COMP 352 – FALL 2021

Tutorial 5

Session Outline

- Array Lists:
  - Definition
  - Methods
- Node Lists:
  - Simply Linked List
  - The position ADT
  - The Node List ADT
  - Doubly Linked Lists
- The Sequence ADT
  - General Overview
  - Methods
  - Implementation
- Problem Solving

The Array List - Definition

An ***Array List*** stores its elements in a linear sequential way and supports access to these elements by their indices.

An ***Array List*** extends an array and is far more superior than it.

A ***Array List*** grows as items are added to it, it is searchable, sortable, etc.

The Array List ADT - Methods

- *get(i):* Return the element of S with index i; an error condition occurs if $i < 0$ or $i >$ size() $- 1$.
- *set(i,e):* Replace with e and return the element at index i; an error condition occurs if $i < 0$ or $i >$ size() $- 1$.
- *add(i,e):* Insert a new element e into S to have index i; an error condition occurs if $i < 0$ or $i >$ size().
- *remove(i):* Remove from S the element at index i; an error condition occurs if $i < 0$ or $i >$ size() $- 1$.
- *size():* Return the number of elements in the Array List.
- *isEmpty():* Return a Boolean indicating if the Array List is empty.

The Array List ADT - Implementation

Simple Array-Based Implementation Performance:

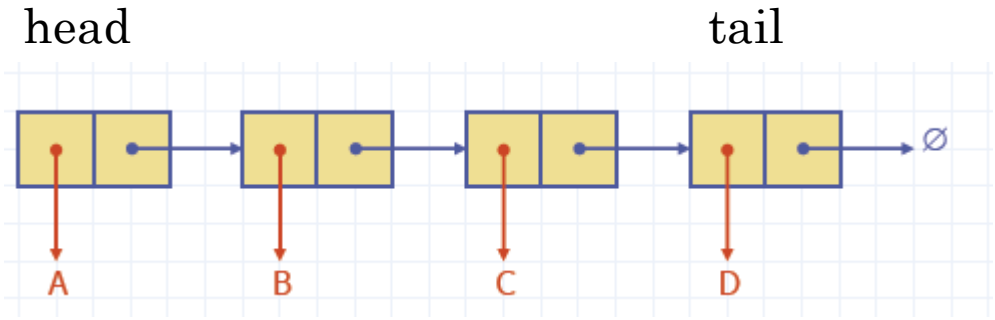
Method	Time
get(i)	O(1)
set(i,e)	O(1)
add(i,e)	O(n)
remove(i)	O(n)
size()	O(1)
isEmpty()	O(1)

Node Lists - Simply Linked List

A simply linked list is a sequence of nodes.

Each node in a simply linked list consists of:

- An element
- A link to the next node



Node Lists - The position ADT

The position ADT abstracts the notion of "place" in a node list.

A position is itself an abstract data type that supports the following simple method:

- *element():* Returns the element stored at this position.

A position is always defined ***relatively***, that is, in terms of its neighbors.

Node Lists - The Node List ADT (1)

The ***node list*** ADT is another type of sequence ADT that uses the concept of position to encapsulate the idea of "node" in a list. It supports the following methods:

- *first():* Return the position of the first element of *S*; an error occurs if *S* is empty.
- *last():* Return the position of the last element of *S*; an error occurs if *S* is empty.
- *prev(p):* Return the position of the element of *S* preceding the one at position *p*; an error occurs if *p* is the first position.
- *next(p):* Return the position of the element of *S* following the one at position *p*; an error occurs if *p* is the last position.

Node Lists - The Node List ADT (2)

The ***node list*** ADT also supports the following update methods:

- *set(p,e):* Replace the element at position *p* with *e*, returning the element formerly at position *p*.
- *addFirst(e):* Insert a new element *e* into *S* as the first element.
- *addLast(e):* Insert a new element *e* into *S* as the last element.
- *addBefore(p,e):* Insert a new element *e* into *S* before position *p*.
- *addAfter(p,e):* Insert a new element *e* into *S* before position p.
- *remove(p):* Remove and return the element at position *p* in S, invalidating this position in *S*.

Node Lists - Doubly Linked Lists (1)

A doubly linked list provides a natural implementation of the List ADT.

It is a sequence of nodes.

Every node consists of:

- An element.
- A link to the previous node
- A link to the next node

Node Lists - Doubly Linked Lists (2)

- A doubly linked list uses the Accessor methods (getNext and getPrev), as well as the Update methods (setNext, setPrev and setElement) to implement the Node List ADT methods (addFirst, addAfter, remove, etc.)

- Using a doubly linked list, we can perform all the methods of the list ADT in $O(1)$ time. Thus, a doubly linked list is an efficient implementation of the list ADT

The Sequence ADT - Introduction and Bridge Methods

- A Sequence is an ADT that provides explicit access to the elements in the list either by their indices or by their positions.
- The two bridging methods that provide the connection between indices (ranks) and positions are:
  - *atIndex(i) (or atRank(r)):* return the position of the element with index i; an error occurs if i<0 or i>size()-1.
  - *indexOf(p) (or rankOf(p)):* return the index of the element at position p.

The Sequence ADT - Implementations

Operation	Array	List
size, isEmpty	1	1
atRank, rankOf, elemAtRank	1	n
first, last, before, after	1	1
replaceElement, swapElements	1	1
replaceAtRank	1	n
insertAtRank, removeAtRank	n	n
addFirst, addLast	1	1
addAfter, addBefore	n	1
remove	n	1

Problem Solving - R-6.8

Give pseudo-code descriptions of algorithms for performing the `methods addBefore(p,e), addFirst(e), and addLast(e)` of the node list ADT, assuming the list is implemented using a doubly linked list.

Problem Solving - R-6.14

Suppose we are keeping track of access counts in a list L of n elements. Suppose further that we have made kn total accesses to the elements in L, for some integer $k \geq 1$. What are the minimum and maximum number of elements that have

been accessed fewer than k times?

Problem Solving - R-6.17

Briefly describe how to perform a new **sequence** method makeFirst(p) that moves an element of a sequence S at position p to be the first element in S while keeping the relative ordering of the remaining elements in S unchanged. That is, makeFirst(p) performs a move-to-front. Your method should run in $O(1)$ time if S is implemented with a doubly linked list.

Problem Solving - R-6.20

Suppose we are maintaining a collection C of elements such that, each time we add a new element to the collection, we copy the contents of C into a new array list of just the right size.

What is the running time of adding n elements to an initially empty collection C in this case?

Problem Solving - R-6.21

Describe an implementation of the methods addLast and addBefore realized by using only methods in the set {isEmpty, checkPosition, first, last, prev, next, addAfter, addFirst}.