/\*\*

\*

\* @author Samuel Swedberg

\* @version 9/23/22

\*

\* A Client that demonstrates timing tests for ArrayBag and LinkedBag data structures.

\*/

public class Client {

public static void main(String[] args) {

int N = 1000000000;

int count = 0;

long start, stop, elapse;

long[][] results = new long[9][3];

// Fills Column 1

for(int n=10; n<N; n\*=10)

{

count++;

results[count][0] = n;

}

// Test for ArrayBag

count = 0; // Resets counter

for(int n=10; n<N; n\*=10)

{

ArrayBag arrayBag = new ArrayBag<>();

start = System.currentTimeMillis(); // Starts timer

for(int i=0; i<n; i++)

{

arrayBag.addTest(i); // Adds test variable to array bag

arrayBag.removeTest(); // Removes test variable to array bag

}

stop = System.currentTimeMillis(); // Ends timer

elapse = stop - start; // Elasped time

count++; // Increases count

results[count][1] = elapse; // Adds elapsed time to matrix

System.out.printf("Time to add and remove %d integrers for ArrayBag is %d (msec)\n", n, elapse);

}

// Test for LinkedBag

count = 0; // Resets counter

for(int n=10; n<N; n\*=10)

{

LinkedBag linkedBag = new LinkedBag<>();

start = System.currentTimeMillis(); // Starts timer

for(int i=0; i<n; i++)

{

linkedBag.addTest(i); // Adds test variable to linked bag

linkedBag.removeTest(); // Removes test variable to linked bag

}

stop = System.currentTimeMillis(); // Ends timer

elapse = stop - start; // Elasped time

count++; // Increases count

results[count][2] = elapse; // Adds elapsed time to matrix

System.out.printf("Time to add and remove %d integrers for LinkedBag is %d (msec)\n", n, elapse);

}

printTable(results); // printTable method to print the table. Uses the results matrix as a parameter.

}

public static void printTable(long results[][])

{

// Header for ascii table

System.out.printf("+---------------+---------+----------+%n");

System.out.printf("| N | A-Bag | L-Bag |%n");

System.out.printf("+---------------+---------+----------+%n");

// Prints out data for N, A-Bag, and L-Bag

for(int i=1; i<results.length; i++)

{

System.out.printf("| %,13d | %7d | %8d |%n", results[i][0], results[i][1], results[i][2]);

System.out.printf("+---------------+---------+----------+%n");

}

}

}

/\*\*

\*

\* @author Samuel Swedberg

\* @version 9/23/22

\* @param <E>

\*

\* The Bag interface holds methods to be access by other classes

\*/

public interface Bag<E> {

/\*\*

\*

\* @return count

\*/

public int size();

/\*\*

\*

\* @return true if empty, false otherwise

\*/

public boolean isEmpty();

/\*\*

\*

\* @param e

\* @return count number of times e exists

\*/

public int getFrequencyOf(E e);

/\*\*

\*

\* @param e

\* @return checks if list contains e

\*/

public boolean contains(E e);

/\*\*

\* add e to list

\* @param e

\*/

public void add(E e);

/\*\*

\*

\* @param e

\* @return removes first occurrence of e from list

\*/

public E remove(E e);

/\*\*

\*

\* @return removes random number from list, throws if list is empty

\*/

public E remove();

/\*\*

\*

\* @param i

\* @return returns number in i index of list, throws if i is out of bounds

\*/

public E get(int i);

/\*\*

\*

\* @param e

\* @return adds to the end of either array or linkedlist

\*/

public void addTest( E e );

/\*\*

\*

\* @return removes the last item at the end of either array or linkedlist

\*/

public E removeTest();

/\*\*

\*

\* @return contents of instance

\*/

@Override

public String toString();

/\*\*

\*

\* @param o

\* @return true if equal, false otherwise

\*/

@Override

public boolean equals(Object o);

}

import java.util.Random;

/\*\*

\*

\* @author Samuel Swedberg

\* @version 9/23/22

\*

\* The ArrayBag class holds information related to accessing and holding information using and interfaces

