Complejidad Algorítmica

Unidad 1: Comportamiento asintótico, métodos de búsquedas y grafos

Módulo 3: Algoritmo Divide y Vencerás

Complejidad Algorítmica

Semana 3 / Sesión 1

MÓDULO 3: Algoritmo Divide y Vencerás

Contenido

1. Definición del Algoritmo Divide y Vencerás
2. Ejemplos clásicos
3. Complejidad Algorítmica

Preguntas

1. Definición Algoritmo Divide y Vencerás

¿Cómo definimos este algoritmo?

- El algoritmo **Divide y Vencerás** tiene una <u>estructura recursiva</u>.
- Para resolver el problema dado, se llama a sí mismo recursivamente una o más veces para tratar subproblemas estrechamente relacionados.
- El algoritmo **divide y vencerás** implica <u>tres pasos</u> en cada nivel de recursividad.

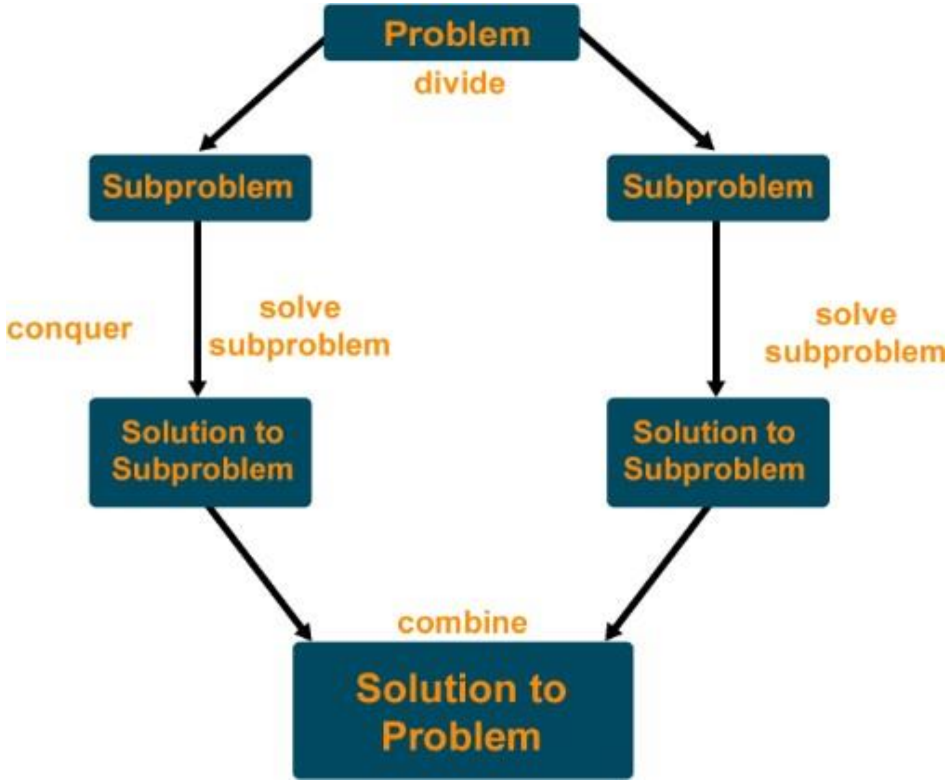
1. **Dividir:** dividir el problema en varios subproblemas que son similares al problema original pero más pequeños,
2. **Conquistar:** resolver el subproblema recursivamente, y si los tamaños del subproblema son lo suficientemente pequeños, resuelva directamente los subproblemas.
3. **Combinar:** combinar esta solución para crear una solución al problema original.

1. Definición Algoritmo Divide y Vencerás

Seudocódigo

```
Algoritmo 1: divide_y_venceras
Data: p: un problema
Result: solución a p
if p es simple then
    solucionar p;
else
    descomponer p en {p1, p2, ..., pn};
    divide_y_venceras(p1);
    divide_y_venceras(p2);
    ⋮
    divide_y_venceras(pn);
    Combinar_soluciones({p1, p2, ..., pn});
```

Diagrama



2. Ejemplos clásicos

EJEMPLOS - PROBLEMAS CLASICOS QUE APLICAN DIVIDE Y VENCERAS

1. Hallar el máximo valor dentro de un arreglo no ordenado.

```
Algoritmo  : Hallar el máximo valor de un arreglo a.
Data: a: un arreglo
Result: max: un entero
max ← a[1];
for i ← 2 to |a| do
    if a[i] > max then
        max ← a[i];
return max
```

