COMPLEJIDAD DE LOS ALGORITMOS

Por Martín Penttonen

1. Clasificación en tiempo $\theta(n \log n)$

Consideráremos el problema de la clasificación de objetos. Para ello construiremos algunos algoritmos y mediremos su complejidad mediante el número de comparaciones necesarias entre dos objetos para ordenar la sucesión.

Dada una sucesión

$$a_1, a_2, a_3, \ldots, a_n \qquad (n\geq 1)$$

de objetos de un conjunto ordenado, digamos de los enteros. ¿Con cuántas comparaciones o intercambios es posible clasificar la sucesión en orden creciente?

Un algoritmo natural es buscar el objeto más pequeño, después el más pequeño de los restantes, etc:

```
for i:=1  to  n  do
  begin  min:=a_i ;
     for j:=i  to n do
         if a_j < a_i then  intercambiar (a_i,a_j)
  end.
```

Independientemente del orden original, este algoritmo requiere

$$(n-1) + (n-2) + \ldots + 2 + 1 = \frac{(n-1)n}{2} = \theta(n^2)$$

comparaciones. Este algoritmo es bastante eficaz, si n es pequeño, pero si n crece, el tiempo de ejecución crece bastante rapidamente.

¿Hay otros algoritmos? Sí. En muchos casos el algoritmo rápido se basa en el principio "divide y venceras". Dicho algoritmo es la clasificación por intercalaciones, "mergesort"

```
procedure   intercala (i,j,k) ;
begin   r:=i ; s:=j ;
  for   l:=i  to k-l  do
    if   a_r<a_s   and  r<j
    then   begin  b_l:=a_r; r:=r+l  end
    else   begin  b_l:=a_s; s:=s+l  end ;
  for  l:=l  to k-l do a_l:=b_l
end;
procedure   clasifica (i,j) ;
begin
  if  i<j   then
    begin   m:= ⌈(i+j)/2  ;
        clasifica (i,m-l) ;
        clasifica (m,j)   ;
        intercala (i,m,j)
    end
end ;
begin
   clasifica (l,n)
end.
```

Ejemplo:

3	1	4	5	9	2	6	0
3	1	4	5	9	2	6	0
3	1	4	5	9	2	6	0
3	1	4	5	9	2	6	0
1	3	4	5	2	9	0	6
1	3	4	5	0	2	6	9
0	1	2	3	4	5	6	9

La complejidad de la clasificación por intercalaciones: Supongamos que n es una potencia de 2. El problema de clasificar n objetos se divide en dos problemas de n/2 objetos, y la intercalación de dos sucesiones con n/2 objetos. Así

$$T(n) = 2\ T(\tfrac{n}{2}) + n$$

El tiempo total de esta recursión es

$$\begin{aligned} T(n) &= 2\ T(\tfrac{n}{2}) + n \\ &= 4\ T(\tfrac{n}{4}) + 2n \\ &= 8\ T(\tfrac{n}{8}) + 3n \\ &\ \ \vdots \\ &= n\ T(1) + \log n\ .\ n \\ &= \theta(n \log n) \end{aligned}$$

Queda demostrado.

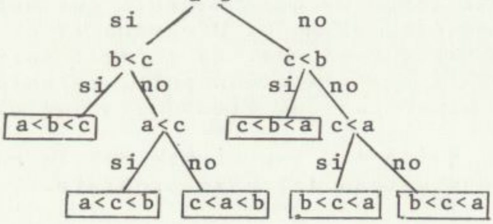
<u>Teorema 1.1</u>. Con la clasificación por intercalciones se puede clasificar n objetos en tiempo $\theta(n \log n)$.

De nuevo podemos preguntarnos. ¿Existen algoritmos mejores?. No. Asintoticamente (aunque no en la práctica) la clasificación por intercalaciones es el mejor algoritmo posible. Vamos a demostrar que el tiempo $\theta(n \log n)$ es optimal.

La ejecución de un algoritmo se puede considerar como una sucesión de decisiones. Nos limitamos al caso en que hay sólo dos alternativas de decisión - sí o no.

Es practico representar el orden de las decisiones en forma de un arbol, llamado <u>arbol de decisión</u>. Este es un grafo cuyos vertices son las preguntas, y de cada vertice salen dos aristas correspondiendo a las decisiones "sí" o "no". Una computación es un camino de la raiz a una hoja del arbol.

<u>Ejemplo</u>: La clasificación de tres objetos distintos a,b y c.



Como se ve, hay seis (=3!) posibles soluciones y seis posibles computaciones.

La profundidad de un árbol es la longitud del camino más largo de la raiz a una hoja. La profundidad del arbol del ejemplo es 3.

Lema 1.1. Si un árbol tiene n hojas y de ningún vértice del árbol salen más de dos aristas, entonces la profundidad del árbol es al menos $\log_2 n$.

Demostración. Demostramos por inducción que si la profundidad de un árbol es p, el arbol no tiene más que 2^p hojas.

p=1 No más de dos hojas.

p>1 Borrando todas las hojas y sus aristas en un árbol T obtenemos un arbol T_1 con la profundidad p-1. Por inducción T_1 no tiene más de 2^{p-1} hojas. Recolocando las hojas borradas vemos que T tiene como máximo $2.2^{p-1}=2^p$ hojas.

Teorema 1.2. Cada algoritmo para la clasificación de n objetos requiere por lo menos $\theta(n \log n)$ comparaciones.

Demostración. Se puede ordenar n objetos con n! maneras. Así, en el árbol de decisión de la clasificación hay n! hojas. Según el Lema 1.1. la profundidad del árbol de decisión, i.e. el número de las comparaciones necesarias, es al menos

$$\log n! \geq \log (n.(n-1) \ldots \lceil \tfrac{n}{2} \rceil) \geq \log (\lceil \tfrac{n}{2} \rceil^{\lfloor \frac{n}{2} \rfloor}) = \theta(n \log n)$$

Una estimación más exacta da

$$\log n! \geq \log_2(\sqrt{2\pi n}\; n^n\, e^{-n+\frac{1}{12n+0.6/n}})$$

$$\geq \log_2(\sqrt{2\pi n}\; e^{n \ln n - n + \frac{1}{12n+0.6/n}}) \quad \text{[Formula de Stirling]}$$

$$\geq n \ln n - n$$

Nota.- Aunque según estos teoremas la clasificación por intercalaciones es optimal, esto no quiere decir que sea así en la practica. Si n es pequeño, el mejor programa es el programa más conciso. Si n crece hay que pensar. Es dificil escribir un programa iterativo para la clasificación por intercalaciones. HEAPSORT es un buen algoritmo del tiempo $\theta(n \log n)$.

En la práctica, quizá más importante que la complejidad del peor caso, sea la complejidad del caso probable.

2. Multiplicación de los enteros en tiempo $(n^{\log_2 3})$

Utilizando el algoritmo de la escuela, dos enteros se pueden multiplicar de la manera siguiente:

$$
\begin{array}{ccccccc}
 & & & 3 & 6 & 2 & 4 \\
 & & & 2 & 3 & 4 & 5 \\
\cline{4-7}
 & & 1 & 8 & 1 & 2 & 0 \\
 & 1 & 4 & 4 & 9 & 6 & \\
1 & 0 & 8 & 7 & 2 & & \\
7 & 2 & 4 & 8 & & & \\
\hline
8 & 4 & 9 & 8 & 2 & 8 & 0
\end{array}
$$

Cuando crecen los enteros, la multiplicación toma más tiempo. Evidentemente el tiempo de la ejecución depende del número de dígitos de los enteros. Consideráremos sólo el caso en que los números de los dígitos en los dos enteros son iguales.

