UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE INGENIERÍA Y CIENCIAS
CC3001 - 1 Algoritmos y Estructuras de Datos

TAREA 4
Cálculo de Derivadas usando Árboles Binarios

Daniela Campos
danielacamposfischer@gmail.com

Profesor:
Patricio Poblete

Fecha:
12 de Junio del 2017

Índice

1 Introducción

En el presente informe, se expondrá como se realizó la Tarea 4 de Algoritmos y Estructuras de Datos, la cuál consistía en crean un programa que resibiera una expresión en notación polaca inversa y retornara la expresión derivada con respecto a a alguna variable.

El problema fue resuelto utilizando árboles binarios. La expresión textual entregada era traspasada a un árbol binario utlizando una pila. Posterior a esto, se seguían las reglas usuales para derivar las expresiones que habían en el árbol. Finalmente, se agregaban paréntesis si era necesario.

Se utilizaron árboles binarios como la estructura de datos principal de programa debido a la facilidad que otorgan a la hora de evaluar expresiones y también por que permiten manipular simbólicamente las expresiones entregadas.

2 Análisis del Problema

2.1 Problema

El problema propuesto, consiste en que será entregado como input una expresión aritmética en notación polaca inversa y una variable respecto a la cual se debe derivar.

La idea es llevar la expresión aritmética a un árbol binario, el cual sea capaz de representar la expresión. Para lograr esto, se utilizará una Pila. El proceso se llevará a cabo de la siguiente manera: se leerán los valores del input uno a uno, si el valor leido corresponde a un número o una variable se creará un nodo con ese valor y este será ingresado a la Pila.

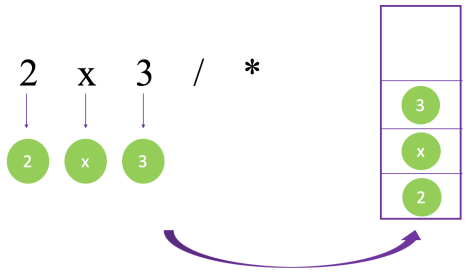

Figura 1: Push a Pila de un número o variable

De otra manera, si el valor leido es un operador, se desapilarán los dos último elementos de la pila y creará un árbol binario, que tendrá como raiz al operador y como hijos a los dos elementos desapilados. Posteriormente, este árbol será apilado. Este proceso se lleva a cabo independiente de si los elementos desapilados son nodos o árboles.

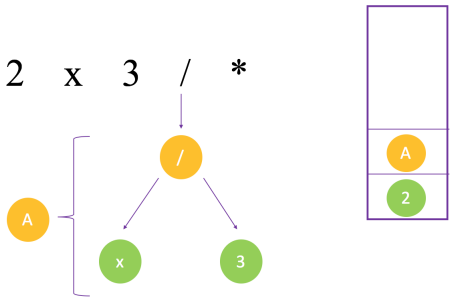

Figura 2: Push a Pila de una árbol

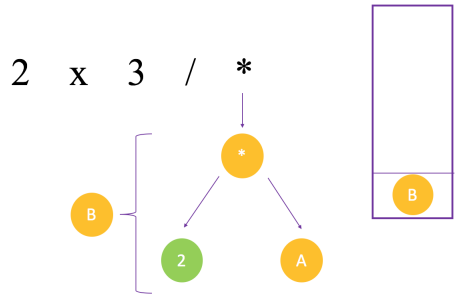

Figura 3: Push a Pila de un árbol que tien por hijo otro árbol

El resultado final de este proceso es una pila que contiene un único elemento(nodo o árbol), el cual pasará a ser derivado.

Al derivar el árbol se siguen las siguientes reglas: si es que el nodo raiz es un número o una variable diferente respecto a la cual se está derivando, ese árbol es reemplazado por un 0, si es que el nodo raiz es la variable con respecto a la cual se deriva, el árbol es reemplazado por un 1, y por último si es que el nodo raiz es un operador, se deben seguir las siguientes reglas de derivación y reemplazar el árbol según corresponda.

$$(f \pm g)' = f' \pm g'$$
$$(f \cdot g)' = f' \cdot g + f \cdot g'$$
$$\left(\frac{f}{g}\right)' = \frac{f' \cdot g - f \cdot g'}{g^2}$$

Este proceso es llevado a cabo de manera recursiva en el árbol. Finalmente, el árbol resultante es recorrido en inorden, es decir primero el subárbol izquierdo, después la raiz y por último el subárbol derecho, para imprimir la expresión en notación infija.

