Semillero de Programación

Algoritmos de Teoría de Números

Ana Echavarría Juan Francisco Cardona

Universidad EAFIT

17 de mayo de 2013

Contenido

1. Problemas semana anterior
2. Hallar los divisores de un número
3. Máximo común divisor y mínimo común múltiplo
4. Criba de Eratóstenes
5. Hallar la factorización prima de un número
6. Exponenciación logarítmica
7. Combinatoria
8. Tarea

Contenido

1 Problemas semana anterior

Problema A - Internet Bandwidth

Hay que hallar el máximo flujo que puede enviarse desde una fuente a un sumidero

Implementación I

```
1  const int MAXN = 105;
2  // Lista de adyacencia de la red residual. Solo las conexiones no
3  // los pesos. Se usa para que el BFS sea rapido
4  vector <int> g [MAXN];
5  int c [MAXN][MAXN]; // Capacidad de aristas de la red de flujos
6  int f [MAXN][MAXN]; // El flujo de cada arista
7  //El predecesor de cada nodo en el camino de aumentacion de s a t
8  int prev [MAXN];
9
10 void connect (int i, int j, int cap){
11     // Agregar SIEMPRE las dos aristas al grafo la red de flujos
12     // sea dirigida. Esto es porque g representa la red residual
13     // que tiene aristas en los dos sentidos
14     g[i].push_back(j);
15     g[j].push_back(i);
16     c[i][j] += cap;
17     // Omitir esta linea si el grafo es dirigido
18     c[j][i] += cap;
```

Implementación II

```
19  }
20
21  int maxflow(int s, int t, int n){
22      // Inicializar el flujo en 0
23      for (int i = 0; i <= n; i++)
24          for (int j = 0; j <= n; j++)
25              f[i][j] = 0;
26
27      int flow = 0;
28      while (true){
29          // Encontrar camino entre s y t con bfs
30          for (int i = 0; i <= n; i++) prev[i] = -1;
31
32          queue <int> q;
33          q.push(s);
34          prev[s] = -2;  // s no tiene nodo anterior en el camino
35
36
37
```

Implementación III

```
38
39          while (q.size() > 0){
40              int u = q.front(); q.pop();
41              if (u == t) break;
42              for (int i = 0; i < g[u].size(); ++i){
43                  int v = g[u][i];
44                  // Si el nodo no esta visitado
45                  // y el peso en la red residual es > 0
46                  if (prev[v] == -1 and c[u][v] - f[u][v] > 0){
47                      q.push(v);
48                      prev[v] = u;
49                  }
50              }
51          }
52          // Si no se llego a t en el camino es porque no hay mas
53          // caminos de aumentacion y el ciclo se detiene
54          if (prev[t] == -1) break;
55
56
```

Implementación IV

```
57
58
59
60
61          // Hallar el cuello de botella
62          // Se recorre el camino desde t hasta s
63          int extra = 1 << 30;
64          int end = t;
65          while (end != s){
66              int start = prev[end];
67              // El cuello de botella es el minimo peso del camino
68              // en la red residual. El peso en la red residual es
69              // la capacidad de la arista menos el flujo
70              extra = min(extra, c[start][end] - f[start][end]);
71              end = start;
72          }
73
74
75
```

Implementación V

```
76
77
78
79          // Bombear el flujo extra por la arista
80          end = t;
81          while (end != s){
82              int start = prev[end];
83              f[start][end] += extra;
84              f[end][start] = -f[start][end];
85              end = start;
86          }
87
88          // Agregar el flujo enviado a la respuesta
89          flow += extra;
90      }
91
92      return flow;
93  }
94
```

Implementación VI

```
95  int main(){
96      int run = 1;
97      int n;
98      while (cin >> n){
99          if (n == 0) break;
100
101         // Limpiar el grafo y la capacidad
102         for (int i = 0; i <= n; i++){
103             g[i].clear();
104             for (int j = 0; j <= n; j++){
105                 c[i][j] = 0;
106             }
107         }
108
109         // Leer el grafo
110         int s, t, edges;
111         cin >> s >> t >> edges;
112         for (int i = 0; i < edges; i++){
113             int n1, n2, cap;
```

Implementación VII

```
114             cin >> n1 >> n2 >> cap;
115             connect(n1, n2, cap);
116         }
117
118         // Imprimir la respuesta
119         printf("Network %d\n", run++);
120         printf("The bandwidth is %d.\n\n", maxflow(s, t, n));
121
122     }
123     return 0;
124 }
```

Problema B - Angry Programmer

- Usar el algoritmo de máximo flujo para hallar el mínimo corte
- Dividir cada nodo en 2 para poder tener en cuenta la capacidad de los nodos.
- Para que no se puedan destruir la fuente y el sumidero ponerles capacidad infinita.

Implementación I

```
1  const int MAXN = 2 * 50 + 5;
2  const int INF = INT_MAX;
3  vector < int > g [MAXN];
4  int c [MAXN][MAXN];
5  int f [MAXN][MAXN];
6  int prev [MAXN];
7
8  void refresh (int n){
9      for (int i = 0; i < 2 * n; i++){
10         g[i].clear();
11         for (int j = 0; j < 2 * n; j++){
12             f[i][j] = 0;
13             c[i][j] = 0;
14         }
15     }
16 }
17
18
```

Implementación II

```
19  void join (int u, int v, int cost){
20      int in_u = 2 * u;
21      int out_u = 2 * u + 1;
22
23      int in_v = 2 * v;
24      int out_v = 2 * v + 1;
25
26      g[out_u].push_back(in_v);
27      g[in_v].push_back(out_u);
28      c[out_u][in_v] += cost;
29
30      g[out_v].push_back(in_u);
31      g[in_u].push_back(out_v);
32      c[out_v][in_u] += cost;
33  }
34
35
36
37
```

Implementación III

```
38  void join_self (int u, int cost){
39      g[2 * u].push_back(2 * u + 1);
40      g[2 * u + 1].push_back(2 * u);
41      c[2 * u][2 * u + 1] += cost;
42      c[2 * u + 1][2 * u] += cost;
43  }
44
45  bool path(int s, int t){
46      for (int i = 0; i <= t; i++)
47          prev[i] = -2;
48
49      queue <int> q;
50      q.push(s);
51      prev[s] = -1;
52      while (q.size() > 0){
53          int u = q.front();
54          q.pop();
55          if (u == t) return true;
56          for (int i = 0; i < g[u].size(); i++){
```

Implementación IV

```
57                int v = g[u][i];
58                if (prev[v] == -2 and c[u][v] - f[u][v] > 0){
59                    q.push(v);
60                    prev[v] = u;
61                }
62            }
63        }
64        return false;
65    }
66
67    int bottleneck(int s, int t){
68        int curr = t;
69        int ans = INF;
70        while (curr != s){
71            ans = min (ans, c[prev[curr]][curr] -
          f[prev[curr]][curr]);
72            curr = prev[curr];
73        }
74        assert(prev[s] == -1);
```

