LECTURE 07

NUMBER THEORY II

SIEVE OF ERATOSTHENES
&
EULER'S TOTIENT FUNCTION

Big-O Coding

Website: www.bigocoding.com

Giới thiệu tổng quan

Prime number (Số nguyên tố): là số nguyên dương chỉ có 2 ước số dương là 1 và chính nó.

Composite number (Hợp số): là số nguyên lớn hơn 1 có nhiều hơn 2 ước số dương.

Ví dụ:

- Số 7: chỉ có 2 ước số dương 1 và 7 ➔ Số 7 là số nguyên tố.
- Số 8: có các ước số dương là 1, 2, 4, 8 ➔ Số 8 là hợp số.

Source Code check Prime Number

```cpp
bool isPrime(int n)
{
    for (int i = 2; i*i <= n; i++)
        if (n % i == 0)
            return false;
    return n > 1;
}
```


```python
def isPrime(n):
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return n > 1
```

Độ phức tạp: $O(\sqrt{N})$.

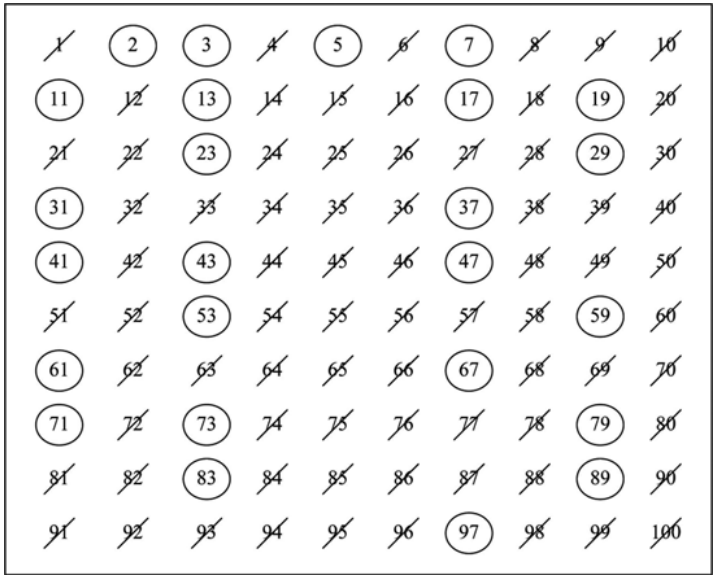
Source Code check Prime Number

```java
private static Boolean isPrime(int n) {
    for (int i = 2; i*i <= n; i++)
        if (n % i == 0)
            return false;
    return n > 1;
}
```

Độ phức tạp: $O(\sqrt{N})$.

Sieve of Eratosthenes

Sieve of Eratosthenes (Sàng nguyên tố Eratosthenes): là thuật toán dùng để tìm ra tất cả các số nguyên tố nhỏ hơn hoặc bằng số N cho trước.



Ý tưởng: Sàng Eratosthenes dựa vào tính chất của thừa số nguyên tố. Nếu 1 số là số nguyên tố thì đánh dấu lại tất cả các bội số của nó vì chắc chắn những số này không phải là số nguyên tố.

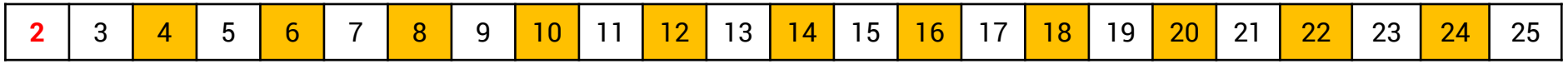
Tiếp tục quay lại tìm trong danh sách những số nào chưa bị đánh dấu và thực hiện tại như bước trên. Nếu không còn số nào thì dừng.

Sieve of Eratosthenes

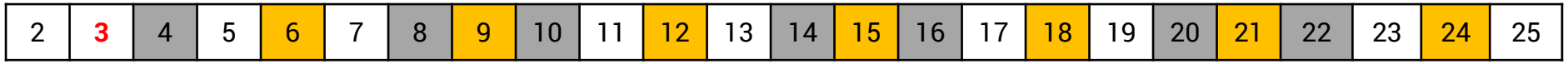
Big-O Orange

Ví dụ mô phỏng thuật toán với N = 25.

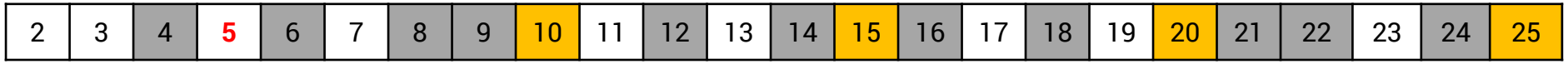
Bước 1: Số đầu tiên là số 2, đánh dấu các bội số của 2 (trừ số 2).



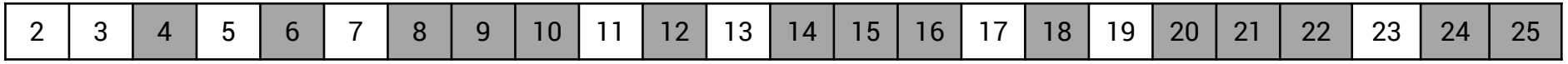
Bước 2: Tiếp theo là số 3, đánh dấu các bội số của 3 (trừ số 3).