2.2 Suposiciones sobre el Problema

Se supondrá que, la expresiónes entregada tiene variables de un sólo carácter y que los números de la expresión estarán entre el 0 y el 9. No es necesario suponer que los símbolos vienen separados por un sólo espacio debido a la implementación realizada, la cual puede ser encontrada en la sección de Solución del Problema.

2.3 Casos Excepcionales

En algunos casos, la implementación de la solución a este problema, no entrega el resultado de la manera que se querría, debido a que se trabaja con **Strings**. Por ejemplo:

Al entregarle como input la expresión

$$4x * x* \Longleftrightarrow 4 \cdot x^2$$

y derivarla con respecto a x, el programa retorna

$$4 \cdot x + 4 \cdot x$$

Expresión que es equivalente al resultado de la derivacion $8 \cdot x$, pero no está escrito como se esperaría.

Al entregarle como input la expresión

$$xx * 2/ \Longleftrightarrow \frac{x^2}{2}$$

y derivarla con respecto a x, el programa retorna

$$(x + x) \cdot 2/2 \cdot 2$$

Lo que es equivalente a x, pero sin las simplificaciones correspondientes. La razón de estos resultados, es que las expresiones son trabajadas como **Strings** y no se operan dentro del programa, por lo que no es posible hacer simplificaciones.

3 Solución del Problema

El programa toma como input del usuario la expresión a derivar en notación polaca inversa y la variable con respecto a la cual se derivará. Posteriormente, crea el árbol utilizando una Pila y luego lo deriva.

3.1 Clases y Métodos Utilizados

A continuación se expondrán las Clases y Métodos utilizados en el programa.

3.1.1 Clase Nodo

Esta clase es al utilizada para crear los árboles con los cuales se trabaja en el programa. Posee un constructor y el método *imprimir*.[1]

3.1.2 Constructor Nodo

El constructor toma como argumento un **String** que representa la información que habrá en la raiz del árbol y 2 **Nodos** que corresponden a el subárbol izquierdo y derecho.

3.1.3 Método Imprimir

El método *imprimir* no toma argumentos. Recorre el árbol en inorden(subarbol izquierdo, raiz, subárbol derecho) para imprimir la expresión aritmética en notaición infija.

3.1.4 Clase Pila

Posee un constructor, el método *estaVacia*, el método *apilar* y el método *desapilar*. [2]

3.1.5 Constructor Pila

La Clase Pila se crea implementando una Lista Enlazada. El constructor crea la Clase Nodo junto con el Objeto Pila, el cual tiene dos elementos, el último nodo y el tamaño de la pila.

[1] El código para la Clase Nodo se encuentra en el Anexo 1.
[2] El código para la Clase Pila se encuentra en el Anexo 2.

3.1.6 Método estaVacia

Este método chequea si la Pila está vacía al revisar si el tamaño de esta es 0. Retorna un valor del tipo **boolean**, *True* si está vacía y *False* sino.

3.1.7 Método apilar

Este método toma como argumento el **Nodo** a apilar y lo ingresa a la Pila, además hace crecer en uno el tamaño de la Pila. No retorna nada dado que es un **void**.

3.1.8 Método desapilar

Este método desapila el último elemento de la Pila y retorna ese nodo. Si la Pila está vacía retorna null. Además hace decrecer la variable tamaño en 1.

3.1.9 Clase Tarea4

Esta clase posee los métodos **parentesis**, **simplificar**, **derivar**, **makeTree** y **main**. Las especificaciones de que hace cada método están a continuación.

3.1.10 Método Parentesis

Este método le agrega al árbol ya derivado los paréntesis necesarios por prioridad de operaciones. Recibe como argumento el árbol y lo retorna con los paréntesis correspondientes.[3]

3.1.11 Método Simplificar

Este método simplifica las expresiones obtenidas al derivar, es decir, por ejemplo si existe una multiplicación por 0 o por 1 hace los cambios correspondientes en el árbol. El método simplificar es utilizado en cada recursión del método derivar. Recibe como argumento el árbol y lo retorna simplificado.[4]

[3]El código para el Método Parentesis se encuentra en el Anexo 3.
[4]El código para el Método Simplificar se encuentra en el Anexo 4.