La complejidad del algoritmo de la escuela es:

$$
\begin{array}{ll}
n \cdot n & \text{multiplicaciones} \\
n \cdot n & \text{adiciones} \\
\hline
2\,n^2 & \text{operaciones}
\end{array}
$$

Así $T(n) = 2n^2 = \theta(n^2)$

¿Hay mejores algoritmos? Vamos a ver que el principio "divide y vencerás" da la complejidad $\theta(n^{\log_2 3})$ que es <u>asintoticamente</u> mejor (pero por el coeficiente grande peor en la práctica) [Karachuba & Ofman, Dokl.Ak.Nank SSSR, 1962,t.145 (en ruso)]

Nos limitamos al caso en que los dos enteros (de base $k \geq 2$) tienen $n=2^m$ dígitos ($m \geq o$), añadiendo ceros si fuera necesario.

Tenemos que computar

$$x \cdot y$$

donde x e y tienen $n=2^m$ dígitos. Dividimos cada uno de los enteros en dos bloques de $\frac{n}{2}$ dígitos:

$$x = s \cdot k^{n/2} + t \,,$$

$$y = u \cdot k^{n/2} + v \,,$$

Luego

$$xy = (s \cdot k^{n/2} + t)\,(u \cdot k^{n/2} + v)$$

$$(1) \qquad = su \cdot k^{n} + (sv + tu)\, k^{n/2} + tv$$

$$(2) \qquad = su \cdot k^{n} + [(s+t)(u+v) - su - tr]\, k^{n/2} + tv$$

En (1) el problema se reduce a cuatro multiplicaciones y algunas adiciones. Se puede demostrar que esto da un algoritmo de $\theta(n^2)$. En (2) se ve que tres multiplicaciones del tamaño $\frac{n}{2}$ bastan. En este caso el número de adiciones y substracciones crece, pero estas son operaciones "baratas".

Repitienddo el método aproximadamente $\log_2 n = m$ veces, se reduce el problema a multiplicaciones, adiciones y substracciones de dígitos.

<u>Ejemplo</u>:

$$3624 = 36 \cdot 100 + 24$$

$$2345 = 23 \cdot 100 + 45$$

$$3624 \cdot 2345 = 36 \cdot 23 \cdot 10^4 + [(36+24)(23+45) - 36 \cdot 23 - 24 \cdot 45]\, 10^2 + 24 \cdot 45$$

$$= 36 \cdot 23 \cdot 10^4 + [60 \cdot 68 - 36 \cdot 23 - 24 \cdot 45] \cdot 10^2 + 24 \cdot 45$$

Para continuar es necesario computar los tres productos que aparecen en la formula:

$$36 \cdot 23 = 3 \cdot 2 \cdot 10^2 + [(3+6)(2+3) - 3 \cdot 2 - 6 \cdot 3] \cdot 10 + 6 \cdot 3$$

$$= 6 \cdot 10^2 + [45 - 6 - 18] \cdot 10 + 18$$

$$= 828$$

$$60 \cdot 68 = 6 \cdot 6 \cdot 10^2 + [(6+0)(6+8) - 6 \cdot 6 - 0 \cdot 8] \cdot 10 + 0 \cdot 8$$

$$= 36 \cdot 10^2 + [84 - 36 - 0] \cdot 10 + 0$$

$$= 4080$$

$$24\ 45 = 2 \cdot 4 \cdot 10^2 + [(2+4)(4+5) - 2 \cdot 4 - 4 \cdot 5] + 4 \cdot 5$$

$$= 8 \cdot 10^2 + [54 - 8 - 20] \cdot 10 + 20$$

$$= 1080$$

Computados estos productos podemos acabar la primera multiplicación:

$$3624 \cdot 2345 = 828 \cdot 10^4 + [4080-828-1080] \cdot 10^2 + 1080$$
$$= 828 \cdot 10^4 + 2172 \cdot 10^2 + 1080$$
$$= 8498280$$

Puede ocurrir que el número en las sumas $s+t$ y $u+v$ se mayor que s,t,u y v. Por eso el tamaño del nuevo problema es un poco más grande que la mitad del original. Esta dificultad se puede evitar facilmente separando el dígito último de estas sumas:

$$s+t = s_1 \cdot k^{n/2} + t_1,$$
$$u+v = u_1 \cdot k^{n/2} + v_1,$$

donde s_1 y u_1 tienen $n/2$ dígitos y t_1 y v_1 tienen un dígito.

Con esta notación tenemos

$$(s+t)(u+v) = s_1 u_1 \cdot k^n + (s_1 v_1 + u_1 t_1) k^{n/2} + t_1 v_1$$

Aquí $s_1 u_1$ es un problema del tamaño $n/2$, mientras $s_1 v_1$ y $u_1 t_1$ no requieren más que n operaciones cada uno, y $t_1 v_1$ requiere sólo una operación.

En total

$$xy = su \cdot k^n + [(s+t)(u+v) - su - tv]\ k^{n/2} + tv$$
$$= s_1 u_1 k^{3n/2} + (su + s_1 v_1 + u_1 t_1)\ k^n + (t_1 v_1 - su - tv)\ k^{n/2} + tv$$

requiere 3 multiplicaciones ($su, s_1 v_1$ y tv) de tamaño $n/2$, y además un número lineal, digamos cn, de operaciones elementales, así que tenemos

$$T(n) = 3T\left(\tfrac{n}{2}\right) + cn$$
$$= 9T\left(\tfrac{n}{4}\right) + cn + \tfrac{3}{2}cn$$
$$= 27T\left(\tfrac{n}{8}\right) + cn + \tfrac{3}{2}cn + \left(\tfrac{3}{2}\right)^2 cn$$
$$\vdots$$
$$= 3^{\log_2 n}\ T(1) + cn + \tfrac{3}{2}cn + \ldots + \left(\tfrac{3}{2}\right)^{\log - 1} cn$$
$$= n^{\log_2 3} + \frac{(3/2)^{\log n} - 1}{3/2 - 1}\ cn$$
$$= (2c+1)\ n^{\log_2 3} - 2cn$$
$$= \theta(n^{\log_2 3}) \le \theta(n^{1 \cdot 6})$$

Teorema 2.1. Se pueden multiplicar dos enteros de n dígitos usando como máximo $\theta(n^{\log_2 3})$ operaciones de dígitos.

Nota.- El coeficíente de $n^{\log_2 3}$ es tan grande que el algoritmo no es útil.

3. Multiplicación de las matrices en tiempo $\theta(n^{\log_2 7})$

En 1.968 V.Strassen [Numerische Mathematik 13, 354-356] demostró que hay algoritmos más rapido para la multiplicación e inversión de las matrices que "los algoritmos de la escuela".

Vamos a demostrar este resultado. Una vez más el algoritmo utiliza el principio "divide y vencerás".

El producto de dos matrices n×n se define como

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & & \\ b_{n1} & & b_{nn} \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & & \\ c_{n1} & \cdots & c_{nn} \end{pmatrix}$$

en donde

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$$

Aquí cada c_{ij} requiere n multiplicaciones y n-1 adiciones, y puesto que la matriz contiene n^2 elementos, la complejidad total de la multiplicación es $2n^3-n^2$ operaciones de los elementos. Así la complejidad del algoritmo de la escuela es $\theta(n^3)$.

El algoritmo de Strassen se basa en el principio "divide y vencerás" y en el lema siguiente según el cual, es posible multiplicar matrices pequeñas rapidamente.

Lema 3.1. Para multiplicar dos matrices 2×2 bastan 7 multiplicaciones, 7 substracciones y 11 adiciones de los elementos.

