



ugr

Universidad
de Granada

TRABAJO FIN DE GRADO

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Entendiendo la teoría de nudos mediante la simulación y la informática gráfica.

Autora

Cristina Zuheros Montes.

Tutores

Alejandro J. León Salas

Antonio Martínez López



Facultad de
Ciencias



Granada, Diciembre de 2016

Entendiendo la teoría de nudos mediante la simulación y la informática gráfica.

Cristina Zuheros Montes.

Palabras clave: nudo, trenza, handle...PONER LAS PALABRAS CLAVE.

Resumen

//HACER RESUMEN AQUI!

PONER TITULO EN INGLES!!!

Cristina Zuheros Montes.

Keywords: knot, braid, handle....PONER AQUI KEYWORDS!!!!

Abstract

//PONER RESUMEN DE MILLON DE PALABRAS INGLES!!!!!!

Índice general

1	Introducción	1
1.1	Objetivos	1
2	Teoría de nudos.	3
3	Teoría de trenzas.	5
4	Desarrollo informático.	7
4.1	Pseudoalgoritmos.	7

Índice de figuras

ÍNDICE DE FIGURAS

Capítulo 1

Introducción

HACER INTRO AQUIII!!!

1.1 Objetivos

En la propuesta inicial se propusieron los siguientes objetivos:

1. Estudiar...//PONER LOS OBJETIVOS INICIALES
//PONER DONDE SE CUBREN LOS OBJETIVOS

Capítulo 2

Teoría de nudos.

Capítulo 3

Teoría de trenzas.

Capítulo 4

Desarrollo informático.

4.1 Pseudoalgoritmos.

En esta sección vamos a ver algunos pseudoalgoritmos referentes a los algoritmos que hemos implementado. Nos vamos a centrar en aquellos que tienen mayor complejidad e interés, sin entrar en gran detalle.

En concreto, vamos a ver los pseudoalgoritmos para el algoritmo de Dehornoy (que se descompone en varios algoritmos auxiliares), el algoritmo que nos permite comprobar si dos trenzas dadas (o sus cierres) son o no equivalentes entre sí y el algoritmo que nos permite comprobar si una trenza dada (o su cierre) es equivalente a la trenza trivial.

Algoritmo de Dehornoy para trenzas.

Vamos a ir viendo los algoritmos auxiliares y concluiremos con el propio algoritmo de Dehornoy. Recordemos que este algoritmo se explica con detalle en la sección ??.

Mediante el algoritmo *Simplifica* eliminamos ocurrencias de tipo $\sigma_i^e \sigma_i^{-e}$, $e \in \{-1, 1\}$ de una palabra que representa a cierta trenza.

Algoritmo 4.1.1. *Algoritmo Simplifica*(*indices_braid*)

ENTRADA: *indices_braid* (cadena de enteros que representa los cruces de una trenza)

SALIDA: *braid_aux* (cadena auxiliar para mejor representacion visual)

nueva_braid (cadena de enteros que representa los cruces de la trenza tras eliminar dos cruces opuestos)

encontrado (bool para indicar si se produce simplificación)

1 *Recorro los cruces de indices_braid*

2 *Si hay dos cruces seguidos con signos opuestos, creo una copia de indices_braid y elimino dichos cruces.*

Haciendo uso del siguiente algoritmo podremos encontrar las posiciones que delimitan un σ_{minimo} -handle.

Algoritmo 4.1.2. Algoritmo encuentra_handle(indices_braid, minimo)

ENTRADA: indices_braid (cadena de enteros que representa los cruces de una trenza)

minimo (generador de handle a encontrar)

SALIDA: pos1 (posición inicial del handle encontrado)

pos2 (posición final del handle encontrado)

```

1   Recorro los cruces de indices_braid
2   Si hay un cruce generado por el elemento minimo, me quedo con esa
    posicion como pos1 y su signo. Sino fin.
3   Si encuentro un cruce generado por el elemento minimo con signo
    opuesto, me quedo con esa posicion como pos2.
```

Haciendo uso del algoritmo reduccion_base aplicamos una reducción local a la trenza representada por indices_braid sobre el σ_{minimo} -handle que está situado entre las posiciones pos1 y pos2. Es importante destacar el hecho de que este σ_{minimo} -handle no contiene $\sigma_{\text{minimo}+1}$ -handle.

