Dividir y conquistar

Pedro O. Pérez M., MTI

Análisis y diseño de algoritmos
Tecnológico de Monterrey

pperezm@tec.mx

03-2019

Contenido

Definición

Es una técnica que permite encontrar la solución de un problema descomponiéndolo en subproblemas más pequeños (dividir) y que tienen la misma naturaleza del problema original, es decir, son similares a este. Luego resuelve cada uno de los subproblemas recursivamente hasta llegar a problemas de solución trivial o conocida con antelación (conquistar) para, finalmente, unir las diferentes soluciones (combinar) y así conformar la solución global al problema.

Forma general

Procedure 1 DIVIDE_AND_CONQUER

```
Input: X
  if X is simple or known then
    return SOLUTION(X)
  else
    Decompose X into smaller problems x_1, x_2, ..., x_n
    for i ← 1 to n do
      y_i ← DIVIDE_AND_CONQUER(X_i)
    end for
    Combine the y_i to get the Y that is solution of X
    return Y
  end if
```

Búsqueda binaria

Buscar un elemento x en un arreglo ordenado A de n elementos.

Procedure 2 BINARY_SEARCH

Input: $A : Array, low : Index, high : Index, k : Key$

```
if low > high then
    return −1
else
    mid ← FLOOR((high + low)/2)
    if k = A[mid] then
        return mid
    else if k < A[mid] then
        return BINARY_SEARCH(A, low, mid − 1, key)
    else if k > A[mid] then
        return BINARY_SEARCH(A, mid + 1, high, key)
    end if
end if
```

Permutaciones

Hallar todas las permutaciones de un número. Por ejemplo, las permutaciones de 123 son: $\{123, 231, 321, 312, 132, 213, 123\}$.

Procedure 3 PERMUTATION

Input: $S : String, pos : Integer$

```
if pos > 1 then
    for i ← 1 to pos do
        SWAP(number, i, pos)
        PERMUTATION(number, pos − 1)
        SWAP(number, i, pos)
    end for
else
    print S
end if
```

Torres de Hanoi

En este problema hay tres ejes verticales. En uno de los ejes se acomoda un número indeterminado de discos, todos de diferente tamaño y ordenados de abajo hacia arreglo del más grande al más pequeño.

El reto consiste en mover todos los discos del eje en el que se encuentran a un eje destino utilizando el otro eje como auxiliar., de acuerdo con las siguientes reglas: [a]

- Solo se puede mover un disco a la vez.
- Solo se pueden mover los discos que están en los topes de los ejes.
- No puede quedar un disco más grande sobre uno más pequeño.

Torres de Hanoi

En este problema hay tres ejes verticales. En uno de los ejes se acomoda un número indeterminado de discos, todos de diferente tamaño y ordenados de abajo hacia arreglo del más grande al más pequeño.

El reto consiste en mover todos los discos del eje en el que se encuentran a un eje destino utilizando el otro eje como auxiliar., de acuerdo con las siguientes reglas: [1]

- Solo se puede mover un disco a la vez.
- Solo se pueden mover los discos que están en los topes de los ejes.
- No puede quedar un disco más grande sobre uno más pequeño.

[1] https://goo.gl/bsLeq3

El problema de pasar *n* discos del eje inicial al eje final se puede dividir en el problema de pasar *n - 1* discos del inicial a un eje auxiliar, luego pasar un disco al poste final y finalmente pasar los *n - 1* del eje auxiliar al final (con el mismo algoritmo).

Procedure 4 HANOI

Input: $n : Integer, start : Index, aux : Index, end : Index$

```
if n > 1 then
    HANOI(n − 1, start, end, aux)
    print Move from start to end
    HANOI(n − 1, aux, start, end)
end if
```

Exponenciación rápida

Dados dos números, x y n, calcular el resultado de x^n, haciendo uso de la técnica de dividir y conquistar.

