Listas Circulares

Una lista circular es aquella en la que el último nodo contiene una referencia al primero, y puede ser recorrida infinitamente.

Las listas circulares, que ya fueron mencionadas al comienzo de esta unidad, son aquellas en las que el último nodo contiene una referencia al primero. Pueden ser tanto simplemente como doblemente enlazadas.

En las listas lineales simples o en las dobles siempre hay un primer nodo (cabeza) y un último nodo (cola). Una lista circular, por propia naturaleza, no tiene ni principio ni fin. Sin embargo, resulta útil establecer un nodo a partir del cual se acceda a la lista y así poder acceder a sus nodos.

La Figura 8.14 muestra una lista circular con enlace simple; podría considerarse que es una lista lineal cuyo último nodo apunta al primero.

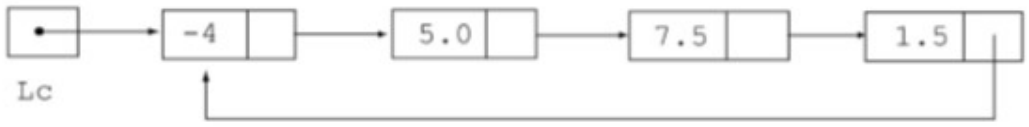

Figura 8.14 Lista circular

Las operaciones que se realizan sobre una lista circular son similares a las operaciones sobre listas lineales, teniendo en cuenta que no hay *primero* ni *último* nodo, aunque sí un nodo de acceso a la lista. Estas operaciones permiten construir el *TAD* `ListaCircular` y su funcionalidad es la siguiente:

- Inicialización o creación.
- Inserción de elementos en una lista circular.
- Eliminación de elementos de una lista circular.
- Búsqueda de elementos de una lista circular.
- Recorrido de cada uno de los nodos de una lista circular.
- Verificación de lista vacía.

La construcción de una lista circular se puede hacer con enlace simple o enlace doble. La implementación que se desarrolla en este apartado enlaza dos nodos con un enlace simple.

Se declara la clase `Nodo`, con el campo dato y enlace, y la clase `ListaCircular` con el puntero de acceso a la lista, junto a los métodos que implementan las operaciones. Los elementos de la lista pueden ser de cualquier tipo, se puede abstraer su tipo en otra clase, por ejemplo `Elemento`; con el fin de simplificar, se supone un tipo conocido.

El constructor de la clase `Nodo` varía respecto al de las listas no circulares, ya que el campo referencia `enlace`, en vez de quedar a `null`, se inicializa para que apunte al mismo nodo, de tal forma que queda como lista circular de un solo nodo.

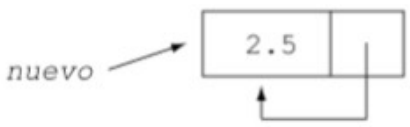

Figura 8.15 Creacción de un nodo de lista circular.

> ***A tener en cuenta***
>
> El pointer de acceso a una lista circular, lc, normalmente apunta al último nodo añadido a la estructura. Esta convención puede cambiar, ya que en una estructura circular no hay *primero* ni *último*.

8.10.1. Insertar un elemento en una lista circular

El algoritmo empleado para añadir o insertar un elemento en una lista circular varía dependiendo de la posición en que se desea insertar. La posición de inserción puede variar. Consideramos que se hace como nodo anterior al del nodo de acceso a la lista `lc`, y que `lc` tiene la dirección del último nodo insertado. A continuación, se declara la clase `ListaCircular` con el constructor *(lista vacía)* y el método de inserción.

```
package listaCircular;

public class ListaCircular
{
   private Nodo lc;

   public ListaCircular()
   {
      lc = null;
   }
   public ListaCircular insertar(Elemento entrada)
   {
     Nodo nuevo;
     nuevo = new Nodo(entrada);
     if (lc != null) // lista circular no vacía
     {
        nuevo.enlace = lc.enlace;
        lc.enlace = nuevo;
     }
      lc = nuevo;
      return this;
   }
   //...
```

8.10.2. Eliminar un elemento en una lista circular

La operación de eliminar un nodo de una lista circular sigue los mismos pasos que los dados para eliminar un nodo en una lista lineal. Hay que enlazar el nodo anterior con el nodo siguiente al que se desea eliminar y que el sistema libere la memoria que ocupa.