3.1.12 Método Derivar

Este método deriva de manera recursiva el árbol creado con la Pila, utilizando las reglas de derivación previamente explicadas. Recibe como argumento el árbol inicial y la variable respecto a la cuál se deriva y retorna el árbol derivado.[5]

3.1.13 Método makeTree

Recibe como argumento un arreglo de Strings, el cuál contiene la expresión aritmética entregada y la Pila. Recorre el arreglo de Strings y chequea si es que el valor evaluado es un operador o una variable. Sigue las reglas explicadas con anterioridad para crear el árbol de la expresión aritmética entregada.[6]

3.1.14 Método main

Obtiene del usuario la expresión en notación Polaca Inversa y la variable respecto a la cuál se debe derivar. Crea la Pila y el arreglo de Strings que contendrá a las expresión. Posteriormente llama al método makeTree y al método derivar. Finalmente llama al método parentesis e imprime el árbol resultante.[7]

[5] El código para el Método Derivar se encuentra en el Anexo 5.
[6] El código para el Método makeTree se encuentra en el Anexo 6.
[7] El código para el Método main se encuentra en el Anexo 7.

4 Modo de Uso

Para que el programa funcione correctamente, se deben guardar los archivos Nodo.java, Pila.java y Tarea4.java en la misma carpeta. Se deben compilar Nodo.java y Pila.java antes de compilar Tarea4.java, tras compilar este y ejecutarlo en el terminal, se le pedirá al usuario que ingrese una expresión aritmética en notación Polaca Inversa y la variable respecto a la cual se quiere derivar.

5 Discusión

5.1 Ejemplos de Entradas y Salidas

A continuación se mostrarán los resultados de ejecutar el programa con diferentes entradas.

5.1.1 Entrada 1:

```
Ingrese expresion en notacion Polaca Inversa: 2 x 3 / * y x - +
Respecto a que quiere derivar esta expresion? x

Output: 2 * 3 / 3 * 3 + - 1
```

5.1.2 Entrada 2:

```
Ingrese expresion en notacion Polaca Inversa: 2 x 3 / * y x - +
Respecto a que quiere derivar esta expresion? y

Output: 1
```

5.1.3 Entrada 3:

```
Ingrese expresion en notacion Polaca Inversa: x x * 2 /
Respecto a que quiere derivar esta expresion? x

Output: ( x + x ) * 2 / 2 * 2
```

5.1.4 Entrada 4:

```
Ingrese expresion en notacion Polaca Inversa: x x * 2 + 3 /
Respecto a que quiere derivar esta expresion? x

Output: ( x + x ) * 3 / 3 * 3
```

5.1.5 Entrada 5:

```
Ingrese expresion en notacion Polaca Inversa: 3 x * x * 2 x * + 7 +
Respecto a que quiere derivar esta expresion? x

Output: 3 * x + 3 * x + 2
```

5.1.6 Entrada 6:

```
Ingrese expresion en notacion Polaca Inversa: y y * x x * /
Respecto a que quiere derivar esta expresion? y

Output:( y + y ) * x * x / x * x * x * x
```

6 Anexos

1. **Anexo 1:** Clase Nodo

```
public class Nodo {

    String info;
    Nodo izq;
    Nodo der;

    public Nodo(String info,Nodo i, Nodo d) {
        this.info=info;
        this.izq = i;
        this.der = d;
    }

    public void imprimir(){
        Nodo puntero = this;
        if (puntero != null){
            if(puntero.izq == null && puntero.der == null){
                System.out.print(puntero.info);
                System.out.print(" ");
            }
            else if(puntero.izq != null && puntero.der == null){
                puntero.izq.imprimir();
                System.out.print(puntero.info);
                System.out.print(" ");
            }
            else if(puntero.izq == null && puntero.der != null){
                System.out.print(puntero.info);
                System.out.print(" ");
                puntero.der.imprimir();
            }
            else if(puntero.izq != null && puntero.der != null){
                puntero.izq.imprimir();
                System.out.print(puntero.info);
                System.out.print(" ");
                puntero.der.imprimir();
            }
        }
    }
}
```