Procedure 5 FAST_POW

Input: $x : Real, n : Integer$

if $n < 0$ **then**
  return $FAST\_POW(1/x, -n)$
else if $n == 0$ **then**
  return 1
else if $n == 1$ **then**
  return x
else if $n \mod 2 = 0$ **then**
  return $FAST\_POW(x * x, n/2)$
else if $n \mod 2 = 1$ **then**
  return $x * FAST\_POW(x * x, (n-1)/2)$
end if

Prefijo común más largo

Dado un conjunto de cadenas, encontrar el prefijo común más largo. Por ejemplo, dadas la siguiente cadenas: “geeksforgeeks”, “geeks”, “geek”, “geezer”, el resultado esperado es: "gee". [2]

[2] https://goo.gl/rqjV76

Procedure 6 FIND_PREFIX

Input: A : $String$, B : $String$

```
result ← ""
i ← 1
j ← 1
while i < A.length and j < B.length do
    if A[i] <> B[j] then
        break
    end if
    result ← result + A[i]
    i ← i + 1
    j ← j + 1
end while
return result
```

Procedure 7 COMMON_PREFIX

Input: $A : Array, low : Index, high : Index$
  if $low == high$ **then**
    return $A[low]$
  end if
  if $low < high$ **then**
    $mid \leftarrow FLOOR((high + low)/2)$
    $str1 \leftarrow COMMON\_PREFIX(A, low, mid)$
    $str2 \leftarrow COMMON\_PREFIX(A, mid + 1, high)$
    return $FIND\_PREFIX(str1, str2)$
  end if

Contando inversiones

Dado arreglo de números enteros distintos, A, determinar el número de inversiones que existen. Decimos que dos indices i < j forma una inversión si $A[i]$ > $A[i]$.

El número de inversiones de un arreglo indica: a qué distancia (o qué tan cerca) está el arreglo de estar ordenado. Si el arreglo ya está ordenado, el conteo de inversión es 0. Si el arreglo está ordenado en orden inverso, el conteo de inversión es el máximo.

Ejemplo: ¿cuántas inversiones hay en el siguiente arreglo: 2, 4, 1, 3, 5? Tiene tres inversiones (2, 1), (4, 1), (4, 3). [3]

[3] `https://goo.gl/DxqxBe`

Procedure 8 SORT_AND_COUNT

Input: $A, B : Array, low, high : Index$

```
r ← 0
left ← 0
right ← 0
if (high − low + 1 = 1) then
    return 0
else
    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)
    COPY(A, B, low, high)
end if
return r + left + right
```

Procedure 9 MERGE_AND_COUNT

Input: $A, B : Array . low, mid, high : Index$

```
...
while left ≤ mid AND right ≤ high do
    if A[left] < A[right] then
        B[i] ← A[left]
        left ← left + 1
    else
        B[j] ← A[right]
        right ← right + 1
        count ← count + (mid − left)
    end if
    i ← i + 1
end while
...
return count
```

Par más cercano

Dado un arreglo de N puntos en un plano, el problema es encontrar el par de puntos más cercano del arreglo. Este problema lo podemos encontrar en una serie de aplicaciones. Por ejemplo, en el control de tráfico áreo, se requiere monitorear los aviones que se acercan demasiado, ya que esto puede indicar una posible colisión.

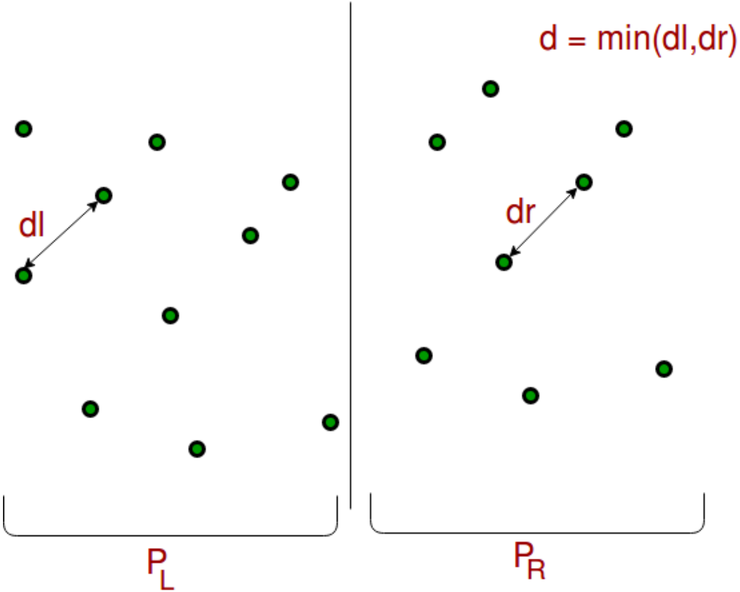
La solución a fuerza bruta, de $O(n^2)$, calcula la distancia existente ente cada par de puntos y devuelve el más cercano. Podemos calcuolar la distancia más cercana en $O(n \log n)$ utilizando la estrategia de Dividir y Conquistar. [4]

