Documentacion Árbol De Búsqueda Binaria

Nicolás García Santelices

19/04/2018

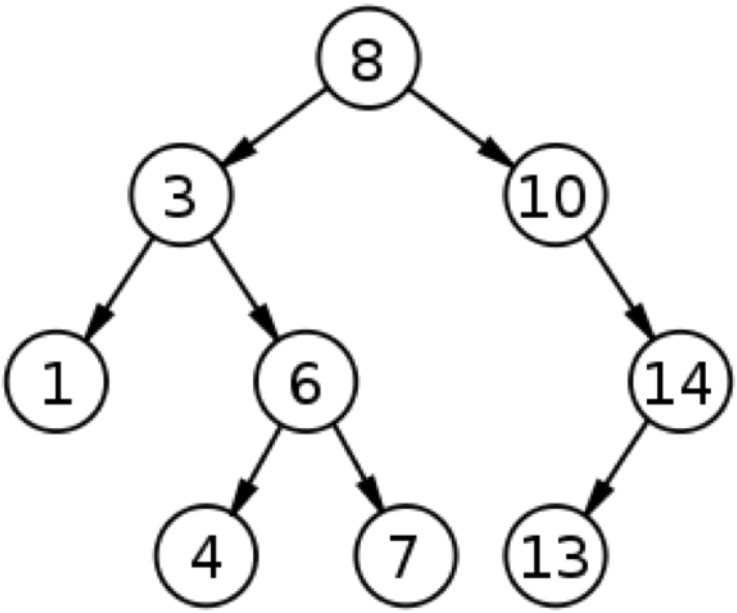
Índice

1. Analisis de un Árbol de Búsqueda Binaria (ABB)

- Definición: Un árbol de búsqueda binario ABB, es aquel que en cada nodo puede tener como mucho grado 2,es decir, un máximo de dos hijos. Los hijos suelen denominarse hijo izquierdo e hijo a la derecho, estableciéndose de esta forma un orden en el posicionamiento de los mismos Se puede resumir en que es un árbol binario que cumple que el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores. Un árbol perfectamente equilibrado tiene el mismo número de nodos en el subárbol izquierdo que en el subárbol derecho. En un árbol binario equilibrado, el peor desempeño de insertar un dato es O(log2n) , donde n es el número de nodos en el árbol. Para calcular la altura del arbol debemos calcular log2n, y del mismo modo esta formula representa el número máximo de comparaciones que insertar necesitará hacer mientras busca el lugar apropiado para insertar un nodo nuevo.

2. Características

- Un árbol de búsqueda binaria es un árbol binario que almacena en cada uno una llave o valor
- El valor de la raíz es menor que los valores almacenados en el lado derecho
- El valor de la raíz es mayor que todos los valores almacenados en el lado izquierdo del nodo



3. Tipos de Recorridos

- Se puede hacer un recorrido de un árbol en profundidad o en anchura. Los recorridos en anchura son por niveles, se realiza horizontalmente desde la raíz a todos los hijos antes de pasar a la descendencia de alguno de los hijos. El coste de recorrer el ABB es O(n), ya que se necesitan visitar todos los vértices. El recorrido en profundidad lleva al camino desde la raíz hacia el descendiente más lejano del primer hijo y luego continúa con el siguiente hijo. Como recorridos en profundidad tenemos inorden, preorden y postorden. Una propiedad de los ABB es que al hacer un recorrido en profundidad inorden obtenemos los elementos ordenados de forma ascendente.

- PreOrden: raíz-izquierda-derecha

- InOrden: izquierda-raíz-derecha

- PosOrden:izquierda-derecha-raíz

4. Operaciones básicas

- inserción: cuando se inserta un nuevo nodo en el árbol hay que tener en cuenta que cada nodo no puede tener más de dos hijos La inserción es similar a la búsqueda y se puede dar una solución tanto iterativa como recursiva. Si tenemos inicialmente como parámetro un árbol vacío se crea un nuevo nodo como único contenido el elemento a insertar. Si no lo está, se comprueba si el elemento dado es menor que la raíz del árbol inicial con lo que se inserta en el subárbol izquierdo y si es mayor se inserta en el subárbol derecho.

```
TREE-INSERT(T, z)
 1  y ← NIL
 2  x ← root[T]
 3  while x ≠ NIL
 4      do y ←  x
 5         if key[z] < key[x]

 6            then x ← left[x]
 7            else x ← right[x]
 8  p[z] ← y
 9  if y = NIL
10     then root[T] ← z              ✣ Tree T was empty
11     else if key[z] < key[y]
12             then left[y] ← z
13             else right[y] ← z
```