2. **Anexo 2:** Clase Pila

```
import java.util.*;

public class Pila{
    private Node ultimo;
    private int tamano;

    private class Node{
        Nodo value;
        Node sgte;
    }

    public Pila(){
        ultimo = null;
        tamano = 0;
    }

    boolean estaVacia(){
        if(tamano == 0){
            return true;
        }
        return false;
    }

    void apilar(Nodo valor){
        if(estaVacia()){
            ultimo = new Node();
            ultimo.value = valor;
            ultimo.sgte = null;
        }
        else{
            Node anterior = ultimo;
            ultimo = new Node();
            ultimo.value = valor;
            ultimo.sgte = anterior;
        }
        ++tamano;
    }
```

```
    Nodo desapilar() {
        Nodo pos = null;
        if (estaVacia()) {
            System.out.println("Pila Vacia");
            pos = null;
        }
        else {
            pos = ultimo.value;
            if (tamano == 1) {
                ultimo = null;
            } else {
                ultimo = ultimo.sgte;
            }
            --tamano;
        }

        return pos;
    }
}
```

3. **Anexo 3:** Método Parentesis Clase Tarea4

```
public static Nodo  parentesis(Nodo tree){
        Nodo a,b;
        if(tree != null && (tree.info.charAt(0) == '*' ||
                 tree.info.charAt(0) == '/')){
            if(tree.izq != null && (tree.izq.info.charAt(0) == '+'||
                 tree.izq.info.charAt(0) == '-')){
                a = parentesisIzq(tree.izq);
                b = parentesisDer(tree.izq);
                tree.izq = new Nodo(b.info, b.izq, b.der);
            }
            if(tree.der != null && (tree.der.info.charAt(0) == '+'||
                 tree.der.info.charAt(0) == '-')){
                a = parentesisIzq(tree.der);
                b = parentesisDer(tree.der);
                tree.der = new Nodo(b.info, b.izq, b.der);
            }
        }
        a = null;
        b = null;
        if(tree.izq != null){
            a = parentesis(tree.izq);
        }
        if(tree.der != null){
            b = parentesis(tree.der);
        }
        tree = new Nodo(tree.info, a, b);

        return tree;
    }
```

4. **Anexo 4:** Método Simplificar Clase Tarea4

```
public static Nodo simplificar(Nodo tree){
        char value = tree.info.charAt(0);
        if(value == '+'){
            if(tree.izq != null && tree.izq.info.charAt(0) == '0'){
                if(tree.der != null){
                    tree = new Nodo(tree.der.info, tree.der.izq,
                          tree.der.der);
                }
                else{
                    tree = new Nodo(null, null, null);
                }
            }
            if(tree.der != null && tree.der.info.charAt(0) == '0'){
                if(tree.izq != null){
                    tree = new Nodo(tree.izq.info, tree.izq.izq,
                          tree.izq.der);
                }
                else{
                    tree = new Nodo(null, null, null);
                }
            }
            if(tree.izq != null && tree.izq.info.charAt(0) == '0' &&
                 tree.der != null && tree.der.info.charAt(0) == '0'){
                tree = new Nodo("0", null, null);
            }
        }
        else if(value == '-'){
            if(tree.izq != null && tree.izq.info.charAt(0) == '0'){
                if(tree.der != null){
                    tree = new Nodo("-", null, tree.der);
                }
            }
            if(tree.der != null && tree.der.info.charAt(0) == '0'){
                if(tree.izq != null){
                    tree = new Nodo(tree.izq.info, tree.izq.izq,
                          tree.izq.der);
                }
            }
            if(tree.izq == null && tree.der != null &&
                 tree.der.info.charAt(0) == '0'){
                tree = new Nodo("0", null, null);
            }
```

```
            if(tree.der == null && tree.izq != null &&
                 tree.izq.info.charAt(0) == '0'){
                tree =  new Nodo("0", null, null);
            }
        }
        else if(value == '*'){
            if((tree.izq != null && tree.izq.info.charAt(0) == '0' )||
                (tree.der != null && tree.der.info.charAt(0) == '0')){
                tree= new Nodo("0", null, null);
            }
            if(tree.izq != null && tree.izq.info.charAt(0) == '1'){
                if(tree.der != null){
                    tree = new Nodo(tree.der.info, tree.der.izq,
                        tree.der.der);
                }
            }
            if(tree.der != null && tree.der.info.charAt(0) == '1'){
                if(tree.izq != null){
                    tree = new Nodo(tree.izq.info, tree.izq.izq,
                        tree.izq.der);
                }
            }
        }
        else if(value == '/'){
            if(tree.der != null && tree.der.info.charAt(0) == '1'){
                tree = new Nodo(tree.izq.info, tree.izq.izq,
                        tree.izq.der);
            }
            if(tree.izq != null && tree.izq.info.charAt(0) == '0'){
                tree = new Nodo("0", null, null);
            }
        }
        return tree;
    }
```

