Cantidad de divisores y Algoritmos ad-hoc

Metodologías de solución de problemas computacionales

Entendemos por metodología a un esquema o “receta”, genérico y bien estructurado, para la solución de problemas que cumplen ciertas características.

A la luz de esta definición, durante este curso vamos a cubrir las siguientes metodologías:

- Búsqueda exhaustiva (brute-force search)
- Algoritmos voraces (greedy algorithms)
- Programación dinámica (dynamic programming)
- Divide y vencerás (divide & conquer)

\+ **Algoritmos ad-hoc**

Ejemplo: divisores

Dado un valor entero positivo *N*, mostrar todos sus divisores diferentes de 1 y *N*

Solución 1

```
read N
for i = 2 to N-1:
   if N % i = 0:
      print i
```

Si *N* es 100, imprimiría:

2, 4, 5, 10, 20, 25, 50

Número de operaciones: $1 + (N-2)(1+1) \rightarrow f(N) = N - 1$

$\rightarrow O(N)$

Solución 2

```
read N
for i = 2 to N//2:
    if N % i = 0:
        print i
```

Si *N* es 100, imprimiría:

2, 4, 5, 10, 20, 25, 50

Número de operaciones:

$$1 + \left(\frac{N}{2} - 1\right)(1 + 1) \rightarrow f(N) = \frac{N}{2}$$

$$\rightarrow O(N)$$

Solución 3

```
read N
for i = 2 to √N:
    if N % i = 0:
        if i ≠ √N :
            print i, N/i
        else:
            print i
```

Si N es 100, imprimiría:

2, 50, 4, 25, 5, 20, 10

Número de operaciones:

$$1 + \left(\sqrt{N} - 1\right)(1 + 2) \rightarrow f(N) = \sqrt{N}$$

$$\rightarrow O(\sqrt{N})$$

Mínimo Común Múltiplo y Algoritmo de Euclides

Mínimo común múltiplo de dos enteros

Dados dos enteros positivos A y B, cuál es el mínimo valor que es múltiplo de ambos

Solución 1:

```
read A, B
mayor = max(A,B)
menor = min(A,B)
for M = mayor to mayor*menor step mayor:
     if M % menor == 0:
          print M
          break
```

Número de operaciones:

$$f(N) = 2 + 1 + 1 + min(A,B)(1 + 1 + 1) = 4 + 3\min(A,B) \;\; \rightarrow O(\min(A,B))$$

Algoritmo de Euclides

Se basa en:

1) Al dividir *M* entre *N*, ambos números enteros, se obtiene un cociente *Q* más un residuo *R*
2) El máximo común divisor de *M* y *N* es igual que el de *N* y *R*
3) *A*B* = MinimoComunMultiplo(*A, B*) * MaximoComunDivisor(*A, B*)

```
read A, B
M, N = A, B
while N ≠ 0:
     M, N = N, M%N
MaxCD = M
MinCM= A*B/MaxCD
```

La demostraciones se pueden encontrar en *Introduction to Algorithms*, capítulo 31.2

Ejemplo: A es 1043, B es 987

La complejidad del algoritmo es $O(\log(\max(A, B)))$

Algoritmo de Euclides

Ejemplo: *A* es 1043, *B* es 987 → M = 1043, N = 987

*En Python existen los métodos gcd() y lcm() en la librería math

Iteración 1: M = 987, N = 56

Iteración 2: M = 56, N = 35

Iteración 3: M = 35, N = 21

Iteración 4: M = 21, N = 14

Iteración 5: M = 14, N = 7

Iteración 6: M = 7, N = 0 (versus 141 iteraciones de la solución 1)

MinCD = 1043 * 987 / 7 = **147063**

Números primos y Criba de Eratóstenes

Criba de Eratóstenes

Dado un valor entero *N* mayor a 1, mostrar todos los números primos menores o iguales a *N*

Solución 1

```
read N
for i=2 to N
   primo = True
   for j=2 to i-1
      if i % j = 0
         primo = False
   if primo
      print i
```

Número de operaciones:

$$1 + (N-1)(1 + (N-2) * 2 + 1 + 1)$$

$$\rightarrow f(N) = 2N^2 - 3N+1$$

$$\rightarrow O(N^2)$$

Criba de Eratóstenes

Solución 2

```
read N
for i=2 to N
    primo = True
    for j=2 to √i
        if i % j = 0
            primo = False
    if prime
        print i
```

Número de operaciones:

$$1 + 1 + (N - 1)\left(1 + \left(\sqrt{N} - 1\right) * 2 + 1 + 1\right)$$

$$\rightarrow f(N) = 2N\sqrt{N} + N - 2\sqrt{N} + 1$$

$$\rightarrow O(N^{\frac{3}{2}})$$

Criba de Eratóstenes

Paso 1: Crear una tabla con dos filas, en una poner los números enteros desde el 2 hasta el *N* y en la otra *True*

Paso 2: El primer número de la tabla con valor *True* es primo

Paso 3: Todos los múltiplos de ese número, a partir de su cuadrado, se ponen en *False*

Paso 4: Si el cuadrado de ese número es menor que *N* se regresa al paso 2, en caso contario el algoritmo termina.

