03 A: Lists, Stacks, and Queues I

CS1102S: Data Structures and Algorithms

Martin Henz

January 27, 2010

Generated on Tuesday 26th January, 2010, 17:00

Abstract Data Types

What is an ADT?

A set of objects together with a set of operations involving them

Abstract Data Types

What is an ADT?

A set of objects together with a set of operations involving them

Inside and Outside

ADTs allow for a clear separation of the *use* of data objects (outside view), and their *implementation* (inside view)

Abstract Data Types

What is an ADT?

A set of objects together with a set of operations involving them

Inside and Outside

ADTs allow for a clear separation of the *use* of data objects (outside view), and their *implementation* (inside view)

Outside view in Java

ADTs are represented in Java by *interfaces* that define the operations on its members

Abstract Data Types

What is an ADT?

A set of objects together with a set of operations involving them

Inside and Outside

ADTs allow for a clear separation of the *use* of data objects (outside view), and their *implementation* (inside view)

Outside view in Java

ADTs are represented in Java by *interfaces* that define the operations on its members

Inside view in Java

ADTs programmed through classes that *implement* interfaces

The List ADT

Characteristics of Lists

- Like in arrays, the elements of a list are numbered using indices from 0 to the current size of the list minus one:

$$A_0, A_1, A_2, \ldots, A_{N-1}$$

The List ADT

Characteristics of Lists

- Like in arrays, the elements of a list are numbered using indices from 0 to the current size of the list minus one:

$$A_0, A_1, A_2, \ldots, A_{N-1}$$

- The *position* if element A_i is the integer i

The List ADT

Characteristics of Lists

- Like in arrays, the elements of a list are numbered using indices from 0 to the current size of the list minus one:

$$A_0, A_1, A_2, \ldots, A_{N-1}$$

- The *position* if element A_i is the integer i
- **But:** Arrays have fixed size, whereas lists start out *empty*, and then grow and shrink

The List ADT

Characteristics of Lists

- Like in arrays, the elements of a list are numbered using indices from 0 to the current size of the list minus one:

$$A_0, A_1, A_2, \ldots, A_{N-1}$$

- The *position* if element A_i is the integer i
- **But:** Arrays have fixed size, whereas lists start out *empty*, and then grow and shrink
- **Operations:** Accessing and changing elements (like in arrays), plus *adding* and *removing* elements

The List ADT

Operations on lists

- `makeEmpty`: create an empty list

The List ADT

Operations on lists

- `makeEmpty`: create an empty list
- `get(i)`: retrieve element at given position `i`; no change of list

The List ADT

Operations on lists

- `makeEmpty`: create an empty list
- `get(i)`: retrieve element at given position `i`; no change of list
- `set(i,x)`: change element at given position `i` to new value; no change of rest of list

The List ADT

Operations on lists

- `makeEmpty`: create an empty list
- `get(i)`: retrieve element at given position `i`; no change of list
- `set(i,x)`: change element at given position `i` to new value; no change of rest of list
- `add(i,x)`: insert element at given position `i`; following elements will change their index

The List ADT

Operations on lists

- `makeEmpty`: create an empty list
- `get(i)`: retrieve element at given position `i`; no change of list
- `set(i,x)`: change element at given position `i` to new value; no change of rest of list
- `add(i,x)`: insert element at given position `i`; following elements will change their index
- `remove(i)`: remove element from given position; following elements will change their index

The List ADT

Operations on lists

- `makeEmpty`: create an empty list
- `get(i)`: retrieve element at given position `i`; no change of list
- `set(i,x)`: change element at given position `i` to new value; no change of rest of list
- `add(i,x)`: insert element at given position `i`; following elements will change their index
- `remove(i)`: remove element from given position; following elements will change their index

How can we *implement* such a list?

Array Implementation

Array Implementation

Question

How can we know how large an array to start with?

Array Implementation

Question

How can we know how large an array to start with?

Problem

What do we do when we want to insert an element and no space is left?

Array Implementation

Question

How can we know how large an array to start with?

Problem

What do we do when we want to insert an element and no space is left?

