Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Tarea 4: Cálculo de derivadas usando árboles binarios

Fecha: 5 de junio de 2017
Autor: Daniel Araya Poblete
Email: Daniel.arayap1@gmail.com
Curso: CC3001-1
Profesor: Nelson Baloian

Introducción

El objetivo de esta tarea es implementar un árbol binario como un método para derivar expresiones matemáticas simples. Esto puede realizarse guardando las expresiones matemáticas, variables, constantes y operaciones; en los nodos del árbol, y de esta manera se pueden manipular de forma libre para poder derivarlas.

La idea inicial es que el usuario entregue la expresión como un String, en notación polaca inversa, además del nombre de la variable con respecto a la cual se quiere derivar la expresión, y que el programa devuelva la expresión ordenada (en una notación de izquierda a derecha) del input y su expresión derivada. La solución para esto fue implementar un programa que utilice una Pila, un árbol binario donde se guarde la expresión a derivar, y otro árbol adicional donde se guarde la expresión derivada. Cabe destacar que al hablar de árboles binarios en esta tarea, realmente se está refiriendo a nodos simples que cuentan con punteros a un hijo izquierdo y un hijo derecho.

Los métodos, implementación y resultados obtenidos en esta tarea se verán en detalle en las siguientes secciones de presente informe.

Análisis del problema

El objetivo de esta tarea es escribir un algoritmo que, al recibir una expresión por parte del usuario, el sea capaz de reconocer la expresión matemática y devuelva su derivada con respecto a la variable que especifique el usuario. Para esto se pide trabajar con una pila de nodos y procesar de forma iterativa cada carácter, de la siguiente manera:

- Si el carácter es un dígito o variable, se crea un nodo con su valor y se apila
- Si es una operación, se crea un nodo cuyo valor es la operación, su hijo derecho sería el siguiente nodo desapilado y su hijo izquierdo el siguiente; luego este nodo se apila.

Se debe mencionar que, para simplificar el programa, éste trabajará sólo con las cuatro operaciones elementales (+, -, *, /). Un ejemplo del procedimiento que genera el árbol de la expresión puede verse en la Figura 1.

Una vez que se tiene este árbol, el programa debe mostrar en pantalla la expresión en el orden normal (de izquierda a derecha), además de su derivada. Para encontrar la derivada, en la tarea se sugiere crear un nuevo árbol binario similar al que contiene la expresión, tal que en cada nodo contiene la derivada del nodo correspondiente en el árbol original, usando las siguientes reglas:

- Si el valor del nodo es número o una variable distinta a la cual se deriva, entrega un nodo de valor 0.
- Si el nodo es la variable respecto a la cual se deriva la expresión, el resultado es un nodo con valor 1.
- Si el nodo es una operación entonces el resultado es un árbol que contiene la fórmula correspondiente a la regla de la derivada para la operación.

