Estructuras de Datos y Algoritmos

Año 2022

Árboles de Expresión

Introducción

Los avances tecnológicos producen día a día una gran cantidad de información que debe ser almacenada y procesada en forma eficiente. Para ello se deben analizar diferentes tipos de estructuras y seleccionar la más adecuada a cada caso. Si se pretende digitalizar información se trata de definir una relación que la represente y almacenarla de tal manera que pueda ser accedida en forma eficiente. Cada nupla de la relación almacenada representa información de la realidad sobre la que estamos trabajando. Por ejemplo si tenemos almacenada una relación '$R \subseteq$ *patente* $\times$ *año* $\times$ *modelo* $\times$ *marca*' la nupla *(ATV800, 2001, Megane bicuerpo, Renault)* me informa que la patente 'ATV800' pertenece a un auto fabricado en el año '2001', que es de marca 'Renault' y cuyo modelo es 'Megane'. Estas relaciones entre los datos almacenados vienen impuestas por los enunciados que se quieren representar, no se pueden modificar, por ejemplo no se podría colocar en la nupla de la patente 'ATV800' el año '1994' porque el auto identificado con esa patente se fabricó recién en el año 2001. El estudio de las relaciones entre los datos que surgen de representar la información de una realidad, y las propiedades de las mismas, es un tema de estudio que podría llamarse **estructuras de información**.

Hay otras relaciones que también afectan a los datos almacenados, pero no tienen que ver con la información que estos representan. Cuando los datos se almacenan en una estructura, se establece una relación entre las celdas y sus contenidos. La estructura define la organización e interrelación de los datos y un conjunto de operaciones que se pueden realizar sobre ellos. Estas relaciones están influenciadas por las propiedades tecnológicas de la memoria en uso y, como dijimos, no afectan a la información que los datos representan. Del modo en que se dispongan las hileras o cadenas de caracteres que representan información en la memoria surgirá una mayor o menor eficacia en la evocación. Esto constituye lo que llamamos **estructuras de almacenamiento**. Estas relaciones sólo surgen al almacenar los datos en la memoria.

Entre las estructuras de almacenamiento vistas anteriormente se encuentran los árboles binarios ordenados. Sin embargo una importante aplicación de los árboles en la informática es su utilización en el análisis de lenguajes como *árboles sintácticos*, es decir, árboles que contienen las derivaciones de una gramática, necesarias para obtener una determinada frase de un lenguaje. Otra aplicación permite etiquetar los nodos de un árbol con operandos y operadores de manera que un árbol represente una expresión.

Un *árbol de expresión* es un árbol binario usado para representar y evaluar expresiones algebraicas o lógicas formadas por operadores unarios o binarios. Es un caso particular de los árboles sintácticos y representa una estructura de información, ya que los elementos almacenados tienen una relación entre ellos que no depende de la estructura en sí, sino que esta relación determina forma particular que tendrá el árbol. Si la información cambia la estructura cambia; por ejemplo, si en lugar de sumar quiero dividir, esto debe ser reflejado en la estructura, si la expresión cambia también debe hacerlo el árbol.

Un árbol de expresión se construye a partir de los operadores simples y los operandos de alguna expresión poniendo los *operandos* en las *hojas* del árbol binario, y los *operadores* en los *nodos internos*. A diferencia de otras notaciones, ésta puede representar la expresión sin ambigüedad, al mirar el árbol,

no hay duda de cuál es el orden de las operaciones.

Para cada operador *binario*, el subárbol izquierdo contiene todos los operandos y operadores del lado izquierdo del operador y el subárbol derecho contiene todos los del lado derecho. Si el nodo i del árbol de expresión está etiquetado con el operador θ (que puede ser un $<$, un $+$, un $*$, etc.) y el subárbol izquierdo representa la expresión E_1 y el derecho la expresión E_2, entonces el subárbol con raíz en i representa la expresión $E_1\ \theta\ E_2$.