El algoritmo para eliminar un nodo de una lista circular es:

1. Búsqueda del nodo que contiene el dato.
2. Se enlaza el nodo anterior con el siguiente.
3. En caso de que el nodo a eliminar sea por el que se accede a la lista, `lc`, se modifica `lc` para que tenga la dirección del nodo anterior.
4. Por último, el sistema libera la memoria ocupada por el nodo al anular la referencia.

La implementación del método debe de tener en cuenta que la lista circular conste de un sólo nodo, ya que al eliminarlo la lista se queda vacía (`lc = null`). La condición de lista con un nodo se corresponde con la forma de inicializar (constructor) un nodo: `lc == lc.enlace`.

El método recorre la lista buscando el nodo con el dato a eliminar, utiliza un puntero al nodo *anterior* para que cuando se encuentre el nodo se enlace con el *siguiente*. Se accede al dato con la sentencia `actual.enlace.dato`; éste permite, si coincide con el dato a eliminar, tener en `actual` el nodo anterior. Después del bucle es necesario volver a preguntar por el campo `dato`, ya que no se comparó el nodo de acceso a la lista, `lc`, y el bucle puede terminar sin encontrar el nodo.

Código Java

```
public void eliminar(Elemento entrada)
{
   Nodo actual;
   boolean encontrado = false;
   //bucle de búsqueda
   actual = lc;
   while ((actual.enlace != lc) && (!encontrado))
   {
      encontrado = (actual.enlace.dato == entrada);
      if (!encontrado)
      {
         actual = actual.enlace;
      }
   }
   encontrado = (actual.enlace.dato == entrada);

   // Enlace de nodo anterior con el siguiente

   if (encontrado)
   {
      Nodo p;
      p = actual.enlace;    // Nodo a eliminar
      if (lc == lc.enlace)  // Lista con un solo nodo
         lc = null;
      else
      {
         if (p == lc)
         {
            lc = actual;  // Se borra el elemento referenciado por lc,
                          // el nuevo acceso a la lista es el anterior
         }
         actual.enlace = p.enlace;
      }
      p = null;
   }
}
```

8.10.3. Recorrer una lista circular

Una operación común a todas las estructuras enlazadas es recorrer o visitar todos los nodos de la estructura. En una lista circular, el recorrido puede empezar en cualquier nodo e ir procesando iterativamente cada nodo hasta alcanzar el de partida. El método (miembro de la clase `ListaCircular`) que se va escribir inicia el recorrido en el nodo siguiente al de acceso a la lista, `lc`, termina cuando alcanza el nodo `lc`. El proceso que se realiza con cada nodo consiste en escribir su contenido.

```
public void recorrer()
{
   Nodo p;
   if (lc != null)
   {
      p = lc.enlace; // siguiente nodo al de acceso
      do {
         System.out.println("\t" + p.dato);
         p = p.enlace;
      }while (p != lc.enlace);
   }
   else
      System.out.println("\t Lista Circular vacía.");
}
```

Ejercicio 8.4

Crear una lista circular con palabras leídas del teclado. El programa debe tener un conjunto de opciones:

- *Mostrar las cadenas que forman la lista.*
- *Borrar una palabra dada.*
- *Al terminar la ejecución, recorrer la lista eliminando los nodos.*

Para crear la lista circular, se utilizan las clases `Nodo` y `ListaCircular` del paquete `listaCircularPalabra` y la clase con el método `main()`. Al ser una lista de palabras, el campo dato del nodo es de tipo `String`.

El método `readLine()` es el apropiado para leer una cadena desde el teclado; cada llamada a `readLine()` crea un objeto cadena con la palabra leída, que a su vez será el nuevo dato de la lista circular. La inserción en la lista se hace llamando al método de `ListaCircular insertar()`.