Bước 3: Tiếp theo là số 5, đánh dấu các bội số của 5 (trừ số 5).



Chạy tương tự các bước còn lại, ta có kết quả:



Độ phức tạp: O(Nlog(logN))

Kết quả bài toán

Danh sách các số nguyên tố từ 1 đến N (N = 25):

2, 3, 5, 7, 11, 13, 17, 19, 23

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Source Code Sieve of Eratosthenes

```cpp
#include <iostream>
#include <vector>
using namespace std;
vector<int> sieveOfEratosthenes(int n)
{
    vector<bool> mark;
    vector<int> primes;
    mark.resize(n + 1, true);
    mark[0] = mark[1] = false;
    for (int i = 2; i*i <= n; i++)
    {
        if (mark[i] == true)
        {
            for (int j = i * i; j <= n; j += i)
                mark[j] = false;
        }
    }
```

Source Code Sieve of Eratosthenes

```
18.     for (int i = 2; i <= n; i++)
19.         if (mark[i] == true)
20.             primes.push_back(i);
21.       return primes;
22. }
```


```
23. int main()
24. {
25.     int n = 25;
26.     vector<int> primes = sieveOfEratosthenes(n);
27.     for (int i = 0; i < primes.size(); i++)
28.             cout << primes[i] << " ";
29.     cout << endl;
30.     return 0;
31. }
```

Source Code Sieve of Eratosthenes

```
1.  def sieveOfEratosthenes(n):
2.      mark = [True] * (n + 1)
3.      primes = []
4.      mark[0] = mark[1] = False
5.      for i in range(2, int(n ** 0.5) + 1):
6.          if mark[i] == True:
7.              for j in range(i * i, n + 1, i):
8.                  mark[j] = False
9.
10.     for i in range(2, n + 1):
11.         if mark[i] == True:
12.             primes.append(i)
13.
14.     return primes
```

Source Code Sieve of Eratosthenes

```
15. if __name__ == "__main__":
16.     n = 25
17.     primes = sieveOfEratosthenes(n)
18.     for i in range(len(primes)):
19.         print(primes[i], end = ' ')
20.     print()
```

Source Code Sieve of Eratosthenes

```java
import java.util.*;
public class Main {
    private static ArrayList<Integer> sieveOfEratosthenes(int n) {
        boolean mark[] = new boolean[n + 1];
        ArrayList<Integer> primes = new ArrayList<>();
        Arrays.fill(mark, true);
        mark[0] = mark[1] = false;
        for (int i = 2; i*i <= n; i++) {
            if (mark[i] == true) {
                for (int j = i * i; j <= n; j += i)
                    mark[j] = false;
            }
        }
        for (int i = 2; i <= n; i++)
            if (mark[i] == true)
                primes.add(i);
        return primes;
    }
```

Source Code Sieve of Eratosthenes

```
19.     public static void main(String[] args) {
20.         int n = 25;
21.         ArrayList<Integer> primes = sieveOfEratosthenes(n);
22.         for (int i = 0; i < primes.size(); i++)
23.             System.out.printf("%d ", primes.get(i));
24.     }
25. }
```

Segmented Sieve

Segmented Sieve: Dùng để tìm số các nguyên tố trong đoạn [left, right], với điều kiện $1 \leq left \leq right \leq 10^{12}$ và $right - left \leq 10^{6}$

Ý tưởng: Sàng Eratosthenes dựa vào tính chất của thừa số nguyên tố. Nếu 1 số là số nguyên tố thì đánh dấu lại tất cả các bội số của nó vì chắc chắn những số này không phải là số nguyên tố.

Tiếp tục quay lại tìm trong danh sách những số nào chưa bị đánh dấu và thực hiện tại như bước trên. Nếu không còn số nào thì dừng.

Độ phức tạp: O(Nlog(logN))

Bước 0: Chuẩn bị dữ liệu

Dùng sàng nguyên tố để khởi tạo giá trị số nguyên tố ban đầu **primes[ ]**

2	3	5	7	11	13	17	19	23	29	31	...

Mảng đánh dấu các số có phải là nguyên tố hay không **mark[ ]**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T

Left = 12 Right = 35

12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

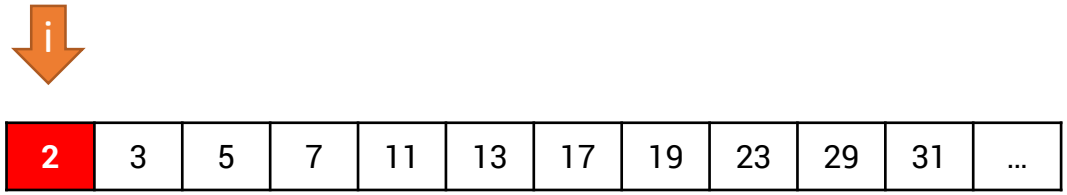
Tìm bội số của một số nguyên tố bắt đầu từ số (number) bất kỳ **base**