\*/

public class ArrayBag<E> implements Bag<E>{

private int count;

private E[] list;

/\*\*

\* Default constructor for ArrayBag

\*/

public ArrayBag()

{

list = ( E[] ) new Object[2];

}

/\*\*

\* Overloaded constructor for ArrayBag

\* @param value

\*/

public ArrayBag( int value )

{

list = ( E[] ) new Object[value];

}

/\*\*

\*

\* @return count

\*/

public int size( ) { return count; }

/\*\*

\*

\* @return true if empty, false otherwise

\*/

public boolean isEmpty( )

{

if(size() == 0)

{

return true;

}

else

{

return false;

}

}

/\*\*

\*

\* @param e

\* @return count number of times e exists

\*/

public int getFrequencyOf( E e )

{

int freqCount = 0;

for( int i=0; i<list.length; i++)

{

if( e.equals(list[i]) )

{

freqCount++;

}

}

return freqCount;

}

/\*\*

\*

\* @param e

\* @return checks if list contains e

\*/

public boolean contains( E e )

{

boolean bool = false;

for(int i=0; i<list.length; i++)

{

if( list[i] == e )

{

bool = true;

}

}

return bool;

}

/\*\*

\*

\* @param e add e to list

\*/

public void add ( E e )

{

if( size() == list.length )

{

E[] temp = ( E[] ) new Object[list.length\*2];

for(int i=0; i<list.length; i++)

{

temp[i] = list[i];

}

list = temp;

temp = null;

list[size()] = e;

count++;

}

else

{

list[size()] = e;

count++;

}

}

/\*\*

\*

\* @param e

\* @return removes first occurrence of e from list

\*/

public E remove ( E e )

{

boolean firstOccurence = false;

if( isEmpty() == false )

{

for(int i=0; i<size(); i++)

{

if(e == list[i] && firstOccurence == false)

{

firstOccurence = true;

for(int x=i; x<size()-1; x++)

{

list[x] = list[x+1];

}

list[list.length-1] = null;

--count;

}

}

}

return null;

}

/\*\*

\*

\* @return removes random number from list, throws if list is empty

\*/

public E remove ( )

{

Random random = new Random();

if ( isEmpty() == false )

{

int rand = random.nextInt(size());

for( int i=rand; i<size()-1; i++)

{

list[i] = list[i+1];

}

list[list.length-1] = null;

--count;

}

else

{

throw new IllegalStateException("IllegalStateException: The remove() method cannot be called on an empty list");

}

return null;

}

/\*\*

\*

\* @param i

\* @return returns number in i index of list, throws if i is out of bounds

\*/

public E get( int i )

{

if ( i > list.length)

{

throw new ArrayIndexOutOfBoundsException("ArrayIndexOutOfBoundsException: index is outside the bounds of the array.");

}

return list[i];

}

/\*\*

\*

\* @param e

\* @return adds to the end of the array

\*/

public void addTest ( E e )

{

for(int i=size(); i<size(); i++)

{

e = list[i];

}

}

/\*\*

\*

\* @return removes the end of the array

\*/

public E removeTest()

{

for(int i=size(); i<size(); i++)

list[i] = null;

count--;

return null;

}

/\*\*

\*

\* @return contents of instance

\*/

public String toString( ) {

String s = "";

for(int i=0; i<size(); i++)

{

s += " " + list[i];

}

return getClass().getName() + "@ size:" + size() + ", " + "list: " + s;

}

/\*\*

\*

\* @param o

\* @return true if equal, false otherwise

\*/

public boolean equals( Object o )

{

if ( !( o instanceof ArrayBag ) )

return false;

ArrayBag b = ( ArrayBag ) o;

return super.equals( b )

&& list == b.list;

}

}

import java.util.Random;

/\*\*

\*

\* @author Samuel Swedberg

\* @version 9/23/22

\*

\* The LinkedBag class holds information related to accessing and holding information using SinglyLinkedLists and interfaces

