TEMA 7

Otros ejemplos

Emma Rollón
erollon@cs.upc.edu

Departamento de Ciencias de la Computación

Criba de Eratóstenes

Input: una secuencia de naturales $x_1, x_2, \ldots, x_n$ en donde:

- n es muy grande
- $1 \leq x_i \leq M$ (siendo M un valor relativamente pequeño).

Output: por cada elemento de la secuencia, escribir "SI" si el elemento es primo, "NO" en caso contrario.

```
/* Primera solución que nos viene a la cabeza */

// Pre: x > 0
// Post: retorna true si x es primo, false en caso contrario
bool primo(int x) {
    ...;
}

int main() {
    int x;
    while (cin >> x) {
        if (primo(x)) cout << "SI" << endl
        else cout << "NO" << endl;
    }
}
```

Criba de Eratóstenes

Lo podemos hacer mejor, ¿cómo?

$\Longrightarrow$ Precalculando si los números desde el 1 al M son o no primos.

¿Qué ganamos con esto?

$\Longrightarrow$ Calculamos una única vez la primalidad de $1, \ldots, M$.

$\Longrightarrow$ La consultamos tantas veces como se necesite.

¿Dónde almaceno la información de que un valor i es primo o no?

$\Longrightarrow$ En un vector de booleanos v $\begin{cases} \text{v[i] es true si i es primo} \\ \text{v[i] es false si i no es primo} \end{cases}$

¿Cómo damos valor a ese vector?

$\Longrightarrow$ Con la criba de Eratóstenes

Criba de Eratóstenes

0 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

```
// Pre: m >= 1
// Post: retorna un vector de booleanos v en donde para 0 <= i <= m
//         v[i] es cierto si i es primo, false en caso contrario.
vector<bool> criba(int m) {
    vector<bool> v(m + 1, true);
    v[0] = v[1] = false;
    for (int i = 2; i <= m; ++i) {
        if (v[i]) {
            for (int j = 2*i; j <= m; j = j + i) v[j] = false;
        }
    }
    return v;
}
```

Criba de Eratóstenes: mejora 1

$$i = 5 \begin{cases} j = 2 \times 5 & \rightarrow \text{múltiplo de 2, ya está tachado} \\ j = 3 \times 5 & \rightarrow \text{múltiplo de 3, ya está tachado} \\ j = 4 \times 5 & \rightarrow \text{múltiplo de 4, ya está tachado} \\ j = 5 \times 5 & \\ j = 6 \times 5 & \\ \ldots & \end{cases}$$

```
// Pre: m >= 1
// Post: retorna un vector de booleanos v en donde para 0 <= i <= m
//         v[i] es cierto si i es primo, false en caso contrario.
vector<bool> criba(int m) {
    vector<bool> v(m + 1, true);
    v[0] = v[1] = false;
    for (int i = 2; i <= m; ++i) {
        if (v[i]) {
            for (int j = i*i; j <= m; j = j + i) v[j] = false;
        }
    }
    return v;
}
```

Criba de Eratóstenes: mejora 2

El bucle interno sólo iterará cuando $i * i$ (que es el valor inicial de j) sea menor o igual que m. Por tanto, ajustamos la condición del bucle externo para que vaya hasta el último valor de i para el cual se realiza alguna iteración del bucle interno.

```
// Pre: m >= 1
// Post: retorna un vector de booleanos v en donde para 0 <= i <= m
//         v[i] es cierto si i es primo, false en caso contrario.
vector<bool> criba(int m) {
    vector<bool> v(m + 1, true);
    v[0] = v[1] = false;
    for (int i = 2; i*i <= m; ++i) {
        if (v[i]) {
            for (int j = i*i; j <= m; j = j + i) v[j] = false;
        }
    }
    return v;
}
```