Demostración. [G.Yuval: Inf. Proc. Lettus 7, 285-286 (1978)].

Dadas dos matrices $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ y $\begin{pmatrix} e & f \\ g & h \end{pmatrix}$

La multiplicación se puede escribir en forma

$$\begin{pmatrix} a & \cdot & b & \cdot \\ \cdot & a & \cdot & b \\ c & \cdot & d & \cdot \\ \cdot & c & \cdot & d \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} q \\ r \\ s \\ t \end{pmatrix}$$

donde los punotos son ceros.

Descomponemos esta matriz 4×4:

$$\begin{pmatrix} a & \cdot & b & \cdot \\ \cdot & a & \cdot & b \\ c & & d & \\ \cdot & c & \cdot & d \end{pmatrix} = \begin{pmatrix} a & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & d & \cdot & d \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & d & \cdot & d \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & b-a & \cdot \\ \cdot & a-d & \cdot & b-d \\ c-a & \cdot & d-a & \cdot \\ \cdot & c-d & \cdot & \cdot \end{pmatrix}$$

$$= \begin{pmatrix} a & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & d & \cdot & d \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & d & \cdot & d \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & a-d & a-d & \cdot \\ \cdot & d-a & d-a & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & b-a & \cdot \\ \cdot & \cdot & d-a & b-d \\ c-a & a-d & \cdot & \cdot \\ \cdot & c-d & \cdot & \cdot \end{pmatrix}$$

$$= \begin{pmatrix} a & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a & \cdot & a & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & d & \cdot & d \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & d & \cdot & d \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & a-d & a-d & \cdot \\ \cdot & a-d & a-d & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} +$$

$$\begin{pmatrix} \cdot & \cdot & b-a & \cdot \\ \cdot & \cdot & b-a & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & d-b & b-d \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & c-d & \cdot & \cdot \\ \cdot & c-d & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ c-a & c-a & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

Nótese que cada una de estas operaciones requiere solamente una multiplicación. Con esta descomposición podemos explicar el producto de las dos matrices en forma siguiente:

$$\begin{pmatrix} q \\ r \\ s \\ t \end{pmatrix} = \left[\begin{pmatrix} \\ \\ \\ \\ \end{pmatrix} + \dots + \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix} \right] \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

$$= \begin{pmatrix} a(e+g) \\ \cdot \\ a(e+g) \\ \cdot \end{pmatrix} + \begin{pmatrix} \cdot \\ d(f+h) \\ \cdot \\ d(f+h) \end{pmatrix} + \begin{pmatrix} \cdot \\ (a-d)(f+g) \\ (a-d)(f+g) \\ \cdot \end{pmatrix} + \begin{pmatrix} (b-a)g \\ (b-a)g \\ \cdot \\ \cdot \end{pmatrix} +$$

$$\begin{pmatrix} \cdot \\ (d-b)(g-h) \\ \cdot \\ \cdot \end{pmatrix} + \begin{pmatrix} \cdot \\ \cdot \\ (c-d)f \\ (c-d)f \end{pmatrix} + \begin{pmatrix} \cdot \\ \cdot \\ (c-a)(e-f) \\ \cdot \end{pmatrix}$$

En esta fórmula se ve que la multiplicación requiere

7 multiplicaciones elementales
7 substracciones
11 adiciones ∎

Ahora aplicamos el mismo método al caso general. Vamos a computar el producto de las matrices

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \quad y \quad B = \begin{pmatrix} b_{11} & \dots & b_{1n} \\ \vdots & & \vdots \\ b_{n1} & \dots & b_{nn} \end{pmatrix}$$

Añadiendo filas y columnas de ceros podemos suponer que $n = 2^m$ para algún m. Dividimos las dos matrices en bloques $\frac{n}{2}\times\frac{n}{2}$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad , \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Entonces

$$A\cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

donde

$$C_{ij} = A_{i1}\, B_{1j} + A_{i2}\, B_{2j} \qquad (i,j = 1,2)$$

Computamos el producto con el método delLema 3.1.

Teorema 3.1. La multiplicación de dos matrices n×n con el algoritmo de Strassen requiere $\theta(n^{\log_2 7})$ multiplicaciones de números.

Demostración. Según Lema 3.1.

$$T(n)= \begin{cases} 1 & \text{si} \quad n=1 \\ 7\, T(\frac{n}{2}) + 18(\frac{n}{2})^2 & \text{si} \quad n\geq 2 \end{cases}$$

Demostramos por inducción que

$$T(n)= 7\, n^{\log_2 7} - 6\, n^2$$

En el caso n=1 la aserción es evidente. Completamos la inducción:

$$\begin{aligned} T(2n) &= 7\, T(n) + 18\, n^2 \\ &= 7\, (7\, n^{\log_2} - 6n^2) + 18n^2 \\ &= 7\cdot 7^{1+\log_2 7} - 24n^2 \\ &= 7\, 7^{\log_2 n} - 6(2n)^2 \\ &= 7(2n)^{\log_2 7} - 6(2n)^2 \quad \blacksquare \end{aligned}$$

Nota 1. El algoritmo con estos coeficientes es peor que "el algoritmo de la escuela" hasta n=700. Sin embargo, es bastante facil modificar el algoritmo de tal manera que $T(n)=3.9.n^{\log_2 7}$, lo que da mejor resultado ya, desde n≥35.

Nota 2. El mismo método da el tiempo $\theta(n^{\log_2 7})$ para la inversión de las matrices.

4. Transformación de Fourier rapida

En este párrafo representamos una técnica avanzada aplicable a la multiplicación de los polinomios y de los enteros la transformada de Fourier.

En el análisis continuo la transformada de Fourier se define:

$$f(x) \qquad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)e^{ixt}dt \qquad \text{(transformación)}$$

$$g(x) \qquad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(t)e^{-ixt}dt \qquad \text{(transformación inversa)}$$

En el caso de las funciones discretas la integral se reemplaza por la suma y e^i por una raiz arbitraria de la unidad.

Sea R un anillo conmutativo, en que cada entero $\neq 0$ tiene un elemento inverso, y que contiene una enésima raiz principal de la unidad, i.e un ω tal que

(i) $\omega \neq 1$,

(u) $\omega^n = 1$,

(m) $1+\omega^p+\omega^{2p}+ \ldots +\omega^{(n-1)p} = 0$ para todo $p \in \{1, \ldots, n-1\}$.

Evidentemente en el anillo de los números complejos $e^{\frac{2\ i}{n}}$ es enésima raiz de la unidad. [También en el anillo $\mathbb{Z}_m$, donde m es un número de Fermat $2^{2^k}+1$, 2 es 2^{k+1} esima raiz de la unidad.]