\*/

public class LinkedBag<E> implements Bag<E> {

SinglyLinkedList<E> bag;

/\*\*

\* Default constructor for LinkedBag

\*/

public LinkedBag() {

bag = new SinglyLinkedList<>();

}

/\*\*

\* Overloaded constructor for LinkedBag

\* @param capacity

\*/

public LinkedBag(int capacity) {

this();

}

/\*\*

\*

\* @return count

\*/

public int size( ) { return bag.size(); }

/\*\*

\*

\* @return true if empty, false otherwise

\*/

public boolean isEmpty( ) { return bag.isEmpty(); }

/\*\*

\*

\* @param e

\* @return count number of times e exists

\*/

public int getFrequencyOf( E e )

{

int freqCount = 0;

for(int i=0; i<bag.size(); i++)

{

E temp = bag.removeFirst();

if(temp.equals(e))

{

freqCount++;

}

bag.addLast(temp);

}

return freqCount;

}

/\*\*

\*

\* @param e

\* @return checks if list contains e

\*/

public boolean contains( E e )

{

boolean bool = false;

for(int i=0; i<bag.size(); i++)

{

if( bag.equals(e) )

{

bool = true;

}

}

return bool;

}

/\*\*

\*

\* @param e add e to list

\*/

public void add ( E e ) { bag.addLast(e); }

/\*\*

\*

\* @param num

\* @return removes first occurrence of num from list

\*/

public E remove ( E e )

{

boolean firstOccurrence = false;

if( !bag.isEmpty() )

{

for(int i=0; i<bag.size()+1; i++)

{

E temp = bag.removeFirst();

if(temp.equals(e) && firstOccurrence == false)

{

firstOccurrence = true;

}

else

{

bag.addLast(temp);

}

}

}

return null;

}

/\*\*

\*

\* @return removes random number from list, throws if list is empty

\*/

public E remove ( )

{

Random random = new Random();

boolean firstOccurrence = false;

if ( !bag.isEmpty() )

{

int rand = random.nextInt(bag.size());

for(int i=0; i<bag.size()+1; i++)

{

E temp = bag.removeFirst();

if(i == rand && firstOccurrence == false)

{

firstOccurrence = true;

}

else

{

bag.addLast(temp);

}

}

}

else

{

throw new IllegalStateException("IllegalStateException: The remove() method cannot be called on an empty list");

}

return null;

}

/\*\*

\*

\* @param i

\* @return returns number in i index of list, throws if i is out of bounds

\*/

public E get( int i )

{

E getFound = null;

if ( i > bag.size())

{

throw new ArrayIndexOutOfBoundsException("ArrayIndexOutOfBoundsException: index is outside the bounds of the array.");

}

else

{

for(int j=0; j<bag.size(); j++)

{

E temp = bag.removeFirst();

if(j == i)

{

getFound = temp;

}

bag.addLast(temp);

}

}

return getFound;

}

/\*\*

\*

\* @param e

\* @return adds to the head of the bag

\*/

public void addTest ( E e )

{

bag.addFirst(e);

}

/\*\*

\*

\* @return removes the head of the bag

\*/

public E removeTest()

{

bag.removeFirst();

return null;

}

/\*\*

\*

\* @return contents of instance

\*/

public String toString( ) {

String s = "";

for(int i=0; i<bag.size(); i++)

{

E temp = bag.removeFirst();

s += " " + temp;

bag.addLast(temp);

}

return getClass().getName() + "@ size:" + bag.size() + ", " + "list: " + s;

}

/\*\*

\*

\* @param o

\* @return true if equal, false otherwise

\*/

public boolean equals( Object o )

{

if ( !( o instanceof ArrayBag ) )

return false;

ArrayBag b = ( ArrayBag ) o;

return super.equals( b )

&& bag.size() == b.size();

