UNIVERSIDAD NACIONAL DE COLOMBIA

Estructuras de Datos

Sesión 11

Dictionary Data Structure (Part 1)

Yoan Pinzón

Table of Content Session 11

- **Dictionary Data Structure**
  - ▷ Linear List Representation

Dictionary Data Structure

Dictionary: collection of pairs of the form (k, e), where k is the *key* and e is the element associated with the key k, supporting the following operations:

- **Insert** an element with a specific key value
- **Search** the dictionary for an element with a specific key value
- **Delete** an element with a specific key value

No two pairs in a dictionary have the same key

The ADT Dictionary

AbstractDataType Dictionary
{
instances: collection of elements with distinct keys

operations:

`get(k)`: return the element with key `k`

`put(k, x)`: put the element `x` whose key is `k` into the dictionary and return the old element (if any) associated with `k`

`remove(k)`: remove the element with key `k` and return it
}

The Interface Dictionary

```java
package unal.datastructures;

interface Dictionary<K extends Comparable<? super K>, E>
{
   E get( K key );
   E put( K key, E theElement );
   E remove( K key );
}
```

Dictionary Data Structure

Linear List Representation

A dictionary is maintained as an ordered linear list $(e_0, e_1, \ldots)$, where the e_is are dictionary pairs in ascending order of key.

Using the linked representation, the class definition for a dictionary looks as follow:

Class Definition of SortedChain

```
package unal.datastructures;

import java.util.*;

public class SortedChain <K extends Comparable<? super K>, E> ↙
  ↳ implements Dictionary<K, E>, Iterable<E>
{
   // top-level nested class
   protected static class SortedChainNode <K extends ↙
     ↳ Comparable<? super K>, E> extends DataDict <K, E> { /* ↙
     ↳ ... */ }

   // fields of SortedChain
   protected SortedChainNode<K, E> firstNode;
   protected int size;
```

```
   // constructor
   public SortedChain ( ) { /* ... */ }

   // methods
   public boolean isEmpty ( ) { /* ... */ }
   public int size ( ) { /* ... */ }
   public E get ( K theKey ) { /* ... */ }
   public E put ( K theKey, E theElement ) { /* ... */ }
   public E remove ( K theKey ) { /* ... */ }
   public String toString ( ) { /* ... */ }
   public Iterator<E> iterator ( ) { /* ... */ }

   private class SortedChainIterator implements Iterator<E> { /* ↙
     ↳ ... */ }

   public static void main ( String[] args ) { /* ... */ }
}
```

DataDict class

```java
class DataDict<K extends Comparable<? super K>, E>
{
   // fields
   K key;     // its key
   E element; // element in node

   // constructor
   DataDict( )
   {
      key = null;
      element = null;
   }

   DataDict( K theKey, E theElement )
   {
      key = theKey;
      element = theElement;
   }
```

```java
   @Override
   public String toString( )
   {
      return "[" + Objects.toString( element ) +
         ",␣key=" + Objects.toString( key ) + "]";
   }
}
```

SortedChainNode class

```
protected static class SortedChainNode<K extends Comparable<? super K>, E> extends DataDict<K, E>
{
   // fields
   protected SortedChainNode<K, E> next;

   // constructors
   protected SortedChainNode( )
   {
      super( );
      next = null;
   }

   protected SortedChainNode( K theKey, E theElement )
   {
      super( theKey, theElement );
      next = null;
   }
```

```
   protected SortedChainNode( K theKey, E theElement,
                           SortedChainNode<K, E> theNext )
   {
      super( theKey, theElement );
      next = theNext;
   }
}
```

constructor

```
/** create an empty sorted chain */
public SortedChain( )
{
    firstNode = null;
    size = 0;
}
```

isEmpty

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

size

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

get

```
/** @return element with specified key
 * @return null if there is no matching element */
public E get( K theKey )
{
   SortedChainNode<K, E> currentNode = firstNode;

   // search for match with theKey
   while( currentNode != null &&
          currentNode.key.compareTo( theKey ) < 0 )
      currentNode = currentNode.next;

   // verify match
   if( currentNode != null && currentNode.key.equals( theKey ) )
      // yes, found match
      return currentNode.element;

   // no match
   return null;
}
```

put

```
/** insert an element with the specified key
 * overwrite old element if there is already an
 * element with the given key
 * @return old element ( if any ) with key theKey */
public E put( K theKey, E theElement )
{
   SortedChainNode<K, E> p = firstNode,
                         tp = null; // tp trails p

   // move tp so that theElement can be inserted after tp
   while( p != null && p.key.compareTo( theKey ) < 0 )
   {
      tp = p;
      p = p.next;
   }

   // check if there is a matching element
   if( p != null && p.key.equals( theKey ) )
   {  // replace old element
```

```
      E elementToReturn = p.element;
      p.element = theElement;
      return elementToReturn;
   }

   // no match, set up node for theElement
   SortedChainNode<K, E> q = new SortedChainNode<K, E>( ↙
     ↳ theKey, theElement, p );

   // insert node just after tp
   if( tp == null ) firstNode = q;
   else tp.next = q;

   size++;

   return null;
}
```

