$ 15000=

UNIVERSITAS SCIENTIARUM
REVISTA DE LA FACULTAD DE CIENCIAS

Volumen 8, N° 1: Enero-Junio de 2003

Esta Revista está indexada y referenciada
en Chemical Abstracts (CA)

PONTIFICIA UNIVERSIDAD JAVERIANA

ICFES

UNIVERSITAS SCIENTIARUM
Revista de la Facultad de Ciencias
PONTIFICIA UNIVERSIDAD JAVERIANA
Vol. 8, N° 1: 25-29

FACTORIZACIÓN DE PERMUTACIONES

Fernando Novoa

Facultad de Ciencias, Pontificia Universidad Javeriana, Carrera 7 N° 43-88, Bogotá

E-mail: fernando.novoa@javeriana.edu.co

RESUMEN

Presentamos la implementación de un programa que permite hallar una factorización de los elementos del grupo simétrico S_n, usando el número mínimo de transposiciones simples. Dicha implementación se realizó en el sistema computacional CoCoA.

Palabras clave: Permutación, transposición simple, código de Lehmer, polinomio de Schubert.

ABSTRACT

We present a program for computing a factorization of the elements of S_n, using the minimal number of simple transpositions. The program is implemented for the computational algebraic system CoCoA.

Key words: Permutation, simple transposition, Lehmer code, Schubert polynomial.

INTRODUCCIÓN

El grupo simétrico S_n puede definirse como el conjunto de todas las biyecciones sobre el conjunto $\{1,2,\ldots,n\}$ con la operación usual de composición de funciones. Los elementos de S_n pueden representarse por medio de ciclos. Por ejemplo, sobre el conjunto $\{1,2,...,5\}$ la función que envía

$$1 \to 5$$
$$2 \to 1$$
$$3 \to 2$$
$$4 \to 3$$
$$5 \to 4$$

se puede representar en un ciclo como (1,5,4,3,2). En general un k-ciclo $(c_1,\ldots,c_k)$ representa la función

$$(c_1 \to c_2)$$
$$(c_2 \to c_3)$$
$$\vdots$$
$$(c_k \to c_1)$$

Los elementos de S_n se llaman permutaciones y también se pueden escribir en notación de una línea. De esta forma, si $w \in S_n$ escribimos $w = [w_1,w_2,...,w_n]$ para indicar que $w(i) = w_i$ para todo $i = 1,...,n$. En el ejemplo anterior, la notación en una línea de la permutación (1,5,4,3,2) es [5,1,2,3,4]. Una transposición es un 2-ciclo y una transposición simple es una transposición que intercambia dos elementos consecutivos. Las transposiciones simples se escriben como $s_k = (k,k+1)$ para $k = 1,...,n-1$. Es bien conocido que el núme-

ro de elementos de S_n es $n!$ y que toda permutación se puede factorizar como un producto de ciclos disyuntos. Además, todo ciclo se puede factorizar como producto de transposiciones simples y por lo tanto, las transposiciones simples generan a S_n. La descomposición anterior en ciclos y transposiciones simples no es única. Sin embargo, el número mínimo de transposiciones simples necesarias para factorizar una permutación $w \in S_n$ se llama la longitud de w y se denota por $l(w)$.

Ejemplo 1. Para la permutación (1,5,4,3,2) tenemos que

$$(1,5,4,3,2) = (4,5)(3,4)(2,3)(1,2)$$

$$= s_4 s_3 s_2 s_1$$

Note que el producto se hace de derecha a izquierda, en la forma usual como se componen funciones. Como se dijo antes, la descomposición en transposiciones no es única y se tiene que

$$s_i s_{i+1} s_i = s_{i+1} s_i s_{i+1}$$
$$s_i s_j = s_j s_i \quad si \mid i-j \mid > 1.$$

DEFINICIONES

En esta sección recordamos las definiciones que necesitamos para desarrollar la implementación del programa. Dicho programa sigue las ideas de Winkel.

La *longitud* de una permutación $w \in S_n$ es

$$l(w) = \#\{(i,j) : i < j \; con \; w_i > w_j\}$$

Decimos que la permutación w tiene un *descendente* en la posición p, si $w_p > w_{p+1}$. Para un n fijo denotamos por w_0 la permutación $[n(n-1)...21]$ la cual tiene una longitud de

$$n(n-1)/2$$

y es la permutación de longitud máxima en S_n.

Para una permutación $w \in S_n$ definimos su código de Lehmer como la secuencia

$$L(w) = \overline{l_{n-1}, l_{n-2}, ..., l_0}$$

donde $l_{n-j} = \#\{i : i > j \; con \; w_i < w_j\}$. Por lo tanto se satisface que si $w \in S_n$ y $L(w) = \overline{l_{n-1}, l_{n-2}, ..., l_0}$ entonces,

$$l(w) = \sum_{j-1}^{n} l_{n-j}$$

Ejemplo 2. Con las rutinas desarrolladas en el programa **reducida.pkg** podemos verificar y dar algunos ejemplos de las anteriores definiciones. Inicialmente calculamos la longitud de una permutación, su código de Lehmer y finalmente verificamos que la suma de las componentes del código de Lehmer de la permutación dada es exactamente su longitud.

```
Red.LondePermutacion([5,1,2,3,4]);
4
```

```
Red.Lehmer([5,1,2,3,4]);
[4, 0, 0, 0, 0]
```

```
Sum(Red.Lehmer([5,1,2,3,4]));
4
```

A una secuencia $a = (a_1,...,a_p)$ se llama una *palabra reducida* de w si $p = l(w)$ y $w = s_{a_1} s_{a_2} ... s_{a_p}$.