<u>La transformada de Fourier discreta</u> es la función

$$F: R^n \to R^n \qquad (n \geq 1)$$

definida por

$$F(a) = Aa$$

donde $\qquad A_{ij} = \omega^{ij} \qquad (0 \leq i,j \leq n-1)$

<u>Lema 4.1</u>. F es una biyección cuya función inversa viene definida por la matriz B: $B_{ij} = \frac{1}{n}\,\omega^{-ij}$ $(i,j=0,\ldots,n-1)$

Demostración

$$(BA)_{ij} = \sum_{k=o}^{n-1} \frac{1}{n}\, \omega^{-ik},\ \omega^{kj} = \begin{cases} 1 & \text{si } i=j \\ \frac{1}{n} \sum_{k=o}^{n-1} \omega^{(j-1)k} = 0 & \text{si } i\neq j \end{cases}$$

Asi BA es la identidad y B es la inversa de A. ∎

La ventaja de la transformación de Fourier se ve en el esquema siguiente:

$$\begin{array}{ccc} (a_o,a_1,\ldots,a_{n-1})\ (b_o,b_1,\ldots,b_{n-1}) & \xrightarrow{\text{convolución}} & (\ldots,\sum_{k=i+J} a_i b_j,\ldots) \\ F\downarrow \qquad F\downarrow & & \uparrow F^{-1} \\ (c_o,c_1,\ldots,c_{n-1})\ (d_o,d_1,\ldots,d_{n-1}) & \xrightarrow{\text{punto a punto}} & (\ldots,c_k \cdot d_k,\ \ldots) \end{array}$$

Tan pronto como tengamos un algoritmo rápido para computar la transformada de Fourier, tendremos un algoritmo rápido para computar la convolución. Vamos a dar un algoritmo rápido.

Si denotamos

$$a = (a_o, a_1, \ldots, a_{n-1}),$$

$$F(a) = (c_o, c_1, \ldots, c_{n-1}),$$

$$f(x) = a_o + a_1 x + \ldots + a_{n-1}\ x^{n-1},$$

entonces

$$c_i = f(\omega^i)\ (i\text{-}o,\ldots,n-1)$$

La computación directa de c_i según esta formula toma el tiempo $\theta(n)$, y la computación de todos los c_i toma el tiempo $\theta(n^2)$. Esto no ayudaría nada, porque la convolución se la puede computar por su definición en tiempo $\theta(n^2)$. Pero hay algoritmos mejores.

<u>Teorema 4.1</u>. La transformada de Fourier se puede computar en $\theta(n \log n)$ operaciones aritméticas de R.

<u>Demostración</u>. Supongamos que $n=2^m$ para algún m

$$\begin{aligned} f(x) &= a_o+a_1x+\ldots+a_{n-1}x^{n-1} \\ &= (a_o+a_2x^2+\ldots+a_{n-2}x^{n-2})+(a_1+a_3x^2+\ldots+a_{n-1}x^{n-2})x \\ &= f_p(x^2)+f_i(x^2)\cdot x, \end{aligned}$$

donde f_p y f_i son polinomios de grado $\frac{n}{2}-1$. Si ω es la enésima raiz de la unidad, entonces ω^2 es la $\frac{n}{2}$ enésima raiz de la unidad, y el problema se divide en dos problemas del tamaño $\frac{n}{2}$. Con los valores de f_p y f_i en $\frac{n}{2}$ puntos $1,\omega^2,\ldots,\omega^{n-2}$ se computan los valores de f en los puntos $1,\omega,\omega^2,\ldots,\omega^{n-1}$ de la manera siguiente:

$$\left.\begin{array}{l} \left.\begin{array}{l} f(\omega^o) = f_p(\omega^o) + f_i(\omega^o)\cdot\omega^o \\ f(\omega^1) = f_p(\omega^2) + f_i(\omega^2)\cdot\omega^1 \\ \vdots \\ f(\omega^{\frac{n}{2}-1}) = f_p(\omega^{n-2}) + f_i(\omega^{n-2})\cdot\omega^{\frac{n}{2}-1} \end{array}\right\} T(\frac{n}{2})+T(\frac{n}{2}) \\ \left.\begin{array}{l} f(\omega^{\frac{n}{2}}) = f_p(\omega^o) + f_i(\omega^o)\cdot\omega^{\frac{n}{2}} \\ f(\omega^{\frac{n}{2}+1}) = f_p(\omega^2) + f_i(\omega^2)\cdot\omega^{\frac{n}{2}+1} \\ \vdots \\ f(\omega^{n-1}) = f_p(\omega^{n-2}) + f_i(\omega^{n-2})\cdot\omega^{n-1} \end{array}\right\} \text{iguales} \end{array}\right\} \begin{array}{l} \text{n multiplicaciones} \\ \text{n adiciones} \end{array}$$

El número total de las operaciones aritméticas es

$$\begin{aligned} T(n) &= 2T(\tfrac{n}{2}) + 2n \\ &= 4T(\tfrac{n}{4}) + 4n \\ &= 8T(\tfrac{n}{4}) + 6n \\ &\vdots \\ &= nT(1) + 2.\log_2 n.n \\ &= \theta(n \log n) \quad \blacksquare \end{aligned}$$

<u>Corolario. 4.1</u>. La transformación inversa se puede computar en tiempo $\theta(n \log n)$ también.

<u>Demostración</u>. Basta reemplazar ω^i por $\frac{1}{n}\,\omega^{-1}$ ∎

Sean

$$a= (a_o,a_1,\ldots,a_{n-1},0,0,\ldots,0)\in R^{2n},\ y$$

$$b= (b_o,b_1,\ldots,b_{n-1},0,0,\ldots,0)\in R^{2n}$$

La <u>convolución</u> de los vectores a y b es

$$a*b = (c_o,c_1,\ldots,c_{n-1},c_n,c_{n+1},\ldots,c_{2n-1}),$$

donde

$$c_k = \sum_{k=i+j} a_i b_j \quad (k=0,\ldots,2n-2), \quad c_{2n-1}=0$$

La computación de la convolución por su definición requiere tiempo $\theta(n^2)$. El algoritmo rápido se basa en la propiedad siguiente de la transformada de Fourier:

<u>Lema 4.2</u>. $F(a*b) = F(a) \cdot F(b)$,
donde $*$ marca la convolución y $\cdot$ marca el producto punto a punto.

<u>Demostración</u>. Demostraremos que

$$F^{-1}(F(a) \cdot F(b) = a*b$$

Denotemos

$$F(a)=(g_o,g_1,\ldots,g_{2n-1}),\ \text{donde}\quad g = \Sigma_{j=o}^{2n-1}\ a_j\omega^{ij}\ ,$$

$$F(b)=(h_o,h_1,\ldots,h_{2n-1}),\ \text{donde}\quad h_i= \Sigma_{j=o}^{2n-1}\ b_k\omega^{ik}\ ,$$

$$F^{-1}(F(a) \cdot F(b) = (p_o,p_1,\ldots,p_{2n-1})$$

Con esta notación,

$$p_t = \frac{1}{2n} \Sigma_{i=o}^{2n-1} (g_i\ h_i)\ \omega^{-ti}$$

$$= \frac{1}{2n} \Sigma_{i=o}^{2n-1} \Sigma_{j=o}^{2n-1} \Sigma_{k=o}^{2n-1} a_j\ b_k\ \omega^{(j+k-t)i}$$

$$= \Sigma_{j=o}^{2n-1} \Sigma_{k=o}^{2n-1} a_j\ b_k \underbrace{\left(\frac{1}{2n} \Sigma_{i=o}^{2n-1}\ \omega^{(j+k-t)i}\right)}_{= \begin{cases} 1 & \text{si } j+k=t \\ o & \text{en otro caso} \end{cases}}$$

$$= \Sigma_{t-j+k} a_j b_k$$

$$= a * b \blacksquare$$

Teorema 4.2. Los coeficientes del producto de dos polinomios del grado $n-1$ se pueden computar con $\theta(n \log n)$ operaciones aritméticas de R.

