Divide y Conquista

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

12-2022

Contenido

Divide y vencerás

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.

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 $i \leftarrow 1$ **to** n **do**
      $y_i \leftarrow DIVIDE\_AND\_CONQUER(X_i)$
    end for
    Combine the y_i to get the Y that is solution of X
    return Y
  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 2 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 mód $2 = 0$ **then**
    return $FAST\_POW(x * x, n/2)$
  else if n mód $2 = 1$ **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". [1]

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

Procedure 3 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 4 COMMON_PREFIX

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

```
if low == high then
    return A[low]
end if
if low < high then
    mid ← FLOOR((high + low)/2)
    str1 ← COMMON_PREFIX(A, low, mid)
    str2 ← COMMON_PREFIX(A, mid + 1, high)
    return FIND_PREFIX(str1, str2)
end if
```

Dado arreglo de números enteros distintos, A, determinar el número de inversiones que existen. Decimos que dos indices $i < j$ forman 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). [2]

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

Procedure 5 SORT_AND_COUNT

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

$r \leftarrow 0$
$left \leftarrow 0$
$right \leftarrow 0$
if $(high - low + 1 = 1)$ **then**
  return 0
else
  $mid \leftarrow FLOOR((high + low)/2)$
  $left \leftarrow SORT\_AND\_COUNT(A, B, low, mid)$
  $right \leftarrow SORT\_AND\_COUNT(A, B, mid + 1, high)$
  $r \leftarrow MERGE\_AND\_COUNT(A, b, low, mid, high)$
  $COPY(A, B, low, high)$
end if
return $r + left + right$

Procedure 6 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
```

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. [3]

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

Procedure 7 GET_CLOSEST

Input: $val1, val2, target : Key$

```
if (target − val1) > (val2 − target) then
    return val2
else
    return val1
end if
```

Procedure 8 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
```

```
mid ← 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 GET_CLOSEST(A[mid − 1], A[mid], target)
    end if
    high ← mid
else
    if mid < A.length and target < A[mid + 1] then
        return GET_CLOSEST(A[mid], A[mid + 1], target)
    end if
    high ← mid
end if
```