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

Árboles Balanceados

Tarea 5

Alumna: Asunción Gómez
agomezcolomer@gmail.com
Profesor: Patricio Poblete

Algoritmos y Estructura de Datos
CC3001
Fecha de entrega: 25 de junio de 2019
Santiago, Chile

Índice de Contenidos

Lista de Figuras

Lista de Códigos

1. Introducción

Los árboles son una estructura de datos que puede llegar a ser muy útil a la hora de ordenar objetos. Existen distintos tipos de árboles: ABB, B+, rojo-negro, AVL, Air, Splay, entre otros. En particular, en esta tarea se trabajará con árboles de búsqueda binaria, los que pueden ser muy útiles para insertar, buscar o eliminar algún elemento, ya que su costo es bajo (en algunos casos llegan a ser de orden log(n)), por lo que pueder ser muy eficientes.

El objetivo de esta tarea es reemplazar la implementación de árboles Air a Splay, modificando el código que fue entregado. Los árboles Splay, como se dijo anteriormente, son árboles de búsqueda binaria que ajustan su estructura cada vez que se inserta un nuevo elemento, llevándolo a la raíz del árbol. Para esto, se debió tranformar las rotaciones del árbol "Air", para que se usara la estrategia de "splaying", analizando todas las posibles rotaciones: zig, zigzag y zigzig junto a sus casos análogos que se describirán posteriormente.

Así fue como se logró implementar el árbol pedido mediante la función "insertar", que fue entregada y usando rotaciones simples y dobles cada vez que se insertara un nuevo nodo para dejarlo como raíz.

2. Diseño de la solución

Para comenzar, se vieron los distintos tipos de rotaciones que existen en un árbol Splay. Las principales se muestran a continuación, sin considerar su caso análogo.

1. Zig: el nodo es un hijo izquierdo de la raíz, es decir, no tiene abuelo.
2. Zig-zag: el nodo es un hijo izquierdo de un hijo derecho.
3. Zig-zig: el nodo es un hijo izquierdo de un hijo izquierdo.

Esta serie de rotaciones se debe hacer hasta que el elemento a insertar quede como hijo directo de la raíz (y ahí hacer la última rotación simple) o como raíz. Para implementar el árbol Splay, se decidió hacer la inserción y búsqueda bottom-up, en la cual se hace un recorrido desde la raíz hasta encontrar el nodo o donde se debiese encontrar. Luego, se aplican la operaciones splay, es decir, se van efectuado las rotaciones a medida que se asciende hasta que el nuevo elemento quede como raíz.

Como las rotaciones zigzag y zagzig corresponden a las rotaciones dobles de un árbol AVL, bastó con usar el mismo código y aplicarlas recursivamente en la función "insertar". Para las rotaciones zigzig y zagzag se usó el de las rotaciones simples, pero aplicada primero al abuelo y luego al padre. Por último, para zig y zag, simplemente se usó el mismo código de las rotaciones simples en un AVL.

El algoritmo usa las funciones entregada "RotSimple" y "RotDoble" y además, modifica la de "insertar". La primera, se encarga de hacer una rotación simple entre el padre y su hijo. La segunda, hace una rotación doble entre el abuelo y el "nieto". Y la última, hace una búsqueda infructuosa al igual que en un ABB, para luego insertar el nuevo elemento. Posteriormente, ejecuta las rotaciones necesarias hasta que el nuevo elemento quede en la raíz.

Uno de los casos bordes que se consideraron fue cuando el árbol es vacío y se insertaba un nuevo elemento. Para cubrirlo, la función "insertar" crea un nodo de hijos nulos que tiene como llave el

nuevo elemento. Otro caso borde es cuando el número a insertar ya se encuentra en el árbol. En este caso, simplemente se ignora la inserción, y así, el árbol nunca tendrá elementos repetidos.

Cabe destacar, que los números a insertar son del tipo double y solo se encuentran en un rango, el cual es dado y se visualizan con una regla (ver figura 1).

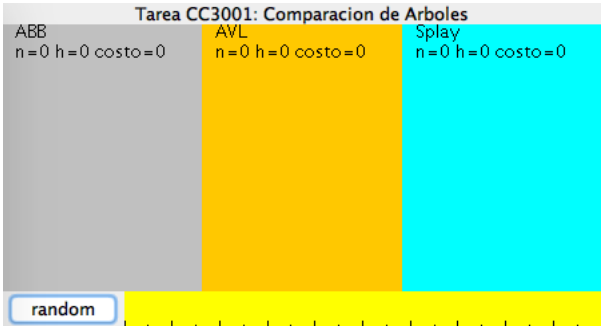

Figura 1: Visualización de los árboles

3. Implementación

Primero que todo, es importante recalcar que la mayoría de los archivos entregados fueron usados para el diseño de la clase Splay. Uno de los más relevantes es el NodoInt.java que se muestra en el "Anexo" (código 5), donde se puede ver que la clase tiene una llave del tipo double y dos Nodos como hijos.