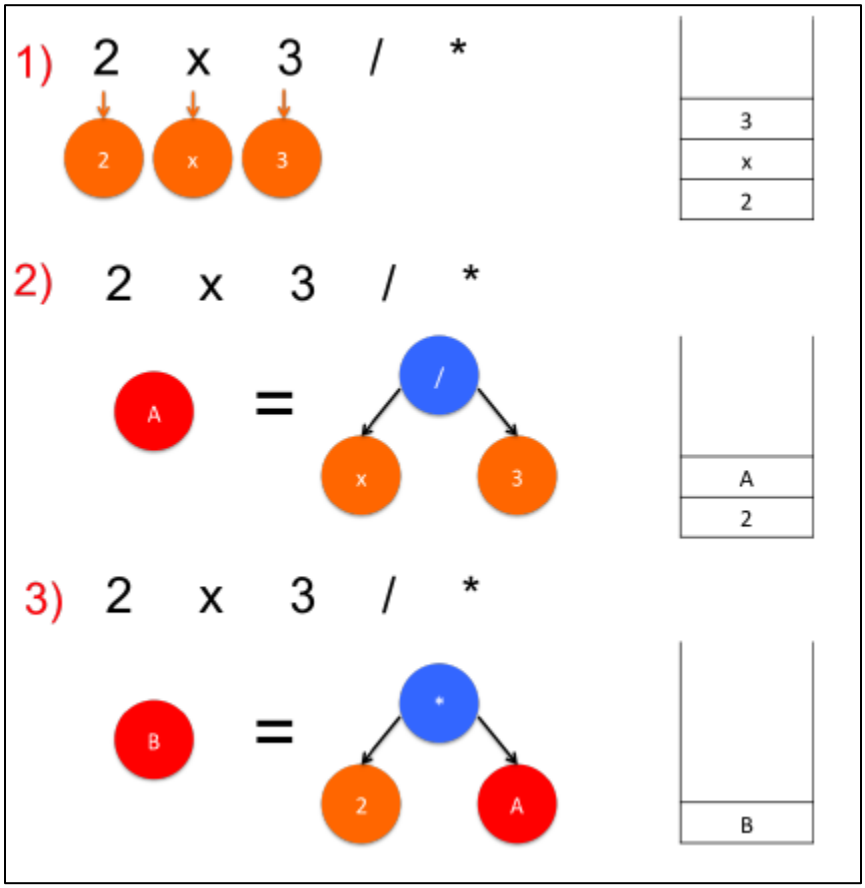

Figura 1. En el primer paso se apilan los tres nodos en la pila, ya que ninguno es una operación. Luego al llegar a un valor que corresponde a una operación, se crea un nodo con su valor y los dos nodos siguientes en la pila como hijos, y este nuevo "árbol" se apila en la pila.

Para realizar esta tarea se tomaron algunos supuestos, tanto para simplificar el problema como para simplificar el resultado:

- Los nombres de variables siempre tienen un sólo caracter.
- Los números de la expresión serán números del 0 al 9.
- Los símbolos vienen siempre separados por un solo espacio

Para simplificar el resultado, se tomaron los siguientes supuestos:

- La multiplicación de un término por 0 se deben reemplazar por 0.
- La multiplicación de un término por 1 se deben reemplazar por el mismo término.
- Las sumas o restas de un término con 0 se deben reemplazar por el mismo término.
- Las divisiones de un término por 1 se deben reemplazar por el mismo término.
- La expresión final debe omitir los paréntesis que no son necesarios (los paréntesis más externos, por ejemplo).

Solución del problema

Para resolver esta tarea se siguió paso a paso las indicaciones ofrecidas en el enunciado. Inicialmente er necesario tener una clase Nodo, que tuviera un puntero hacia su hijo izquierdo y otro hacia su hijo derecho (Figura 2).

```
class Nodo{
   String info;
   Nodo izq, der;
   Nodo(String x, Nodo y, Nodo z){
      info =x;
      izq=y;
      der=z;
   }
```

Figura 2. Clase Nodo.

Para la implementación de la tarea era necesario tener distintos métodos que procesaran la entrada:

- El método *construir* toma el String en el que estará contenida la expresión del input, este método crea el árbol que contiene la expresión matemática. Para esto se utiliza un Stack pila de Nodos como se requiere para este programa, es decir, si encuentra una variable o un dígito lo guarda como un Nodo y se apila; en caso contrario (si es una operación) la guarda en un Nodo, y sus Nodos hijos serán los siguientes Nodos que se extraigan de la pila. (ver figura 3).

```
if (caracter == '+' || caracter == '*' || caracter == '/' || caracter == '-') {
    Nodo aux = new Nodo();
    aux.info = "" + caracter;
    aux.der = pila.pop();
    aux.izq = pila.pop();
    pila.push(aux);
} else if (isDigit(caracter) || isLetter(caracter)) {
    Nodo aux = new Nodo("" + caracter, null, null);
    pila.push(aux);
}
```

Figura 3. Extracto del método private static Nodo construir(String fx).

- El método *derivar* pide los parámetros del Nodo que contiene el árbol a derivar y la variable según la cual se quiere derivar y retorna un nuevo árbol que contiene la expresión original derivada. Para esto evalúa los distintos casos: que el Nodo contenga una variable que no sea la variable por la cual derivar o un dígito (resultado 0), que el Nodo contenga un '+' o un '-' (retorna $f' \pm g'$), que el Nodo contenga un '*' (retorna $f'g + g'f$) o que el Nodo contenga un caracter '/' (retorna $\frac{f'g - fg'}{g^2}$). Para cada operación el programa crea un Nodo cuyo atributo *info* contiene la operación de la derivada y sus Nodos hijos contienen los miembros de la operación (ver Figura 4).

```
if(caracter.equals(var)) derivada.info ="1";
else if(caracter.equals("+")||caracter.equals("-")){
   derivada.info =caracter;
   derivada.izq=derivar(fx.izq,var);
   derivada.der=derivar(fx.der,var);
}
else if(caracter.equals("*")){
   derivada.info ="+";
   derivada.izq=new Nodo("*",derivar(fx.izq,var),fx.der);
   derivada.der=new Nodo("*",fx.izq,derivar(fx.der,var));
}
else if(caracter.equals("/")){
   derivada.info =caracter;
   derivada.izq=new Nodo();
   derivada.izq.info ="-";
   derivada.izq.izq=new Nodo("*",derivar(fx.izq,var),fx.der);
   derivada.izq.der=new Nodo("*",fx.izq,derivar(fx.der,var));
   derivada.der=new Nodo("*",fx.der,fx.der);
}
else derivada.info ="0";
return derivada;
```