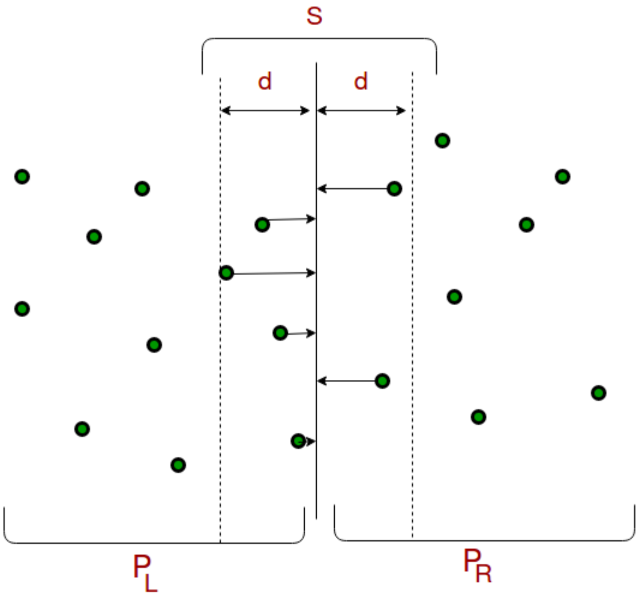
[4] `https://goo.gl/36zwaA`

Procedure 10 CLOSEST_PAIR

Input: $A : Array < Point >$

1. Ordenamos todos los puntos según sus coordenadas x.
2. Divide todos los puntos en dos mitades.
3. Encuentra, recursivamente, las distancias más pequeñas en ambas mitades.
4. Tomamos el mínimo de ambas distancias, d.
5. Crea un arreglo que almacene todos los puntos que se encuentran a una distancia máxima de la línea media que divide ambas mitades.
6. Encuentramos la distancia más corta de este subconjunto.
7. Devolvemos el mínimo, d, y la distancia más pequeña calculada en 4.


d = min(dl,dr)
dl
dr
P
L
P
R


S
d
d
P_L
P_R

Encontrar el número más cercano en el arreglo

Dado un arreglo de números enteros ordenados. Necesitamos encontrar el valor más cercano al número dado. El arreglo puede contener valores duplicados y números negativos. [5]

[5] https://goo.gl/XTgBzX

Procedure 11 FIND_CLOSEST

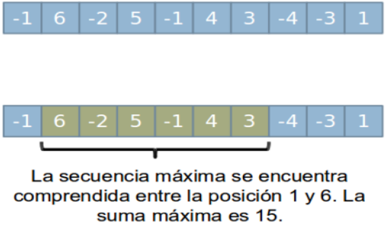
Input: $A : Array, target : Key$

```
if target < A[1] then
    return A[1]
end if
if target < A[A.length] then
    return A[A.length]
end if
low ← 1
high ← A.length
while low < high do
    NEXT SLIDE
end while
return count
```

$mid \leftarrow FLOOR((high + low)/2)$
if $target = A[mid]$ **then**
  return $A[mid]$
else if $target < A[mid]$ **then**
  if $mid > 0$ **and** $target > A[mid-1]$ **then**
    return $CLOSEST(A[mid-1], A[mid], target)$
  end if
  $high \leftarrow mid$
else
  if $mid < A.length$ **and** $target < A[mid+1]$ **then**
    return $CLOSEST(A[mid], A[mid+1], target)$
  end if
  $high \leftarrow mid$
end if

Secuencia de suma máxima

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



Procedure 12 MAX_SUM

Input: $A : Array, low, high : Index$

```
if (high − low + 1) = 1 then
    return A[low]
else
    mid ← FLOOR((high − low)/2)
    left ← MAX_SUM(A, low, mid)
    right ← MAX_SUM(A, mid + 1, high)
    center ← MAX_AUX(A, low, high)
    return MAX(left, right, center)
end if
```

Procedure 13 MAX_AUX

Input: $A : Array, low, high : Index$

$left \leftarrow 0$
$acum \leftarrow 0$
for $i \leftarrow mid$ to low **do**
  $acum \leftarrow acum + A[i]$
  if $acum < 0$ **then**
    $acum \leftarrow 0$
  end if
  $left \leftarrow MAX(left, acum)$
end for

$right \leftarrow 0$
$acum \leftarrow 0$
for $i \leftarrow mid + 1$ to $high$ **do**
  $acum \leftarrow acum + A[i]$
  if $acum < 0$ **then**
    $acum \leftarrow 0$
  end if
  $right \leftarrow MAX(right, acum)$
end for
return $left + right$