Idea

Start out with a fixed size array and store the elements starting at position 0.

Array Implementation

Question

How can we know how large an array to start with?

Problem

What do we do when we want to insert an element and no space is left?

Idea

Start out with a fixed size array and store the elements starting at position 0. When the array size is exceeded, create an array of double its size, and copy the elements over.

In Detail: Doubling and Copying Array

```
int [] arr = new int[10];
...
// Later on we decide arr needs to be larger
int [] newArr = new int[arr.length * 2];
for (int i = 0; i < arr.length; i++)
   newArr[i] = arr[i];
arr = newArr;
```

Quick Analysis: Array Implementation of Lists

- `get(i)` and `set(i,x)` require

Quick Analysis: Array Implementation of Lists

- `get(i)` and `set(i,x)` require $O(1)$ time (array access)

Quick Analysis: Array Implementation of Lists

- `get(i)` and `set(i,x)` require $O(1)$ time (array access)
- `add(i,x)` and `remove(i)` require:

Quick Analysis: Array Implementation of Lists

- `get(i)` and `set(i,x)` require $O(1)$ time (array access)
- `add(i,x)` and `remove(i)` require:
  - $O(N)$ if i is low (for example 0), and

Quick Analysis: Array Implementation of Lists

- `get(i)` and `set(i,x)` require $O(1)$ time (array access)
- `add(i,x)` and `remove(i)` require:
    - $O(N)$ if i is low (for example 0), and
    - $O(1)$ if i is high (for example N)

Simple Linked Lists

Idea

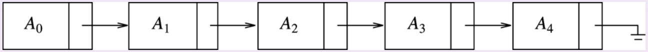
Build a chain of objects called *nodes*, where each has a reference to the *next* one

Pros and Cons

No need for copying, but now access is expensive

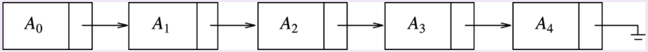
Removing and Adding Elements

Example linked list:

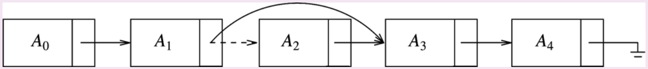


Removing and Adding Elements

Example linked list:

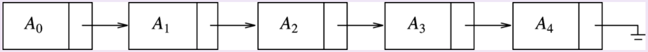


Removing an element:

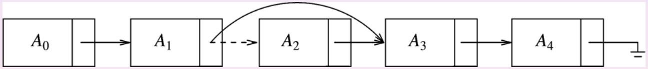


Removing and Adding Elements

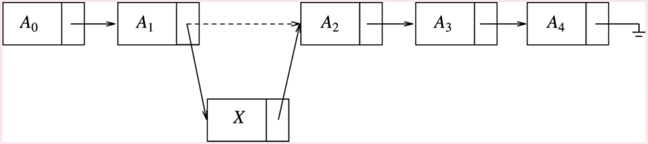
Example linked list:



Removing an element:



Inserting an element:



Quick Analysis: Array Implementation of Lists

`get(i)`, `set(i,x)`, `add(i,x)` and `remove(i)` require:

Quick Analysis: Array Implementation of Lists

`get(i)`, `set(i,x)`, `add(i,x)` and `remove(i)` require:

- $O(1)$ if i is low (for example 0), and

Quick Analysis: Array Implementation of Lists

`get(i)`, `set(i,x)`, `add(i,x)` and `remove(i)` require:

- $O(1)$ if i is low (for example 0), and
- $O(N)$ if i is high (for example N)

Quick Analysis: Array Implementation of Lists

`get(i)`, `set(i,x)`, `add(i,x)` and `remove(i)` require:

- $O(1)$ if i is low (for example 0), and
- $O(N)$ if i is high (for example N)

Question

Can we improve the runtime for insertion at the end of the list?

Optimization: Doubly-linked Lists

Idea

Keep track of the current end of the chain, to add a new node, using a `last` field

Optimization: Doubly-linked Lists

Idea

Keep track of the current end of the chain, to add a new node, using a `last` field

Problem

How to update `last` field when removing last element?

