Exámen 2012, preguntas y respuestas - Resolucion del Segundo Parcial

Estructuras de datos y algoritmos (Universitat Politecnica de Valencia)

Resolución del Segundo Parcial de EDA (7/7/2012)

1.- En la clase `ABB`, diseña un método `numNodos2Hijos` que devuelva el número de nodos con dos hijos que hay en un Árbol Binario de Búsqueda. **(2 puntos)**

```
public int numNodos2Hijos(){ return numNodos2Hijos(this.raiz); }
protected int numNodos2Hijos(NodoABB<E> actual){
   int res = 0;
   if ( actual!=null )
     if ( actual.izq!=null && actual.der!=null )
          res += 1 + numNodos2Hijos(actual.izq)+numNodos2Hijos(actual.der);
     else res +=     numNodos2Hijos(actual.izq)+numNodos2Hijos(actual.der);
   return res;
}
```

```
public class ABB <E extends Comparable <E>> {
   protected NodoABB<E> raiz;
   protected int talla;
   public ABB(){ raiz = null; talla = 0;}
   //Resto de métodos de la clase
   …
}
class NodoABB<E> {
   E dato;
   NodoABB<E> izq, der;
   public NodoABB(E e){ dato = e; izq = null; der = null; }
   public NodoABB(E e, NodoABB<E> izq, NodoABB<E> der){
         dato = e; this.izq = izq; this.der = der;
    }
}
```

2.- Para representar un ABB de palabras se ha definido la clase `ABBDeString`, una derivada de `ABB` cuyos elementos son de tipo `String`:

```
public class ABBDeString extends ABB<String> { … }
```

Diseña en esta clase un método que, con el menor coste temporal posible, devuelva el número de palabras de un ABB que empiecen por una letra dada (representada mediante un `String` de longitud 1): `public int numPalabrasCon(String letraPrefijo)` **(2 puntos)**

Nota: es recomendable que uses el método `boolean startsWith(String prefijo)` de la clase `String` para comprobar si una palabra del ABB empieza por `letraPrefijo`.

```
public int numPalabrasCon(String letraPrefijo){
   return numPalabrasCon(letraPrefijo, this.raiz);
}
protected int numPalabrasCon(String lP, NodoABB<String> n){
   if ( n==null ) return 0;
   else{
      if ( n.dato.startsWith(lP) )
       return 1 + numPalabrasCon(l, n.izq) + numPalabrasCon(lP, n.der);
      else if ( n.dato.compareTo(lP)>0 ) return numPalabrasCon(lP, n.izq);
           else                          return numPalabrasCon(lP, n.der);
}
```

3.- Diseña un método recursivo que devuelva el número de elementos menores que uno dado `e` contenidos en un Montículo Binario Minimal (*Min-Heap*). Este método debe aprovechar la propiedad de ordenación del *Min-Heap* para que su coste temporal sea el menor posible. Indica también su coste asintótico en el mejor y el peor caso. **(3 puntos)**

```
public int numMenoresQue(E e){
  return numMenoresQue(e, 1);
}
protected int numMenoresQue(E e, int actual){
  if ( actual>talla ) return 0;
  else{  int resC = elArray[actual].compareTo(e);
         if ( resC<0 )
     return 1+numMenoresQue(e, 2*actual)+numMenoresQue(e, 2*actual+1);
         else return 0;
   }
}
```

Análisis del coste Temporal Asintótico: sea x=`talla`, el tamaño del problema.

En el Mejor Caso, cuando el dato que ocupa la Raíz del *Min-Heap* ya es mayor o igual que `e`, se tiene que $T^{M}_{numMenoresQue}(x) \in \Theta(1)$. Por tanto, $\mathbf{T_{numMenoresQue}(x) \in \Omega(1)}$.

En el Peor Caso, cuando los x=`talla` datos del *Min-Heap* son menores que `e`, se tiene que $T^{P}_{numMenoresQue}(x) \in \Theta(x)$. Por tanto, $\mathbf{T_{numMenoresQue}(x) \in O(x)}$.

4.- Recordando que en una Tabla Hash Enlazada si dos Entradas están en la misma cubeta es porque han colisionado, diseña un método consultor en la clase `TablaHash` que devuelva el número total de colisiones que se han producido tras insertar un cierto número de Entradas distintas, con claves distintas. **(1 punto)**

```
public int numColisiones(){
   int nTotalColisiones = 0;
   for ( int i=0; i<elArray.length; i++ ){
       int nColCubetaI = 0;
       for ( EntradaHash<C, V> e = elArray[i]; e!=null; e = e.siguiente )
                 nColCubetaI++;
      if ( nColCubetaI>0 ) nTotalColisiones += nColCubetaI-1;
   }
   return nTotalColisiones;
}
```

5.- En un sistema de información se dispone de dos `Map<Clave, Valor>` `m1` y `m2` implementados con sendas Tablas Hash. Como resultado de la actualización de dicho sistema se desea obtener un tercer `Map<Clave, Valor>` que sea la diferencia de `m1` y `m2`, es decir que contenga solo aquellas Entradas de `m1` que no estén también en `m2`. Por ejemplo, si las Entradas de `m1` son `{("uno", 1) ("dos", 2) ("tres", 3) ("cuatro", 4) ("seis", 6)}` y las de `m2` son `{("tres", 3) ("cuatro", 4) ("siete", 7) ("ocho", 8)}`, las Entradas del `Map` diferencia serán `{("uno", 1), ("dos", 2),("seis", 6)}`.

Diseña un método (estático) `diferencia` que devuelva el `Map` diferencia de dos `Map` `m1` y `m2` dados; ten en cuenta que para ello solo podrás utilizar los métodos definidos en la interfaz `Map` y el método constructor de `TablaHash`, que figuran al final de esta hoja. **(2 puntos)**

Nota: para simplificar, asume que si una Entrada de `m1` tiene la misma clave que otra de `m2` entonces los valores de ambas Entradas serán también iguales.

```
public static Map<Clave,Valor> diferencia(Map<Clave,Valor> m1, Map<Clave,Valor> m2){
   Map<Clave,Valor> res = new TablaHash<Clave,Valor>(m1.talla());
   Object[] clavesM1 = m1.claves();
   for ( int i=0; i<clavesM1.length; i++ ){
     Clave c = (Clave)clavesM1[i];
     Valor v = m2.recuperar(c);
     if ( v==null ) res.insertar(c, m1.recuperar(c));
   }
   return res;
}
```

```
public interface Map<C, V> {

  /** inserta la Entrada (c,v) en un Map y devuelve null; si ya
   *  existe una Entrada de Clave c en el Map, devuelve su valor
   *  asociado y lo substituye por v (o actualiza la Entrada)*/
  V insertar(C c, V v);

  /** elimina la Entrada con Clave c de un Map y devuelve su
   *  valor asociado, null si no existe una Entrada con dicha clave*/
  V eliminar(C c);

  /** devuelve el valor asociado a la clave c en un Map,
   *  null si no existe una Entrada con dicha clave*/
  V recuperar(C c);

  /** comprueba si un Map está vacío */
  boolean esVacio();

  /** devuelve la talla, o número de Entradas, de un Map */
  int talla();

  /** devuelve un array que contiene las talla() Claves de un Map */
  C[] claves();
}
```

Método constructor de la clase `TablaHash<C, V>`:

```
public TablaHash(int tallaMaximaEstimada){…}
```