Si el operador es *unario* uno de los subárboles será vacío. Tradicionalmente algunos de los operadores unarios se escriben a la izquierda de sus operandos, por ejemplo: la función logaritmo '$\log()$', la función seno '$\sin()$' y el menos unario '$-$'. Otros se escriben a la derecha de los operandos, como la función factorial '$()!$' o la función que me devuelve el cuadrado '$()^2$'. A veces cualquier lado es permisible, como ocurre con el operador de derivación, el cual puede ser escrito como '$\frac{d}{dx}$' a la izquierda de los operandos o como '$()'$' que va a su derecha. Finalmente, hay operadores que cambian su significado según estén a la derecha o a la izquierda de su operando, como es el caso del operador '$++$' en el lenguaje C. Si el operador se escribe a la izquierda, entonces en el árbol de expresión tomaremos su subárbol izquierdo como vacío, así sus operandos aparecen en el lado derecho del operador en el árbol, exactamente como lo hacen en la expresión. Si el operador se escribe a la derecha, entonces su subárbol derecho será el que aparece vacío, y los operandos estarán en el subárbol izquierdo del operador.

Algunos ejemplos de árboles de expresión se muestran a continuación:

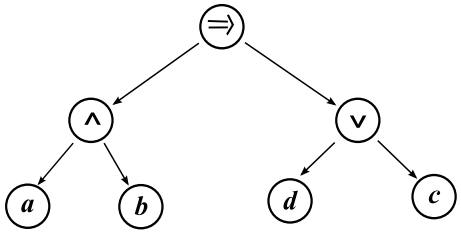

Figura 1: Árbol de expresión para $(a \wedge b) \Rightarrow (d \vee c)$

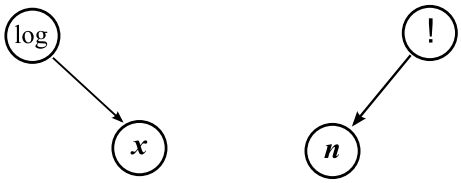

Figura 2: Árboles de expresión de $\log x$ (izquierda) y $n!$ (derecha)

En el ejemplo de la Figura 1 el hijo izquierdo de la raíz tiene al operador '$\wedge$' como etiqueta y sus hijos izquierdo y derecho representan las expresiones a y b respectivamente. Por lo tanto este nodo representa la expresión $(a \wedge b)$. El nodo raíz representa $(a \wedge b) \Rightarrow (d \vee c)$, su etiqueta es el operador '$\Rightarrow$' y las expresiones representadas pos sus hijos son: $(d \vee c)$ y $(a \wedge b)$.

La Figura 2 muestra los árboles de dos expresiones que utilizan operadores unarios.

El árbol de la Figura 3 representa una expresión un poco más compleja y en él se pueden apreciar las siguientes características, que son propias de todos los árboles de expresión:

- La raíz siempre debe ser un operador.

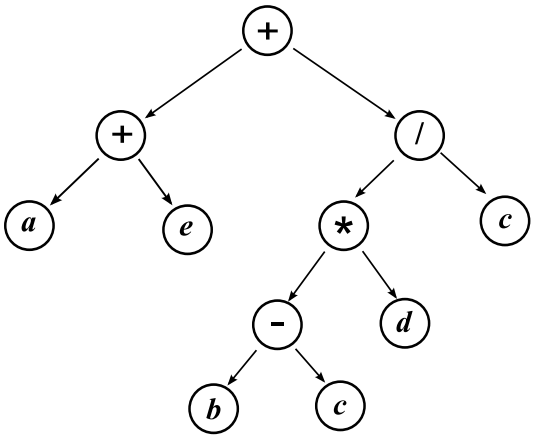

Figura 3: $(a + e + (b - c) * d/c)$

- La raíz de todo subárbol siempre es un operador.
- Las hojas siempre deben ser operandos.
- Los nodos internos deben estar etiquetados por operadores.
- Si un operador tiene mayor prioridad que el que está almacenado en un nodo, se coloca como hijo del mismo.
- Si un operador tiene igual o menor prioridad que el que está en un nodo, se coloca como padre de éste.
- Una expresión entre paréntesis tiene mayor prioridad que cualquier otra.
- Un nodo puede contener como hijo un subárbol que contiene un pequeña expresión.
- los paréntesis no se representan porque no son necesarios.