base = (number / prime) * prime

Nếu base < number ➔ base = base + number

Bước 1: Chạy thuật toán prime = 2

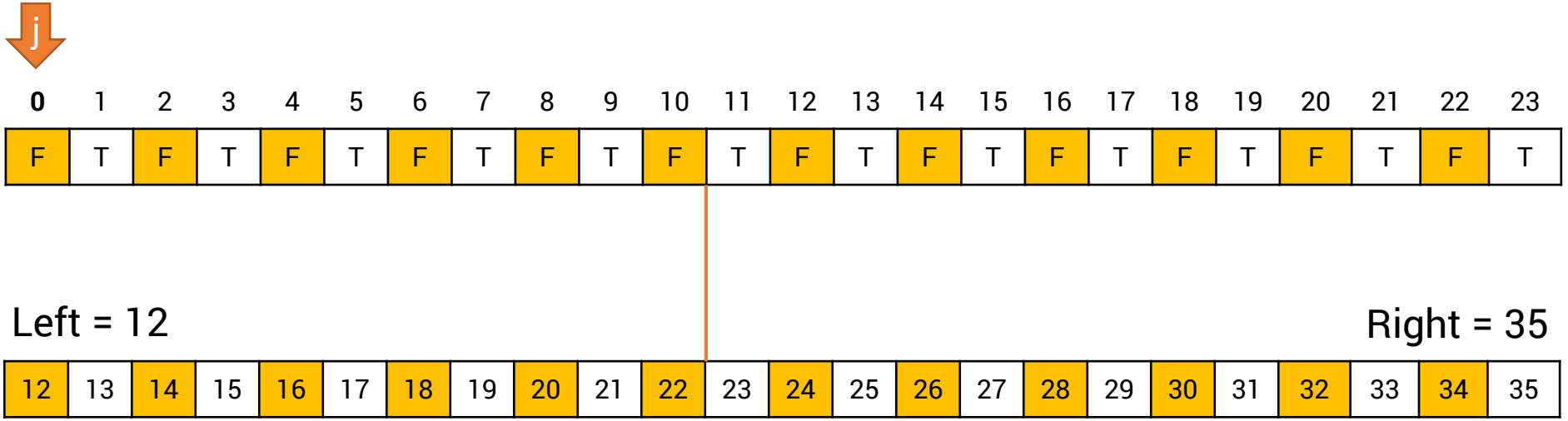
Tìm bội số của số nguyên tố prime = 2.



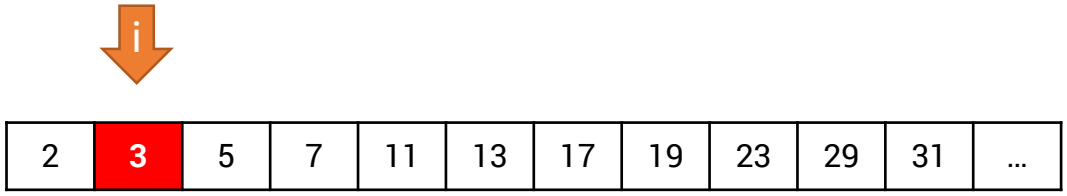
base = (left / prime) * prime = (12 / 2) * 2 = 12

Duyệt j = base - Left → Right - Left (duyệt j = 0 → 23)

➔ Đánh dấu các bội số của số prime (2)