Optimization: Doubly-linked Lists

Idea

Keep track of the current end of the chain, to add a new node, using a `last` field

Problem

How to update `last` field when removing last element?

Solution

Keep track of the `previous` node

Optimization: Doubly-linked Lists

Idea

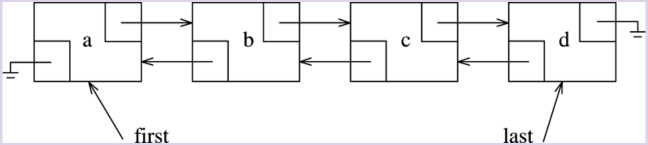
Keep track of the current end of the chain, to add a new node, using a `last` field

Problem

How to update `last` field when removing last element?

Solution

Keep track of the `previous` node



Java’s Collections API

API

An “API” (Application Programming Interface) is a library of interfaces and classes that support the programming of applications

Java's Collections API

API

An "API" (Application Programming Interface) is a library of interfaces and classes that support the programming of applications

Java's Collections API

API for *collections*, sets of identically-typed objects

Java's Collections API

API

An "API" (Application Programming Interface) is a library of interfaces and classes that support the programming of applications

Java's Collections API

API for *collections*, sets of identically-typed objects

Purpose

Provides interfaces and implementations of the most commonly used collections, including most of the data structures studied in CS1102S!

Excursion: Generic Types in Java

Remember IntList from crash course:

```
public class IntList
{ ...
  public static IntList cons(int i,
                             IntList list) {...}
  public static IntList nil = ...;
  public static int car(IntList list){...}
  public static IntList cdr(IntList list){...}
  public static boolean isNil(IntList list) {...}
}
```

Excursion: Generic Types in Java

Such lists can only contain integers! How about lists of integers?

```
public class IntListList
{
 public static IntListList cons(IntList i,
                                IntListList list){.}
 public static IntListList nil = ...;
 public static IntList car(IntListList list){.....}
 public static IntListList cdr(IntListList list){.}
 public static boolean isNil(IntListList list){...}
}
```

Tired of writing “boilerplate”?

Problem

For each content type, we need to introduce a new kind of list type, with identical implementation!

Tired of writing “boilerplate”?

Problem

For each content type, we need to introduce a new kind of list type, with identical implementation!

Solution

Introduce *generic types*: type placeholders that can be instantiated when a list object is created

Generic Lists, Scheme Style

```
public class List<Any> {
{
  public static List<Any>
        cons(<Any> i, List<Any> list) {...}
  public static List<Any>  nil = ...;
  public static <Any> car(List<Any>  list){...}
  public static <Any> List cdr(List<Any>  list){.}
  public static boolean isNil(List<Any> list){.}
}
...
List<Integer> mylist
    = List.cons(new Integer(5),List.nil);
```

The Top-level Collection Interface

```
public interface Collection<Any>
        extends Iterable<Any>
{
    int size();
    boolean isEmpty();
    void clear();
    boolean contains(Any x);
    boolean add(Any x);        // sic
    boolean remove(Any x);     // sic
    java.util.Iterator<Any> iterator();
}
```

Iterable Objects and Iterators

Requirement of Iterable Interface

Iterable objects must support a method `iterator()`, which returns an iterator of correct type

```
public interface Collection<Any>
          extends Iterable<Any>
{
   ...
   java.util.Iterator<Any> iterator();
}
```

What is an Iterator?

```
public interface Iterator<Any> {
  boolean hasNext( );
  Any next( );
  void remove( );
}
```

Iterable Objects Provide Enhanced for-loop

```
public static <Any> void
  print(Collection<Any> coll) {
    for(Any item : coll)
      System.out.println(item);
}
```

Java Compiler Support for Iterators

```
for( Any item : coll )
    System.out.println( item );
```

becomes

```
Iterator itr = coll.iterator();
while( itr.hasNext()) {
  Any item = itr.next( );
  System.out.println( item );
} }
```