Figura 4. Cuerpo del método *private static Nodo derivar(Nodo fx, String var).*

- El método *simplificar* toma como argumento el árbol que contiene la función y la simplifica bajo los supuestos vistos en la sección anterior según los distintos casos, como se puede ver en la Figura 5. El programa no puede realizar la simplificación de los paréntesis ya que al hacer que éste método sea recursivo

```
if (!s.equals("+") && !s.equals("*") && !s.equals("-") && !s.equals("/")) return fx;
else{
   fx.izq=simplificar(fx.izq);
   fx.der=simplificar(fx.der);
          switch (s) {
              case "+":
                  if (fx.izq.info.equals("0")) {
                      return fx.der;
                  } else if (fx.der.info.equals("0")) {
                      return fx.izq;
                  } else return fx;
              case "*":
                  if (fx.izq.info.equals("0") || fx.der.info.equals("0")) {
                      return new Nodo("0", null, null);
                  } else if (fx.izq.info.equals("1")) {
                      return fx.der;
                  } else if (fx.der.info.equals("1")) {
                      return fx.izq;
                  } else return fx;
              case "/":
                  if (fx.izq.info.equals("0")) {
                      return new Nodo("0", null, null);
                  } else if (fx.der.info.equals("1")) {
                      return fx.izq;                  }
                  break;
              default:
                  if (fx.der.info.equals("0")) {
                      return fx.izq;
                  }
                  break;
          }
   return fx;
```

Figura 5. Cuerpo del método *private static Nodo simplificar(Nodo fx)*

- El método *getString* toma como argumento un nodo que contiene la expresión (ya sea la expresión original o la derivada) y retorna un String con el resultado (ver Figura 6).

```
if (!raiz.equals("+") && !raiz.equals("*") && !raiz.equals("-") && !raiz.equals("/")){
   System.out.print(raiz);
}
else{
   System.out.print("(");
   imprimir(fx.izq);
   System.out.print(raiz);
   imprimir(fx.der);
   System.out.print(")");
}
```

Figura 6. Extracto del método *private static String getString(Nodo fx)*

Con estos métodos, en el método main se obtienen la expresión, guardada en un String (en notación polaca inversa), y la variable según la cual se quiere derivar. Luego se construye el árbol, se simplifica y se imprime el String obtenido con *getString*, quitándole los paréntesis externos. Finalmente se deriva este árbol, se simplifica y se imprime nuevamente el resultado (simplificando también los paréntesis de la misma forma que con la expresión original, usando el método *substring*).

Modos de uso y resultados

El programa le pedirá al usuario una expresión matemática a derivar (en notación polaca inversa), además de una variable según la cual el usuario quiera derivar ésta. A continuación, se mostrarán algunos ejemplos de input para el programa:

- Entrada: 2 x 3 / * y x - +
  Derivando en: x
  Corresponde a: (2*(x/3))+(y-x)
  Salida: (2*(3/(3*3)))+(0-1)
- Entrada: 2 x 3 / * y x - +
  Derivando en: y
  Corresponde a: (2*(x/3))+(y-x)
  Salida: 1
- Entrada: 2 y + 3 * 5 /
  Derivando en: y
  Corresponde a: ((2+y)*3)/5
  Salida: (3*5)/(5*5)
- Entrada: 2 y + 3 * 5 /
  Derivando en: x
  Corresponde a: ((2+y)*3)/5
  Salida: 0
- Entrada: x y / 5 /
  Derivando en: x
  Corresponde a: (x/y)/5
  Salida: ((y/(y*y))*5)/(5*5)
- Entrada: x y / 5 /
  Derivando en: y
  Corresponde a: (x/y)/5
  Salida: (((0-x)/(y*y))*5)/(5*5)
- Entrada: 3 2 + x y + /
  Derivando en: x
  Corresponde a: ((3+2)/(x+y))
  Salida: (0-(3+2))/((x+y)*(x+y))
- Entrada: 3 2 + x y + /
  Derivando en: y
  Corresponde a: ((3+2)/(x+y))
  Salida: (0-(3+2))/((x+y)*(x+y))