Implementación V

```
75      return ans;
76  }
77
78  void pump (int s, int t, int extra){
79      int curr = t;
80      while (curr != s){
81          f[prev[curr]][curr] += extra;
82          f[curr][prev[curr]] -= extra;
83          curr = prev[curr];
84      }
85  }
86
87  int max_flow(int s, int t){
88      int ans = 0;
89      while (true){
90          // Find path
91          if (!path(s, t)) break;
92          // Find bottleneck
93          int extra = bottleneck(s, t);
```

Implementación VI

```
94          // Pump bottleneck
95          pump(s, t, extra);
96          // Add flow to answer
97          ans += extra;
98      }
99      return ans;
100 }
101
102 int main(){
103     int nodes, edges;
104     while (scanf("%d %d", &nodes, &edges) == 2){
105         if (nodes == 0 and edges == 0) break;
106
107         refresh(nodes);
108         // Crear los nodos (entrada y salida)
109         for (int i = 1; i < nodes - 1; i++){
110             int j, cost;
111             cin >> j >> cost;
112             j--;
```

Implementación VII

```
113             join_self(j, cost);
114         }
115         // El peso de la fuente y el sumidero es INF
116         join_self(0, INF);
117         join_self(nodes - 1, INF);
118 
119         // Unir las aristas
120         for (int i = 0; i < edges; i++){
121             int u, v, cost;
122             cin >> u >> v >> cost;
123             u--; v--;
124             join(u, v, cost);
125         }
126 
127         cout << max_flow(0, 2 * nodes - 1) << endl;
128     }
129     return 0;
130 }
```

Contenido

2 Hallar los divisores de un número

Divisores de un número

- Los divisores vienen de a parejas, si $a|b$ entonces $\frac{b}{a}|b$
- Es por esto que basta con mirar hasta la raíz cuadrada del número para hallar sus divisores

Divisores de un número

```
1  void divisors(int n){
2      int i;
3      for (i = 1; i * i < n; ++i){
4          if (n % i == 0) printf("%d\n%d\n", i, n/i);
5      }
6      // Si el numero es un cuadrado perfecto, imprimir su raiz
       cuadrara una sola vez
7      if (i * i == n) printf("%d\n", i);
8  }
```

Complejidad

Pregunta

¿Cuántos números hay que revisar para hallar todos los divisores de un número n?
De acuerdo a eso, ¿cuál es la complejidad del algoritmo?

Complejidad

Pregunta

¿Cuántos números hay que revisar para hallar todos los divisores de un número n?
De acuerdo a eso, ¿cuál es la complejidad del algoritmo?

Respuesta

La complejidad de este algoritmo es $O(\sqrt{n})$

Contenido

3 Máximo común divisor y mínimo común múltiplo

Máximo común divisor

GCD

Para hallar el máximo común divisor entre dos números `a` y `b` ejecutar el comando `__gcd(a, b)`

Mínimo común múltiplo

LCM

El mínimo común múltiplo se puede hallar en términos del máximo común divisor así:

$$\operatorname{lcm}(a, b) = \frac{|a \cdot b|}{\gcd(a, b)}$$

Propiedades

- Si $c|a$ y $c|b$ entonces $c|\gcd(a,b)$
- Si $a|b\cdot c$ y $\gcd(a,b)=1$ entonces $a|c$
- $\gcd(a,b)$ es una combinación lineal entera de a y b

