Parte de Algoritmos
de la asignatura de Programación
Master de Bioinformática

Divide y vencerás

Web asignatura: http://dis.um.es/~domingo/algbio.html

E-mail profesor: domingo@um.es

Transparencias preparadas a partir de las del curso de Algoritmos y Estructuras de Datos II, del Grado de Ingeniería Informática y An Introduction to Bioinformatics Algorithms

Método general

- La técnica **divide y vencerás** consiste en descomponer el problema en un conjunto de subproblemas más pequeños. Después se resuelven estos subproblemas y se combinan las soluciones para obtener la solución para el problema original.

- **Esquema general:**

  divide_venceras (p: problema)

  dividir (p, p_1, p_2, ..., p_k)

  para i = 1, 2, ..., k

  s_i = *resolver* (p_i)

  solucion = *combinar* (s_1, s_2, ..., s_k)

Puede ser recursivo siendo “resolver” una nueva llamada a “divide_venceras”

Si el problema es “pequeño”, entonces se puede resolver de forma directa.

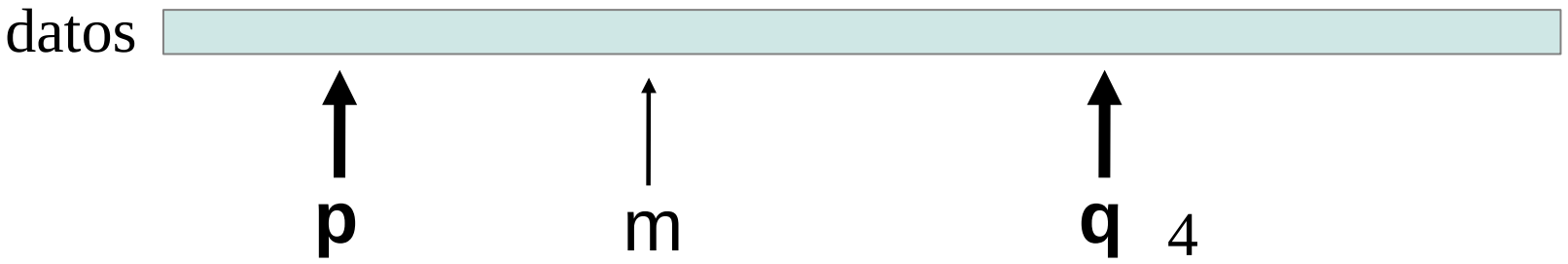
Método general

- Para que pueda aplicarse la técnica divide y vencerás necesitamos:
  - El problema original debe poder dividirse fácilmente en un conjunto de subproblemas, del mismo tipo que el problema original pero con una resolución más sencilla (menos costosa).
  - La solución de un subproblema debe obtenerse independientemente de los otros.
  - Necesitamos un método (más o menos directo) de resolver los problemas de tamaño pequeño.
  - Es necesario tener un método de combinar los resultados de los subproblemas.

Método general, esquema recursivo

- Normalmente para resolver los subproblemas se utilizan llamadas recursivas al mismo algoritmo (aunque no necesariamente).
- **Esquema recursivo (con división en 2 subproblemas):**

```
divide_venceras (p, q: indice)
var m: indice
    si  pequeño (p, q)
        solucion = solucion_directa (p, q)
    en otro caso
        m = dividir (p, q);
        solucion = combinar (divide_venceras (p, m),
                             divide_venceras (m+1, q));
```



Método general, funciones

- Hay que definir las **funciones genéricas**.
  - *pequeño*: determina cuándo el problema es pequeño para aplicar la resolución directa.
  - *solucion_directa*: método alternativo de resolución para tamaños pequeños.
  - *dividir:* función para descomponer un problema grande en subproblemas.
  - *combinar:* método para obtener la solución al problema original a partir de las soluciones de los subproblemas.

