COMP 250

Lecture 11

doubly linked lists

Jan. 31, 2022
Lists

- array list
- singly linked list
- doubly linked list
Doubly linked list

Each node in the list has a reference to the next node and to the previous node, and to an element object.
class DNode< E > {

    DNode< E >     next;
    DNode< E >     prev;
    E              element;

    // constructor

    DNode( E e ) {
        element = e;
        prev = null;
        next = null;
    }
}

Doubly linked list

As with a singly linked list, the doubly linked list uses a head and tail reference.

next  prev  element

null

head

null

tail

null
For a doubly linked list, removing the last element is fast.

```javascript
removeLast()
{
    size = size – 1
}
```
Unlike for a singly linked list, removing the last element of a doubly linked list is fast.

```java
removeLast()
{
    tail = tail.prev
    tail.next = null
    size = size - 1
}
```
Unlike for a singly linked list, removing the last element of a doubly linked list is fast.

```java
removeLast()
{
    e = tail.element
    tail = tail.prev
    tail.next = null
    size = size - 1
    return e
}
```
Suppose we want to access node i in a doubly linked list.

One issue is that edge cases (i = 0, i = size – 1) require special treatment: node 0 has a null prev field and node size-1 has a null next field.

We would like to avoid testing special cases for each method, since this is error prone.

For example, in the removeLast() method on the last slide, what if there was only one node? That code would not work. We forgot to adjust head!

[ADDED Feb. 10] Moreover, the instruction tail.next = null would cause a null pointer exception.]
Avoid edge cases with “dummy nodes”

These indices are not part of the list.
class DLinkedList<E> { // Java code

    DNode<E> dummyHead;
    DNode<E> dummyTail;
    int size;

    // constructor
    DLinkedList<E>(){
        dummyHead = new DNode<E>();
        dummyTail = new DNode<E>();
        dummyHead.next = dummyTail;
        dummyTail.prev = dummyHead;
        size = 0;
    }

    private class DNode<E>{ ... }
}
Q: How many objects in total in this figure?

A: \[1 + 6 + 4 = 11\]
Other List Operations

Many list operations require access to node i.

(This is so for singly linked lists also.)

: get(i) set(i,e) add(i,e) remove(i) :
get(i) { // returns the *element* at index i of list

}
get(i) {  // returns the element at index i of list

    return getNode(i).element

}  

getNode() is a helper method discussed on next slide

In Java, it would normally be a private method.
```cpp
getNode(i) { // helper, returns a DNode

// Omitting verification that 0 <= i < size

}
```

```plaintext
dummyHead ——>   null

i = 0

i = 1

i = 2

i = 3

dummyTail ——>   null
```
getNode(i) {  // returns a DNode

  // Omitting verification that 0 <= i < size

  node = dummyHead.next
  for (k = 0; k < i; k++)
    node = node.next
  return node
}
Ideas for how to speed this up?
getNode( i ) {  // returns a DNode

if ( i < size/2 ){  // iterate from head
    node = dummyHead.next
    for (k = 0; k < i; k ++)
        node = node.next  // exits loop when k==i
}
else{  // iterate from tail
    node = dummyTail.prev
    for (k = size-1; k > i; k --)  // exits loop when k==i
        node = node.prev
}
return node
}
```c
remove(i) {
    node = getNode(i)
}
```

**BEFORE**

```
i - 1
```

```
i
    node

```

```
i + 1
```

**AFTER**

```
? 
```
remove( i ) {
    node = getNode( i )
}

See exercises
Java LinkedList class

https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

It uses a *doubly linked list* as the underlying data structure.

It has some methods that ArrayList doesn’t have e.g.

- addFirst()
- removeFirst()
- addLast()
- removeLast()

Why ?
### Computational Complexity (N = list size)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array List</th>
<th>SListedList</th>
<th>DListedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFirst</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>removeFirst</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>addLast</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>removeLast</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>get(i)</td>
<td>$O(1)$</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

*Only if there is available space.*

*Worst case is $O(N)$.*

*Best cases are $O(1)$.*

*Worst cases are $O(N)$.*
Q: What is the time complexity of the following?

// Assume E is some actual type
// N is some constant

LinkedList< E > list = new LinkedList< E >();

for (k = 0; k < N; k++)
    list.addFirst( new E( .... ) );

A: \[ 1 + 1 + 1 + \ldots + 1 = N \quad \Rightarrow \quad O(N) \]

where ‘1’ means constant time, i.e. do instructions 1 time
Q: What is the time complexity of the following?

```java
// Let size == N
for (k = 0; k < list.size(); k++)
    list.get(k);
```

A: \[ 1 + 2 + 3 + \cdots + N = \frac{N(N+1)}{2} \Rightarrow O(N^2) \]
Java ‘enhanced for loop’

A more efficient way to iterate through elements in a Java LinkedList is to use:

```java
for (E e : list) { ... }
```

‘list’ references a LinkedList< E > object.

e is a local variable to the loop. It is of type E, namely the type of element in the linked list.

You can use e and list within the loop, but don’t modify list.
Java ‘enhanced for loop’

for (E e : list) {
    // do something
}

When E is a LinkedList, this is implemented roughly as

    node = head      // or write it using the dummyhead idea
while (node != null){
    e = node.element
    // do something with e
    node = node.next
}
What about “Space Complexity”?

We say all three data structures use space $O(N)$ for a list of size $N$. But linked lists use more than 2x (single) or 3x (double) as much space as arraylists.
How to “clone” a list i.e. make a copy?

```
LinkedList<Shape> list2 = list1.clone();
```

For technical reasons that I will discuss in a future lecture, you need to include a cast here:

```
(LinkedList)
```
“Shallow copy”

The list object and the list nodes are copied. But the Shape objects are not copied.
LinkedList<T>.clone() makes a shallow copy.

Returns a shallow copy of this LinkedList. (The elements themselves are not cloned.)

Overrides:
- clone in class Object

Returns:
a shallow copy of this LinkedList instance
Next week you will understand why this says Object rather than LinkedList. This is the reason that we need to cast, as I mentioned two slides ago.
“Deep copy”

The linkedlist object, the list nodes, and the list elements are all copied. The Java LinkedList class does not have a built-in method to make a deep copy.
Real Example – Shallow Copy

Suppose have a list of midterm exams for a course. The exams need to be graded by hand.

Each grader (TA) is responsible for grading certain questions. So each grader will have a list of exams, and will write on each of exams.

Each grader needs a shallow copy of the list of exams.

For this example, we don’t care if it is a linked list or array list.
Real Example – Deep Copy

Suppose you have a list of job applications, which will be examined by different people in a company. Suppose the employer wants independent assessment of applications by different people.

Each person assessing the applications will mark up the PDF of each application.

Each assessor needs a deep copy of the list of applications. They should not be allowed to see each other’s assessments.
## Coming up...

### Lectures

<table>
<thead>
<tr>
<th>Day</th>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed.</td>
<td>Feb. 2</td>
<td>Quadratic Sorting i.e. $O(N^2)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- bubble sort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- selection sort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- insertion sort</td>
</tr>
<tr>
<td>Fri.</td>
<td>Feb. 4</td>
<td>Object Oriented Design 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Inheritance)</td>
</tr>
</tbody>
</table>

### Assessments

- Assignment 1
  - due on Friday, Feb. 11