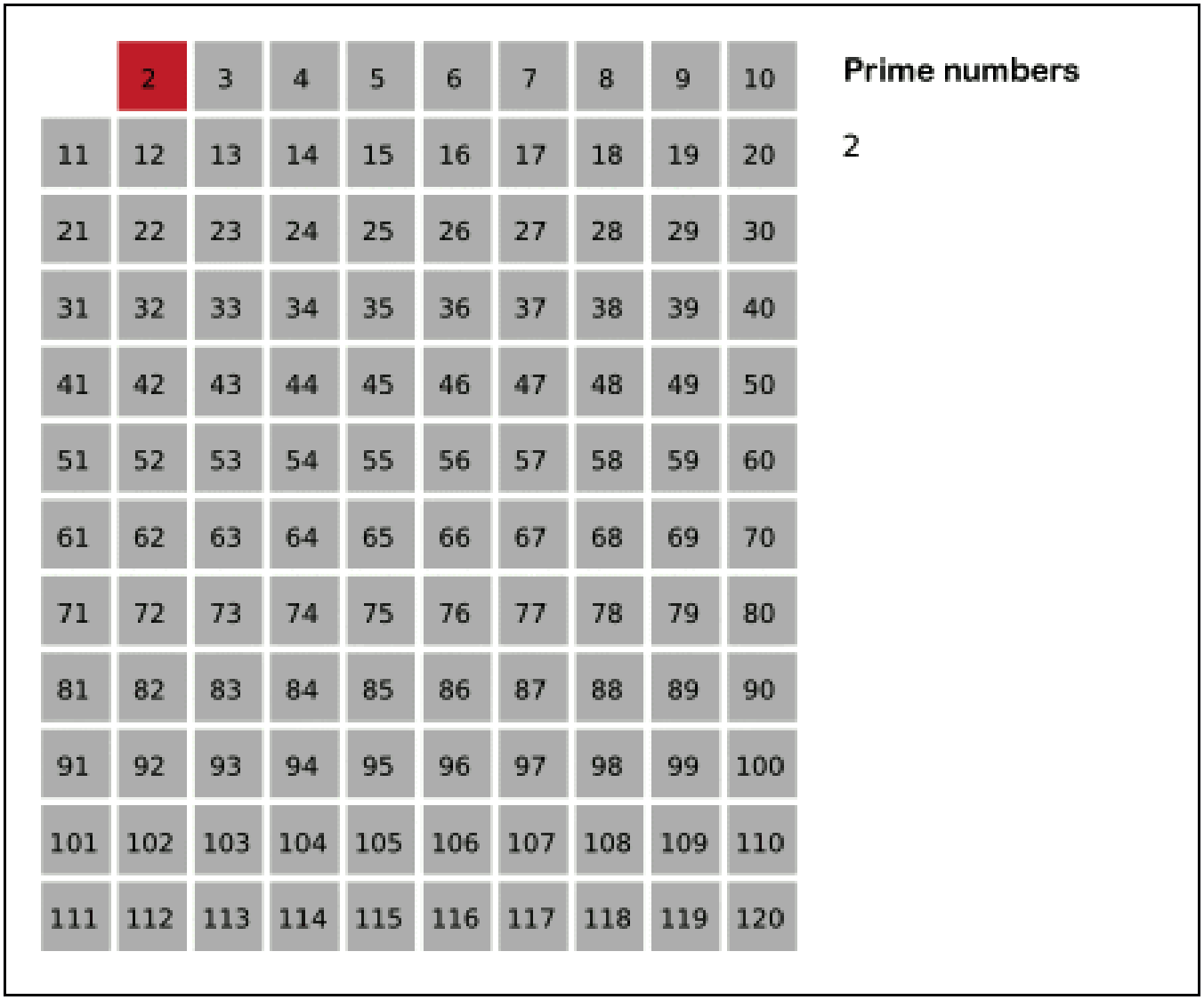
Contenido

4 Criba de Eratóstenes

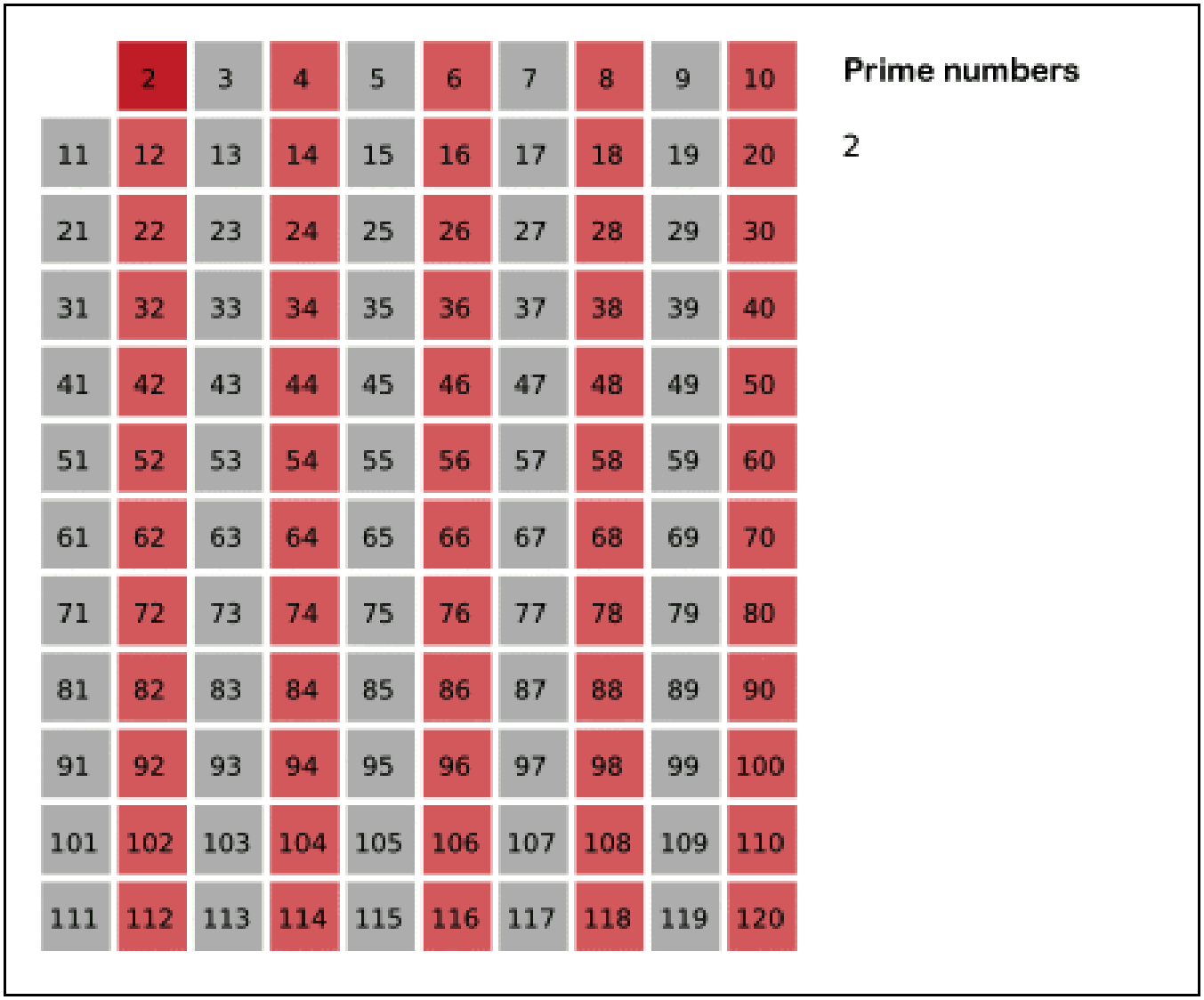
Criba de Eratóstenes

- Es un algoritmo para hallar todos los números primos hasta un límite n
- Inicialmente asume marca todos los números de 2 hasta n como no visitados y el 1 como visitado
- Empieza a recorrer los números uno por uno
- Cuando encuentra un número no visitado empieza a marcar todos sus múltiplos (no él) como visitados
- Cuando llega al número n, los números que estén marcados como no visitados son los números primos del intervalo $[1, n]$