- búsqueda: el algoritmo comprara el elemento a buscar con la raíz, si es menor continua la búsqueda por la rama izquierda, si es mayor continua por la derecha, este procedimiento se realiza recursivamente hasta que encuentre el nodo o hasta que llegue al final del árbol Cabe destacar que la búsqueda en este tipo de árboles es muy eficiente, representa una función logarítmica. El máximo número de comparaciones que necesitaríamos para saber si un elemento se encuentra en un árbol binario de búsqueda estaría entre [log2(N+1)] y N, siendo N el número de nodos.

```
TREE-SEARCH (x, k)
1  if x= NIL or k = key[x]
2     then return x
3  if k < key[x]
4     then return TREE-SEARCH(left[x], k)
5     else return TREE-SEARCH(right[x], k)
```

- Borrar:La operación de borrado no es tan sencilla como las de búsqueda e inserción. Existen varios casos a tener en consideración:

  - Borrar un nodo sin hijos o nodo hoja: simplemente se borra y se establece a nulo el apuntador de su padre.
  - Borrar un nodo con un subárbol hijo: se borra el nodo y se asigna su subárbol hijo como subárbol de su padre.
  - Borrar un nodo con dos subárboles hijo: la solución está en reemplazar el valor del nodo por el de su predecesor o por el de su sucesor en inorden y posteriormente borrar este nodo. Su predecesor en inorden será el nodo más a la derecha de su subárbol izquierdo (mayor nodo del subarbol izquierdo), y su sucesor el nodo más a la izquierda de su subárbol derecho (menor nodo del subarbol derecho). En la siguiente figura se muestra cómo existe la posibilidad de realizar cualquiera de ambos reemplazos:

```
TREE-DELETE(T, z)
 1  if left[z] = NIL or right[z] = NIL
 2      then y ← z
 3      else y ← TREE-SUCCESSOR(z)
 4  if left[y] ≠ NIL
 5      then x ← left[y]
 6      else x ← right[y]
 7  if x ≠ NIL
 8      then p[x] ← p[y]
 9  if p[y] = NIL
10      then root[T] ← x
11      else if y = left[p[y]]
12              then left[p[y]] ← x
13              else right[p[y]] ← x
14  if y ≠ z
15      then key[z] ← key[y]
16           copy y's satellite data into z
17  return y
```

- Minimo y maximo: Para encontrar el valo minimo en un ABB se debe buscar en los hijos izquierdos hasta llegar al final del arbol,de modo contrario para encontrar un maximo se debe llegar al final del arbol por el lado derecho

```
TREE-MINIMUM (x)
1  while left[x] ≠ NIL
2       do x ← left[x]
3  return x

TREE-MAXIMUM(x)
1  while right[x] ≠ NIL
2       do x ← right[x]
3  return x
```

5. Aspectos técnicos del Desarrollo

- En primera instancia se pidio implementar un ABB en el lenguaje de programacion Java,en este ABB se debe implementar un CRUD(excepto update),e ingresar la cantidad de 10.000 datos y ver como el árbol se comporta.

1. Estructura de Nodo implementada

```
class nodoArbol {

        arbolBusquedaBinaria hd;
        arbolBusquedaBinaria hi;
        private int numero;

        public nodoArbol(){
            hd = null;
            hi = null;
            numero = 0;
        }
        public int getNumero() {
            return numero;
        }
        public void setNumero(int numero) {
            this.numero = numero;
        }

    }
```

2. Estructura de Árbol implementada

```
public class arbolBusquedaBinaria {
    private nodoArbol raiz;

    public arbolBusquedaBinaria() {
        @SuppressWarnings("unused")
        nodoArbol raiz = new nodoArbol();
    }
```

Esta estructurta fue planteada de este modo ya que solo se usaron enteros como 'clave' en los árboles utilizados , a modo de hacer mas simple la implementacion y uso de estos.

- Diagrama de Clase