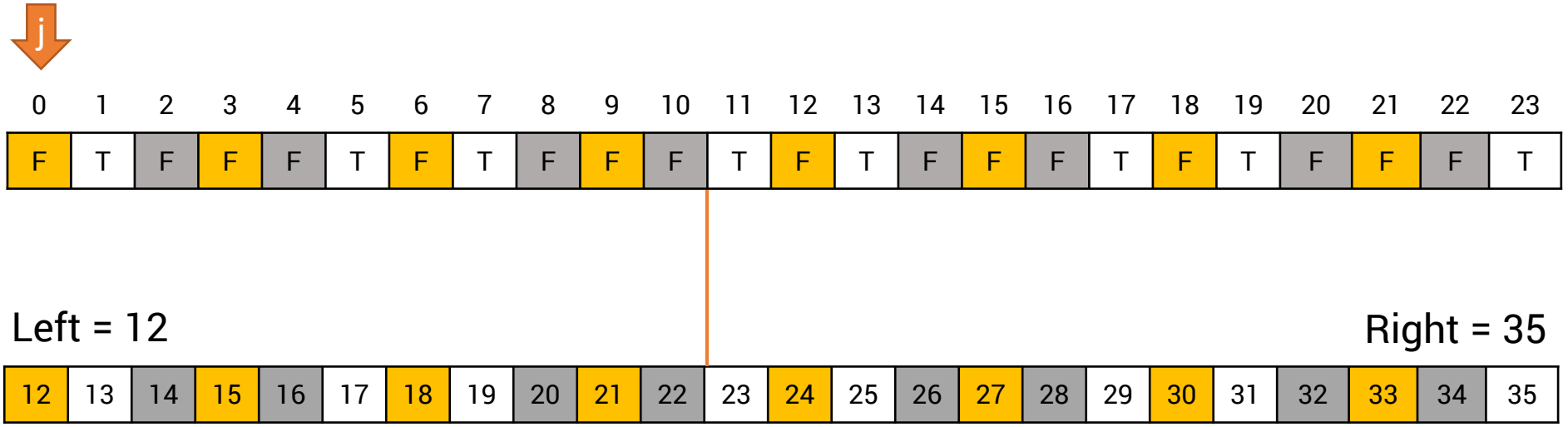
Bước 2: Chạy thuật toán prime = 3



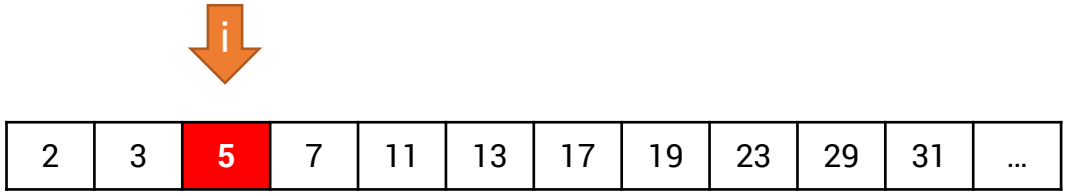
base = (left / prime) * prime = (12 / 3) * 3 = 12

Duyệt j = base - Left → Right – Left (duyệt j = 0 → 23)

➔ Đánh dấu các bội số của số prime (3)



Bước 3: Chạy thuật toán prime = 5

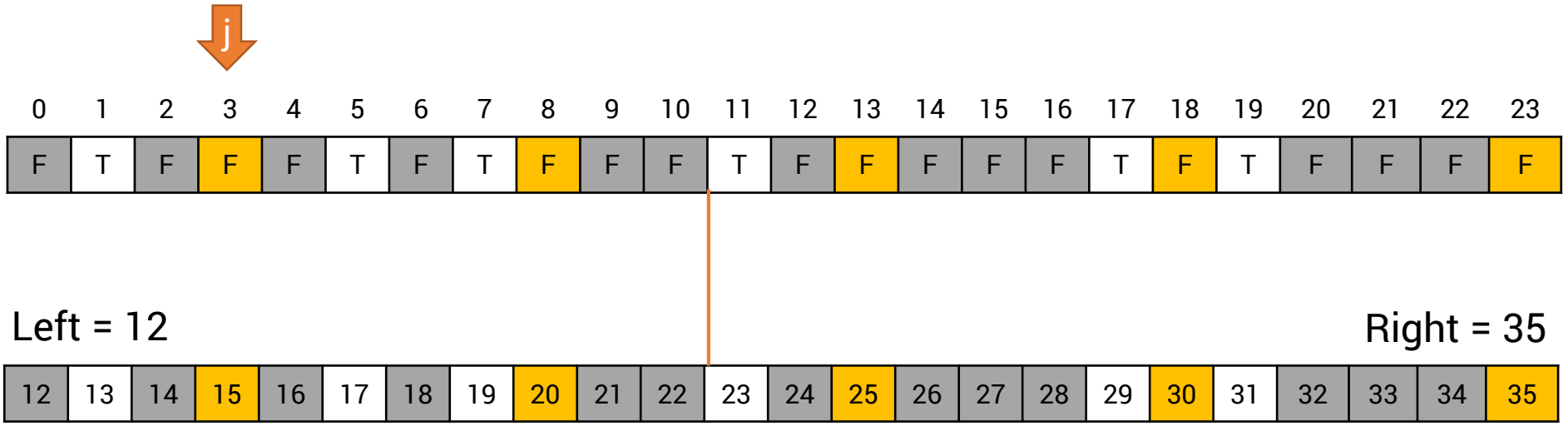


base = (left / prime) * prime = (12 / 5) * 5 = 10 < Left

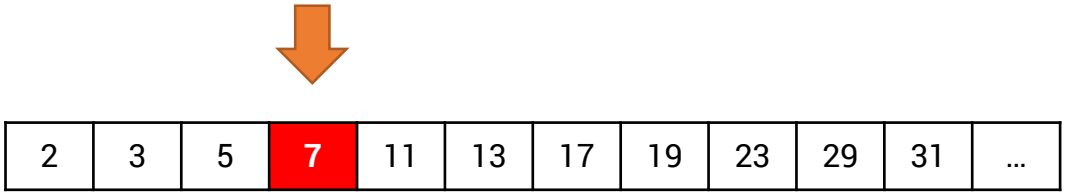
➔ base = base + prime = 10 + 5 = 15

Duyệt j = base - Left → Right - Left (duyệt j = 3 → 23)