Criba de Eratóstenes

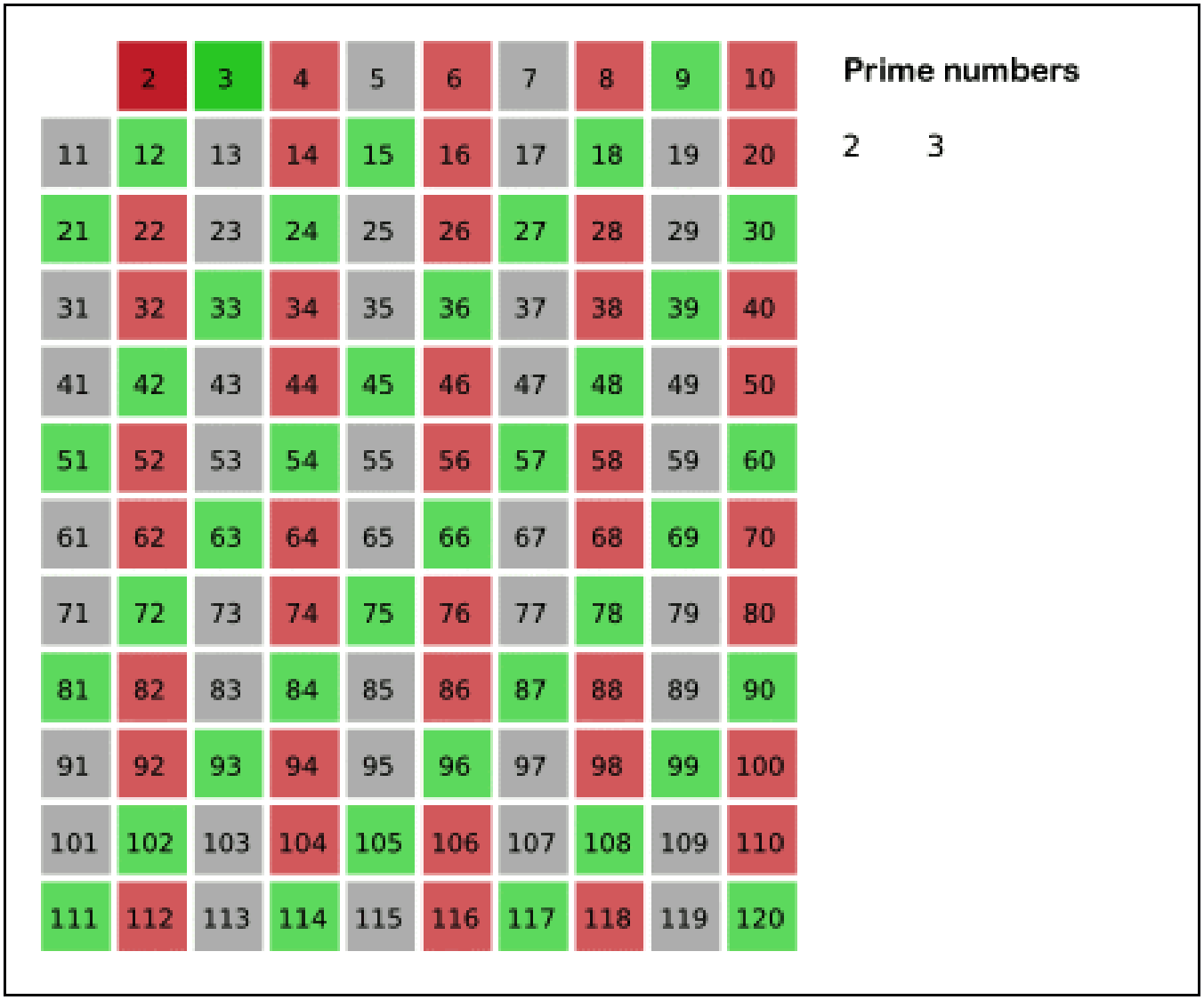


Criba de Eratóstenes

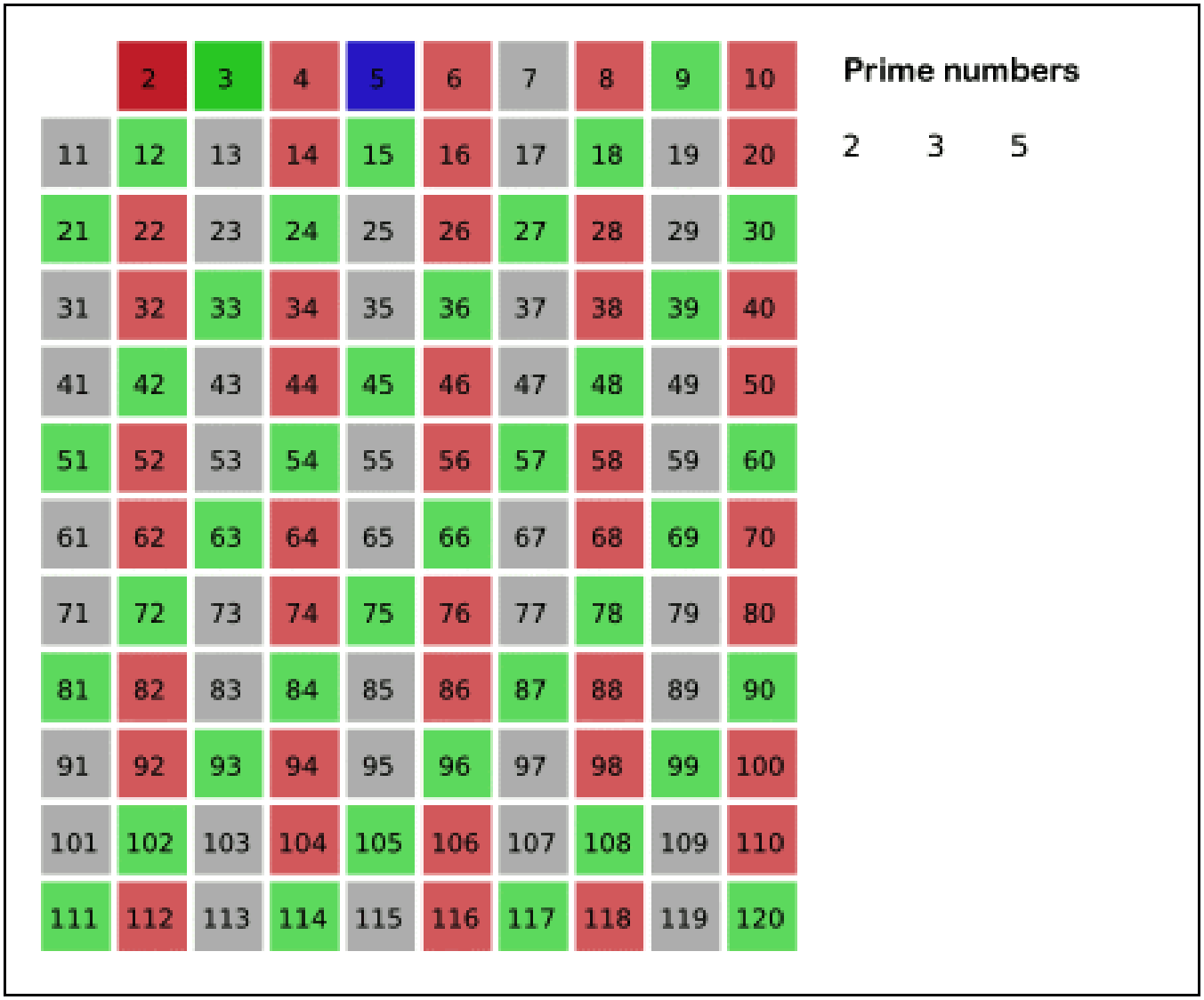


Criba de Eratóstenes

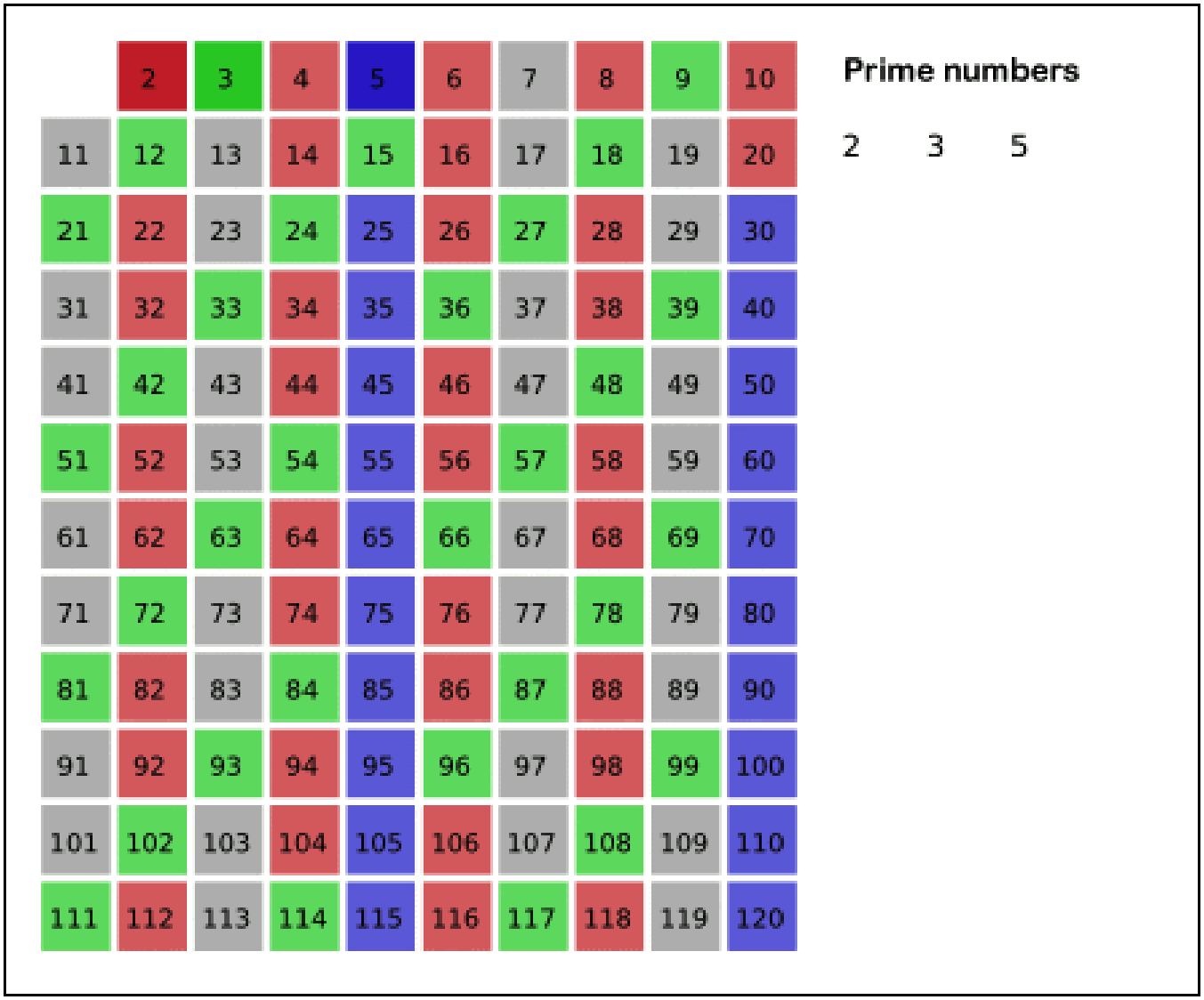
Criba de Eratóstenes



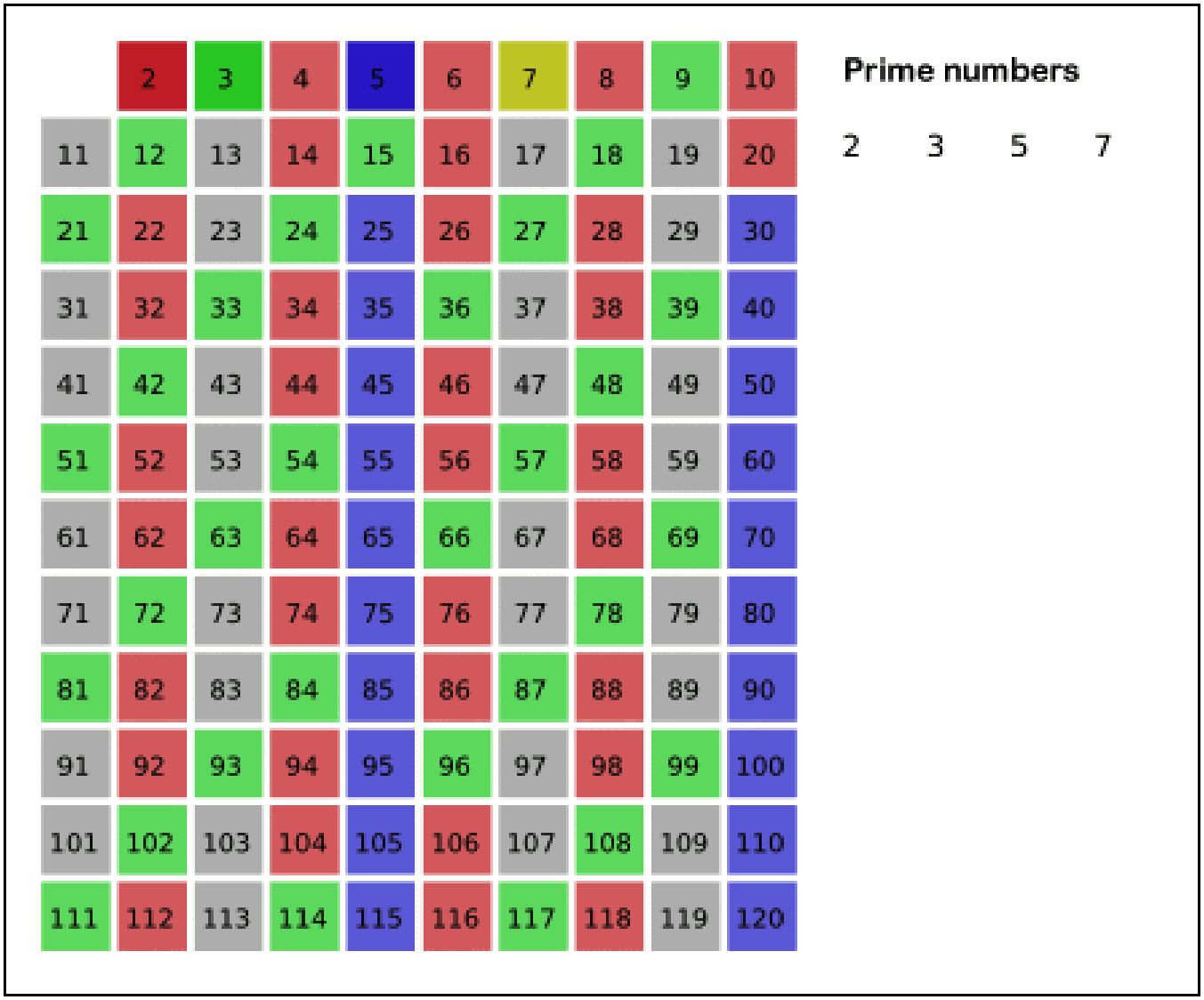
Criba de Eratóstenes



