CC30a - Algoritmos y Estructuras de Datos **Semestre Otoño, 2004**

Pauta Control 2

1. Pregunta 2: Hashtables

Se desea implementar tablas de hashing con encadenamiento (listas separadas), usando la función h(x) (que se supone dada).

<imaginarse el dibujo que habia en el control>

1. Escriba la definición de las clases para esta estructura de datos. Para el constructor de la tabla suponga que el valor de m es un parámetro. (2 puntos)

```
/* Nodos de las listas enlazadas (1 punto)*/
    public static class Nodo{
    protected int info;
    protected Nodo next;

    public Nodo(int info, Nodo next){
        this.info = info;
        this.next = next;
    }

/* Tabla de Hash con listas (1 punto)*/
    public static class Hash{
    protected Nodo[] tabla;
    protected int size;

    public Hash(int m){
        this.tabla = new Nodo[m];
        this.size = m;
    }
  }
```

2. Implemente un método de búsqueda para una llave x con las siguientes características: (4 puntos)

   - si el elemento no se encuentra, se debe agregar al comienzo de la lista respectiva.
   - si el elemento está en la tabla, se le debe mover al inicio de la lista respectiva ("move-to-front").
   - en ambos casos, se debe retornar una referencia al nodo que contiene a la llave x.

```
/* Supuesto (no se descontara puntaje por definir mal o no definir h(x)
 * mientras que la usen bien :P */
    public int h(int x){
        /* Definicion de h(x) ya dada */
    }
    public Nodo buscar(int x){
        int hashVal = h(x);
        Nodo toFront, aux;
        if(this.tabla[hashVal] == null)
          this.tabla[hashVal] = new Nodo(x,null);
        else{
          if(this.tabla[hashVal].info == x)
            return this.tabla[hashVal];
          else if((toFront = buscaBorra(this.tabla[hashVal], x)) != null){
            aux = this.tabla[hashVal];
            this.tabla[hashVal] = toFront;
            this.tabla[hashVal].next = aux;
            }
          else{
            this.tabla[hashVal] = new Nodo(x, this.tabla[hashVal]);
          }
         }
        return this.tabla[hashVal];
    }


        /* Metodo auxiliar, busca el nodo, si lo encuentra, lo borra
         * de la lista y lo retorna, sino, retorna null             */
        public Nodo buscaBorra(Nodo nodo, int x){
          Nodo tmp = nodo;
          Nodo out;
          Nodo anterior = null;
          if(tmp == null)
            return null;
            for(;;tmp = tmp.next){
              if(tmp.next == null)
                 return null;
              else if(tmp.next.info == x){
                 out = tmp.next;
                 tmp.next = out.next;
                 return out;
              }
             }

         }
```

Distribución del puntaje:

Búsqueda: 1 punto.

Borrar: 1 punto.

Insertar: 1,5 puntos.

Crear correctamente el nodo: 0,3.

Retornar Nodo: 0,2 .

Hay varias formas de implementar las tablas de Hash, la que se presenta aqui considera que dentro del arreglo de listas hayan valores, también se acepta que haya un Nodo sin valor en la primera posición de los casilleros de la tabla.

Se espera que como mínimo se utilize correctamente la característica de las tablas de hash de tener acceso en tiempo constante a las listas enlazadas gracias a la función de hash *h(x)*. Si se realizan busquedas secuenciales, se bajará 0.5 además de no tener puntaje para la búsqueda.

La idea de move-to-front es tomar el nodo a mitad de la lista y moverlo al principio de ésta, NO de intercambiar el valor del 1er elemento con el buscado, ya que asi se viola la optimización de que los elementos con mayor probabilidad van al comienzo de la lista enlazada.

En general, no se descontará por detalles, como fines de instrucción u otros, mientras que sea en lenguaje Java.

Si ciertas ovejas negras hacen pseudo-código, las penas del infierno recaeran sobre ellas, descontandoles entre 0.2 y 1.0 puntos según el grado de pseudo-codismo.