Resolución del Primer Parcial de EDA (7 de Abril de 2017)

1.- Se quiere ampliar la Jerarquía `ListaConPI` para incluir el método `sucesor` en su funcionalidad. Para ello:

a) En el paquete `modelos`, se define la subinterfaz `LPIComparable` reutilizando vía Herencia la interfaz `ListaConPI`. Completa su cabecera. **(0. 5 puntos)**

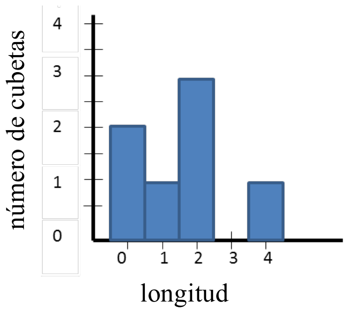
```
public interface LPIComparable <E extends Comparable<E>> extends ListaConPI<E> {
    /** SII !esvacia(): devuelve la posición del sucesor de e o -1 si no está */
    public int sucesor(E e);
}
```

b) En el paquete `lineales`, se define la clase `LEGLPIComparable` que implementa `LPIComparable` reutilizando vía Herencia `LEGListaConPI`, la clase que implementa `ListaConPI`. Escribe su cabecera. **(0.5 puntos)**

```
public class LEGLPIComparable<E extends Comparable<E>>
    extends LEGListaConPI<E>
    implements LPIComparable<E> { … }
```

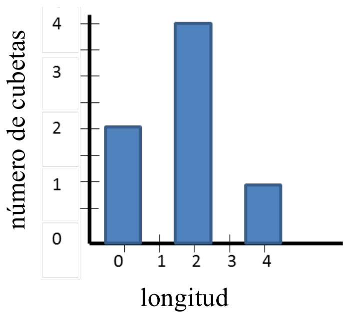
2.- Sea el siguiente el histograma de ocupación de una Tabla Hash:

a) Indica el número de elementos y el de cubetas que tiene la Tabla. **(0.5 puntos)**



11 elementos y 7 cubetas.

b) Indica en qué cubeta de la Tabla se debe insertar un nuevo elemento para que su histograma de ocupación pase a ser el siguiente: **(0.5 puntos)**



Se debe insertar en la (única) cubeta de longitud 1; así, el nº de cubetas de longitud 2 de la Tabla pasará a ser 4 y el de cubetas de longitud 1 pasará a ser cero.

3.- Diseña en la clase `ABB` el método `addMinimo` para añadir un dato `e` menor que todos los del `ABB`. No puedes utilizar el método `insertar` de la clase `ABB`. **(2 puntos)**

```
public void addMinimo(E e) {
    if (raiz == null) { raiz = new NodoABB<E>(e); }
    else { this.raiz = addMinimo(e, raiz); }
}
//SII actual != null: devuelve el Nodo resultado de insertar e en actual
protected NodoABB<E> addMinimo(E e, NodoABB<E> actual) {
    NodoABB<E> res = actual;
    if (actual.izq == null) { res.izq = new NodoABB<E>(e); }
    else {
        res.izq = addMinimo(e, actual.izq);
        res.talla++;
    }
    return res;
}
Una solución iterativa equivalente sería como sigue:
public void addMinimo(E e) {
    NodoABB<E> nuevo = new NodoABB<E>(e);
    if (raiz == null) { raiz = nuevo; }
    else {
        NodoABB<E> actual = raiz;
        while (actual.izq != null) {
            actual.talla++;
            actual = actual.izq;
        }
        actual.izq = nuevo;
    }
}
```

Suponiendo que el ABB está equilibrado, indica la talla del problema, x, y el coste Temporal del método `addMinimo` que has diseñado, $T_{addMinimo}(x)$, utilizando la notación asintótica (O y Ω o bien Θ). **(0.5 puntos)**

Talla del problema x = número de nodos del ABB, o talla de su nodo Raíz.

Coste Temporal Asintótico: el método realiza un Recorrido del camino que une la Raíz del ABB con su nodo más a la izquierda; como el ABB está equilibrado, este camino tiene una longitud del orden de log x y, por tanto, $T_{addMinimo}(x) \in \Theta(\log x)$

4.- La Dirección General de Tráfico tiene en un `Map dgt` la información de cada coche, concretamente su matrícula (clave) y el año de matriculación (valor). Se pide diseñar un método estático `cochesPorAnyo` que dado el `Map dgt` devuelva un `Map` que tenga, para cada año, el número de coches matriculados. **(2 puntos)**

```
public static Map<Integer, Integer> cochesPorAnyo(Map <String, Integer> dgt) {
    Map<Integer, Integer> res = new TablaHash<Integer, Integer>(dgt.talla());
    ListaConPI <String> matriculas = dgt.claves();
    for (matriculas.inicio(); !matriculas.esFin(); matriculas.siguiente()) {
        String matricula = matriculas.recuperar();
        Integer anyo = dgt.recuperar(matricula);
        Integer cont = res.recuperar(anyo);
        if (cont == null) { res.insertar(anyo, 1); }
        else { res.insertar(anyo, ++cont); }
    }
    return res;
}
```

Indica la talla del problema, x, y el coste Temporal del método `cochesPorAnyo` que has diseñado, $T_{cochesPorAnyo}(x)$, utilizando la notación asintótica (O y Ω o bien Θ). **(0.5 puntos)**

Talla del problema x = `dgt.talla()`.

Coste temporal asintótico: $T_{cochesPorAnyo}(x) \in \Theta(x)$.