Ejemplo 3. Siguiendo el ejemplo 1, (4,3,2,1) es una palabra reducida para (1,5,4,3,2) = [5,1,2,3,5].

El conjunto de palabras reducidas para una permutación $w \in S_n$ se denota por $R(w)$. La cardinalidad de dicho conjunto se puede establecer por medio de funciones generatrices, las cuales fueron introducidas por Stanley.

Claramente podemos recuperar la permutación a partir de su código de Lehmer. Para esto, podemos usar el siguiente algoritmo:

Algoritmo 1. Sea $\overline{l_{n-1},l_{n-2},...,l_0}$ el código de Lehmer de una permutación $w \in S_n$. Para hallar w procedemos así:

w_1 es el $l_{n-1}+1$ elemento del conjunto $\{1,2,...,n\}$

w_2 es el $l_{n-2}+1$ elemento del conjunto $\{1,2,...,n\}\backslash\{w_1\}$

y en general

w_j es el $l_{n-j}+1$ elemento del conjunto $\{1,2,...,n\}\backslash\{w_1, w_2,...,w_{j-1}\}$

Ejemplo 4. En la notación de una fila la permutación $w = (1,5,4,3,2)$ se representa como $w = [5,1,2,3,4]$.Su código de Lehmer es $L(w) = \overline{4,0,0,0,0} = \overline{l_4,l_3,l_2,l_1,l_0}$. Por lo tanto, w_1 es el $l_{5-1} + 1 = l_4 + 1 = 4 + 1 = 5$ elemento del conjunto $\{1,2,3,4,5\}$, es decir el quinto elemento de dicho conjunto, en este caso, $w_1 = 5$. Así, w_2 es el $l_{5-2} + 1 = l_3 + 1 = 0 + 1 = 1$ elemento del conjunto $\{1,2,3,4\}$ por lo tanto, $w_2 = 1$. De esta forma recobramos w.

Ejemplo 5. De nuevo, haciendo uso del programa desarrollado podemos hacer ese cálculo rápidamente mediante las siguientes instrucciones:

```
Time Red.InverseLehmer([4,0,0,0,0]);
[5, 1, 2, 3, 4]
Cpu time = 0.00, User time = 0
```

La última línea le da una idea del tiempo que tardo la CPU en este cómputo.

A partir del código de Lehmer podemos encontrar una palabra reducida para las permutaciones de S_n.Este algoritmo ya se ha generalizado por Winkel [W] para otros tipos de grupos de reflexiones distintos de los grupos de simetría.

Algoritmo 2. Sea $n \in N$ $w \in S_{n+1}$ y $L(w) = \overline{l_n,...,l_0}$ el código de Lehmer de w. Entonces la secuencia

$$a(w) = (\Theta(l_n) | ... | \Theta(l_1))$$

es una palabra reducida para w, en donde la secuencia $\Theta(l_j)$ se define por

$$\Theta(l_j) = n - j + l_j ... n - j + 1$$

si $l_j \neq 0$ En caso contrario, se define como la secuencia vacía.

Ejemplo 6. Sea $w = [5,1,2,3,4]$. Entonces su código de Lehmer es $L(w) = \overline{4,0,0,0,0}$. Por lo tanto la palabra reducida que proporciona el algoritmo 2 depende únicamente de $l_4 = 4$. En este caso observe que $n + 1 = 5$ por lo tanto

$$\Theta(l_4) = 4 - 4 + 4 ... 4 - 4 + 1 = 4 ... 1 = 4321$$

como se tenía desde el ejemplo 1.

Ejemplo 7. Con nuestro programa podemos calcular una palabra reducida para las permutaciones de S_n por medio de la función Word1(w)

```
Red.Word1([5,1,2,3,4]);
[4, 3, 2, 1]
```

```
Time Red.Word1([5,1,2,3,4]);
[4, 3, 2, 1]
Cpu time = 0.00, User time = 0
```

De nuevo observe el tiempo de ejecución. Además a partir de la palabra reducida podemos también encontrar la permutación a la que corresponde.

```
Red.Word([4,3,2,1],5);
[5, 1, 2, 3, 4]
```

El segundo parámetro n de esta función Word (a, n) es necesario, por cuanto $S_n \subset S_{n+1} \subset \cdots$.

UNA APLICACIÓN

Una de las primeras aplicaciones de nuestro programa es el cálculo de los polinomios de Schubert. Estos polinomios fueron definidos por Lascoux y Schutzenberger y son aún muchas las preguntas que están sin resolver acerca de ellos y sus relaciones en diversas ramas de las matemáticas y en especial en combinatoria.