Paso 5: Todos los números con valor *True* son los primos

Ejemplo: *N* es 20

Paso 1:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

Paso 2 y 3:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
O	O	X	O	X	O	X	O	X	O	X	O	X	O	X	O	X	O	X

Paso 4: $2^2 < 20$ → volver al paso 2

Paso 2 y 3:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
O	O	X	O	X	O	X	X	X	O	X	O	X	X	X	O	X	O	X

Paso 4: $3^2 < 20$ → volver al paso 2

Paso 2 y 3:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
O	O	X	O	X	O	X	X	X	O	X	O	X	X	X	O	X	O	X

Paso 4: $5^2 \geq 20$ → terminar y se obtiene: 2, 3, 5, 7, 11, 13, 17, 19

Criba de Eratóstenes (~240 A.C.)

La complejidad de procesamiento del algoritmo es *O(N*log(log(N)))*

Mientras que su complejidad en memoria es *O(N)*, comparada con las de las soluciones 1 y 2 es que *O(1),* ¿Esto supone un problema?

Depende: por ejemplo, ¿En vez de contarlos o mostrarlos necesitamos almacenarlos y consultarlos en *O(1)*?

En cualquier caso la tabla se puede optimizar empleando un arreglo de booleanos donde el índice desplazado dos unidades hace las veces de la primera fila

Otras cribas

- Criba de Euler, o Criba Lineal Moderna (redescubierta y formalizada entre 1960 y 1970)
- Criba Segmentada (Múltiples autores entre 1960 y 1990)
- Criba de Bays-Hudson (1977)
- Criba de Gries-Misra (1978)
- Criba de Pritchard, o Criba de la rueda (1981)
- Criba de Pomerance, o Criba cuadrática (1981)
- Criba de Atkin (2003)

Criba lineal

```
funcion criba_lineal(N):
    es_primo = arreglo de N+1 valores en True
    primos = arreglo inicialmente vacío
    for i=2 to N:
        if es_primo_i:
            primos.add(i)
        for p in primos:
            if p*i > N:
                break
            es_primo_p*i = False
            if i % p == 0: #p es el menor factor primo de i
                break
    retornar primos
```

Elemento dominante y Algoritmo Boyer-Moore

Elemento dominante

Dado un arreglo *X* con *N* elementos enteros (aunque se puede extrapolar a otros tipos) determinar cuál, si es que lo hay, aparece más de *N*/2 veces

Ejemplo: [9, 6, 8, 5, 8, 8, 7, 8, 8]

Solución 1 (trivial)

```
solucion1(X, N):
   max_conteo = 1
   for i = 0 to N-2
      conteo = 1
      max_conteo = 1
      for j = i+1 to N-1
         if Xi = Xj
            conteo += 1
      if conteo > max_conteo
         max_conteo = conteo
         dominante = Xi
   if max_conteo > N/2
      print dominante
```

Cuya complejidad de procesamiento es $O(N^2)$ y de memoria es $O(1)$

Solución 2 (ordenamiento)

```
solucion2(X, N):
   X.sort()
   k, conteo, max_conteo = 0
   for i = 0 to N-1:
      if Xi = Xk
         conteo += 1
         if conteo > max_conteo:
            max_conteo = conteo
            dominante = Xi
      else:
         conteo = 0
         k = i+1
   if max_conteo > N/2
      print dominante
```

Cuya complejidad de procesamiento es $O(N * \log N)$ y de memoria es $O(1)$

Solución 3 (diccionario)

```
solucion3(X, N):
   d = diccionario
   max_conteo = 1
   for i = 0 to N-1
      if Xi not in d:
          d.add(Xi, 1)
      else:
          d.update(Xi, d.value(Xi)+1)
          if d.value(Xi) > max_conteo
             max_conteo = d.value(Xi)
             dominante = Xi
   if max_conteo > N/2
      print dominante
```

Cuya complejidad de procesamiento es $O(N')$ y de memoria es $O(N')$

Solución 4 (Algoritmo Boyer-Moore)

```
solucion4(X, N):
   conteo = 0
   for i = 0 to N-1
      if conteo = 0
         conteo = 1
         dominante = Xi
      else if dominante = Xi
         conteo += 1
      else
         conteo -= 1
   max_conteo = 0
   for i = 0 to N-1
      if Xi = dominante
         max_conteo += 1
   if max_conteo > N/2
      print dominante
```

Cuya complejidad de procesamiento es $O(N)$ y de memoria es $O(1)$

Ejemplo: [9, 6, 8, 5, 8, 8, 7, 8, 8]