Divide y Conquista

ANÁLISIS Y DISEÑO DE ALGORITMOS
MTI. PEDRO O. PÉREZ M.
TECNOLÓGICO DE MONTERREY, CAMPUS QUERÉTARO

Contenido

- ¿En qué consiste el paradigma Divide y Conquista?
- Algoritmos que aplican este paradigma.
  - Contando inversiones.
  - Par más cercano.
  - Multiplicación de números enteros.
  - Secuencia de suma máxima.

¿En qué consiste este paradigma?

- Es un paradigma muy simple:
  - Divide el problema original en sub-problemas (usualmente se divide a la mitad).
  - Encuentra las sub-soluciones para cada uno de estos subproblemas.
  - Si es necesario, combina las sub-soluciones para obtener la solución del problema inicial.

¿En qué consiste este paradigma?

- Para que esta paradigma pueda ser aplicado, es necesario que el problema puede ser dividido en subproblemas que sean INDEPENDIENTES entre sí.
- Es un paradigma fuertemente recursivo.
- Genera algoritmos muy eficientes (O(n log n) ó O(log n)).

¿En qué consiste este paradigma?

- ¿Conocen algún algoritmo que utilice este paradigma?
  - Quick Sort.
  - Merge Sort.
  - ABB, AVL.
  - Búsqueda binaria.
  - Método de la bisección.

Contando inversiones

- Problema:
  - Dado arreglo de números enteros distintos, A, determinar el número de inversiones que existen. Decimos que dos indices $i < j$ forma una inversión si $a_i > a_j$.
  - Ejemplo: ¿cuántas inversiones hay en el siguiente arreglo: {6 2 4 1 3 5}?

Contando inversiones

- Definiendo un algoritmo básico:
  - Dado un arreglo original A de tamaño n.
  - Dividimos el arreglo en dos mitades, contamos las inversiones que existen en cada uno de las mitades.
  - Contamos el número de inversiones que hay entre las dos mitades.
  - Regresamos la suma de todo lo anterior.

Contando inversiones

```
procedure SORT_AND_COUNT (A, B, low, high):
r ← 0
left ← 0
right ← 0

if (high - low) + 1) == 1 then
    return 0
else
begin
    mid ← FLOOR((high + low / 2))
    left ← SORT_AND_COUNT(A, B, low, mid)
    right ← SORT_AND_COUNT(A, B, mid + 1, high)
    r ← MERGE_AND_COUNT(A, B, low, mid, high)
end
return (r + left + righ)
```

Contando inversiones

```
procedure MERGE_AND_COUNT (A, B, low, mid, high):
left ← low
right ← mid + 1
i ← 0
count ← 0

while left <= mid and right <= high do
begin
     if A[left] < A[right] then
     begin
          B[i] ← A[left]
          left ← left + 1
     end
     else
     begin
          B[i] ← A[right]
          right ← right + 1
          count ← count + 1
     end
     i ← i + 1
end
/* una vez que se terminó una de las mitades del arreglo, mueve los
   Restantes del segundo */
COPY(B, A, low, high)
return count
```

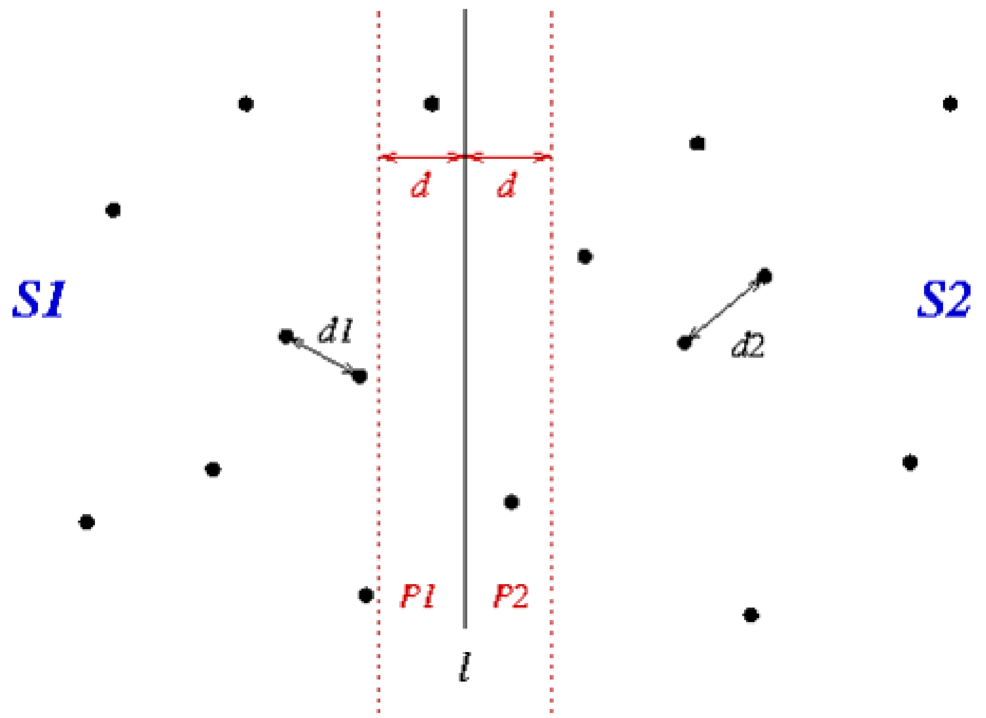
Par mas cercano

- Problema:
  - Dados n puntos en un plano, encontrar el par de punto que se encuentra más cercano.
- Geometría computacional: gráficas, visión computacional, sistemas de información geográficos y modelación molecular.