Demostración. Sean

$$p(x) = a_o + a_1 x + \ldots + a_{n-1} x^{n-1}, \quad a = (a_o, a_1, \ldots, a_{n-1}, 0, 0, \ldots, 0),$$

$$q(x) = b_o + b_1 x + \ldots + b_{n-1} x^{n-1}, \quad b = (b_o, b_1, \ldots, b_{n-1}, 0, 0, \ldots, 0)$$

Dado que los coeficientes de $p(x).q(x)$ son los componentes de la convolución $a*b$, se les puede computar de la siguiente manera:

$$\begin{tikzcd}
a \arrow[d, "F", "\theta(n \log n)"'] & b \arrow[d, "F"] & a*b \\
F(b) & F(b) \arrow[r, "\theta(n)"'] & F(a)\ F(b) \arrow[u, "F^{-1}", "\theta(n \log n)"']
\end{tikzcd}$$

El tiempo total de la computación es $\theta(n \log n)$ operaciones aritméticas de R. $\blacksquare$

La misma técnica es aplicable a la multiplicación de los enteros, ya que la representación $d_{n-1}\ d_{n-2} \ldots d_o$ de un número en base k se puede interpretar como el polinomio.

$$d_{n-1}\ k^{n-1} + d_{n-2}\ k^{n-2} + \ldots + d_o$$

En el método de Karatsuba los dos enteros fueron divididos en dos bloques. Sin embargo, ni la división en dos bloques, ni la división en n bloques es optimal. Consideraremos el caso general en el que los enteros de longitud n son divididos en b bloques de longitud l, i.e. n=bl. En este caso los enteros binarios son plinomios del grado b-1 en 2^l.

Teorema 4.3. (Schönhage & Strassen 1971). Es posible multiplicar dos enteros de n dígitos usando no más de

$$\theta(n \log n(\log \log n)^2)$$

operaciones de dígitos.

Demostración. Sea $n=2^k$ para algún k y sean las representaciones de los enteros

$$a = a_{n-1}\ a_{n-2} \cdots a_1 a_o = \Sigma_{i=o}^{n-1}\ a_1 2^i \qquad (a_i \in \{o,1\}),$$

$$b = b_{n-1}\ b_{n-2} \cdots b_1 b_o = \Sigma_{i=o}^{n-1}\ b_i 2^i \qquad (b_i \in \{o,1\}).$$

Sea l la potencia de 2 que está más cerca de $k=\log_2 n$, y sea m=n/l. Representamos a y b como polinomios de los bloques de longitud l:

$$a = c_{m-1}\ c_{m-2} \cdots c_1\ c_o = \Sigma_{i=o}^{m-1}\ c_i 2^{li},$$

$$b = d_{m-1}\ d_{m-2} \cdots d_1\ d_o = \Sigma_{i=o}^{m-1}\ d_i 2^{li}$$

El algoritmo para computar a·b es lo siguiente:

(i) Computar las transformadas de Fourier de los vectores $(c_o, c_1, \ldots, c_{m-1}, 0, \ldots, 0)$ y $(d_o, d_1, \ldots, d_{m-1}, 0, \ldots, 0)$. Se necesitan $\theta(m \log n)$ operaciones aritmeticas aplicadas a enteros de longitud l. En la computación de Teorema 4.1 hay log m ≈ l niveles, y en cada nivel hay una multiplicación por ω^k ($|\omega|=1$) y una adición. Por eso al fin de la computación los números no pueden tener más que 2^l dígitos, ($1+\log m \approx 2^l$)

(ii) La multiplicación de los vectores transformados punto a punto no requiere más que m multiplicaciones de los números con 2^l dígitos.

(iii) La transformación inversa toma $\theta(m \log m)$ operaciones de números con no más que 3l dígitos.

(iv) De los componentes $h_o, h_i, \ldots, h_{2m-1}$ de la convolución se calcula el producto $ab=h_{2m-1}2^{l(2m-1)}+\ldots+h_1 2^l+h_o$.

El tiempo total del algoritmo es:

$$\begin{aligned} T9N) &= \theta(m \log m)\cdot T(2l)+2mT(2l) + (m \log m)T(3l) + 2m\cdot 3l \\ &= \theta(m \log m)\ T(3l) \\ &= \theta(\tfrac{n}{\log n} \log \tfrac{n}{\log n})\ T(3 \log n) \\ &= \theta(n\ T(3 \log n)) \\ &= \theta(n \log n\ T(\log \log n)) \\ &= \theta(n \log n\ (\log \log n)^2) \end{aligned}$$

[usando el algoritmo de la escuela] ■

Nota. Dividiendo los enteros en $\sqrt{n}$ bloques de longitud $\sqrt{n}$ y usando el anillo $\mathbb{Z}_m$ en vez de $\mathbb{C}$ es posible alcanzar la complejidad (n log n log log n). No se sabe si este resultado es optimal.

5. Problemas NP-completos

Todos los problemas anteriores tenían un algoritmo con complejidad polinomial. Está motivado decir, que si un problema tiene complejidad más grande que la polinomial, la solución no es viable, si el tamaño del problema crece. Hay problemas tales, como por ejemplo decidir si el lenguaje de unaexpresión regular con intersección y complementario es vacío, que tienen complejidad que no está limitada por ninguna función elemental.

Hay una gran clase de problemas para los que no se han podido descubrir soluciones polinomiales, pero tampoco se han podido demostrar que tengan complejidad peor que polinomial. Aun más curioso es, que si uno de estos problemas tiene solución polinomial, la tienen los demás también. Y si uno no tiene solución polinomial, los demás tampoco. Dichos problemas se llaman NP-completos.

Para precisar el concepto del algoritmo definimos la máquina de Turing.

La maquina de Turing no determinista con k cintas

$M = (Q,T,I,\delta,b,q_o,q_f)$ contiene los componentes siguientes:

(1) 1 es un conjunto finito de estados internos,

(2) T es un alfabeto finito de simbolos de cinta,

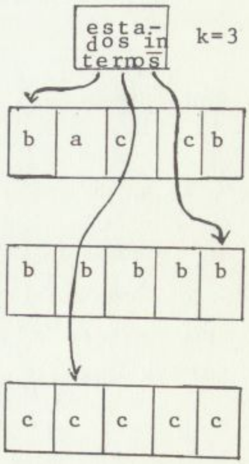
(3) $I \leq T$ es el conjunto de simbolos de entrada,

(4) δ es una relación $QxT^k \to Qx(Tx\{-1,0,1\})^k$, la relación de transición

(5) $b \in T$ I es el simbolo del cuadro blanco,

(6) $q_o \in Q$ es el estado inicial, y

(7) $q_f \in Q$ es el estado final.



La situación en cualquier momento de la computación viene expresada por la configuración

$(x_1 q y_1, \ldots, x_k q y_k)$,

que quiere decir que en este momento M está en el estado $q \in Q$, la cinta i contiene palabra $x_i y_i$, y la cabeza de leer/escribir está en el primer cuadro de y_i.

Una configuración $c=(x_1qy_1,\ldots,x_kqy_k)$ <u>deriva</u> a otra configuración $c'=(x_i'q'y_1',\ldots,x_k'q'y_n')$, y lo denotamos $c \vdash c'$, si hay $(q',(b_1,j_1),\ldots,(b_k,j_k)) \in \delta(q,a_1,\ldots,a_k)$ tal que $x_i'y_i'$ es obtenido de x_iy_i reemplazando el primer simbolo a_i de y_i por b_i, y moviendo la cabeza de tal manera que $|x_i'|=|x_i|+j_i$. Denotamos $c \overset{n}{\vdash} c'$ si hay $c_o,\ldots,c_n$ tales que $c=c_o\ c_1\ \ldots\ c_n=c'$, y $c \vdash^* c'$ si hay un n tal que $c \overset{n}{\vdash} c'$.

