UNIVERSIDAD NACIONAL DE COLOMBIA

Estructuras de Datos

Sesión 4

List Data Structure (Part 4)

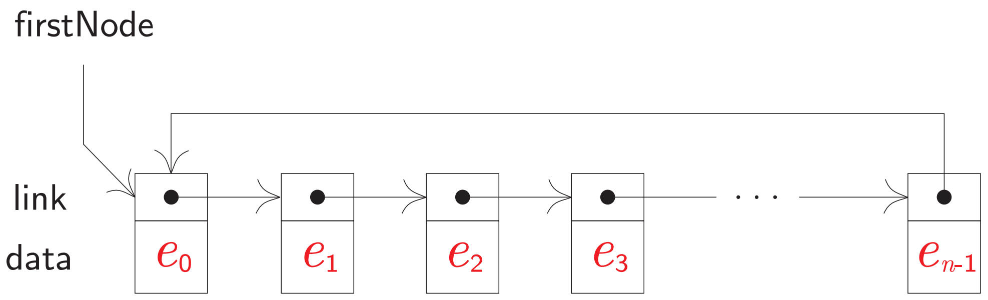
Yoan Pinzón

Table of Content Session 4

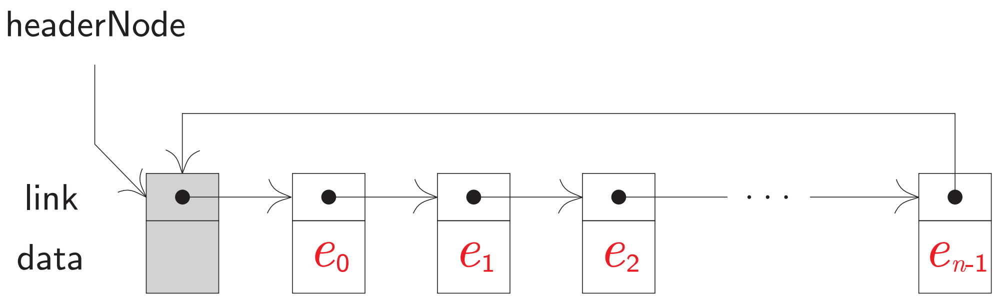
Circular Chain with Header Node

Special cases are always problematic in algorithm design and frequently lead to bugs. The code of a chain can be simplified (to run faster) by doing the following:

1) Represent the chain as a **circular chain**.



2) Adding an additional node, called the **header node**, at the front.



Class Definition of CircularWithHeader

```
package unal.datastructures;

import java.util.*;

public class CircularWithHeader<T> implements LinearList<T>
{
   // fields
   protected ChainNode<T> headerNode;
   protected int size;

   // constructor
   public CircularWithHeader( ) { /* ... */ }

   // methods
   public boolean isEmpty( ) { /* ... */ }
   public int size( ) { /* ... */ }
```

```
   void checkIndex( int index ) { /* ... */ }
   public T get( int index ) { /* ... */ }
   public int indexOf( T theElement ) { /* ... */ }
   public T remove( int index ) { /* ... */ }
   public void add( int index, T theElement ) { /* ... */ }
   public String toString( ) { /* ... */ }
   public static void main( String[] args ) { /* ... */ }
}
```

constructor

```java
/** create a circular list that is empty */
public CircularWithHeader( )
{
    headerNode = new ChainNode<T>( );
    headerNode.next = headerNode;
    size = 0;
}
```

isEmpty

```java
/** @return true iff list is empty */
public boolean isEmpty( )
{
   return size == 0;
}
```

size

```
/** @return current number of elements in list */
public int size( )
{
   return size;
}
```

checkIndex

```
/** @throws IndexOutOfBoundsException when
 * index is not between 0 and size - 1 */
void checkIndex( int index )
{
   if( index < 0 || index >= size )
      throw new IndexOutOfBoundsException
            ( "index␣=␣" + index + "␣␣size␣=␣" + size );
}
```

get

```java
/** @return element with specified index
 * @throws IndexOutOfBoundsException when
 * index is not between 0 and size - 1 */
public T get( int index )
{
   checkIndex( index );

   // move to desired node
   ChainNode<T> currentNode = headerNode.next;
   for( int i = 0; i < index; i++ )
      currentNode = currentNode.next;

   return currentNode.element;
}
```