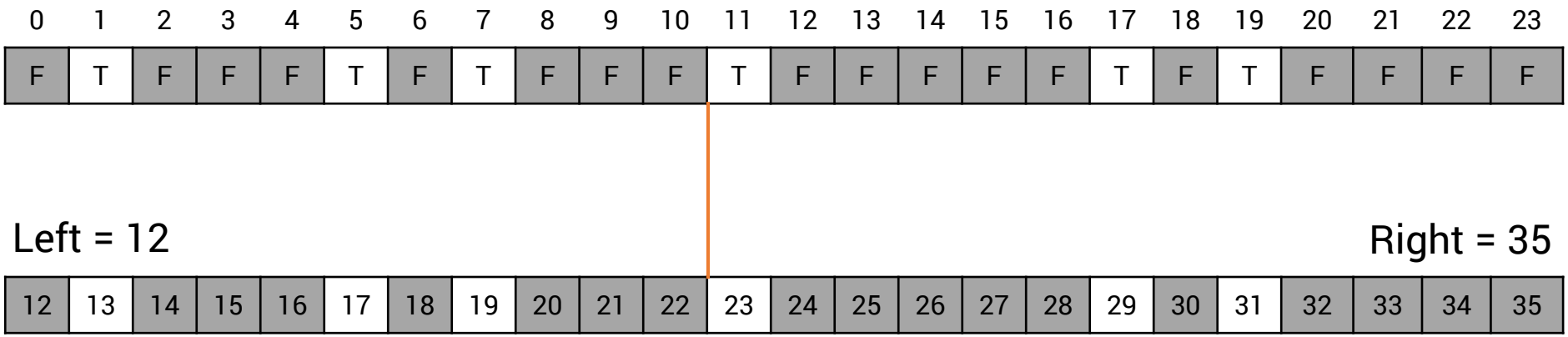
➔ Đánh dấu các bội số của số prime (5)



Bước 4: Dừng thuật toán prime = 7



➔ Dừng thuật toán prime (7) > $\sqrt{right}$ $\left(\sqrt{35} = 5.916\right)$



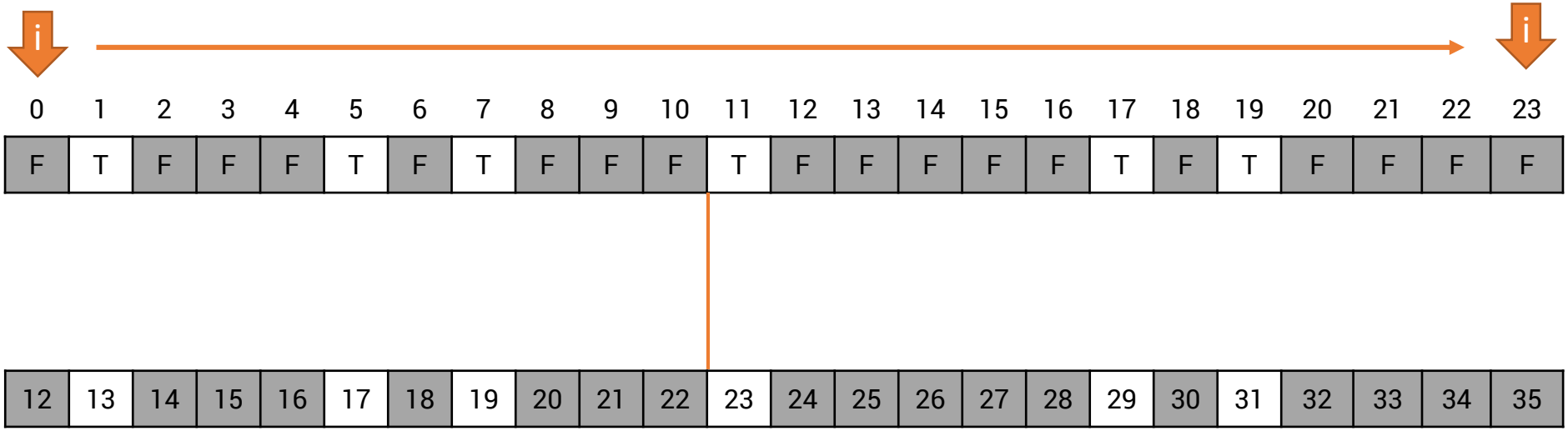
Bước 5: Xuất kết quả bài toán

Chạy từ i từ Left (12) đến Right (35) lấy các giá trị true (màu trắng) là các số nguyên tố.

Left = 12

Nếu mark[i – left] == true ➔ push(i)

Right = 35



➔ Danh sách các số nguyên tố: 13, 17, 19, 23, 29, 31

Source Code Segmented Sieve

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
vector<int> sieveOfEratosthenes(int limit)
{
    vector<bool> mark;
    vector <int> primes;
    mark.resize(limit + 1, true);
    mark[0] = mark[1] = false;
    for (int i = 2; i * i <= limit; i++)
        if (mark[i] == true)
            for (int j = i * i; j <= limit; j += i)
                mark[j] = false;
    for (int i = 2; i <= limit; i++)
        if (mark[i] == true)
            primes.push_back(i);
    return primes;
}
```

Source Code Segmented Sieve

```
20. vector<int> segmentedSieve(int left, int right, vector<int> primes)
21. {
22.     if (left == 1)
23.         left = left + 1;
24.     vector<bool> mark;
25.     mark.resize(right - left + 1, true);
26.     for (int i = 0; i < primes.size() && primes[i] <= sqrt(right); i++)
27.     {
28.         int base = (left / primes[i]) * primes[i];
29.         if (base < left)
30.             base += primes[i];
31.         for (int j = base; j <= right; j += primes[i])
32.         {
33.             if (j != primes[i])
34.                 mark[j - left] = false;
35.         }
36.     }
```

Source Code Segmented Sieve

```
37.     vector<int> result;
38.     for (int i = left; i <= right; i++)
39.         if (mark[i - left] == true)
40.             result.push_back(i);
41.     return result;
42. }
```

