Práctica 6 - Árboles Binarios de Búsqueda Aumentados

Estructuras de Datos

Diciembre 2017

Un **árbol binario de búsqueda aumentado** es un árbol binario de búsqueda en el que en cada nodo se almacenan su clave y su peso (similar a como ocurre en los montículos zurdos o los maxifóbicos).

Por ejemplo, la siguiente figura muestra un árbol de búsqueda aumentado, donde las claves aparecen en rojo y los pesos en azul:

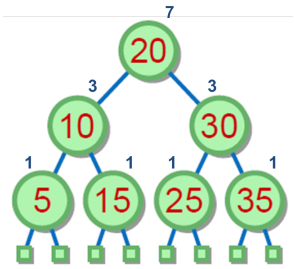

Figura 1: Árbol de Búsqueda Aumentado.

Implementación en Java

La clase java `AugmentedBST` implementa árboles binarios de búsqueda aumentados. Esta clase utiliza una clase interna `Tree` con la siguiente definición:

```
private static class Tree<E> {
    E key;          // key stored in node
    int weight;     // weight of node (number of keys in tree)
    Tree<E> left;  // left child
    Tree<E> right; // right child

    public Tree(E k) {
        key = k;
        weight = 1;
```

```
        left = null;
        right = null;
    }
}
```

En el fichero `AugmentedBST.java` se facilitan las implementaciones de varios métodos básicos sobre BST aumentados, incluyendo los métodos de búsqueda, inserción y borrado (similares a los de los BST vistos en clase).

El árbol de la figura anterior se puede construir con el siguiente código:

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

   AugmentedBST<Integer> bst = new AugmentedBST<Integer>();
   int xs[] = new int[] {20,10,30,5,15,25,35};

   for(int x : xs) {
      bst.insert(x);
   }
   System.out.println(bst);
}
```

El objetivo de la práctica es completar definiciones **eficientes** de los siguientes métodos auxiliares (aparecen al final de `AugmentedBST.java`):

- `T select(int i)` devuelve la i-ésima clave más pequeña del BST. Las claves se numeran a partir de 0. Por ejemplo, para el árbol de la figura tenemos que:

  ```
  bst.select(0) = 5
  bst.select(1) = 10
  bst.select(4) = 25
  bst.select(6) = 35
  bst.select(10) = null
  ```

  Para obtener una implementación **eficiente** de `select` debes aprovechar que la estructura almacena los pesos de los árboles.

  Para el árbol de la figura, `select(3)` debe devolver la cuarta clave más pequeña del árbol. Como la raíz `20` tiene exactamente tres claves a su izquierda, sabemos que el resultado es precisamente la propia raíz `20`, pues es la cuarta clave más pequeña del árbol. Por otro lado, `select(4)` tiene que devolver la quinta clave más pequeña del árbol. Como la raíz tiene tres claves a su izquierda, sabemos que la quinta clave más pequeña debe aparecer en su hijo derecho (y por tanto no es necesario explorar el hijo izquierdo).

- `T floor(T k)` devuelve la clave más grande del BST que es menor o igual que `k`. Por ejemplo, para el árbol de la figura tenemos que:

  ```
  bst.floor(4) = null
  bst.floor(5) = 5
  ```

```
bst.floor(6) = 5
bst.floor(9) = 5
bst.floor(10) = 10
bst.floor(27) = 25
bst.floor(50) = 35
```

- `T ceiling(T k)` devuelve la clave más pequeña del BST que es mayor o igual que `k`. Para el árbol de la figura tenemos:

```
bst.ceiling(4) = 5
bst.ceiling(5) = 5
bst.ceiling(6) = 10
bst.ceiling(9) = 10
bst.ceiling(10) = 10
bst.ceiling(27) = 30
bst.ceiling(50) = null
```

- `int rank(T k)` devuelve el número de claves del BST que es menor que `k`. Para el árbol del ejemplo tenemos:

```
bst.rank(4) = 0
bst.rank(5) = 0
bst.rank(6) = 1
bst.rank(9) = 1
bst.rank(10) = 1
bst.rank(27) = 5
bst.rank(50) = 7
```

- `int size(T low, T high)` devuelve el número de claves del BST que pertenecen al intervalo cerrado `[low, high]`.

```
bst.size(1,4) = 0
bst.size(1,5) = 1
bst.size(5,5) = 1
bst.size(5,6) = 1
bst.size(1,9) = 1
bst.size(1,10) = 2
bst.size(1,27) = 5
bst.size(10,30) = 5
bst.size(40,50) = 0
bst.size(5,35) = 7
bst.size(35,5) = 0
```

En general, para obtener soluciones eficientes de los anteriores métodos debes aprovechar dos propiedades:

1. Los árboles son BST y por tanto satisfacen el **invariante de orden**: para cada nodo, las claves de su hijo izquierdo son menores y las claves de su hijo derecho son mayores.
2. Los árboles almacenan el peso en sus nodos, lo que puede evitar la exploración de alguno de sus hijos (como en el caso de `select`).