Anexo: Código fuente

```
package main;
import java.io.*;
import java.util.*;
import static java.lang.Character.*;
public class Tarea4 {
   static public void main(String[] args) throws IOException{
        Scanner sc=new Scanner(System.in);
        System.out.print("Entrada: ");
        String fx=sc.nextLine();
        System.out.print("Derivando en: ");
        String var=sc.nextLine();
        sc.close();
        String expresion=getString(simplificar(construir(fx)));
        System.out.print("Corresponde a: "+ expresion.substring(1,expresion.length()-1));
        expresion=getString(simplificar(derivar(construir(fx),var)));
        System.out.print("\nSalida: " + expresion.substring(1,expresion.length()-1) );
   }
   private static Nodo construir(String fx){
        Stack<Nodo> pila=new Stack<>();
        for(int i=0;i<fx.length();i++) {
            char caracter = fx.charAt(i);
            if (caracter == '+' || caracter == '*'||caracter == '/' || caracter == '-') {
                Nodo aux = new Nodo();
                aux.info = "" + caracter;
                aux.der = pila.pop();
                aux.izq = pila.pop();
                pila.push(aux);
            } else if (isDigit(caracter) || isLetter(caracter)) {
                Nodo aux = new Nodo("" + caracter, null, null);
                pila.push(aux);
            }
        }
        return pila.pop();
    }
   private static String getString(Nodo fx){
        String raiz=fx.info;
        String expresion;
        if (!raiz.equals("+")&&!raiz.equals("*")&&!raiz.equals("-")&& !raiz.equals("/")){
            expresion=raiz;
        }
        else{
            expresion="("+getString(fx.izq)+raiz+getString(fx.der)+")";
        }
        return expresion;
    }
   private static Nodo simplificar(Nodo fx){
      String s=fx.info;
      if (!s.equals("+")&& !s.equals("*") && !s.equals("-") && !s.equals("/")) return fx;
      else{
         fx.izq=simplificar(fx.izq);
         fx.der=simplificar(fx.der);
            switch (s) {
                case "+":
                    if (fx.izq.info.equals("0")) {
                        return fx.der;
```

```
                    } else if (fx.der.info.equals("0")) {
                        return fx.izq;
                    } else return fx;
                case "*":
                    if (fx.izq.info.equals("0") || fx.der.info.equals("0")) {
                        return new Nodo("0", null, null);
                    } else if (fx.izq.info.equals("1")) {
                        return fx.der;
                    } else if (fx.der.info.equals("1")) {
                        return fx.izq;
                    } else return fx;
                case "/":
                    if (fx.izq.info.equals("0")) {
                        return new Nodo("0", null, null);
                    } else if (fx.der.info.equals("1")) {
                        return fx.izq
                    }
                    break;
                default:
                    if (fx.der.info.equals("0")) {
                        return fx.izq;
                    }
                    break;
            }
        return fx;
    }
  }
  private static Nodo derivar(Nodo fx, String var){
     String caracter=fx.info;
     Nodo derivada=new Nodo();
     if(caracter.equals(var)) derivada.info ="1";
     else if(caracter.equals("+")||caracter.equals("-")){
        derivada.info =caracter;
        derivada.izq=derivar(fx.izq,var);
        derivada.der=derivar(fx.der,var);
     }
     else if(caracter.equals("*")){
        derivada.info ="+";
        derivada.izq=new Nodo("*",derivar(fx.izq,var),fx.der);
        derivada.der=new Nodo("*",fx.izq,derivar(fx.der,var));
     }
     else if(caracter.equals("/")){
        derivada.info =caracter;
        derivada.izq=new Nodo();
        derivada.izq.info ="-";
        derivada.izq.izq=new Nodo("*",derivar(fx.izq,var),fx.der);
        derivada.izq.der=new Nodo("*",fx.izq,derivar(fx.der,var));
        derivada.der=new Nodo("*",fx.der,fx.der);
     }
     else derivada.info ="0";
     return derivada;
  }
}
```