También, como las rotaciones zig y zag corresponden a las rotaciones simples en un árbol AVL, simplemente se usó la función "RotSimple" (uno de los casos se muestra en el código 1) que deja al hijo como padre, y el padre pasa a ser su hijo derecho. Para la rotación zag es análogo. Y, como ya se mencionó, para las rotaciones zigzag y zagzig se utilizó el código de las rotaciones doble de un árbol AVL.

Código 1: Rotación Simple ZIG

```
1 //rotacion hacia la izquierda x-->p (hijo izquierdo del padre) ZIG
2 if (sentido >= 1) {
3     NodoInt pp = new NodoInt(((NodoInt) p.izq).der, p.info, p.der);
4     return new NodoInt(((NodoInt) p.izq).izq, ((NodoInt) p.izq).info, pp);
5 }
```

Tras analizar el archivo de la clase Air, es fácil darse cuenta que la única diferencia con los Splay es el tipo de rotación. En el código 2 se visualiza la rotación que hace este tipo de árbol si el valor a insertar es menor que llave del árbol. Así, es claro ver que la única rotación distinta entre estos árboles es la de zigzig (zagzag), que es única para los Splay. Esta es una rotación simple que se ejecuta sobre el abuelo del nodo y luego sobre el padre, por lo que de alguna manera se debe tener referencia a estos dos.

Código 2: Air: insertar

```
1 //si el valor a insertar es menor que la llave del nodo a
2 if( x<b.info ) {
```

```
   //se interta en el subarbol izquierdo
   NodoInt r = new NodoInt(insertar(b.izq,x),b.info,b.der);
   //puntero interno al subarbol izquierdo
   NodoInt i = (NodoInt)r.izq;
   //retorna un nuevo nodo
   return new NodoInt(i.izq,i.info,new NodoInt(i.der,r.info,r.der));
}
```

Además, se tuvo que modificar la función de "insertar" para que hiciera las rotaciones necesarias a medida que iba crecienco el árbol, ya que, como se dijo anteriormente, la función es recursiva. En el código 3 se muestra el caso en que el nuevo elemento "x" se encuentre en el subárbol izquierdo. Ahí es posible observar cómo hace una búsqueda infructuosa, para luego insertar el nodo y comenzar con las rotaciones dependiendo del caso en que se esté. Es importante ver la forma en que se tuvo referencia al padre y abuelo del nuevo nodo para lograr hacer las rotaciones zigzig y zagzig.

Código 3: Splay: insertar

```
NodoInt b = (NodoInt)a; //a es el arbol donde se insertara el nuevo valor x
//si el valor a insertar es menor que la llave del nodo a
if( x<b.info )
    //búsqueda ABB
    NodoInt retorno =  new NodoInt(insertar(b.izq,x),b.info,b.der);
    if(b.izq instanceof NodoExt){return retorno;}
    //zigzig <--> rotacion c/r al abuelo y luego c/r al padre
    else if(x<((NodoInt)b.izq).info){return RotSimple(RotSimple(retorno,1),1);}
    //zigzag <--> rotacion doble
    else if(x>((NodoInt)b.izq).info){return RotDoble(retorno,-1);}
    //zig
    else{// if(x==((NodoInt)b.izq).info){return RotSimple(retorno,1);}
}
```

Finalmente, para que se visualizara el árbol Splay en la ventana se modificó el archivo de árboles, lo que se observa en el código 4.

Código 4: Arboles: insertall

```
public static void insertall(double llave)
  {
   a1.raiz = Abb.insertar(a1.raiz,llave);
   a2.raiz = Avl.insertar(a2.raiz,llave);
   //se cambia arbol Air por Splay
   a3.raiz = Splay.insertar(a3.raiz,llave);
  }
```

4. Resultado y Conclusiones

Se puede concluir que se lograron la mayoría de los objetivos propuestos, ya que se implementó satisfactoriamente las rotaciones zigzig y zigzag, pero para algunos casos las rotaciones simples (zig y zag) no se ejecutan muy bien.

En la figura 2 y 3, es claro ver que el árbol logra ejecutar la rotación zigzig, pero no así la

rotación simple (fijarse en el árbol Air y Splay para notarlo). Este problema no se logró solucionar, ya que si en la primera condición de la función "insertar" (código 3) se ejecutaba la rotación simple, nunca ejecutaría las otras rotaciones y quedaría como un árbol Air o peor aún, solo funcionaría si se entregaban números en orden (descendiente o ascendente).