Algoritmo 4.1.3. Algoritmo reduccion_base(indices_braid, minimo, pos1, pos2)

ENTRADA: indices_braid (cadena de enteros que representa los cruces de una trenza)

minimo (generador de handle)

pos1 (posición inicial del handle)

pos2 (posición final del handle)

SALIDA: braid_aux2 (cadena auxiliar para mejor representación visual)

nuevo (cadena de enteros que representa los cruces de la trenza tras aplicar la reducción local al σ_{minimo} -handle entre pos1 y pos2)

simplificado2 (bool auxiliar para mejor representación visual)

```

1   Creo vector_auxiliar
2   Recorro los cruces de indices_braid desde pos1 hasta pos2
3   Si hay un cruce generado por el elemento (minimo+1) aniado al
    vector_auxiliar los 3 cruces correspondientes. Sino, aniado el
    mismo cruce.
4   Creo vector_nuevo con los elementos desde inicio de indices_braid
    hasta pos1, vector_auxiliar, y los elementos desde pos2 hasta
    final de indices_braid.
5   Si indices_braid y vector_auxiliar tienen distinto tamaño, asigno
    a braid_aux2 una cadena con ciertos ceros para mejor
    visualización.
```

Estos algoritmos auxiliares no se proporcionan para uso directo al usuario ya que el realmente interesante es el algoritmo de Dehornoy, pero sí podrían ser usados. Veamos entonces el algoritmo de Dehornoy.

Algoritmo 4.1.4. Algoritmo dehornoy(br, N_cortes, Radio, representar)*ENTRADA:* br (trenza)*N_cortes* (numero de cortes de las cadenas de la trenza)*Radio* (radio de las cadenas de la trenza)*representar* (bool para representar las equivalencias de la trenza)*SALIDA:* es_trivial (bool que nos indica si la trenza dada es o no trivial)*trenza_final* (cadena de enteros que representa a la trenza reducida equivalente a br)

```

1  Si numero_argumentos=1 -> N_cortes=20, Radio=0.5, representar=1.
2  indices_braid = cadena de enteros que representa a la trenza
3  Si br tiene cadenas a la derecha triviales, las eliminamos
   visualmente.
4  Mientras queden handles en la trenza dada...
5      Obtenemos la palabra libremente reducida de indices_braid.
6      Si no se produce reduccion...
7          minimo = generador principal de indices_braid.
8          [pos1,pos2]=encuentra_handle(indices_braid,minimo)
9          Si pos1 y pos2 son posiciones validas....
10         Busco primer subhandle a realizar en el
            handle.
11         Actualizo pos1 y pos2
12         Aplico reduccion_base a dicho subhandle.
13         Creo matriz con secuencia de palabras generadas en el
            proceso.
14 Si representar, muestro las trenzas usando dicha matriz.

```

Finalmente cabe comentar que el algoritmo de Dehornoy se podrá aplicar sobre trenzas cerradas. El proceso que se realizará internamente será exactamente el mismo que para trenzas, ya que el cerrar la trenza no aporta información nueva para el algoritmo de Dehornoy.

Algoritmo de equivalencia para trenzas.**Algoritmo 4.1.5. Algoritmo equivalentes**(br1,br2,explicacion)*ENTRADA:* br1 (trenza1)*br2* (trenza2)*explicacion* (bool para mostrar mensajes explicativos)*SALIDA:* equi (bool para indicar si br1 y br2 son o no equivalentes)

```

1  Si numero_argumentos=2 -> explicacion=0.
2  Obtengo exponente de ambas trenzas.
3  Si son distintos -> No son equivalentes. Finalizo
4  Obtengo permutacion de ambas trenzas.
5  Si son disintas -> No son equivalentes. Finalizo
6  Genero trenza_auxiliar = br1br2^{-1}.
7  Aplico Dehornoy a trenza_auxiliar y obtengo final_braid.
8  Si final_braid no es vacia -> No son equivalentes.
9  Sino -> Si explicacion=1 -> represento secuencia de trenzas de
   dehornoy.

```

Algoritmo de equivalencia para trenzas cerradas.

Algoritmo 4.1.6. Algoritmo equivalentes(*br1, br2, explicacion*)

ENTRADA: *br1* (trenza cerrada 1)

br2 (trenza cerrada 2)

explicacion (bool para mostrar mensajes explicativos)

SALIDA: *equi* (bool para indicar si *br1* y *br2* son o no equivalentes)

```

1      Si numero_argumentos=2 -> explicacion=0.
2      Si le numero de enlaces de br1 y de br2 son disintos -> No son
      equivalentes. Finalizo
3      Si ...
4
5      Obtengo exponente de ambas trenzas.
6      Si son distintos -> No son equivalentes. Finalizo
7      Obtengo permutacion de ambas trenzas.
8      Si son disintas -> No son equivalentes. Finalizo
9      Genero trenza_auxiliar = br1br2-1.
10     Aplico Dehornoy a trenza_auxiliar y obtengo final_braid.
11     Si final_braid no es vacia -> No son equivalentes.
12     Sino -> Si explicacion=1 -> represento secuencia de trenzas de
      dehornoy.
```