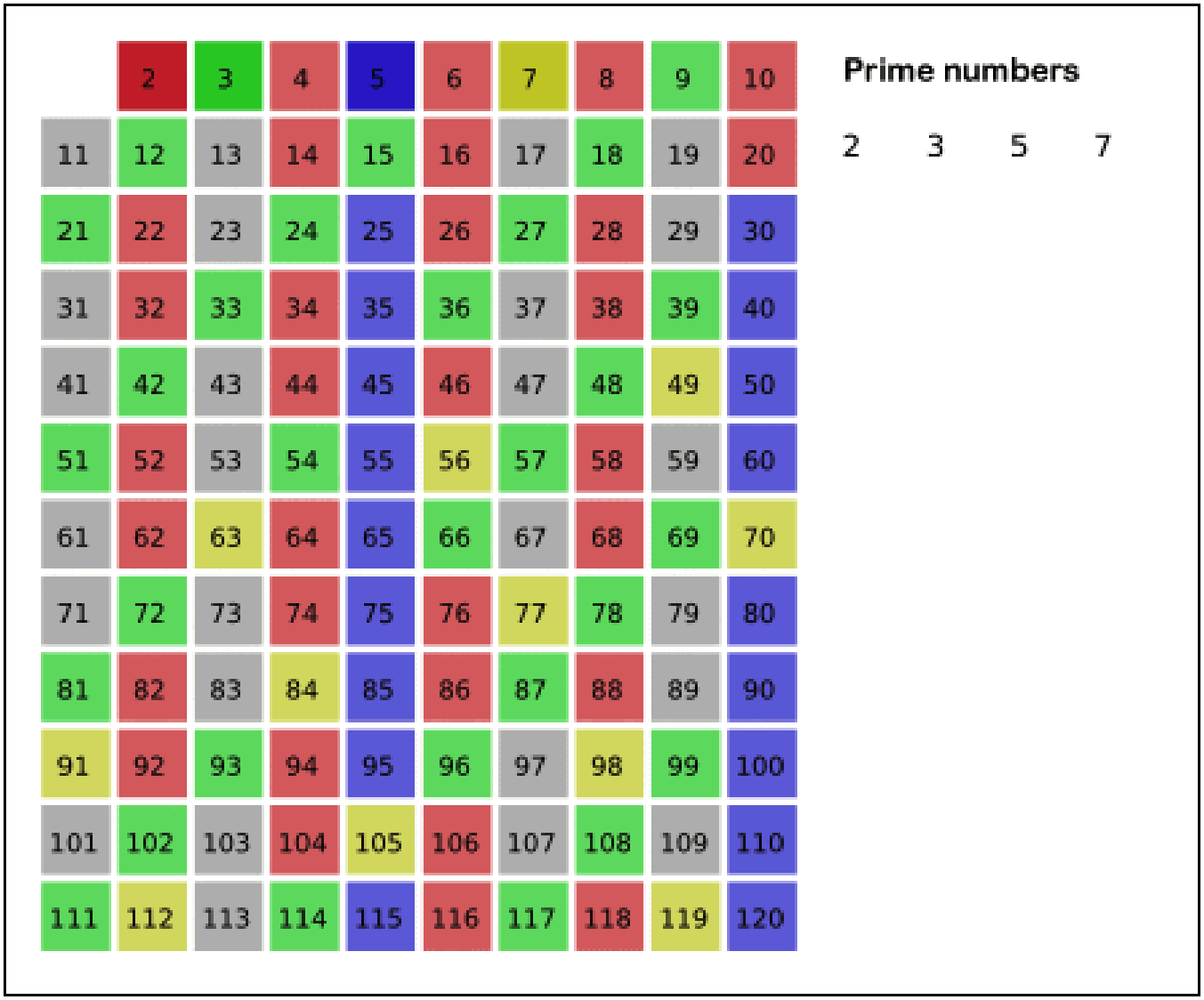
Criba de Eratóstenes



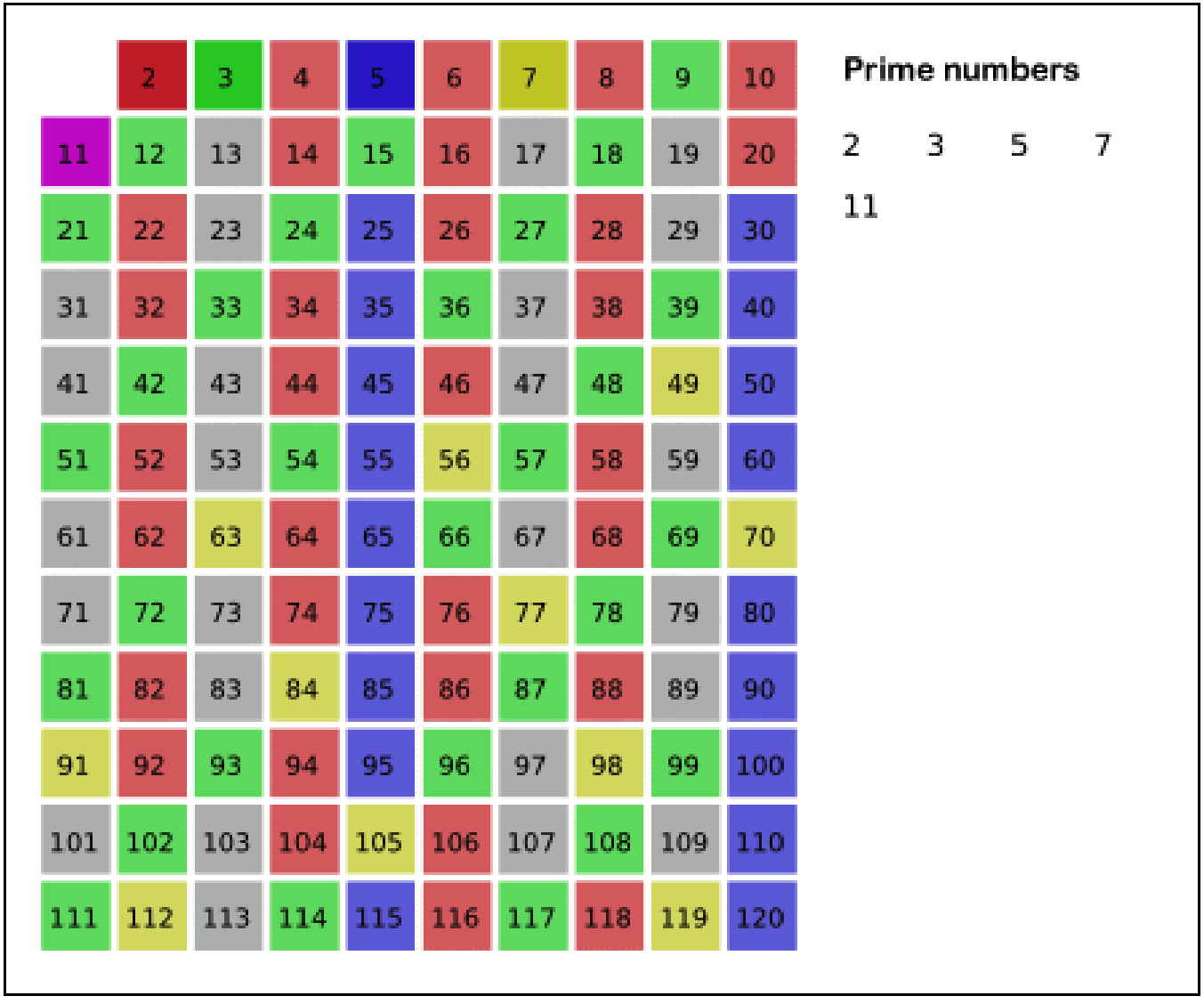
Criba de Eratóstenes



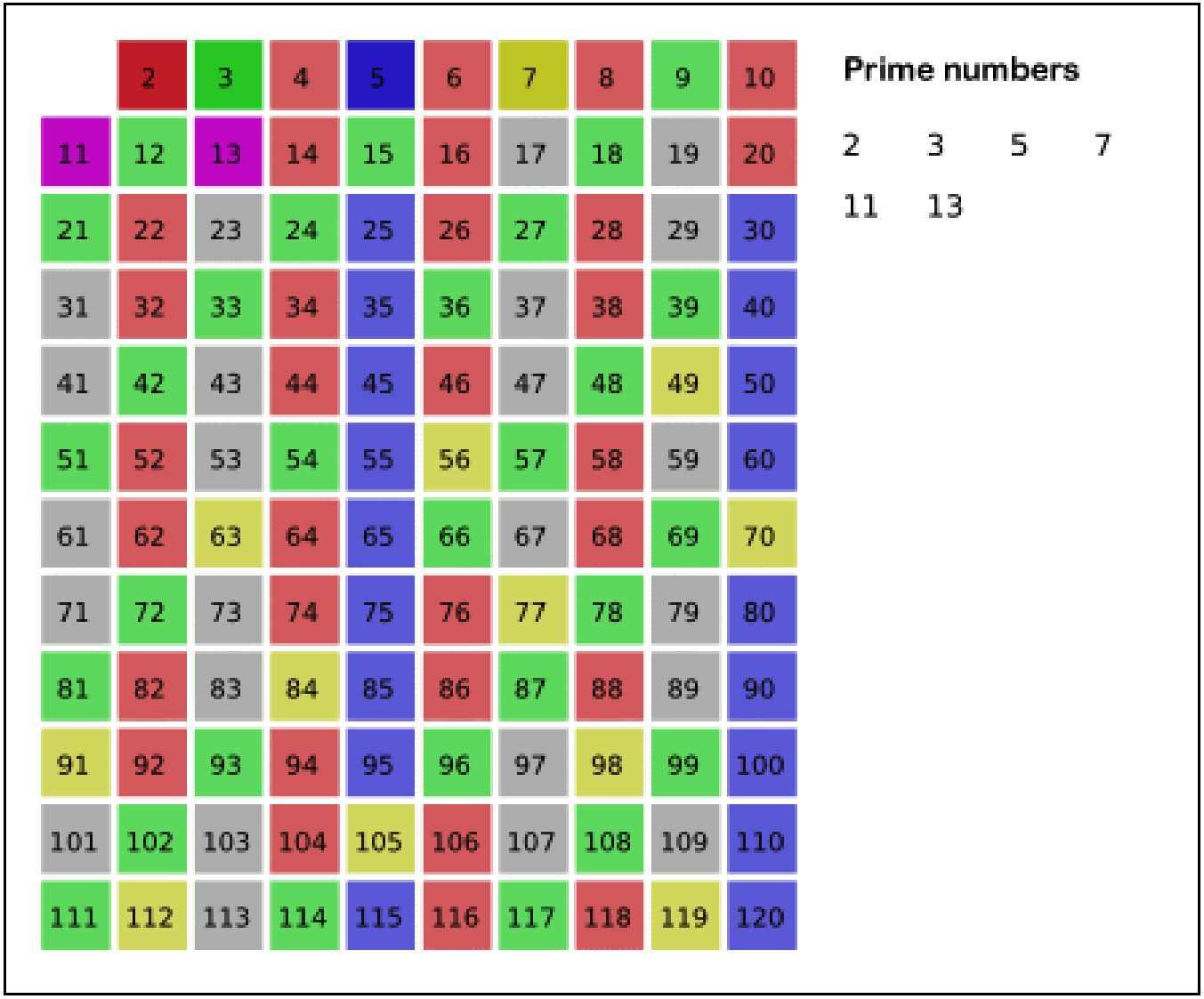
Criba de Eratóstenes



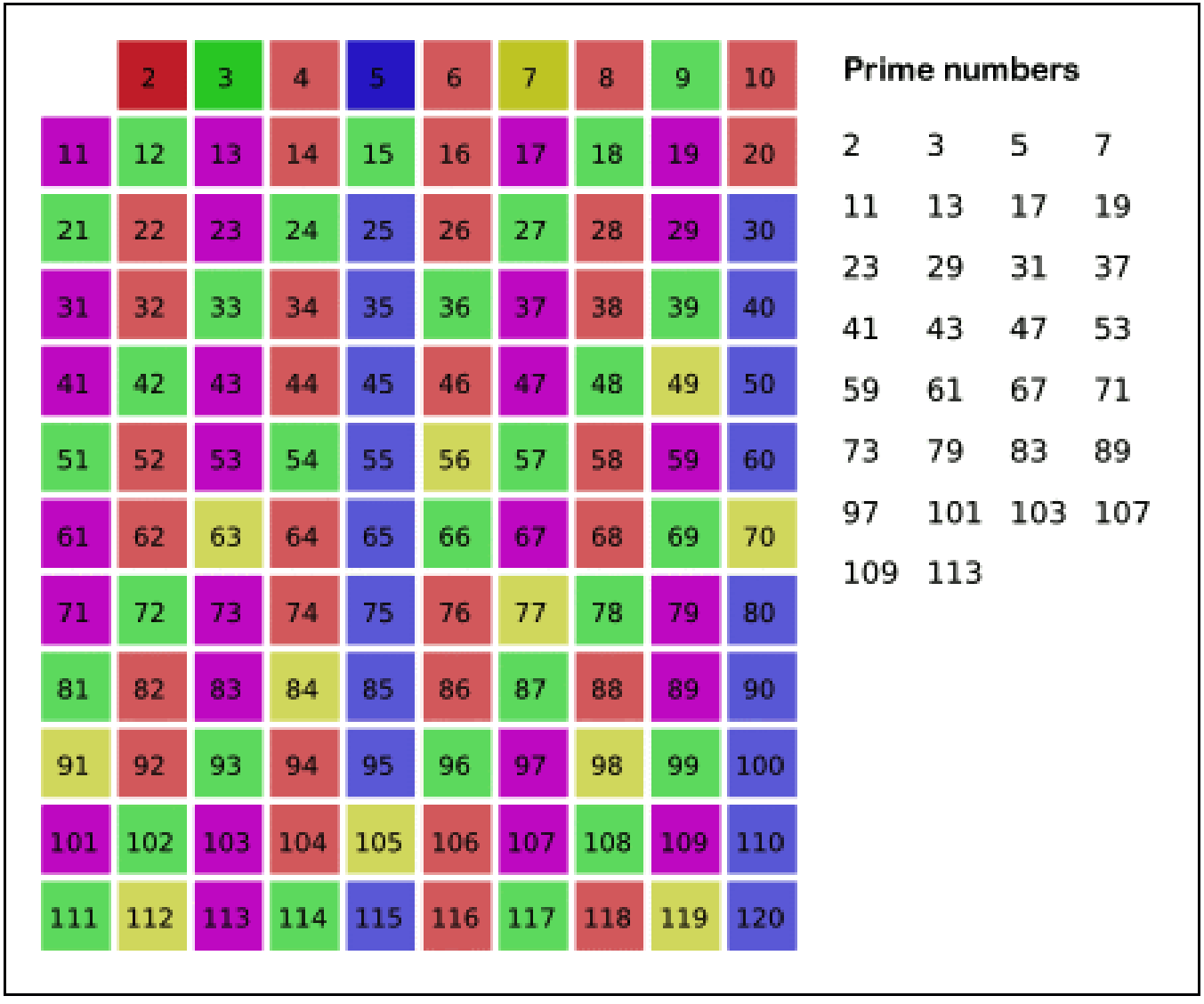
Criba de Eratóstenes



Criba de Eratóstenes



Criba de Eratóstenes



```cpp
const int MAXN = 1000000;
bool sieve[MAXN + 5];
vector <int> primes;

void build_sieve(){
    memset(sieve, false, sizeof(sieve));
    sieve[0] = sieve[1] = true;

    for (int i = 2; i * i <= MAXN; i ++){
        if (!sieve[i]){
            for (int j = i * i; j <= MAXN; j += i){
                sieve[j] = true;
            }
        }
    }
    for (int i = 2; i <= MAXN; ++i){
        if (!sieve[i]) primes.push_back(i);
    }
}
```

Complejidad

Pregunta

¿Cuántas veces se visita cada número?
De acuerdo a eso, ¿cuál es la complejidad de la criba?

