{d} \leq \sqrt{n}$

Entonces, si encontramos un divisor menor a $\sqrt{n}$, si o si existe un divisor mayor a $\sqrt{n}$.
Por lo tanto basta con ver los números hasta la raiz de n!

```
bool is_prime(int x) {
    for (int d = 2; d * d <= x; d++) {
        if (x % d == 0)
            return false;
    }
    return x >= 2;
}
```

Complejidad: $O(\sqrt{n})$

Part III

Factorización Prima

Factorización Prima

Problema

Dado un entero n que no es primo, quiero factorizarlo en factores primos.

Ejemplo

$n = 18 \Rightarrow n = 2^1 \cdot 3^2$

Factorización Prima

Problema

Dado un entero n que no es primo, quiero factorizarlo en factores primos.

Ejemplo

$n = 18 \Rightarrow n = 2^1 \cdot 3^2$

¿Cuál es el rango que deberíamos buscar?

Factorización Prima

Problema

Dado un entero n que no es primo, quiero factorizarlo en factores primos.

Ejemplo

$n = 18 \Rightarrow n = 2^1 \cdot 3^2$

¿Cuál es el rango que deberíamos buscar? Gracias a lo que vimos antes el rango será $[2, \sqrt{n}]$!

IDEA INICIAL

```
vector<long long> fact_prima(long long n) {
  vector<long long> factorization;
  for (long long d = 2; d * d <= n; d++) {
    while (n % d == 0) {
      factorization.push_back(d);
      n /= d;
    }
  }
  if (n > 1)
    factorization.push_back(n);
  return factorization;
}
```

Pequeña Optimización

Sabemos que la mitad de los números son pares. ¿Podremos hacer algo con eso?

Pequeña Optimización

Sabemos que la mitad de los números son pares. ¿Podremos hacer algo con eso? Sí!

```
vector<long long> fact_prima(long long n) {
  vector<long long> factorization;
  while (n % 2 == 0) {
      factorization.push_back(2);
      n /= 2;
  }
  for (long long d = 3; d * d <= n; d += 2) {
    while (n % d == 0) {
      factorization.push_back(d);
      n /= d;
    }
  }
  if (n > 1)
    factorization.push_back(n);
  return factorization;
}
```

PERO SON PRIMOS...

Si queremos hacer la factorización prima, ¿por qué pasamos por todos los enteros? ¿Tendremos una forma de tener los primos que podrían haber entre 1 y n?

PERO SON PRIMOS...

Si queremos hacer la factorización prima, ¿por qué pasamos por todos los enteros? ¿Tendremos una forma de tener los primos que podrían haber entre 1 y n? Sí! La criba! Pero debemos tenerlos precomputados para que valga la pena.

```
vector<long long> fact_prima(long long n) {
  vector<long long> factorization;
  for (long long d : primes) {
    if (d * d > n)
      break;
    while (n % d == 0) {
      factorization.push_back(d);
      n /= d;
    }
  }
  if (n > 1)
    factorization.push_back(n);
  return factorization;
}
```

References I