El <u>lenguaje aceptado</u> por M es

$$L(M) = \{\omega\in I^* \mid (q_o\omega,q_ob,\ldots,q_ob) \overset{*}{\vdash} (\omega_1q_f\omega_1',\ldots,\omega_kq_f\omega_k')\}$$

Sea f una función $N\to N$. El lenguaje L(M) es <u>aceptado en tiempo</u> f, si

$$L(M) = \{\omega \mid q_o\omega,q_ob,\ldots,) \overset{m}{\vdash} (\omega_1q_f\omega_k'),\ m\le f(|\omega|)\}$$

Ahora definimos dos clases de lenguajes importantes:

NP = {L|L es aceptado por una máquina de Turing no determinista con k cintas ($k\ge1$) en tiempo p, p polinomio},

P = {L|L es aceptado por una máquina de Turing determinista con k cintas ($k\ge1$) en tiempo p, p polinomio},

La máquina de Turing es determinista, por supuesto, si δ es función en vez de relación. Por eso

$$P\le NP$$

El gran problema de la complejidad es

$$¿P=NP?$$

Un lenguaje L es <u>reducible</u> a otro lenguaje L' en tiempo polinomial, y lo denotamos $L\le_p L'$, si hay una máquina de Turing determinista M y un polinomio p tales que

(i) $L'=\{x|\omega\in L,(q_o\omega,q_ob,\ldots,q_ob) \vdash^* (q_fx,\omega_2q_f\omega_2',\ldots,\omega_kq_f\omega_k')\}$

(ii) M funciona en tiempo p, i•e * puede reemplazarse por $m\leq f(|\omega|)$.

Un lenguaje L' es <u>NP-duro</u>, si para todo $L\in NP$, $L\leq_p L'$.
Un lenguaje L' es <u>NP-completo</u>, si L' es NP-duro y $L'\in NP$.

La importancia de la NP-completitud se manifiesta en

<u>Teorema 5.1</u>. Si L es NP-completo,

$L\in P$ si y solo si $P=NP$

<u>Demostración</u>. Supongamos que L es NP-completo.

(i) Si P=NP, por la definición de NP-completitud $L\in NP=P$
(ii) Supongamos que $L\in P$, i•e que hay una máquina de Turing determinista M que acepta L en tiempo polinomial p. Sea $L_1\in NP$. Por la NP-dureza de L, $L_1\in pL$, i.e. hay una máquina determinista M_1 que transforme L_1 en L en tiempo polinomial p_1. Uniendo el estado final de M_1 y el estado inicial de M obtenemos una máquina determinista que acepta L_1 en tiempo polinomial $p(p_1(n))$. ■

A continuación damos una lista de problemas, que interpretados como lenguajes, son NP-completos:

(1) <u>FNC-satisfacibilidad</u>
Dada una formula ω del cálculo de proposiciones en forma normal conjuntiva.] I.e. ω es de forma $\omega = (a_{11}\vee\ldots\vee a_{1k_1})$ $\wedge\ldots\wedge (a_{l_1}\vee\ldots\vee a_{lk_l})$, donde todo a_{ij} es o átomo o negación de un átomo] Hay que dicidir si ω es satisfacible o no.

(2) <u>Colorabilidad</u>
Dados un entero k>o, los vertices $v_1,v_2,\ldots,v_n$ y las aristas

$(v_{i_1},v_{j_1}),\ldots,(v_{i_m},v_{j_m})$ de un grafo G. Hay que dicidir si G es k-colorable. [Coloración: si (v_i,v_j) es una arista, v_i y v_j tienen color distinto]

(3) Recubrimiento exacto
Dado una colección $C_o,C_1,\ldots,C_n$ de conjuntos. ¿Hay una so-subcolección $C_{i_1},C_{i_2},\ldots,C_{i_m}$ de $C_1,\ldots,C_m$ tal que $C_o=C_{i_2}\cup C_{i_2}\cup\ldots\cup C_{i_m}$, y $C_{i_j}\cap C_{i_k}=\emptyset$ para todos $j\neq k$?.

(4) Mochila
Dados los enteros k, $i_1,i_2,\ldots,i_n$. ¿Hay un subconjunto $\{i_{j_1},\ldots,i_{j_1}\}$ de $\{i_1,\ldots,i_n\}$ tal que $k=i_{j_1}+\ldots+i_{j_1}$?.
[k es el volumen de la mochila. Hay que elegir los objetos que llenen la mochila].

(5) Viajante de comercio dirigido
Dado un entero k>o, los vertices $v_1,\ldots,v_n$ y las aristas $(v_{i_1},v_{j_1}),\ldots,(v_{i_m},v_{j_m})$ de un grafo dirigido G, y para toda arista (v_{i_1},v_{j_1}) un entero $d(v_{i_1},v_{j_1})\geq 0$ (la distancia).
¿Hay una permutación π de los vertices tal que

$\pi(v_1),\pi(v_2),\ldots,\pi(v_n),\pi(v_1)$ es un ciclo en G y
$d(\pi(v_1),\pi(v_2))+d(\pi(v_2),\pi(v_3))+\ldots+d(\pi(v_{n-1}),\pi(v_n))+d(\pi(v_n),\pi(v_1))=k$?

Los problemas se codifican en lenguajes reemplazando todos los elementos por números binarios distintos y separándolos con un simbolo especial.

De este modo, por ejemplo,

$(0 \vee 1 \vee \neg 10) \wedge (\neg 1 \vee \neg 0 \vee \neg 1) \wedge (0 \vee \neg 1 \vee 10)$

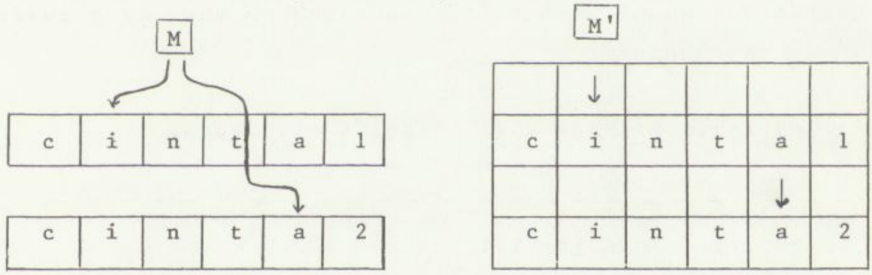
está en el lenguaje de las formas normales conjuntivas satisfacibles, 0=F, 1=10=T satisface la formula.

Ahora vamos a demostrar que el lenguaje FNC-sat es NP-duro. Lo haremos describiendo cómo se construye para cada palabra ω en tiempo polinomial una formula $F(\omega)$ tal que $f(\omega)$ es satisfacible si y solo si $\omega \in L$.

Para simplificar la demostración necesitamos el lema siguiente:

<u>Lema 5.1</u>. Cada lenguaje de NP se puede aceptar con una máquina de Turing no determinista con una cinta en tiempo polinomial.

"<u>Demostración</u>." Las k cintas de una máquina M se pueden unir en una cinta con 2k rayas de máquina M':



M' simula una transición de M recorriendo todos los cuadros que contienen la flecha $\downarrow$. Si M acepta ω en tiempo $m=p(|\omega|)$, M' acepta ω en tiempo $2+4+6+\ldots+2m \leq 2m^2$, i.e. en tiempo $2\,p^2(|\omega|)$.

<u>Teorema 5.1. (Cook 1971)</u>. FNC sat es NP-completo.

<u>Demostración</u>. I FNC sat $\in$ NP se verifica construyendo una máquina que primero adivina los valores que satisfagan la formula (si las hay), y después calculo el valor de la formula, y por fin acepta si la formula tiene el valor T. Evidentemente dicha máquina funciona en tiempo polinomial.