Complejidad

Pregunta

¿Cuántas veces se visita cada número?
De acuerdo a eso, ¿cuál es la complejidad de la criba?

Respuesta

Como cada número se visita una única vez, la complejidad del algoritmo es $O(n)$.

Contenido

5 Hallar la factorización prima de un número

Factorización prima

Pregunta

Si se conocen todos los primos desde 0 hasta $\sqrt{a}$ ¿cómo usarlos para hallar la factorización prima de a?

Factorización prima

Pregunta

Si se conocen todos los primos desde 0 hasta $\sqrt{a}$ ¿cómo usarlos para hallar la factorización prima de a?

Respuesta

- La factorización prima de a contiene máximo un solo primo mayor que $\sqrt{a}$
- Se divide a por todos los primos menores o iguales a su raíz cuadrada
- Hay que tener en cuenta que un primo puede aparecer varias veces en la factorización
- Si luego de dividirlo queda un número mayor que 1, ese número es primo

Factorización prima

```cpp
const int MAXN = 1000000; // MAXN > sqrt(a)
bool sieve[MAXN + 5];
vector <int> primes;

vector <long long> factorization(long long a){
    // Se asume que se tiene y se llamo la funcion build_sieve()
    vector <long long> ans;
    long long b = a;
    for (int i = 0; 1LL * primes[i] * primes[i] <= a; ++i){
        int p = primes[i];
        while (b % p == 0){
            ans.push_back(p);
            b /= p;
        }
    }
    if (b != 1) ans.push_back(b);
    return ans;
}
```

Complejidad

Complejidad

La complejidad del algoritmo para hallar la factorización prima de un número es aproximadamente $O(\sqrt{n})$

Contenido

6 Exponenciación logarítmica

La operación módulo

La operación $a \bmod n$ es el residuo que queda de dividir a por n.

Propiedades

- $(a \bmod n) \bmod n = a \bmod n$
- $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$
- $(a \cdot b) \bmod n = ((a \bmod n) \cdot (b \bmod n)) \bmod n$
- $\left(\frac{a}{b}\right) \bmod n \neq \left(\frac{a \bmod n}{b \bmod n}\right) \bmod n$

La operación módulo

Pregunta

¿Cómo utilizar el módulo para hallar los últimos k dígitos del número a?

La operación módulo

Pregunta

¿Cómo utilizar el módulo para hallar los últimos k dígitos del número a?

Respuesta

Los últimos k dígitos del número a se pueden hallar con la operación $a \bmod 10^k$

Exponenciación logarítmica

Bigmod

Se quiere hallar el valor de $b^p \bmod m$ para valores de b y de p muy grandes (b^p no cabe en un `unsigned long long`). ¿Cómo hacer eso eficientemente?

Exponenciación logarítmica

Bigmod

Se quiere hallar el valor de $b^p \bmod m$ para valores de b y de p muy grandes (b^p no cabe en un `unsigned long long`). ¿Cómo hacer eso eficientemente?

Algoritmo

El valor de $b^p \bmod m = \text{bigmod}(b, p, m)$ se puede calcular en orden $\log(p)$ de la siguiente manera:

$$\text{bigmod}(b,p,m) = \begin{cases} 1 & \text{si } p = 0 \\ \left(\text{bigmod}(b, \frac{p}{2}, m)\right)^2 \bmod m & \text{si } p \text{ es par} \\ \left(b \cdot \text{bigmod}(b, p-1, m)\right) \bmod m & \text{si } p \text{ es impar} \end{cases}$$

Bigmod

```
1  int bigmod(int b, int p, int m){
2      if (p == 0) return 1;
3      if (p % 2 == 0){
4          int mid = bigmod(b, p/2, m);
5          return (1LL * mid * mid) % m;
6      }else{
7          int mid = bigmod(b, p-1, m);
8          return (1LL * mid * b) % m;
9      }
10 }
```

Contenido

7 Combinatoria

Coeficientes binomiales

El valor de $\binom{n}{k}$ que es el número de subconjuntos de k elementos escogidos de un conjunto con n elementos se puede definir como:

$$\binom{n}{k} = \begin{cases} 1 & \text{si } n = k \\ 1 & \text{si } k = 0 \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{si } k < n \end{cases}$$

Nótese que $\binom{n}{k}$ está definido solo si $k \leq n$.

Coeficientes binomiales

```
1  const int MAXN = 30;
2  long long choose[MAXN][MAXN];
3
4  void build_binomial(int N){
5      for (int n = 0; n <= N; ++n) choose[n][0] = choose[n][n] = 1;
6
7      for (int n = 1; n <= N; ++n){
8          for (int k = 1; k < n; ++k){
9              choose[n][k] = choose[n-1][k-1] + choose[n-1][k];
10         }
11     }
12 }
```

Complejidad

Pregunta

¿Cuál es la complejidad del algoritmo para hallar los coeficientes binomiales?

Complejidad

Pregunta

¿Cuál es la complejidad del algoritmo para hallar los coeficientes binomiales?

Respuesta

El algoritmo tiene una complejidad de $O(n^2)$

Permutaciones

Una permutación de un conjunto es un ordenamiento particular de sus elementos.

Permutaciones de elementos diferentes

El número de permutaciones de n elementos diferentes es $n!$

Permutaciones

Permutaciones con elementos repetidos

El número de permutaciones de n elementos donde hay m_1 elementos repetidos de tipo 1, m_2 elementos repetidos de tipo 2, $\ldots$, m_k elementos repetidos de tipo k es

$$\frac{n!}{m_1!m_2!\cdots m_k!}$$

Ejemplo

¿De cuántas formas diferentes se pueden reordenar los números 313?

Permutaciones

Permutaciones con elementos repetidos

El número de permutaciones de n elementos donde hay m_1 elementos repetidos de tipo 1, m_2 elementos repetidos de tipo 2, $\ldots$, m_k elementos repetidos de tipo k es

$$\frac{n!}{m_1!m_2!\cdots m_k!}$$

Ejemplo

¿De cuántas formas diferentes se pueden reordenar los números 313?

Se pueden reordenar de $\frac{3!}{1!\,2!} = 3$ maneras diferentes

Permutaciones

Permutaciones de subconjuntos

El número de permutaciones de k elementos diferentes tomados de un conjunto de n elementos es

$$\frac{n!}{(n-k)!} = k!\binom{n}{k}$$

Contenido

8 Tarea

Tarea

Tarea

Resolver los problemas de
`http://contests.factorcomun.org/contests/60`