remove

```
/** @return matching element and remove it
 * @return null if no matching element */
public E remove( K theKey )
{
   SortedChainNode<K, E> p = firstNode,
                       tp = null; // tp trails p

   // search for match with theKey
   while( p != null && p.key.compareTo( theKey ) < 0 )
   {
      tp = p;
      p = p.next;
   }

   // verify match
   if( p != null && p.key.equals( theKey ) )
   {  // found a match
      E e = p.element; // the matching element

      // remove p from the chain
```

```
      if( tp == null ) firstNode = p.next; // p is first node
      else tp.next = p.next;

      size--;

      return e;
   }

   // no matching element to remove
   return null;
}
```

toString

```java
/** convert to a string */
@Override
public String toString( )
{
   StringBuilder s = new StringBuilder( "[" );
   if( firstNode != null )
   {  // nonempty chain
      // do first element
      s.append( Objects.toString( firstNode.element ) );
      // do remaining elements
      SortedChainNode<K, E> currentNode = firstNode.next;
      while( currentNode != null )
      {
         s.append( ",␣" + Objects.toString( ↙
           ↪ currentNode.element ) );
         currentNode = currentNode.next;
```

```java
      }
   }
   s.append( "]" );

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

iterator

```java
/** create and return an iterator */
public Iterator<E> iterator( )
{
   return new SortedChainIterator( );
}
```

SortedChainIterator class

```java
/** sorted chain iterator */
private class SortedChainIterator implements Iterator<E>
{
   // fields
   private SortedChainNode<K, E> nextNode;

   // constructor
   public SortedChainIterator( )
   {
      nextNode = firstNode;
   }

   // methods
   /** @return true iff list has more elements */
   public boolean hasNext( )
   {
      return nextNode != null;
   }
```

```
      /** @return next element in list
       * @throws NoSuchElementException
       * if there is no next element */
      public E next( )
      {
         if( nextNode != null )
         {
            E obj = nextNode.element;
            nextNode = nextNode.next;
            return obj;
         }
         else throw new NoSuchElementException
                       ( "No␣next␣element" );
      }

      /** unsupported method */
      public void remove( )
      {
         throw new UnsupportedOperationException
                  ( "remove␣not␣supported" );
      }
   }
```

main

```
   /** test program */
   public static void main( String[] args )
   {
      // test default constructor
      SortedChain<Integer, Integer> x = new SortedChain<>( );

      // test put
      x.put( new Integer( 2 ), new Integer( 12 ) );
      System.out.println( "The␣list␣is␣" + x );
      x.put( new Integer( 6 ), new Integer( 16 ) );
      System.out.println( "The␣list␣is␣" + x );
      x.put( new Integer( 1 ), new Integer( 11 ) );
      System.out.println( "The␣list␣is␣" + x );
      x.put( new Integer( 4 ), new Integer( 14 ) );
      System.out.println( "The␣list␣is␣" + x );
      x.put( new Integer( 6 ), new Integer( 26 ) );
      System.out.println( "The␣list␣is␣" + x );

      // test iterator
      for( Integer r : x ) System.out.println( r);
```

```java
        // test get
        System.out.println( "element␣" +
            x.get( new Integer( 2 ) ) + "␣has␣key␣2" );
        System.out.println( "element␣" +
            x.get( new Integer( 1 ) ) + "␣has␣key␣1" );
        System.out.println( "element␣" +
            x.get( new Integer( 6 ) ) + "␣has␣key␣6" );

        // test remove
        System.out.println( "removed␣element␣" +
            x.remove( new Integer( 2 ) ) + "␣with␣key␣2" );
        System.out.println( "The␣list␣is␣" + x );
        System.out.println( "removed␣element␣" +
            x.remove( new Integer( 1 ) ) + "␣with␣key␣1" );
        System.out.println( "The␣list␣is␣" + x );
        System.out.println( "removed␣element␣" +
            x.remove( new Integer( 6 ) ) + "␣with␣key␣6" );
        System.out.println( "The␣list␣is␣" + x );
    }
```

Compiling `SortedChain.java`

```
C:\2016699\code> javac unal\datastructures\SortedChain.java ↵
C:\2016699\code> java unal.datastructures.SortedChain ↵
The list is [12]
The list is [12, 16]
The list is [11, 12, 16]
The list is [11, 12, 14, 16]
The list is [11, 12, 14, 26]
11
12
14
26
element 12 has key 2
element 11 has key 1
element 26 has key 6
removed element 12 with key 2
The list is [11, 14, 26]
removed element 11 with key 1
The list is [14, 26]
removed element 26 with key 6
The list is [14]
```