}

}

/\*\*

\* SinglyLinkedList Class

\* Code Fragments 3.14, 3.15

\* from

\* Data Structures & Algorithms, 6th edition

\* by Michael T. Goodrich, Roberto Tamassia & Michael H. Goldwasser

\* Wiley 2014

\* Transcribed by

\* @author Samuel Swedberg

\* @version 9/23/22

\*/

public class SinglyLinkedList<E> {

//---- nested Node class -----

private static class Node<E> {

private E element; // reference to the element stored at this node

private Node<E> next; // reference to the subsequent node in the list

public Node(E e, Node<E> n) {

element = e;

next = n;

}

public E getElement( ) { return element; }

public Node<E> getNext( ) { return next; }

public void setNext(Node<E> n) { next = n; }

}

// instance variables of the SinglyLinkedList

private Node<E> head = null; // head node of the list (or null if empty)

private Node<E> tail = null; // last node of the list (or null if empty)

private int size = 0; // number of nodes in the lis

public SinglyLinkedList() {} // constructs an initially empty list

// access methods

public int size( ) { return size; }

public boolean isEmpty( ) { return size == 0; }

public E first( ) { // returns (but does not remove) the first element

if (isEmpty( )) return null;

return head.getElement( );

}

public E last( ) { // returns (but does not remove) the last element

if (isEmpty( )) return null;

return tail.getElement( );

}

// update methods

public void addFirst(E e) { // adds element e to the front of the list

head = new Node<>(e, head); // create and link a new node

if (size == 0)

tail = head; // special case: new node becomes tail also

size++;

}

public void addLast(E e) { // adds element e to the end of the list

Node<E> newest = new Node<>(e, null); // node will eventually be the tail

if (isEmpty( ))

head = newest; // special case: previously empty list

else

tail.setNext(newest); // new node after existing tail

tail = newest; // new node becomes the tail

size++;

}

public E removeFirst( ) { // removes and returns the first element

if (isEmpty( )) return null; // nothing to remove

E answer = head.getElement( );

head = head.getNext( ); // will become null if list had only one node

size--;

if (size == 0)

tail = null; // special case as list is now empty

return answer;

}

}

run:

Time to add and remove 10 integrers for ArrayBag is 0 (msec)

Time to add and remove 100 integrers for ArrayBag is 0 (msec)

Time to add and remove 1000 integrers for ArrayBag is 0 (msec)

Time to add and remove 10000 integrers for ArrayBag is 0 (msec)

Time to add and remove 100000 integrers for ArrayBag is 2 (msec)

Time to add and remove 1000000 integrers for ArrayBag is 8 (msec)

Time to add and remove 10000000 integrers for ArrayBag is 33 (msec)

Time to add and remove 100000000 integrers for ArrayBag is 241 (msec)

Time to add and remove 10 integrers for LinkedBag is 0 (msec)

Time to add and remove 100 integrers for LinkedBag is 0 (msec)

Time to add and remove 1000 integrers for LinkedBag is 0 (msec)

Time to add and remove 10000 integrers for LinkedBag is 0 (msec)

Time to add and remove 100000 integrers for LinkedBag is 2 (msec)

Time to add and remove 1000000 integrers for LinkedBag is 9 (msec)

Time to add and remove 10000000 integrers for LinkedBag is 47 (msec)

Time to add and remove 100000000 integrers for LinkedBag is 502 (msec)

**+---------------+---------+----------+**

**| N | A-Bag | L-Bag |**

**+---------------+---------+----------+**

**| 10 | 0 | 0 |**

**+---------------+---------+----------+**

**| 100 | 0 | 0 |**

**+---------------+---------+----------+**

**| 1,000 | 0 | 0 |**

**+---------------+---------+----------+**

**| 10,000 | 0 | 0 |**

**+---------------+---------+----------+**

**| 100,000 | 2 | 2 |**

**+---------------+---------+----------+**

**| 1,000,000 | 8 | 9 |**

**+---------------+---------+----------+**

**| 10,000,000 | 33 | 47 |**

**+---------------+---------+----------+**

**| 100,000,000 | 241 | 502 |**

**+---------------+---------+----------+**

BUILD SUCCESSFUL (total time: 1 second)