```
43. int main()
44. {
45.     int left, right;
46.     left = 12;
47.     right = 35;
48.     vector<int> primes = sieveOfEratosthenes(sqrt(right);
49.     vector<int> result = segmentedSieve(left, right, primes);
50.     for (int i = 0; i < result.size(); i++)
51.         cout << result[i] << " ";
52.     return 0;
53. }
```

Source Code Segmented Sieve

```
1.  def sieveOfEratosthenes(limit):
2.      mark = [True] * (limit + 1)
3.      primes = []
4.      mark[0] = mark[1] = False
5.      for i in range(2, int(limit ** 0.5) + 1):
6.          if mark[i] == True:
7.              for j in range(i * i, limit + 1, i):
8.                  mark[j] = False
9.      for i in range(2, limit):
10.         if mark[i] == True:
11.             primes.append(i)
12.     return primes
```

Source Code Segmented Sieve

```
13. def segmentedSieve(left, right, primes):
14.     if left == 1:
15.         left += 1
16.     mark = [True] * (right - left + 1)
17.     i = 0
18.     while i < len(primes) and primes[i] <= right ** 0.5:
19.         base = left // primes[i] * primes[i]
20.         if base < left:
21.             base += primes[i]
22.         for j in range(base, right + 1, primes[i]):
23.             if j != primes[i]:
24.                 mark[j - left] = False
25.         i += 1
26.     result = []
27.     for i in range(left, right + 1):
28.         if mark[i - left] == True:
29.             result.append(i)
30.     return result
```

Source Code Segmented Sieve

```
31. if __name__ == "__main__":
32.     left = 12
33.     right = 35
34.     primes = sieveOfEratosthenes(right ** 0.5)
35.     result = segmentedSieve(left, right, primes)
36.     for p in result:
37.         print(p, end=' ')
```

Source Code Segmented Sieve

```java
import java.util.*;
public class Main {
    private static ArrayList<Integer> sieveOfEratosthenes(int limit) {
        boolean mark[] = new boolean[limit + 1];
        ArrayList<Integer> primes = new ArrayList<>();
        Arrays.fill(mark, true);
        mark[0] = mark[1] = false;
        for (int i = 2; i*i <= limit; i++) {
            if (mark[i] == true) {
                for (int j = i * i; j <= limit; j += i)
                    mark[j] = false;
            }
        }
        for (int i = 2; i <= limit; i++)
            if (mark[i] == true)
                primes.add(i);
        return primes;
    }
```

Source Code Segmented Sieve

```
19.     private static ArrayList<Integer> segmentedSieve(int left, int right,
                                                         ArrayList<Integer> primes) {
20.         if (left == 1)
21.             left = left + 1;
22.         boolean mark[] = new boolean[right - left + 1];
23.         Arrays.fill(mark, true);
24.         for (int i = 0; i < primes.size()
                            && primes.get(i) <= Math.sqrt(right); i++) {
25.             int base = (left / primes.get(i))  * primes.get(i);
26.             if (base < left)
27.                 base += primes.get(i);
28.             for (int j = base; j <= right; j += primes.get(i)) {
29.                 if (j != primes.get(i))
30.                     mark[j - left] = false;
31.             }
32.         }
```

Source Code Segmented Sieve

```
33.             ArrayList<Integer> result = new ArrayList<>();
34.             for (int i = left; i <= right; i++)
35.                 if (mark[i - left] == true)
36.                     result.add(i);
37.             return result;
38.         }
```

```
39.         public static void main(String[] args) {
40.             int left, right;
41.             left = 12;
42.             right = 35;
43.             ArrayList<Integer> primes = sieveOfEratosthenes(Math.sqrt(right));
44.             ArrayList<Integer> result = segmentedSieve(left, right, primes);
45.             for (int i = 0; i < result.size(); i++) {
46.                 System.out.printf("%d ", result.get(i));
47.             }
48.         }
49.     }
```

EULER'S TOTIENT FUNCTION

Euler's totient function

Euler's totient function (Hàm phi Euler): hàm phi Euler của một số nguyên dương N là số lượng số nguyên dương không quá N và nguyên tố cùng nhau với N.

Ký hiệu: $\phi(N)$ hoặc $\varphi(N)$.

Relatively prime (Số nguyên tố cùng nhau): hai số được gọi là nguyên tố cùng nhau nếu chúng có ước số chung lớn nhất là 1.

Ví dụ: số 6 và 35 là nguyên tố cùng nhau vì chúng có ước số chung lớn nhất là 1.

** Lưu ý: số 6 và 35 tuy là hợp số nhưng chỉ cần có GCD là 1 thì chúng là số nguyên tố cùng nhau.*

Euler's totient function

Ví dụ: $\phi(8) = 4$

- GCD(1, 8) = 1.
- GCD(2, 8) = 2 (không tính).
- GCD(3, 8) = 1.
- GCD(4, 8) = 4 (không tính).
- GCD(5, 8) = 1.
- GCD(6, 8) = 2 (không tính).
- GCD(7, 8) = 1.
- GCD(8, 8) = 8 (không tính).

Mở rộng: nếu chúng ta có n số nguyên cùng có GCD là 1 thì n số đó cũng là nguyên tố cùng nhau.

Ví dụ: Ba số 2, 10, 15 là nguyên tố cùng nhau, nhưng không nguyên tố cùng nhau từng đôi một.

Tìm phi Euler của N

Ý tưởng cơ bản: Duyệt qua toàn bộ các số từ 1 đến N, đếm các số có GCD với N bằng 1.

Ví dụ: $\phi(60) = 16$

60 nguyên tố cùng nhau với: 1, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, 59.

Độ phức tạp: $\mathbf{O}(\boldsymbol{NlogN})$.

Source Code Euler's totient function