Entre los muchos usos que tienen los árboles de expresión se pueden mencionar: traducir las expresiones a notación sufijos, prefijos e infijos; ser utilizados dentro de compiladores para analizar, optimizar y traducir programas; evaluar expresiones algebraicas o lógicas; ser utilizados en el análisis de lenguajes, entre otras.

Recorridos sobre un Árbol

Una de las operaciones más importantes sobre un árbol binario es su *barrido* o *recorrido*, es decir, el moverse a través de todos los nodos del árbol visitando a cada uno una única vez y en algún orden determinado.

Si pensáramos en recorrer una lista, por ejemplo, el visitar cada nodo se haría utilizando el orden natural en el que ellos aparecen: *del primero al último*. Para los árboles, en cambio, hay varios órdenes diferentes que surgen naturalmante de su estructura y que permiten que todos sus nodos puedan ser visitados. Recordemos que un árbol binario se define como $< A_1, v, A_2 >$ donde v es la raíz del árbol y A_1 y A_2 son los subárboles, o hijos, izquierdo y derecho de v respectivamente. Durante el barrido de un

árbol, estando en un nodo, hay tres tareas que se desearían realizar en algún orden: visitar el nodo mismo (nodo corriente), recorrer su subárbol izquierdo, recorrer su subárbol derecho. La clave que distingue a los distintos barridos está en decidir si visitamos el nodo corriente *antes* de recorrer sus subárboles, o lo visitamos *entre* cada recorrido de sus subárboles, o se hace *luego* de recorrer ambos subárboles.

Si llamáramos a la visita del nodo corriente V, al barrido del subárbol izquierdo L y al barrido del subárbol derecho R, se puede ver que hay seis formas de organizar estas tareas:

$$VLR \quad LVR \quad LRV \quad VRL \quad RVL \quad RLV$$

Por convención estándar estas seis variantes se ven reducidas a tres, permitiendo sólo aquellos recorridos en los que el subárbol izquierdo es barrido antes que el derecho. Así las tres formas de recorrer un árbol en las que se visita el subárbol izquierdo antes que el derecho tiene nombres especiales con los que son reconocidas:

VLR	LVR	LRV
Preorden	*Inorden*	*Postorden*

Estos nombres son asignados a cada recorrido de acuerdo a los pasos que cada uno sigue durante el barrido de un árbol. El barrido **Preorden** (o Preorder) visita el nodo corriente *antes* que sus subárboles, el contenido de la raíz aparece antes que el contenido de los subárboles de los nodos hijos. El barrido **Inorden** (o Inorder) visita el nodo corriente *entre* sus subárboles: visita el subárbol izquierdo antes que el nodo corriente, luego este nodo y por último el subárbol derecho. El barrido **Postorden** (o Postorder) visita el nodo corriente *después* de visitar sus subárboles, el contenido de los subárboles do los nodos hijos aparece antes que el contenido de la raíz.

La siguiente tabla nos muestra la lógica de cada recorrido la cual está expresada, al igual que la definición de la estructura del árbol, en forma recursiva:

Preorden	*Inorden*	*Postorden*
* Visitar la raíz	* Aplicar Inorden al subárbol izquierdo	* Aplicar Postorden al subárbol izquierdo
* Aplicar Preorden al subárbol izquierdo	* Visitar la raíz	* Aplicar Postorden al subárbol derecho
* Aplicar Preorden al subárbol derecho	* Aplicar Inorden al subárbol derecho	* Visitar la raíz

Sabemos que existen diferentes maneras de notar una expresión algebraica, una de estas notaciones es la conocida como *Notación Polaca*, en honor a su descubridor el matemático polaco Jan Lukasiewicz. El orden en que los operandos y los operadores de una expresión son escritos determina cada una de las notaciones. Así cuando los operadores aparecen *antes* que sus operandos nos encontramos ante una expresión escrita en **forma prefijo**. Cuando los operadores aparecen *después* de los operandos la expresión se encuentra en **forma postfijo**, también llamada **forma sufijo**. Por último cuando la expresión aparece escrita siguiendo la convención usual de poner los operadores binarios *entre* sus operandos se dice que ésta está en **forma infijo**.