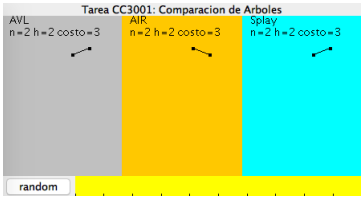

Figura 2: Resultado de un árbol Splay (n=2)

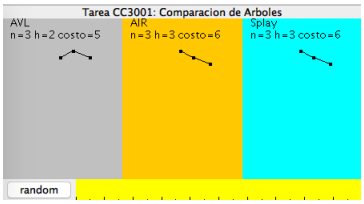

Figura 3: Resultado de un árbol Splay (n=3)

Al comparar los árboles AVL con Splay, se logra verificar que estos últimos son más "fáciles" de implementar, ya que no deben cumplir ninguna condición de balance y las rotaciones son más simples. Por lo mismo, gastan menos memoria, ya que no deben guardar la altura de cada nodo. Además, garantizan que ninguna secuencia de operaciones sea mala (todos los costos amortizados son de orden log(n)). Los AVL, por el otro lado, aseguran un costo de orden log(n) para todas sus operaciones, por lo que se puede concluir que es menos costoso. Esto se puede visualizar en la figura 4 y 5 en donde es claro ver que los AVL son los menos costosos en comparación a los ABB, Air y Splay, es decir, es más fácil accesar a los elementos en este tipo de árbol. Por consiguente, es posible concluir que los árboles auto-balanceables (diferencia de la altura de todos los sub-árboles izquierdo y derecho es a lo más 1), ayuda a mantener aún más ordenado el árbol, conservando una altura parecida en el subárbol izquierdo y derecho y así generar un acceso más fácil a todos sus nodos.

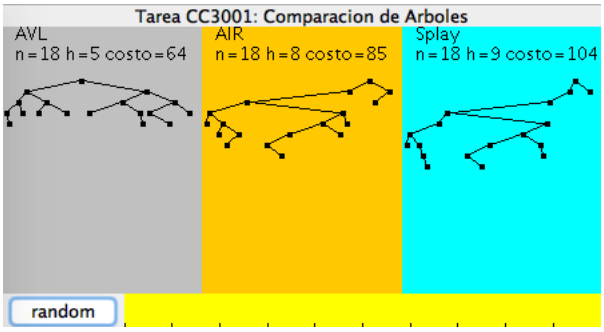

Figura 4: Comparación de árboles AVL, Air y Splay (n=18)

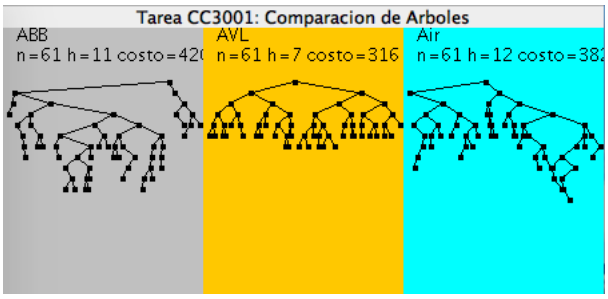

Figura 5: Costo de árboles ABB, AVL y Air

Si comparamos los árboles ABB, AVL y Splay se puede observar que estos últimos pueden llegar a ser incluso más costosos que los ABB (ver figura 6). Sin embargo, esto varía para el número de nodos existentes en el árbol (para otros casos el Splay era menos costoso).

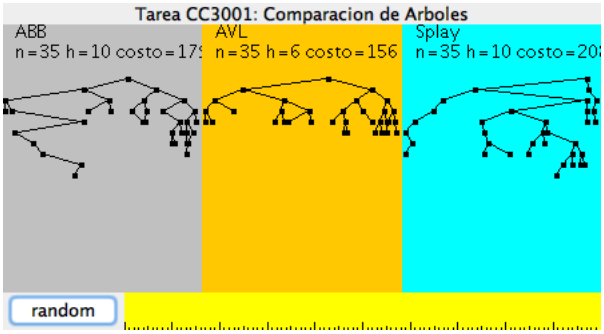

Figura 6: Costo de árboles ABB, AVL y Splay

Si se comparan los árboles Air con los Splay, en teoría los primeros son menos eficientes, ya que para algunas de inserciones dejan la búsqueda en O(n), a diferencia de los Splay, ya que sus costos amortizados son de O(log(n)). Sin embargo, como se puede ver en las figuras ya mencionadas, el costo suele ser mayor en un Splay, aunque posiblemente sea porque el algoritmo no se logró al 100 %. En cuanto a la implementación, la única diferencia es la rotación zigzig (zagzag), que hace más difícil implementar un árbol Splay que un Air, ya que se necesitan las referencias al padre y abuelo.