```cpp
#include <iostream>
using namespace std;
int gcd(int a, int b)
{
    int remainder;
    while (b != 0)
{
        remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}
```

Source Code Euler's totient function

```
13. int phi(int n)
14. {
15.     int result = 1;
16.     for (int i = 2; i < n; i++)
17.         if (gcd(i, n) == 1)
18.             result++;
19.     return result;
20. }
21.
22. int main()
23. {
24.     int n = 60;
25.     cout << "phi(" <<n <<") = "<<phi(n) << endl;
26.     return 0;
27. }
```

Source Code Euler's totient function

```
1.  def gcd(a, b):
2.      while b != 0:
3.          remainder = a % b
4.          a = b
5.          b = remainder
6.      return a

7.  def phi(n):
8.      result = 1
9.      for i in range(2, n):
10.         if gcd(i, n) == 1:
11.             result += 1
12.     return result
```

Source Code Euler's totient function

```
13. if __name__ == "__main__":
14.     n = 60
15.     print('phi(', n, ') = ', phi(n), sep = '')
```

Source Code Euler's totient function

```
1.  public class Main {
2.      private static int gcd(int a, int b) {
3.          int remainder;
4.          while (b != 0) {
5.              remainder = a % b;
6.              a = b;
7.              b = remainder;
8.          }
9.          return a;
10.     }
```

Source Code Euler's totient function

```
11.     private static int phi(int n) {
12.         int result = 1;
13.         for (int i = 2; i < n; i++)
14.             if (gcd(i, n) == 1)
15.                 result++;
16.         return result;
17.     }

18.     public static void main(String[] args) {
19.         int n = 60;
20.         System.out.printf("phi(%d) = %d\n", n, phi(n));
21.         return;
22.     }
23. }
```

Euler's totient function

Các tính chất của hàm phi Euler:

1. Nếu p là số nguyên tố: $\phi(\boldsymbol{p}) = \boldsymbol{p} - \mathbf{1}$

Ví dụ: $\phi(5) = 5 - 1 = 4$

2. Nếu p là số nguyên tố và k ≥ 1: $\phi(\boldsymbol{p^k}) = \boldsymbol{p^k} - \boldsymbol{p^{k-1}}$

Ví dụ: $\phi(5^2) = 5^2 - 5^{2-1} = 25 - 5 = 20$

3. Nếu a và b là nguyên tố cùng nhau: $\phi(\boldsymbol{a} \times \boldsymbol{b}) = \phi(\boldsymbol{a}) \times \phi(\boldsymbol{b})$

Ví dụ: $\phi(6 \times 5) = \phi(6) \times \phi(5) = 2 \times 4 = 8.$

➔ **Dựa vào các tính chất của hàm phi Euler để tìm cách giảm độ phức tạp.**

Euler's totient function

Prime factorization (Phân tích thừa số nguyên tố): Phân tích một số ra thừa số nguyên tố là phân tích số đó thành tích của các số nguyên tố.

Ví dụ: $18 = 2 \times 3 \times 3 = 2 \times 3^2$

Tổng quát:

Xét $n = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k}$ (với p_i là ước nguyên tố thứ i của n).

- Dựa vào tính chất 3 ta được:

$$\phi(n) = \phi\left(p_1^{a_1}\right) \times \phi\left(p_2^{a_2}\right) \times \cdots \times \phi\left(p_k^{a_k}\right)$$

- Dựa vào tính chất 2 ta được:

$$\phi(n) = \left(p_1^{a_1} - p_1^{a_1-1}\right) \times \left(p_2^{a_2} - p_2^{a_2-1}\right) \times \cdots \times \left(p_k^{a_k} - p_k^{a_k-1}\right)$$

Euler's totient function

Tổng quát:

$$\phi(n) = \left(p_1^{a_1} - p_1^{a_1-1}\right) \times \left(p_2^{a_2} - p_2^{a_2-1}\right) \times \cdots \times \left(p_k^{a_k} - p_k^{a_k-1}\right)$$

$$= \left(p_1^{a_1}\left(1 - \frac{1}{p_1}\right)\right) \times \left(p_2^{a_2}\left(1 - \frac{1}{p_2}\right)\right) \times \cdots \times \left(p_k^{a_k}\left(1 - \frac{1}{p_k}\right)\right)$$

$$= \left(p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k}\right) \times \left(\left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_k}\right)\right)$$

$$= n \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_k}\right)$$

Kết luận: $\phi(n) = n \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_k}\right)$

Euler's totient function

Kết luận: $\phi(n) = n \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_k}\right)$

Ví dụ: Tính $\phi(60)$

Ta có: $60 = 2^2 \times 3 \times 5$

➔ $\phi(60) = 60 \times \left(1 - \frac{1}{2}\right) \times \left(1 - \frac{1}{3}\right) \times \left(1 - \frac{1}{5}\right)$

➔ $\phi(60) = 60 \times \frac{1}{2} \times \frac{2}{3} \times \frac{4}{5}$

➔ $\phi(60) = 16$

Độ phức tạp: $O(\sqrt{N})$

Bước 1: Chạy thuật toán lần 1 (i=2)

Bắt đầu từ số i là nguyên tố đầu tiên, chia N cho i đến khi nào không chia hết được nữa:

- Nếu $i * i \leq N$ $(2 * 2 \leq 60)$ ✓
- Nếu $N \% i == 0$ $(60 \% 2 == 0)$ ✓

➔ $N = N/i$

i	N
2	60
2	30
2	15

➡ **Dừng bước 1 vì** $(N \% i) != 0$

$result = N = 60$

➔ $result = result * (1 - 1/i)$

➔ $result = 60 * (1 - 1/2) = 60 * 1/2 = 30$

result = 30

Bước 2: Chạy thuật toán lần 2 (i=3)

Tăng giá trị biến i lên và tiếp tục xét xem N có chia hết cho i không, nếu có thì tiếp tục chia N cho i cho tới khi không chia hết được nữa:

- Nếu $i * i \leq N$ $(3 * 3 \leq 15)$ ✓
- Nếu $N \% i == 0$ $(15 \% 3 == 0)$ ✓

➔ $N = N/i$

i	N
3	15
3	5

➔ **Dừng bước 2 vì** $(N \% i) != 0$

$result = 30$

➔ $result = result * (1 - 1 / i)$

➔ $result = 30 * (1 - 1 / 3) = 30 * 2/3 = 20$

result = 20

Bước 3: Chạy thuật toán lần 3 (i=4)

Tăng giá trị biến i lên và tiếp tục xét xem N có chia hết cho i không, nếu có thì tiếp tục chia N cho i cho tới khi không chia được nữa:

- Nếu $i * i \leq N$ $(4 * 4 \leq 5)$ 

 **Dừng vòng lặp (vì không thỏa điều kiện đầu)**

Kiểm tra nếu N > 1 thì chúng ta sẽ tính giá trị lại result lần cuối.

$result = 20$

➔ $result = result * (1 - 1 / N)$

➔ $result = 20 * (1 - 1 / N) = 20 * 4/5 = 16.$

result = 16

Kết luận: $\phi(60) = 16$

Source Code Euler's totient function