II FNC sat es NP-duro. Sea $L \in NP$, $L=L(M)$. Vamos a definir una transformación Comp (sin dar explicitamente la máquina para la transformación) tal que $\omega \in L$ si y solo si Comp $(\omega) \in$ FNC sat. Comp será computable en tiempo polinomial Intuitivamente, Comp (ω) es satisfacible si ω tiene una computación aceptable.

Sea $M=(Q,T,I,\delta,b,q_o,q_k)$ la máquina no determinista con una cinta, que acepta L en tiempo polinomial. Sea $Q=\{q_o,q_1,\ldots,q_k\}$, y $T=\{a_o,a_1,\ldots,a_m\}$ $(a_o=b)$. Sea $\omega\in I^*$. La formula Comp (ω) contiene variables $a_i^{s,t}$ $(i=0,\ldots,m; s_1 t=1,\ldots,P)$ y $q_i^{s,t}$ $(i=o,\ldots,k;\ s,t=1,\ldots p)$. La interpretación de estas variables es: $a_i^{s,t}$ es satisfacible ssi en el momento t M contiene la letra a_i en el cuadro s,

$q_i^{s,t}$ es satisfacible ssi en el momento t,M esta en el cuadro s en estado q_1.

Notese que en el momento t,M contiene no más de t cuadros no blancos en su cinta.

Por nitidez utilizamos la notación siguiente:

$$x\to y := \quad x\vee y,$$

$$x\leftrightarrow y := (\ x\vee y)\wedge(x\vee\ y),$$

$$\bigwedge_{i=1}^{n} x_i := x_1\wedge x_2\wedge\ldots\wedge xn,$$

$$\bigvee_{i=1}^{n} x_i := x_1\vee x_2\vee\ldots\vee xn,$$

$$\dot{\bigvee_{i=1}^{n}} x_i := (\bigvee_{i=1}^{n} x_1)\wedge(\bigwedge_{i\neq f}(\ x_i\wedge\ x_j))$$

Todas estas formulas están en forma normal conjuntiva:

La formula Comp (ω) tiene la forma

Comp (ω) = Comp$(\omega)\wedge$Inic $(\omega)\wedge$Transición $(\omega)\wedge$Fin (ω)
cuyos factores estan definidos como:

$$\text{Config}(\omega) = \bigwedge_{t=1}^{p}\{|\bigwedge_{s=1}^{p}(\ \dot{\bigvee_{i=o}}\ a_i^{s,t})|\wedge|\dot{\bigvee_{s=1}^{p}}\ \dot{\bigvee_{i=o}^{k}}\ q_i^{s,t}|\}$$

Esta formula es satisfacible si y solo si en cada momento la máquina tiene en cada cuadro una letra única y está en un estado único en un cuadro único de su cinta. La longitud de esta palabra es $\theta(p^3 \log p)$.

$$\text{Inic }(\omega)=q_o^{1,1}\wedge a_{i_1}^{1,1}\wedge a_{i_2}^{2,1}\wedge\ldots\wedge a_{i_n}^{n,1}\wedge a_o^{n+1,1}\wedge\ldots\wedge a_o^{p,1},$$

donde q_o es el estado inicial y $\omega=a_{i_1}a_{i_2}\ldots a_{i_n}$ y a_o es el simbolo del cuadro blanco. La longitud de esta palabra es $\theta(p \log p)$.

$$\text{Transición}(\omega)=\{\bigwedge_{t=1}^{p}\bigwedge_{s=1}^{p}\bigwedge_{i=o}^{m}[(a_i^{s,t}\leftrightarrow a_i^{s,t+1})\ (\bigvee_{j=o}^{k} q_j^{s,t})\}\wedge$$

$$\{\bigwedge_{t=1}^{p}\bigwedge_{s=1}^{p}\bigwedge_{i=o}^{m}\bigwedge_{j=o}^{k}|a_i^{s,t}\ q_j^{s,t}\rightarrow\bigvee_{r=1}^{l}(a_{i_r}^{s,t+1}\ q_{j_r}^{s+d_r,t+1})|\},$$

donde $\delta(q_j,a_i)=\{(q_{j_r},a_{i_r},dr)|r=1,\ldots,l\}$. Esta formula es satisfacible si y sólo si la máquina puede hacer la transición de una configuración del momento t a otra configuración del momento t+1. La longitud de la formula es $\theta(p^2\log p)$.

$$\text{Fin}(\omega) = \bigvee_{s=1}^{p} q_k^{s,p}$$

Aqúi suponemos que q_k es el estado final, y tan pronto como M llegue al estado final, se quedará en el y no hará cambios en la cinta. La formula es satisfacible si y sólo si en el momento p, M está en el estado final. La longitud de la formula es $\theta(p \log p)$.

La formula Comp (ω) está en forma conjuntiva y $\omega\in L(M)$ si y sólo si Comp(ω) es satisfacible. La longitud de Comp(ω) es $\theta(p(|\omega|)^3 \log p(|\omega|)) \leq \theta(p(|\omega|)^4)$, y por eso limitada por un polinomio de $|\omega|$. Esta transformación puede ser realizada por una máquina de Turing determinista ■

Demostramos con FNC 3 sat el subconjunto de FNC sat, que contiene solamente formulas con no más de 3 literales en sus factores.

Teorema 5.2. FNC 3 sat es NP-completo.

Demostración. FNC 3 sat $\in$ NP se ve como en el teorema anterior.

La reducción FNC sat $\leq$ FNC 3 sat es una consecuencia inmediata de la equivalencia

$$a_1 \vee a_2 \vee \ldots \vee a_n = (a_1 \vee a_2 \vee b_2) \wedge (b_2 \vee a_3 \vee b_3) \wedge \ldots$$
$$\wedge (b_{n-2} \vee a_{n-1} \vee a_n),$$

donde $b_2, b_3, \ldots, b_{n-2}$ son variables nuevas ∎

Teorema 5.3. "Colorabilidad" es NP-completo

Demostración. Es fácil ver que este problema está en NP. Para NP-dureza basta verificar FNC 3 sat $\leq$ p Colorabilidad. Sean $a_1, \ldots, a_n$ los átomos de formula F de FNC 3 sat, y $f_i, \ldots f_m$ los factores de esta formula. Construimos un grafo G_F, que tiene vertices $f_1, \ldots, f_m$, $a_1, \ldots, a_n$, $a_1, \ldots,$ a_n y además vertices $v_1, \ldots, v_n$. Las aristas de G_F son:

$$\{(v_i, v_j) \mid i \neq j\} \cup \quad [\text{subgrafo completo}]$$
$$\{(v_i, a_j), (v_1, a_j) \mid i \neq j\} \cup$$
$$\{(a_1, \neg a_i) \mid \ i = 1, \ldots, n\} \cup$$
$$\{(f_i, a_j) \mid a_j \text{ no está en } f_i\} \cup \{(f_i, \neg a_j) \mid \neg a_j \text{ no está en } f_i\}.$$

Vamos a demostrar que $F \in$ FNC3 sat sii G_F es $(n+1)$-colorable.

Supongamos que $n \geq 4$ porque FNC3 sat con 3 variables está en P.

Colorabilidad $\rightarrow$ satisfacibilidad. Supongamos que G_F es $(n+1)$-colorable. Ya que el subgrafo de los v_i es completo, este subgrafo está colorado con n colores. Puesto que desde a_j salen aristas a cada $v_i (i \neq j)$, o $col(a_j)=col(v_j)$ o $col(a_j)$ es un nuevo color c_{n+1}. Lo mismo se puede decir de $col(\neg a_j)$. Por otro lado, a_i y $\neg a_i$ tienen colores distintos, porque $(a_i, \neg a_i)$ está en el grafo. Así uno de ellos tiene el color $col(v_i)$, y otro eo color c_{n_i}. Digamos $col(y_1)=col(y_2)= \ldots = col(y_n)= c_{n+1}$, donde y_i es a_i ó $\neg a_i$. Entonces $col(\neg y_1)= col(v_1), \ldots, col(\neg y_n)=col(v_n)$.