Ordenación por mezcla

```
MergeSort (i, j: integer)
/* Es pequeño si el tamaño es menor que un caso base */
    si pequeño(i, j)
        OrdenaciónDirecta(i, j)
    en otro caso
/* El array original se divide en dos trozos de tamaño igual (o lo más
   parecido posible), es decir ⌊n/2⌋ y ⌈n/2⌉ */
        s = (i + j) div 2
    /* Resolver recursivamente los subproblemas */
        MergeSort(i, s)
        MergeSort(s+1, j)
    /* Mezcla dos listas ordenadas. En O(n) */
        Combina (i, s, j)
```

$t(n) = 2*t(n/2)+ f(n)$; Con $f(n) \in O(n)$ Suponiendo n potencia de 2, tenemos: $t(n) = 2 \cdot t(n/2) + a \cdot n + b$ $t(n) \in O(n \cdot \log n)$

Ordenación por mezcla

→Programarlo en Perl.

→Comparar experimentalmente el tiempo de ejecución con el de la ordenación rápida.

Problema LCS

- Por Programación dinámica el coste de tiempo y de memoria es mn, pero no es necesario almacenar la tabla, sino que se puede rellenar por columnas, usando dos vectores, uno para la columna previa y otro para la que se está calculando

		a	c	t	t
	0	0	0	0	0
t	0	0			
a	0	1			
c	0	1			
t	0	1			

		a	c	t	t
	0	0	0	0	0
t	0	0	0		
a	0	1	1		
c	0	1	2		
t	0	1	2		

		a	c	t	t
	0	0	0	0	0
t	0	0	0	1	
a	0	1	1	1	
c	0	1	2	2	
t	0	1	2	3	

		a	c	t	t
	0	0	0	0	0
t	0	0	0	1	1
a	0	1	1	1	1
c	0	1	2	2	2
t	0	1	2	3	3

Así se obtiene la longitud, pero no la cadena, pues no se puede recomponer al no tener almacenadas las decisiones.

Problema LCS

- Esquema Divide y vencerás:

```
LCS(Origen,Destino):
        Si (Origen y Destino en columnas consecutivas)
            Devolver LCS de Origen a Destino
        en otro caso
            Medio=índice medio entre Origen y Destino
            LCS(Origen,Medio)
            LCS(Medio,Destino)
```

El problema es que no conocemos el punto Medio por el que pasa el camino óptimo

Problema LCS

… pero se pueden guardar las longitudes hasta la columna intermedia, y de la columna intermedia en adelante resolver el problema invirtiéndolo

		a	c
	0	0	0
t	0	0	0
a	0	1	1
c	0	1	2
t	0	1	2

2	1	0	t
1	1	0	a
1	1	0	c
1	1	0	t
0	0	0	
t	t		

y la longitud máxima es con la que se obtiene el máximo sumando el prefijo y el sufijo (en el ejemplo hasta el carácter c, que da 3).

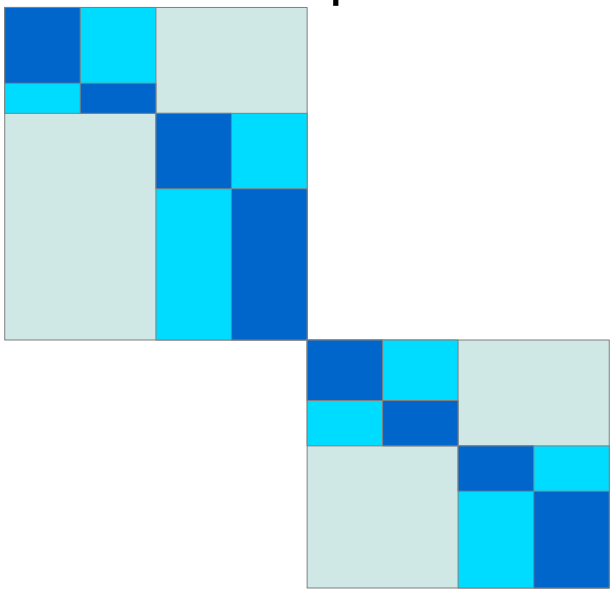
Problema LCS

El coste ha sido de mn, y la ocupación de memoria 2n (dos columnas en la parte izquierda y dos en la derecha, pero sólo se trabaja en una en un momento dado)

Pero sólo hemos calculado la distancia, no la cadena.

Se puede obtener la cadena aplicando Programación dinámica a la izquierda hasta la fila del vértice Medio, y a la derecha desde esa fila. Así el coste es nm/2.

O se puede aplicar Divide y vencerás recursivamente para calcular los puntos medios de los subintervalos:



Coste: n(m+m/2+m/4+...)=

2nm

Memoria: 4n

→Programarlo en Perl.