Criba de Eratóstenes: utilización

```
// Pre: m >= 1
// Post: retorna un vector de booleanos v en donde para 0 <= i <= m,
//         v[i] es cierto si i es primo, false en caso contrario.
vector<bool> criba(int m) {
    vector<bool> v(m + 1, true);
    v[0] = v[1] = false;
    for (int i = 2; i*i <= m; ++i) {
        if (v[i]) {
            for (int j = i*i; j <= m; j = j + i) v[j] = false;
        }
    }
    return v;
}

// Pre: secuencia de naturales con valores entre 1 y M (conocido)
// Post: por cada elemento, escribe "SI" si es primo, "NO" en caso contrario
int main() {
    const int M = ??;                  // el valor concreto nos lo darán en el enunciado
    vector<bool> p = criba(M);
    int x;
    while (cin >> x) {
        if (p[x]) cout << "SI" << endl
        else cout << "NO" << endl;
    }
}
```

Salta, salta

```
// Pre: 0 <= p < v.size()
// Post: retorna "dreta", "esquerra" o "mai" según el enunciado;
//           no importa el contenido final de v
string salta(int p, vector<int>& v) {
    ...
}
```

0	1	2	3	4	5	6	7
20	-2	5	2	3	-4	-2	-1

$p = 3 \rightarrow$ “esquerra”

$p = 0 \rightarrow$ “dreta”

$p = 2 \rightarrow$ “mai”

Salta, salta: recursivo

```
// Pre: 0 <= p < v.size()
// Post: retorna "dreta", "esquerra" o "mai" según el enunciado;
//           no importa el contenido final de v
string salta(int p, vector<int>& v) {
    if (v[p] == 0) return "mai";
    int nueva_pos = p + v[p];
    v[p] = 0;
    if (nueva_pos < 0) return "esquerra";
    if (nueva_pos >= v.size()) return "dreta";
    return salta(nueva_pos, v);
}
```

¿Y si cambiamos el orden de los condicionales?

```
// Pre: 0 <= p < v.size()
// Post: retorna "dreta", "esquerra" o "mai" según el enunciado;
//           no importa el contenido final de v
string salta(int p, vector<int>& v) {
    if (v[p] == 0) return "mai";
    int nueva_pos = p + v[p];
    v[p] = 0;
    if (nueva_pos >= v.size()) return "dreta";     // <———— ¡OJO!
    if (nueva_pos < 0) return "esquerra";
    return salta(nueva_pos, v);
}
```

v.size() es un entero sin signo (el bit de más peso aporta valor y no indica signo). La comparación >= la interpreta sobre dos enteros sin signo.

Salta, salta: más restrictivo

¿Y si cambiamos la cabecera y no podemos modificar el vector v?

```
// Pre: 0 <= p < v.size(); traza.size() = v.size();
//        traza[i] es cierto si se ha pasado por esa posición, false en caso contrario
// Post: retorna "dreta", "esquerra" o "mai" según el enunciado;
//        modifica traza indicando que se ha pasado por la posición p
string salta_rec(int p, const vector<int>& v, vector<bool>& traza) {
    if (traza[p]) return "mai";
    int nueva_pos = p + v[p];
    traza[p] = true;
    if (nueva_pos < 0) return "esquerra";
    if (nueva_pos >= v.size()) return "dreta";
    return salta_rec(nueva_pos, v, traza);
}

// Pre: 0 <= p < v.size()
// Post: retorna "dreta", "esquerra" o "mai" según el enunciado;
string salta(int p, const vector<int>& v) {
    vector<bool> traza(v.size(), false);
    return salta_rec(p, v, traza);
}
```

Ríos digitales

Río 1: 1 2 4 8 16 23 28 38 ... 107 ...

Río 3: 3 6 12 15 21 24 30 33 39 ... 96 ...

Río 9: 9 18 27 36 45 54 ... 99 ...

Río 14: 14 19 29 ... 107 ...

Río 15: 15 21 24 ...

```
int suma_digits(int x) {
    ...
}

int trobada_de_rius(int n) {
  int rio1 = 1;
  int rio3 = 3;
  int rio9 = 9;
  while (n != rio1 and n != rio3 and n != rio9) {    // <------ centinela
    if (rio1 < n) rio1 = rio1 + suma_digits(rio1);
    else if (rio3 < n) rio3 = rio3 + suma_digits(rio3);
    else if (rio9 < n) rio9 = rio9 + suma_digits(rio9);
    else n = n + suma_digits(n);
  }
  return n;
}
```