Par mas cercano

- Puntos a tomar en cuenta:
  - Sea $P = \{p_1, \ldots, p_n\}$ un conjunto de puntos dados. Donde p_i tiene las coordenadas (x_i, y_i); y para dos puntos determinados, $p_i, p_j \in P$, $d(p_i, p_j)$, denota la distancia Euclidiana entre ambos.
  - También, supondremos que no existen dos puntos que tenga la misma coordenada x o *y*.
- ¿Cómo sería resolver el problema para encontrar el par de puntos más cercano en una línea? (O(n log n))

Par mas cercano



Par mas cercano

```
procedure CLOSEST_PAIR_REC(Px, Py):
if |P| <= 3 then
      Encuentra el par más cercano de todos los pares posibles.
else
begin
      Construir Qx, Qy, Rx, Ry ( O(n) )
      (q0, q1) ← CLOSEST_PAIR_REC(Qx, Qy)
      (r0, r1) ← CLOSEST_PAIR_REC(Rx, Ry)

      δ ← MINIMUN( DISTANCE(q0, q1), DISTANCE(r0, r1) )
      x* ← la máximo coordenada en x de un punto en el conjunto Q
      L ← {(x, y) : x = x*}
      S ← los puntos en P que están a distancia δ de L.

      Construir Sy ( O(n) )
      foreach s in Sx do
      begin
            Calcular la distancia de s a s' (Cada uno de los siguientes 15 puntos en Sy)
            Sea (s, s') el par que ha logrado la mínima distancia.
      end

      if DISTANCE(s, s') < δ then
            return (s, s')
      else if DISTANCE(q0, q1) < DISTANCE(r0,r1) then
            return (q0, q1)
      else
            return (r0,r1)
```

Par mas cercano

```
procedure CLOSEST_PAIR(P):
Construir Px y Py ( O(n logn n) )
(p0, p1) ← CLOSEST_PAIR_REC(Px, Py)
```

Multiplicación de números enteros

- Problema:
  - Realizar la multiplicación de dos números enteros
- Algoritmo de Karatsuba.

Multiplicación de números enteros

procedure RECURSIVE_MULTIPLY(x, y):

$x \leftarrow x^1 * 2^{n/2} + x^0$

$y \leftarrow y^1 * 2^{n/2} + y^0$

Calcular $x^1 + x^0$ y $y^1 + y^0$

p ← RECURSIVE_MULTIPLY($x^1 + x^0$, $y^1 + y^0$)

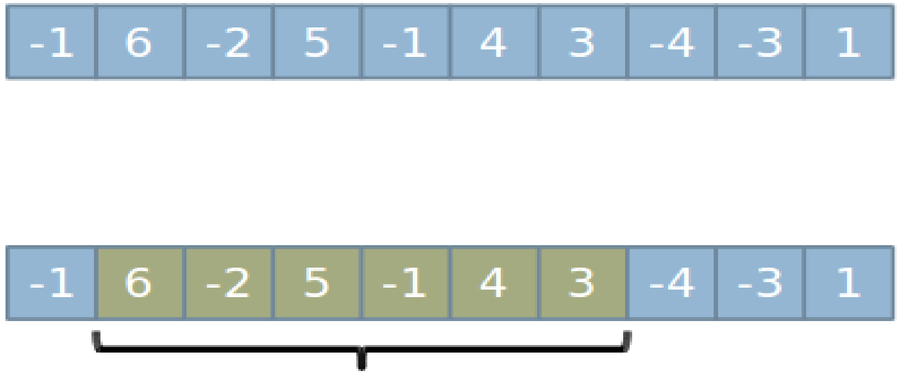
x^1y^1 ← RECURSIVE_MULTIPLY(x^1, y^1)

x^0y^0 ← RECURSIVE_MULTIPLY(x^0, y^0)

return $x^1y^1 * 2^n + (p - x^1y^1 - x^0y^0) * 2^{n/2} + x^0y^0$

Secuencia de suma máxima

- Problema:
  - Dado un arreglo de n números enteros positivos y negativos, encontrar los i elementos del arreglo cuya suma sea la máxima posible.



La secuencia máxima se encuentra comprendida entre la posición 1 y 6. La suma máxima es 15.

Secuencia de suma máxima

```
procedure MAX_SUM(A, low, high):
if (high - low + 1) == 1 then
    return A[low]
else
begin
    mid ← FLOOR( (high + low) / 2)
    max_sum_left ← MAX_SUM(A, low, mid)
    max_sum_right ← MAX_SUM(A, mid + 1, high)

    for i ← mid until 1 step -1 do
        Encontrar el valor máximo que se puede llegar a genera (max_sum1)

    for i ← mid + 1 until high do
        Encontrar el valor máximo que se puede llegar a genera (max_sum2)

    max ← max_sum1 + max_sum2
    if max_sum_left > max then
        max ← max_sum_left
    else if max_sum_right > max then
        max ← mmax_sum_right
    return max
```