Una de las dificultades que se tuvo fue comprender el código y lograr hacerle cambios, ya que es una forma diferente de hacer un programa (partirlo desde cero ha sido lo más usual). En particular, los archivos más difíciles de entender fueron los de la clase AVL (entender la recursión) y la rotación de los árboles Air.

A pesar de que no se logró implementar del todo el árbol Splay, funciona en general y se espera a futuro entender cuál fue el error y cómo era la manera correcta de diseñar el algoritmo.

5. Anexo

Código 5: Archivo de NodoInt

```
public class NodoInt extends Nodo{
   public double info; // llave
   public Nodo izq, der; // hijos
   int n; // numero de nodos
   int h; // altura
   int c; // costo de accesar todos los nodos del subarbol
 }
```

Código 6: Clase Splay

```
public class Splay {
    //rotacion simple ZIG o ZAG
    public static NodoInt RotSimple(NodoInt p, int sentido) {
        //rotacion hacia la izquierda x-->p (hijo izquierdo del padre) ZIG
        if (sentido >= 1) {
            NodoInt pp = new NodoInt(((NodoInt) p.izq).der, p.info, p.der);
            return new NodoInt(((NodoInt) p.izq).izq, ((NodoInt) p.izq).info, pp);
        }
        //rotacion hacia la derecha p<--x (hijo derecho del padre) ZAG
        else {
            NodoInt pp = new NodoInt(p.izq, p.info, ((NodoInt) p.der).izq);
            return new NodoInt(pp, ((NodoInt) p.der).info, ((NodoInt) p.der).der);
        }
    }
    //zigzag o zagzig en arbol SPLAY
    public static NodoInt RotDoble (NodoInt p, int sentido)
    {
        if (sentido <= -1)
        {
            NodoInt izquierda = new NodoInt( ((NodoInt) p.izq).izq,
                    ((NodoInt) p.izq).info, ((NodoInt) ((NodoInt) p.izq).der).izq);
            NodoInt pp = new NodoInt( ((NodoInt) ((NodoInt) p.izq).der).der,
                    p.info, p.der );
            return new NodoInt( izquierda, ((NodoInt) ((NodoInt) p.izq).der).info, pp);
        }
        else
        {
            NodoInt derecha = new NodoInt( ((NodoInt) ((NodoInt) p.der).izq).der,
                    ((NodoInt) p.der).info,   ((NodoInt) p.der).der );
            NodoInt pp = new NodoInt( p.izq,
                    p.info, ((NodoInt) ((NodoInt) p.der).izq).izq );
            return new NodoInt( pp, ((NodoInt) ((NodoInt) p.der).izq).info, derecha);
        }
    }


    //inserta el nodo y ejecuta la estrategia splaying
```

```
38      public static Nodo insertar(Nodo a, double x) {
39          if( a instanceof NodoExt ) {return new NodoInt( Nodo.nulo, x, Nodo.nulo );}
40          //puntero al nodo interno a
41          NodoInt b = (NodoInt)a;
42          //si el valor a insertar es menor que la llave del nodo a
43          if( x<b.info ) {
44              NodoInt retorno =  new NodoInt(insertar(b.izq,x),b.info,b.der);
45              //
46              if(b.izq instanceof NodoExt){
47                  return retorno;
48              }
49              //zigzigs
50              else if(x<((NodoInt)b.izq).info){
51                  //rotacion c/r al abuelo y luego c/r al padre
52                  return RotSimple(RotSimple(retorno,1),1);
53              }
54              //zigzag
55              else if(x>((NodoInt)b.izq).info){
56                  return RotDoble(retorno,-1);
57              }
58              else{// if(x==((NodoInt)b.izq).info){
59                  return RotSimple(retorno,1);
60              }
61          }
62          else if( x>b.info ) {
63              NodoInt retorno = new NodoInt(b.izq,b.info,insertar(b.der,x));
64              if(b.der instanceof NodoExt){
65                  return retorno;
66              }
67              else if(x>((NodoInt)b.der).info){
68                  return RotSimple(RotSimple(retorno,-1),-1);
69              }
70
71              else if(x<((NodoInt)b.der).info){
72                  return RotDoble(retorno,1);
73              }
74              else{// if(x==((NodoInt)b.der).info){
75                  return RotSimple(retorno,-1);
76              }
77          }
78          else {
79              return b; // ignoramos insercion si es llave repetida
80          }
81  }
82  }
83  }
84  }
```