PROYECTO TEORIA DE LA PROGRAMACIÓN 2018-1

Daniel Felipe Moreno D’Aleman

Ejercicio 8.28 – “brassard, fundamentos de algoritmia”

1. **ENUNCIADO:** Considere el alfabeto ∑ = {a, b, c}. Los elementos de ∑ tienen la siguiente tabla de multiplicación, donde las filas muestran el símbolo de la izquierda y las columnas muestran el símbolo de la derecha.

	a	*b*	*c*
a	*b*	*b*	*a*
b	*c*	*b*	*a*
c	*a*	*c*	*c*

   De esta manera, ab = b, ba = c y así sucesivamente. Observe que la multiplicación definida por esta tabla no es conmutativa ni asociativa.

   Buscar un algoritmo eficiente que examine una cadena $\boldsymbol{x} = \boldsymbol{x_1} \ldots \boldsymbol{x_n}$ de caracteres de ∑ y decida si es posible o no poner paréntesis en x de tal manera que el valor de la expresión resultante sea a, por ejemplo, si x=bbbba, el algoritmo debería responder <si> porque (b(bb)(ba))=a. Esta expresión no es única. Por ejemplo, (b(b(b(ba))))=a también.

2. **ENTRADA:** Una cadena **x** formada por n caracteres del alfabeto ∑ = {a, b, c}.

   SALIDA: <Si> si existe una manera de poner paréntesis en **x** de tal manera que el valor de la expresión resultante sea **a**, de lo contrario <No>.

3. **EJEMPLOS**
   a. **Entrada:** x = bbbba
      Salida: <Si>
   b. **Entrada:** x = babaca
      Salida: <Si>
   c. **Entada:** x = aa
      Salida: <No>

EJEMPLO PASO A PASO:

Se toma la cadena bbbba.

$$\boldsymbol{paren}(\boldsymbol{a},\text{bbbba}) = \left((\boldsymbol{paren}(\boldsymbol{b},\boldsymbol{b}) \wedge \boldsymbol{paren}(\boldsymbol{c},\text{bbba}))\ \vee (\boldsymbol{paren} \ldots)\right) \equiv \boldsymbol{True}$$

$$\boldsymbol{paren}(\boldsymbol{c},\text{bbba}) = \left((\boldsymbol{paren}(\boldsymbol{b},\boldsymbol{b}) \wedge \boldsymbol{paren}(\boldsymbol{a},\text{bba}))\ \vee (\boldsymbol{paren} \ldots)\right) \equiv \boldsymbol{True}$$

$$\boldsymbol{paren}(\boldsymbol{a},\text{bba}) = \left((\boldsymbol{paren}(\boldsymbol{b},\boldsymbol{b}) \wedge \boldsymbol{paren}(\boldsymbol{c},\text{ba}))\ \vee (\boldsymbol{paren} \ldots)\right) \equiv \boldsymbol{True}$$

$$\boldsymbol{paren}(\boldsymbol{c},\text{ba}) = \left((\boldsymbol{paren}(\boldsymbol{b},\boldsymbol{b}) \wedge \boldsymbol{paren}(\boldsymbol{a},\boldsymbol{a}))\ \vee (\boldsymbol{paren} \ldots)\right) \equiv \boldsymbol{True}$$

$$\boldsymbol{paren}(\boldsymbol{b},\boldsymbol{b}) \equiv \boldsymbol{True}$$

$$\boldsymbol{paren}(\boldsymbol{a},\boldsymbol{a}) \equiv \boldsymbol{True}$$

No es necesario continuar con las otras combinaciones, dado que por teorema $(True \vee \theta) \equiv True$ se pueden omitir las demás, en dado contrario que diera False, se debe continuar con las demás combinaciones.

4. DISEÑO RECURRENTE.

$$\boldsymbol{paren}(\boldsymbol{z},\boldsymbol{x_1} \ldots \boldsymbol{x_n}) = \begin{cases} \boldsymbol{z} \equiv \boldsymbol{x} & si\ \ n = 1 \\ \left((\boldsymbol{paren}(\boldsymbol{f},\boldsymbol{x_1} \ldots \boldsymbol{x_i}) \wedge \mathbf{paren}(\boldsymbol{g},\boldsymbol{x_{i+1}} \ldots \boldsymbol{x_n}))\ \vee (\boldsymbol{paren} \ldots)\right) & si\ \ (1 \leq\ i < n) \wedge (f \times g = z) \end{cases}$$

Donde:

- n es la longitud de la cadena $\boldsymbol{x_1} \ldots \boldsymbol{x_n}$
- f y g son letras del alfabeto.
- Z es la letra a la cual se quiere llegar (si es alcanzable o no) a través de las diferentes combinaciones de la cadena x.

5. PASO A PASO PARA EL DISEÑO ITERATIVO.

a. **Estructura de datos:** Arreglo el cual almacena todos los resultados finales de cada posible forma de poner paréntesis en x.

b. **Ejemplo:**

Se parte de la cadena abc

1. Se toma la forma a(bc)
2. Se mira el resultado de bc, el cual es ‘a’.
3. Se reemplaza el resultado, quedando ‘aa’
4. Se opera nuevamente, donde el resultado es ‘b’
5. Cuando se tenga un solo carácter, se compara con ‘a’, dando False.
6. Se agrega el resultado Arreglo.
7. Se toma la forma (ab)c
8. Se mira el resultado de ab, el cual es ‘b’
9. Se reemplaza el resultado, quedando ‘bc’
10. Se opera nuevamente, donde el resultado es ‘a’
11. Cuando se tiene un solo carácter, se compara con ‘a’, dado True

El arreglo final será de la forma arreglo = {False, True}.

12. Por ultimo se mira en el arreglo, si hay un True, en este caso como ahí uno, se retorna un <Si>.

c. **Interpretación:** Valor de verdad, el cual indica si es posible o no poner paréntesis a una cadena X conformada por lo caracteres en el alfabeto ∑ = {a, b, c} de tal manera que al operar su resultado final sea ‘a’.