5.- Sea **v** un array de enteros, de longitud mayor que 0, que contiene una secuencia de valores estrictamente creciente hasta un cierto índice **p** y estrictamente decreciente desde **p** hasta el final. Diseña un método Divide y Vencerás para encontrar este índice **p** con el mejor coste posible. **(2 puntos)**

Por ejemplo, el resultado **p** sería 5 para el siguiente array:

30	40	80	100	110	160	50	10	4	2
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

```
public static int posicion(int[] v) {
    return posicion(v, 0, v.length - 1);
}
private static int posicion(int[] v, int i, int f) {
    // Búsqueda con garantía de éxito
    if (i == f) { return i; } // subarray con 1 elemento
    else if (i + 1 == f) { // subarray con 2 elementos
        if (v[i] > v[f]) { return i; }
        else { return f; }
    }
    else {
        // subarray con, mínimo, 3 elementos; de ellos, o el central cumple la
        // condición, o forma parte de la secuencia creciente o de la decreciente
        int m = (i + f) / 2;
        if (v[m - 1] < v[m]) {
            if (v[m] > v[m + 1]) { return m; } // v[m] cumple la condición
            else { return posicion(v, m + 1, f); } // v[m] en secuencia creciente
        }
        else { return posicion(v, i, m - 1); } // v[m] en secuencia decreciente
    }
}
```

Estudia el coste Temporal del método (recursivo) que has diseñado. En concreto, ... **(1 punto)**

Indica la talla del problema, x, en función de los parámetros del método: x = `f - i + 1`.

Para una talla x dada, indica si existen instancias significativas; si las hubiera, indica cuáles son y por qué:

Mejor caso: `p` = `m`, la posición central del subarray `v[i, f]`.

Peor caso: `p` = `i` o `p` = `f`, respectivamente la primera posición o la última del subarray `v[i, f]`.

Escribe las Relaciones de Recurrencia que requiera tu respuesta en el punto anterior; luego, usa los Teoremas de Coste para resolverlas y acotarlas:

$T^{P}_{posicion}(x > 2) = 1 * T^{P}_{posicion}(x / 2) + k$.

Luego, por Teorema 3 (sobrecarga constante), con a = 1 y c = 2, $T^{P}_{posicion}(x > 2) \in \Theta(\log_2 x)$.

A partir de tu respuesta en el punto anterior, escribe el coste Temporal Asintótico del método, utilizando la notación asintótica (O y Ω o bien Θ):

$T_{posicion}(x) \in \Omega(1)$ y $T_{posicion}(x) \in O(\log_2 x)$.

ANEXO

La interfaz `ListaConPI` del paquete `modelos`.

```
public interface ListaConPI<E> {
      void insertar(E e);
      /** SII !esFin() */ void eliminar();
      void inicio();
      /** SII !esFin() */ void siguiente();
      void fin();
      /** SII !esFin() */ E recuperar();
      boolean esFin();
      boolean esVacia();
      int talla();
}
```

La interfaz Map del paquete `modelos`.

```
public interface Map<C, V> {
    V insertar(C c, V v);
    V eliminar(C c);
    V recuperar(C c);
    boolean esVacio();
    int talla();
    ListaConPI<C> claves();
}
```

Las clases `ABB` y `NodoABB` del paquete `jerarquicos`.

```
public class ABB<E extends Comparable<E>> {
    protected NodoABB<E> raiz;
    public ABB() { this.raiz = null; }
    …
}

class NodoABB<E> {
    protected E dato;
    protected NodoABB<E> izq, der;
    int talla;
    NodoABB(E e) {
        this.dato = e;
        this.izq = null; this.der = null;
        talla = 1;
    }
}
```

Teoremas de coste

Teorema 1: $f(x) = a\cdot f(x - c) + b$, con $b \geq 1$
- si $a=1$, $f(x) \in \Theta(x)$;
- si $a>1$, $f(x) \in \Theta(a^{x/c})$;

Teorema 2: $f(x) = a\cdot f(x - c) + b\cdot x + d$, con b y $d\geq 1$
- si $a=1$, $f(x) \in \Theta(x^2)$;
- si $a>1$, $f(x) \in \Theta(a^{x/c})$;

Teorema 3: $f(x) = a\cdot f(x/c) + b$, con $b \geq 1$
- si $a=1$, $f(x) \in \Theta(\log_c x)$;
- si $a>1$, $f(x) \in \Theta(x^{\log_c a})$;

Teorema 4: $f(x) = a\cdot f(x/c) + b\cdot x + d$, con b y $d\geq 1$
- si $a<c$, $f(x) \in \Theta(x)$;
- si $a=c$, $f(x) \in \Theta(x\cdot \log_c x)$;
- si $a>c$, $f(x) \in \Theta(x^{\log_c a})$;

Teoremas maestros

Teorema para recurrencia divisora: la solución a la ecuación $T(n) = a\cdot T(n/b) + \Theta(n^k)$, con $a\geq 1$ y $b>1$ es:
- $T(n) = O(n^{\log_b a})$ si $a>b^k$;
- $T(n) = O(n^k\cdot \log n)$ si $a=b^k$;
- $T(n) = O(n^k)$ si $a<b^k$;

Teorema para recurrencia sustractora: la solución a la ecuación $T(n) = a\cdot T(n-c) + \Theta(n^k)$ es:
- $T(n) = \Theta(n^k)$ si $a<1$;
- $T(n) = \Theta (n^{k+1})$ si $a=1$;
- $T(n) = \Theta (a^{n/c})$ si $a>1$;