indexOf

```java
/** @return index of first occurrence of theElement,
 * return -1 if theElement not in list */
public int indexOf( T theElement )
{
   // put theElement in header node
   headerNode.element = theElement;

   // search the list for theElement
   ChainNode<T> currentNode = headerNode.next;
   int index = 0; // index of currentNode
   while( !currentNode.element.equals( theElement ) )
   {
      // move to next node
      currentNode = currentNode.next;
      index++;
   }

   // make sure we found matching element
   if( currentNode == headerNode )
```

```java
            return -1;
        else
            return index;
    }
```

remove

```java
    /** Remove the element with specified index.
     * All elements with higher index have their
     * index reduced by 1.
     * @throws IndexOutOfBoundsException when
     * index is not between 0 and size - 1
     * @return removed element */
    public T remove( int index )
    {
        checkIndex( index );

        T removedElement;

        // use q to get to predecessor of desired node
        ChainNode<T> q = headerNode;
        for( int i = 0; i < index; i++ )
            q = q.next;
```

```
      removedElement = q.next.element;
      q.next = q.next.next; // remove desired node

      size--;
      return removedElement;
   }
```

add

```
   /** Insert an element with specified index.
    * All elements with equal or higher index
    * have their index increased by 1.
    * @throws IndexOutOfBoundsException when
    * index is not between 0 and size */
   public void add( int index, T theElement )
   {
      if( index < 0 || index > size )
         // invalid list position
         throw new IndexOutOfBoundsException
               ( "index␣=␣" + index + "␣␣size␣=␣" + size );

      // find predecessor of new element
      ChainNode<T> p = headerNode;    // Fixed YP
      for( int i = 0; i < index; i++ )
         p = p.next;
```

```java
        // insert after p
        p.next = new ChainNode<T>( theElement, p.next );

        size++;
    }
```

toString

```java
    /** convert to a string */
    @Override
    public String toString( )
    {
        StringBuilder s = new StringBuilder( "[" );

        // put elements into the buffer
        ChainNode<T> currentNode = headerNode.next;
        while( currentNode != headerNode )
        {
            s.append( Objects.toString( currentNode.element ) + ", " );
            currentNode = currentNode.next;
        }
        if( size > 0 )
            s.setLength( s.length( ) - 2 ); // remove last ", "
        s.append( "]" );

        // create equivalent String
        return new String( s );
    }
```

main

```java
/** test program */
public static void main( String[] args )
{
   CircularWithHeader<Integer> x = new CircularWithHeader<>( );

   for( int i = 0; i < 10; i++ )
      x.add( i, new Integer( i ) );
   System.out.println( "List=" + x );

   for( int i = 0; i < 5; i++ )
      x.remove( 2 );
   System.out.println( "List=" + x );

   for(int i = 0; i < 10; i++)
      System.out.println( i + "␣is␣element␣" + x.indexOf( new ↙
        ↳ Integer( i ) ) );
}
```

Compiling CircularWithHeader.java

```
C:\2016699\code> javac unal\datastructures\CircularWithHeader.java ↵
C:\2016699\code> java unal.datastructures.CircularWithHeader ↵
List=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
List=[0, 1, 7, 8, 9]
0 is element 0
1 is element 1
2 is element -1
3 is element -1
4 is element -1
5 is element -1
6 is element -1
7 is element 2
8 is element 3
9 is element 4
```

Time Complexity Comparison of Representations

Operation	ArrayLinearList[†]	Chain[‡]
`isEmpty`	$\Theta(1)$	$\Theta(1)$
`size`	$\Theta(1)$	$\Theta(1)$
`checkIndex`	$\Theta(1)$	$\Theta(1)$
`get`	$\Theta(1)$	$O($`index`$)$
`indexOf`	$O($`size`$)$	$O($`size`$)$
`remove`	$O($`size-index`$)$	$O($`index`$)$
`add`	$O($`size`$)$	$O($`index`$)$
`toString`	$\Theta($`size`$)$	$\Theta($`size`$)$

[†]Array-based Representation
[‡]Linked Representation