Si tomamos como ejemplo la expresión $a \times b$ notamos que está escrita en forma infijo, y pasa a forma prefijo si la escribimos $\times a\ b$ o a forma postfijo (sufijo) si se escribe $a\ b \times$. Si la expresión fuera más compleja: $a + b \times c$ y se quiere pasar a forma postfijo, como el producto debe resolverse primero se convierte primero, entonces se obtiene: $a\ + (b\ c \times)$ y luego llegamos a :' $a\ b\ c \times +$'.

Observar que si se tiene una expresión representada por su árbol de expresión se puede obtener cualquiera de sus representaciones en notación polaca por medio de alguno de los barridos del árbol. Así el barrido **preorden** de los nodos del árbol nos da la forma *prefijo* de la expresión. En el ejemplo de la Figura 1, el preorden del árbol es: '$\Rightarrow \wedge a\ b \vee d\ c$'.

Análogamente, el barrido **postorden** de los nodos de un árbol de expresión nos da la representación *postfijo* de la expresión. Así en el ejemplo de la Figura 1, la expresión postfijo que se obtiene del árbol es '$a\ b\ \wedge\ d\ c\ \vee\ \ \Rightarrow$'.

Finalmente, el barrido **inorden** de un árbol de expresión da la notación *infijo* en sí misma, pero sin paréntesis. En el ejemplo, el recorrido inorden del árbol anterior es '$(a \wedge b) \Rightarrow (d \vee c)$'. Sin embargo, se debería señalar que se ha omitido una complicación importante, los *paréntesis*. A veces, cuando se escribe una expresión en infijo, hay que usar *paréntesis* para mantener el orden de las operaciones, si éste es distinto al que establecen las reglas de precedencia. Así que un recorrido inorden *no es suficiente* para generar una expresión infija.

Sin embargo, con algunas mejoras, el árbol de expresión y los tres recorridos recursivos proporcionan una forma general de pasar expresiones de un formato a otro.

Una cuarta estrategia de recorrido de un árbol, que puede ser muy útil y aparece cómo bastante natural, es la visita de los nodos tal como ellos aparecen en la página, es decir de *arriba hacia abajo* y de *izquierda a derecha*. Este método se denomina **barrido por niveles** ya que todos los nodos del mismo nivel aparecen juntos y en orden. El parámetro nivel nos dice el nivel en el que estamos dentro del árbol. Por defecto, inicialmente es '0'. Cada vez que visitamos un hijo del nodo corriente, pasamos al *nivel corriente+1* porque el nivel del nodo hijo es siempre uno mayor que el del nodo padre. Este algoritmo no es naturalmente recursivo por lo que debe usarse una *cola* como estructura auxiliar para recordar los subárboles izquierdo y derecho de cada nodo. Cuando se visita un nodo se colocan sus hijos al final de la cola, donde serán visitados después de los nodos que ya están en la misma. Si se realiza el recorrido del árbol de la Figura 1 por niveles el resultado es el siguiente: '$\Rightarrow \wedge \vee\ a\ b\ d\ c$ '.

Este recorrido no tiene mucho sentido sobre una estructura de información como el árbol de expresión, porque salvo el hecho de listar todos los nodos del mismo, no representa nada desde el punto de vista algebraico. En cambio si resulta útil cuando se realiza sobre un árbol utilizado como estructura de almacenamiento, porque nos permite 'ver' claramente la relación entre los distintos nodos: padres, hijos, hermanos, al mismo tiempo que lista en forma sistemática todos los nodos de la estructura.

Del mismo modo, los recorridos preorden, inorden y postorden tienen poco sentido si se realizan sobre un ABB, que como ya dijimos es una estructura de almacenamiento, ya que los listados que éstos proveen no tienen ningún significado a menos que correspondan a alguna expresión. El único recorrido que puede resultar útil es el inorden, que devuelve los nodos de un ABB listados en *orden creciente*. Esta característica del barrido inorden permite obtener un método de ordenamiento llamado *Treesort* u *Ordenamiento por ABB*. Este método toma el conjunto de elementos a ordenar y:

- Construye un árbol binario de búsqueda a partir del conjunto.

- Realiza un barrido Inorden sobre el mismo.

Obteniéndose así el conjunto ordenado de menor a mayor.