The List Interface in Collection API

```
public interface List<Any>
       extends Collection<Any>
{
  Any get(int idx);
  Any set(int idx, Any newVal);
  void add(int idx, Any x);
  void remove(int idx);

  ListIterator<Any> listIterator(int pos);
}
```

ListIterators

Idea

Provide, in addition to iterating forward also *iterating backward* and in addition to removal of an entry also *addition and changing* of an entry

ListIterators

```
public interface ListIterator<Any>
        extends Iterator<Any>
{
  boolean hasPrevious();
  Any previous();
  void add(Any x);
  void set(Any newVal);
}
```

ArrayList and LinkedList

```
public class ArrayList<Any>
        implements List<Any> {...}
public class LinkedList<Any>
        implements List<Any> {...}
```

Example: Remove Even Elements

Task

In a given list of Integer, remove all even integers, without copying the list (*in-place* operation)

```
ArrayList<Integer> myArrayList = ...;
LinkedList<Integer> myLinkedList = ...;
removeEvens(myArrayList);
removeEvens(myLinkedList);
```

ADT in Action

```
ArrayList<Integer> myArrayList = ...;
LinkedList<Integer> myLinkedList = ...;
removeEvens(myArrayList);
removeEvens(myLinkedList);
```

Observation

Both ArrayList and LinkedList implement the interface List. We can define removeEvens(...) in terms of List operations!

Inside and Outside

The same function removeEvens behaves differently for myLinkedList than for myArrayList!

In Detail: First Version

```
public static void removeEvensVer1(
                          List<Integer> lst) {
  int i = 0;
  while( i < lst.size( ) )
    if( lst.get( i ) % 2 == 0 )
      lst.remove( i );
    else
      i++;
}
```

In Detail: First Version

```
public static void removeEvensVer1(
                                List<Integer> lst) {
  int i = 0;
  while( i < lst.size( ) )
    if( lst.get( i ) % 2 == 0 )
      lst.remove( i );
    else
      i++;
}
```

Runtime for removeEvensVer1(myArrayList):
Runtime for removeEvensVer1(myLinkedList):

In Detail: First Version

```
public static void removeEvensVer1(
                                   List<Integer> lst) {
  int i = 0;
  while( i < lst.size( ) )
    if( lst.get( i ) % 2 == 0 )
      lst.remove( i );
    else
      i++;
}
```

Runtime for removeEvensVer1(myArrayList): $O(N^2)$
Runtime for removeEvensVer1(myLinkedList):

In Detail: First Version

```
public static void removeEvensVer1(
                          List<Integer> lst) {
  int i = 0;
  while( i < lst.size( ) )
    if( lst.get( i ) % 2 == 0 )
      lst.remove( i );
    else
      i++;
}
```

Runtime for removeEvensVer1(myArrayList): $O(N^2)$
Runtime for removeEvensVer1(myLinkedList): $O(N^2)$

In Detail: Second Version

Idea

Use an iterator to go through the list, and remove element when found to be even

```
public static void removeEvensVer2(
                         List<Integer> lst) {
  for( Integer x : lst )
     if( x % 2 == 0 )
       lst.remove( x );
}
```

In Detail: Second Version

Idea

Use an iterator to go through the list, and remove element when found to be even

```
public static void removeEvensVer2(
                          List<Integer> lst) {
  for( Integer x : lst )
     if( x % 2 == 0 )
       lst.remove( x );
}
```

Runtime for removeEvensVer2(myArrayList): runtime error!
Runtime for removeEvensVer2(myLinkedList): runtime error!

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                              List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                         List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList):

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                                List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList): $O(N^2)$

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                              List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList): $O(N^2)$
Runtime for removeEvensVer3(myLinkedList):

In Detail: Third Version

Idea

Use the iterator's remove operation!

```
public static void removeEvensVer3(
                              List<Integer> lst) {
  Iterator<Integer> itr = lst.iterator();
  while(itr.hasNext())
    if(itr.next() % 2 == 0)
      itr.remove();
}
```

Runtime for removeEvensVer3(myArrayList): $O(N^2)$
Runtime for removeEvensVer3(myLinkedList): $O(N)$