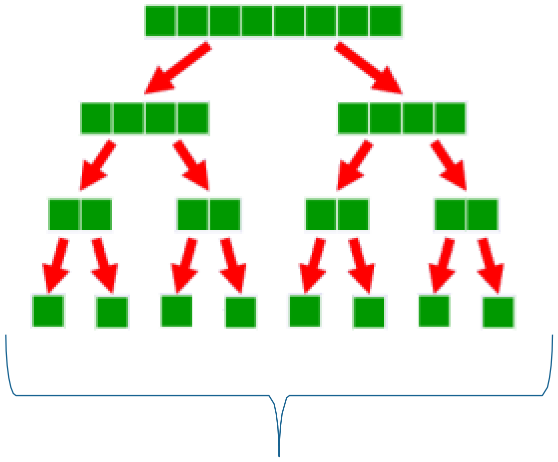
Solución clásica

2. Ejemplos clásicos

EJEMPLOS - PROBLEMAS CLASICOS QUE APLICAN DIVIDE Y VENCERAS

1. Hallar el máximo valor dentro de un arreglo no ordenado.

Gráficamente:



Aplicando Divide y Vencerás

Seudocódigo:

1. Dividimos el arreglo en dos partes.
2. Hallamos el máximo de cada parte.
3. Seleccionamos el mayor de los dos.
4. Volvemos a aplicar recursivamente el enfoque.

Algoritmo 3: maximo(a, i, j)

```
if i = j then
    return max;
else
    med ← (i + j)/2;
    max_i ← maximo(a, i, med);
    max_d ← maximo(a, med + 1, j);
    if max_i > max_d then
        return max_i;
    else
        return max_d;
```

2. Ejemplos clásicos

EJEMPLOS - PROBLEMAS CLASICOS QUE APLICAN DIVIDE Y VENCERAS

2. Multiplicación de enteros de n cifras

$\boxed{M = 9876 \times 5678}$

Algoritmo Clásico

$M = 9876 \times 5678$

$M = 9876 \times (5 \times 1000 + 6 \times 100 + 7 \times 10 + 8)$

$$
\begin{aligned}
M = {} & 9876 \times (5 \times 1000) + \\
& 9876 \times (6 \times 100) + \\
& 9876 \times (7 \times 10) + \\
& 9876 \times 8
\end{aligned}
$$

Operaciones básicas:
- Multiplicaciones de dígitos: n2 −1
- Sumas de dígitos: n

Algoritmo Divide y Vencerás

$M = 9876 \times 5678$

$M = (98 \times 100 + 76) \times (56 \times 100 + 78)$

$$
\begin{aligned}
M = {} & (98 \times 56) \times 10000 + \\
& (98 \times 78 + 76 \times 56) \times 100 + \\
& (76 \times 78)
\end{aligned}
$$

Operaciones básicas:
- Multiplicaciones de dígitos: 1 + 2 + 1
- Sumas de dígitos: 3

Eficiencia algoritmo:
Θ(n^2)

2. Ejemplos clásicos

EJEMPLOS - PROBLEMAS CLASICOS QUE APLICAN DIVIDE Y VENCERAS

3. Multiplicación de enteros de n cifras

Multiplicar a = 98765678 y b = 24680135

Algoritmo 4: $\mathrm{mult}(a, b, n)$

if a o b son pequeños then
    return $a \times b$;
else
    obtener a_i, a_d, b_i, b_d;
    $z_1 \leftarrow \mathrm{mult}(a_i, b_i, n/2) \times 10^n$;
    $z_2 \leftarrow (\mathrm{mult}(a_i, b_d, n/2) + \mathrm{mult}(a_d, b_i, n/2)) \times 10^{n/2}$;
    $z_3 \leftarrow \mathrm{mult}(a_d, b_d, n/2)$;
    return $z_1 + z_2 + z_3$;

Divide:

a = 98765678 $= a_i \times$ 10^4 + a_d $\rightarrow a_i$ = 9876, a_d = 5678

b = 24680135 $= b_i \times$ 10^4 + b_d $\rightarrow b_i$ = 2468, b_d = 0135

Combinar:

$a \times b = (a_i \times$ 10^4 + $a_d) \times (b_i \times$ 10^4 + $b_d)$

$a \times b = a_i \times b_i \times$ 10^4 + $(a_i \times b_d + a_d \times b_i) \times$ 10^4 + $a_d \times b_d$

2. Ejemplos clásicos

EJEMPLOS - PROBLEMAS CLASICOS QUE APLICAN DIVIDE Y VENCERAS

3. Multiplicación de matrices cuadradas - Simple

- Sea C = A × B, para dos matrices cuadradas A, B de dimensión n.
- El esquema de la solución "más simple" es:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

- La solución "más simple" realiza:

n^3 multiplicaciones simples (multiplicaciones de dos cifras).
$n^2(n-1)$ sumas.

Su complejidad es $\Theta(n^3)$.