Los polinomios de Schubert G_w, están indexados por los elementos $w \in S_n$ y se pueden definir recursivamente por medio de las siguientes ecuaciones:

$$G_w = \delta_{(a_1 \ldots a_p)}(x_1^{n-1} x_2^{n-2} \cdots x_{n-1})$$

$a = (a_1, \ldots a_p)$ es una palabra reducida de $w^{-1}w_0$ y el operador $\delta_{(a_1, \ldots, a_p)}$ se define

$$\delta_{(a_1, \ldots, a_p)} = \delta_{a_1} \delta_{a_2} \cdots \delta_{a_p}$$

y los *operadores de diferencias dividida* δ_k están definidos por

$$\delta_k(f) = \frac{f - s_k f}{x_k - x_{k+1}}$$

donde $s_k f$ significa intercambiar x_k y x_{k+1} en el polinomio $f \in Z[x_1, \cdots, x_n]$

Por lo tanto conociendo una palabra reducida de la permutación $w^{-1}w_0 \in S_n$ aplicamos el operador de diferencias dividido según lo indique dicha palabra reducida al polinomio $x_1^{n-1} x_2^{n-2} \cdots x_{n-1}$

y obtenemos el polinomio de Schubert.

Ejemplo 8. Calculemos el polinomio de Schubert asociado a la permutación $w = [3,5,6,1,4,2]$. Primero calculamos w^{-1}, lo componemos con w_o y calculamos una palabra reducida para esta permutación.

```
W;
[3, 5, 6, 1, 4, 2]
```

```
IW:=Inverso(W);
IW;
[4, 6, 1, 5, 2, 3]
```

```
W0:=PerLarga(6);
W0;
[6, 5, 4, 3, 2, 1]
```

```
A:=Componer(IW,W0);
A;
[3, 2, 5, 1, 6, 4]
```

```
L:=Red.Word1(A);
L;
[2, 1, 2, 4, 3, 5]
```

Luego aplicamos el operador de diferencias dividida según el orden dado por la palabra reducida L que hemos calculado, al polinomio $x_1^5 x_2^4 \cdots x_5$

```
F;
x[1]^5x[2]^4x[3]^3x[4]^2x[5]
```

```
DiDiOperator1(L,6,F);
x[1]^3x[2]^3x[3]^2x[4] +x[1]^3x[2] ^2x[3]^
3x[4] +x[1]^2x[2]^3x[3]^3x[4] +x[1]^3x
[2]^3x[3] ^2x[5] +x[1]^3x[2]^2x[3] ^3x[5]
+x[1]^2x[2]^3x[3]^3x[5]
```

Todas estas operaciones las simplificamos con una sola rutina

```
Schubert1(W);

x[1]^3x[2]^3x[3]^2x[4] + x[1]^3x[2]^2x[3]
^3x[4] + x[1]^2x[2]^3x[3]^3x[4] +
x[1]^3x[2]^3x[3]^2x[5] + x[1]^3x[2]^
2x[3]^3x[5] + x[1]^2x[2]^3x[3]^3x[5]
```

CONCLUSIONES

Por el teorema de Cayley sabemos que todo grupo finito es isomorfo a un subgrupo de algún S_n. Esto le da gran importancia a las permutaciones y en particular a las transposiciones simples, por cuanto ellas generan los grupos de simetría. Este programa permite calcular la descomposición de las permutaciones en transposiciones simples, con lo cual podemos saber la paridad de cada permutación.

Por otra parte, la descomposición en transposiciones simples está relacionada con el orden de Bruhat en S_n, el cual es una de las principales herramientas usadas para la descripción de las fórmulas para la generación y multiplicación de polinomios de Schubert.

El programa también puede utilizarse para determinar cuales permutaciones son Grassmannianas. Si w es Grassmanniana, entonces existen una partición λ y un entero m asociados a w tales que el polinomio de Schubert G_w coincide con el polinomio de Schur s_λ^m asociado a la partición λ en m variables.

Son muchas las aplicaciones de los polinomios de Schubert en geometría, álgebra, topología y combinatoria y esta herramienta computacional puede servir de ayuda pedagógica al permitirnos plantear y también verificar conjeturas sobre éstos polinomios y principalmente sobre las características de sus coeficientes. Adicionalmente contamos con otro programa **Schubert.pkg** para el cálculo de polinomios de Schubert haciendo uso diagramas de Rothe.

LITERATURA CITADA

NOVOA F. Cálculo de polinomios de Schubert. Reporte técnico 01/2002. Pontificia Universidad Javeriana, Departamento de Matemáticas, Bogotá, Colombia, 2002.

PRAGACZ P. Symmetric Polynomials and divided differences in formulas of intersection theory. Preprint, 1991.

STANLEY R. *Enumerative Combinatorics,* vol. 2. Cambridge University Press, 1999.

WINKEL R. A combinatorial derivation of the Poincaré polynomials of the finite irreducible Coxeter groups. Preprint. 1998.