Para borrar una palabra, se llama al método `eliminar()`; este busca el nodo que tiene la palabra, utiliza el método `equals()`, en lugar del operador ==, para determinar si coincide la palabra buscada con la del nodo.

Con el fin de recorrer la lista circular liberando cada nodo, se declara el método `borrarLista()` en la clase `ListaCircular`.

Consulte el apartado 8.10 para conocer el detalle de los método de las clases `Nodo` y `ListaCircular`; ahora sólo se escriben las pequeñas diferencias y el método `borrarLista()`.

```
package listaCircularPalabra;

class Nodo
{
```

```
   String dato;
   Nodo enlace;
   public Nodo (String entrada) {; }
}

public class ListaCircular
{
   private Nodo lc;

   public ListaCircular(){;}
   public ListaCircular insertar(String entrada){;}
   public void eliminar(String entrada)
   {
      Nodo actual;

      actual = lc;
      while ((actual.enlace != lc) &&
                     !(actual.enlace.dato.equals(entrada)))
      {
         if (!actual.enlace.dato.equals(entrada))
            actual = actual.enlace;
      }
         // Enlace de nodo anterior con el siguiente
         // si se ha encontrado el nodo.

      if (actual.enlace.dato.equals(entrada))
      {
         Nodo p;
         p = actual.enlace;          // Nodo a eliminar
         if (lc == lc.enlace)        // Lista con un solo nodo
            lc = null;
         else
         {
            if (p == lc)
            {
               lc = actual; // Se borra el elemento referenciado por lc,
                            // el nuevo acceso a la lista es el anterior
            }
```

```
            }
            actual.enlace = p.enlace;
         }
         p = null;
      }
   }

   public void borrarLista()
   {
      Nodo p;
      if (lc != null)
      {
         p = lc;
         do {
            Nodo t;
            t = p;
            p = p.enlace;
            t = null;  // no es estrictamente necesario
         }while(p != lc);
      }
      else
```

```
         System.out.println("\n\t Lista vacía.");
      lc = null;
   }
   public void recorrer(){;}
}
/* clase con el método main().  Se escribe un sencillo menu para
   elegir operaciones con la lista circular.
*/
import java.io.*;
import listaCircularPalabra.*;
class ListaPalabras
{
   public static void main(String [] a) throws IOException
   {
      String palabra;
      ListaCircular listaCp;
      int opc;
      BufferedReader entrada = new BufferedReader(
                               new InputStreamReader(System.in));

      listaCp = new ListaCircular();
      System.out.println("\n Entrada de Nombres. Termina con ^Z.\n");
      while ((palabra = entrada.readLine())!= null)
      {
         String nueva;
         nueva = new String(palabra);
         listaCp.insertar(nueva);
      }
      System.out.println("\t\tLista circular de palabras");
      listaCp.recorrer();

      System.out.println("\n\t Opciones para manejar la lista");
      do {
         System.out.println("1. Eliminar una palabra.\n");

          System.out.println("2. Mostrar la lista completa.\n");
          System.out.println("3. Salir y eliminar la lista.\n");
          do {
             opc = Integer.parseInt(entrada.readLine());
         }while (opc<1 || opc>3);

          switch (opc) {
          case 1: System.out.print("Palabra a eliminar: ");
                  palabra = entrada.readLine();
                  listaCp.eliminar(palabra);
                  break;
          case 2: System.out.println("Palabras en la Lista:\n");
                  listaCp.recorrer();
                  break;
          case 3: System.out.print("Eliminación de la lista.");
                  listaCp.borrarLista();
          }
       }while (opc != 3);
    }
 }
```

Referencias Bibliográficas

Libro: Estructuras de datos en Java

Autores: Luis Joyanes Aguilar Ignacio Zahonero Martínez

Edificio Valrealty, 1. ª Planta

Basauri, 17 28023 Aravaca (Madrid)

Editor: José Luis García Técnico

Editorial: Blanca Pecharromán

Compuesto en: Gesbiblo, S. L.

Diseño de cubierta: Gesbiblo, S. L.