Vamos a demostrar que todos los vertices f_i tienen color distinto de c_{n+1}. Salen $2n-3$ aristas de f_i. Porque $2n-3 \geq n+1$ para $n \geq 4$, hay un j tal que (f_i, a_j) y $(f_i, \neg a_j)$ son aristas. Pero ó a_j ó $\neg a_j$ está colorado con c_{n+1}. Por eso $col(f_i) \neq c_{n+1}$ par todo i.

Afirmamos que cada factor f_i contiene un literal y_j tal que $col(y_j) \neq c_{n+1}$. En otro caso, ya que salen aristas de f_i a al menos n+1 vertices, de f_i salen aristas a todos los vertices $\neg y_1, \neg y_2, \ldots, \neg y_n$ colorados con n colores distintos. Por esto, y ya que f_i no está colorado con c_{n+1}, n+1 colores no bastarían para la coloración.

Según esta aserción los valores $y_1=T$, $y_2=T, \ldots, y_n=T$ satisfacen la formula.

Satisfacibilidad $\rightarrow$ Colorabilidad. Supongamos que $y_1=T, \ldots, y_n=T$ $(y_1=a_i$ o $y_i=\neg a_i)$ satisface $f_i \wedge \ldots \wedge f_m$. Entonces cada f_i contiene un y_j y se puede colorar f_i con $col(y_j)$. Por eso n+1 colores bastan ∎

Teorema 5.4. Recubrimiento exacto es NP-completo.

Demostración. El algoritmo no determinista polinomial para el problema es obvio. Vamos a demostrar que colorabilidad $\leq$recubrimiento exacto.

Sean $v_1,\ldots,v_n$ los vertices de un grafo G, $(v_{i_1},v_{j_1}),\ldots,(v_{i_m},v_{j_m})$ las aristas, y k el número de los colores. Definimos los conjuntos

$C_{il} = \{v_i\}\cup\{[(v_i,v_j),l] \mid (v_i,v_j) \text{ está en } G,\ 1\leq l\leq k\}$,

$D_{ijl} = \{[(v_i,v_j),l]\}$,   [Nota: $(v_i,v_j)=(v_j,v_l)$ en grafos]

$S = \{v_i,\ldots,v_n\}\ \{[(v_i,v_j),l] \mid (v_i,v_j) \text{ en grafo},\ 1\leq l\leq k\}$

Afirmamos que S tiene un recubrimiento exacto sii G es k-colorable.

Colorabilidad $\rightarrow$Cubrimiento exacto. Sea c(i) el color de v_i en la k-coloración. Los conjuntos $C_{ic(i)}$ (i=1,...,n) con los conjuntos D_{ijl} tales que $[(v_i,v_j),l] \notin \cup_{i,j} C_{il}$ forman un recubrimiento exacto. Porque los conjuntos D_{ijl} contienen un solo elemento que no esta en ningun C_{il}, el recubrimiento es exacto por su parte. Por otro lado, si $C_{ic(i)}$ y $C_{ic(j)}$ tuvieran un elemento $[(v_i,v_j),l]$ comun, los vertices v_i y v_j tendrían el mismo color en contra de la suposición.

Recubrimiento exacto $\rightarrow$ Colorabilidad. Si S tiene un recubrimiento exacto, para todo $i\in\{1,\ldots,n\}$ el recubrimiento contiene un conjunto C_{il_i} . Podemos colorear el vertice c_i con l_i si fuera una arista (v_i,v_j) tal que $l_i=l_j$, entonces $[(v_i,v_j),l_i]$ en los dos conjuntos C_{il_i} y C_{jl_j} ∎

<u>Teorema 5.5</u>. La mochila es NP-completo.

<u>Demostración</u>. Demostramos que recubrimiento exacto $\leq$Mochila.

Sean los conjuntos del recubrimiento exacto $C_o, C_1, \ldots, C_n$ y todos elementos sean $C_1, \ldots, C_m$. Para cada conjunto C_I definimos números $(n+2)$.ario $C_1 = 1_m \, 1_{m-1} \ldots 1_1$ en donde

$$1_j = \begin{cases} 1 & \text{si } e_j \in C_i \\ 0 & \text{si } e_j \notin C_i \end{cases}$$

Sean k el número cuyo representación es 11 ... 1.

C_o tiene un recubrimiento exacto sii la Mochila definido por enteros k y $c_o, c_1, \ldots, c_n$ tiene una solución. Si C_o tiene un recubrimiento exacto $C_{i_1}, \ldots, C_{l_r}$, entonces obviamente $K = c_{i_t} + \ldots + c_{i_r}$. Por otro lado, si $k = c_{i_1} + \ldots + c_{i_r}$, cada dígito 1 de k proviene de un solo c_i porque ni siquiera n + 1 dígitos pueden llenar la base n+2 de k. ∎

<u>Teorema 5.6</u>. Viajante de comercio dirigido es NP-completo.

<u>Demostración</u>. Sean $k, i_1, \ldots, i_n$ los números de la Mochila.

Construimos un grafo G con vertices $v_o, v_1, \ldots, v_n, v_{n+1}$, y todas las aristas (v_i, v_j) $(i \neq j)$. Las distancias serán

$$d(v_r, v_s) = \begin{cases} i_s & \text{si } r<s \\ 0 & \text{si } r<s \text{ o } s=n+1 \end{cases}$$

y la distancia total será k.

Demostramos que la Mochila tiene una solución sii el Viajante de comercio dirigido construido anteriormente tiene una solución.

Si el Viajante de comercio tiene una solución, hay un ciclo en el grafo tal que la suma de las distancias es k. Pero entonces la Mochila también tiene solución, porque las distancias son números de la Mochila.

Por otro lado, supongamos que la Mochila tiene la solución $k=i_{j_1}+\ldots+i_{j_t}$. Entonces el Viajante de comercio tiene el ciclo $v_0, v_{j_1}, \ldots, v_{j_t}, v_{n+1}, \ldots, v_{l_s}, v_0$ de longitud k, en donde $l_1, \ldots, l_s$ son los elementos de $\{1,\ldots,n\} \setminus \{j_1,\ldots j_t\}$ en el orden decreciente. ∎

Referencias.

Libros:

A.Aho, y Hopcroft, y ULLman: *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

E.Horowitz, S.Sahni: *Fundamentals of Computer Algorithms*, Pitman, 1978.

J.Sawage: *The Complexity of Algorithms*, Academic Press

Artículos:

S.A.Cook (1971): The complexity of theorem proving procedures. Proceedings of the 3 rd Annual ACM Symposium on Theory of Computing, 151-158.

A.Karatsaba, Jn.Ofman (1962): Umnozenie mnogozmacnich cisel na avtomatah. Doklady Akademu Nauk SSSR 145 (1962), No.2, 293-294.

J.von Neumann (1945): Collected Works V,196-214. Pergamon Press, 1961

A.Schonhage, V.Strassen (1971): Schnelle multiplikation grosser Zahlen. Computing 7, 281-252.

V.Strassen (1965): Gaussian elimination is not optimal. Numerische Mathematik, 13; 354-356.

G.Yuval (1978): A simple proof of Strasseris result. Information Processing Letters 7, 285-286.