```cpp
#include <iostream>
using namespace std;
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 * (i - 1);
        }
    }
    if (n > 1)
        result = result / n * (n - 1);
    return result;
}
```

Source Code Euler's totient function

```
21. int main()
22. {
23.     int n = 60;
24.     cout << "phi(" << n << ") = " << phi(n) << endl;
25.     return 0;
26. }
```

Source Code Euler's totient function

```
1.  def phi(n):
2.      result = n
3.      for i in range(2, int(n ** 0.5) + 1):
4.          if n % i == 0:
5.              while n % i == 0:
6.                  n //= i
7.              result = result // i * (i - 1)
8.        if n > 1:
9.          result = result // n * (n - 1)
10.       return result

11. if __name__ == "__main__":
12.     n = 60
13.     print('phi(', n, ') = ', phi(n), sep = '')
```

Source Code Euler's totient function

```java
public class Main {
    private static 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 * (i - 1);
            }
        }
        if (n > 1) {
            result = result / n * (n - 1);
        }
        return result;
    }
```

Source Code Euler's totient function

```
17.     public static void main(String[] args) {
18.         int n = 60;
19.         System.out.printf("phi(%d) = %d\n", n, phi(n));
20.         return;
21.     }
22. }
```

Euler's theorem

Định lý Euler: Cho a và m là số nguyên tố cùng nhau, nghĩa là GCD(a, m) = 1, ta có:

$$a^{\phi(m)} \equiv 1 \ (mod\ m)$$

Ví dụ 1: $a = 3$, $m = 10$, $\phi(10) = 4$

➔ $3^4 = 81 \equiv 1 \pmod{10}$.

Ví dụ 2: $a = 2$, $m = 11$, $\phi(11) = 10$

➔ $2^{10} = 1024 \equiv 1 \pmod{11}$

Euler's theorem

$$a^{\phi(m)} \equiv 1 \ (mod\ m)$$

Ứng dụng: Nếu a là số nguyên tố cùng nhau với 10 thì ta có thể áp dụng để tìm chữ số tận cùng của một số có số mũ lớn.

7^{222} (mod 10)

$= 7^{4x55 + 2}$ (mod 10)

$= (7^4)^{55} \times 7^2$ (mod 10)

$= 1^{55} \times 7^2$ (mod 10)

= 49 (mod 10).

➔ Vậy 7^{222} có chữ số tận cùng là 9

Lưu ý: Các số có chữ số tận cùng là 3, 7, 9 khi nâng lên lũy thừa bậc 4n (n thuộc N) thì chữ số tận cùng là 1.

Hỏi đáp