5. **Anexo 5:** Método Derivar Clase Tarea4

```
public static Nodo derivar(Nodo tree, String variable){
        Nodo a,b, ap, bp, c, d, e, f, g, h, i;
        char value = tree.info.charAt(0);
        char variable2 = variable.charAt(0);
        if(value != variable2 && value != '*' && value != '/' &&
                value != '+' && value != '-'){
            tree.info = "0";
        }
        else if(value == variable2){
            tree.info = "1";
        }
        else{
            if(value == '+' || value == '-'){
                a = derivar(tree.izq, variable);
                b = derivar(tree.der, variable);
                tree.izq = new Nodo(a.info, a.izq, a.der);
                tree.der = new Nodo(b.info, b.izq, b.der);
                c = simplificar(tree);
                tree = new Nodo(c.info, c.izq, c.der);

            }
            else if(value == '*'){
                tree.info = "+";
                a = new Nodo(tree.izq.info, tree.izq.izq, tree.izq.der);
                b = new Nodo(tree.der.info, tree.der.izq,
                        tree.der.der);
                ap = new Nodo(tree.izq.info, tree.izq.izq,
                        tree.izq.der);
                bp = new Nodo(tree.der.info, tree.der.izq,
                        tree.der.der);
                c = derivar(ap,variable);
                d = derivar(bp,variable);
                ap = new Nodo(c.info, c.izq, c.der);
                bp = new Nodo(d.info, d.izq, d.der);

                tree.izq = new Nodo("*", ap, b);
                e = simplificar(tree.izq);
                tree.izq = new Nodo(e.info, e.izq, e.der);

                tree.der = new Nodo("*", a, bp);
                f = simplificar(tree.der);
                tree.der = new Nodo(f.info, f.izq, f.der);
```

```
                g = simplificar(tree);

                tree = new Nodo(g.info, g.izq, g.der);
            }
            else if(value == '/'){
                a = new Nodo(tree.izq.info, tree.izq.izq, tree.izq.der);
                b = new Nodo(tree.der.info, tree.der.izq,
                        tree.der.der);
                ap = new Nodo(tree.izq.info, tree.izq.izq,
                        tree.izq.der);
                bp = new Nodo(tree.der.info, tree.der.izq,
                        tree.der.der);
                c = derivar(ap,variable);
                d = derivar(bp,variable);
                ap = new Nodo(c.info, c.izq, c.der);
                bp = new Nodo(d.info, d.izq, d.der);
                e = new Nodo("*", ap, b);
                f = new Nodo("*", a, bp);

                tree.izq = new Nodo("-", simplificar(e),
                        simplificar(f));
                g = simplificar(tree.izq);
                tree.izq = new Nodo(g.info, g.izq, g.der);

                tree.der = new Nodo("*", b, b);
                h = simplificar(tree.der);
                tree.der = new Nodo(h.info, h.izq, h.der);

                i = simplificar(tree);
                tree = new Nodo(i.info, i.izq, i.der);
            }
        }
        return tree;
    }
```

6. **Anexo 6:** Método makeTree Clase Tarea4

```
public static void makeTree(String[] S, Pila p){
        Nodo a,b;
        for(int i = 0; i < S.length; ++i){
            char value = S[i].charAt(0);
            if(value == '*' || value == '/' || value == '+' ||
                 value == '-'){
                a = p.desapilar();
                b = p.desapilar();
                Nodo op = new Nodo(S[i],b, a);
                p.apilar(op);
            }
            else{
                Nodo valor = new Nodo(S[i], null, null);
                p.apilar(valor);
            }
        }
    }
```

7. **Anexo 7:** Método main Clase Tarea4

```
static public void main(String[] args){

        Scanner s = new Scanner(System.in);
        System.out.print("Ingrese exp. en notacion Polaca Inv.: ");
        String exp = s.nextLine();
        System.out.print("Respecto a que deriva la expresion? ");
        String variable = s.nextLine();

        Pila pila = new Pila();
        String[] S = exp.split(" ");

        makeTree(S, pila);

        Nodo treeDerivado = new Nodo(null, null, null);

        Nodo treeFinal = pila.desapilar();

        Nodo a = derivar(treeFinal, variable);
        treeFinal = new Nodo(a.info, a.izq, a.der);

        Nodo b = parentesis(treeFinal);
        treeFinal = new Nodo(b.info, b.izq, b.der);

        treeFinal.imprimir();
        System.out.println();

    }
```