Se descompone cada matriz de dimensión n en 4 sub-matrices de dimensión $\frac{n}{2}$.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

3. Complejidad Algorítmica

La <u>complejidad del tiempo</u> para el **algoritmo divide y vencerás** se calcula utilizando el <u>teorema maestro</u>.

$$T(n) = aT(n / b) + f(n)$$

Donde:

T(n) es la complejidad del algoritmo para una entrada n

n es el tamaño de entrada

a es el número de subproblemas en la recursividad

n/b es el tamaño de cada subproblema donde se supone que todos los subproblemas tienen el mismo tamaño.

Podemos decir que **f(n)** es el trabajo realizado fuera de la llamada recursiva.

Complejidad de la solución

Complejidad por descomponer el problema y combinar las soluciones

$$T(n) = aT(n/b) + O(n^k)$$

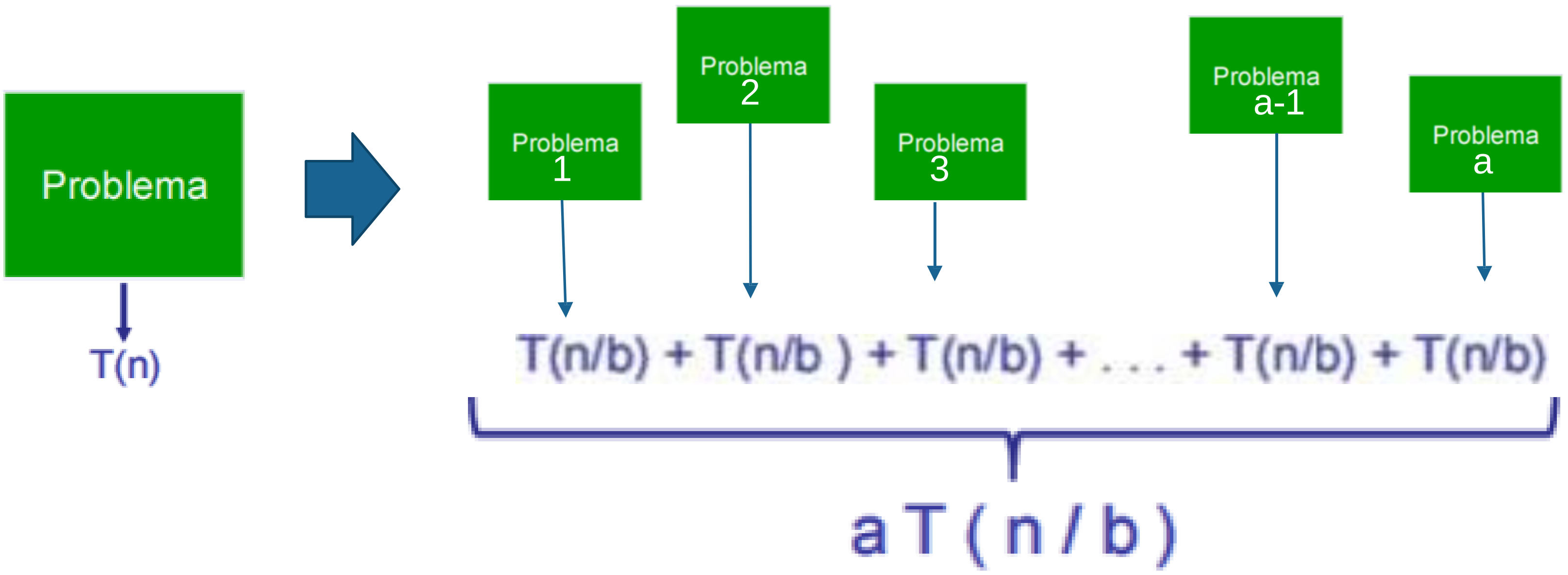
$$a \geq 1, \quad b \geq 2, \quad k \geq 0$$

Que se resuelve de la siguiente forma:

$$T(n) = \begin{cases} O(n^k), & a < b^k \\ O(n^k \log n), & a = b^k \\ O(n^{\log_b a}), & a > b^k \end{cases}$$

https://youtu.be/VplAn4NHYA0
https://www.youtube.com/watch?v=_-5l_gKnnsM

3. Complejidad Algorítmica



Donde:

T(n): complejidad del algoritmo para tamaño n

T(n/b): complejidad para un subproblema

n/b: tamaño de cada nuevo subproblema

a: número de subproblema

3. Complejidad Algorítmica

La complejidad algorítmica de los ejemplos anteriormente descritos de detallan a continuación:

Hallar el máximo valor: $T(n) = 2T(n/2) + n$

Multiplicación de enteros de n cifras: $T(n) = 4T(n/2) + n$

Multiplicación de matrices cuadradas: $T(n) = 4T(n/2) + n^2$

PREGUNTAS

Dudas y opiniones