Lecture 5: Linked Lists

Bohyung Han
CSE, POSTECH
bhhan@postech.ac.kr

Linked List

- What is the linked list?
  - A data structure consisting of a sequence of nodes
  - Each node is composed of data and link(s) to other nodes.

- Properties
  - Linked lists can be used to implement several other common abstract
data types, such as stacks and queues
  - The elements can easily be inserted or removed without reallocation
    or reorganization of the entire structure
  - Linked lists allow insertion and removal of nodes at any point in the list
    in constant time.
  - Simple linked lists do not allow random access to the data.
  - http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

Singly Linked Lists

- A singly linked list is a concrete data structure consisting of a
  sequence of nodes
- Each node stores
  - Element(s)
  - Link to the next node

The Node Class for Singly Linked List

```java
// Node class
public class Node {
    private Object element;
    private Node next;

    // Constructors
    public Node() {
        this(null, null);
    }

    public Node(Object e, Node n) {
        element = e;
        next = n;
    }
}
```

Defines element and link

Creates a node with null, which references
to its element and next node

Creates a node with given element
and next node
The Node Class for Singly Linked List

```java
public Object getElement() {
    return element;
}
public Node getNext() {
    return next;
}
public void setElement(Object newElem) {
    element = newElem;
}
public void setNext(Node newNext) {
    next = newNext;
}
```

### Accessor methods

- `getElement()`: Returns the element
- `getNext()`: Returns the next node

### Modifier methods

- `setElement(Object newElem)`: Sets the element
- `setNext(Node newNext)`: Sets the next node

---

Inserting a Node at the Head

- **Operation sequence**
  - Allocate a new node.
  - Insert new element.
  - Have new node point to old head.
  - Update head to point to new node.

Node v = new Node();
v.setElement("A");
v.setNext(head);
head = v;

---

Removing a Node from the Head

- **Operation sequence**
  - Update head to point to next node in the list.
  - Allow garbage collector to reclaim the former first node.

head = head.getNext();

- **Java: garbage collection**
  - Unused memory is reclaimed automatically for reuse.
  - No need to free or delete

---

Inserting a Node at the Tail

- **Operation sequence**
  - Allocate a new node.
  - Insert new element.
  - Have new node point to null.
  - Have old last node point to new node.
  - Update tail to point to new node.

Node v = new Node();
v.setElement("A");
v.setNext(null);
tail.setNext(v);
tail = v;
Removing a Node at the Tail

- **Operation sequence**
  - Move tail to the second last node.
  - Have tail node point to null.
  - Allow garbage collector to reclaim the former last node.
- **Issues**
  - There is no constant time way to make the tail to point to the previous node.
  - Removing at the tail of a singly linked list is not efficient!

Stack as a Linked List

- We can implement a stack with a singly linked list.
- The top element is stored at the first node of the list.
- The space used is $O(n)$ and each operation of the Stack ADT takes $O(1)$ time.

Queue as a Linked List

- We can implement a queue with a singly linked list.
  - The front element is stored at the first node.
  - The rear element is stored at the last node.
- The space used is $O(n)$ and each operation of the Queue ADT takes $O(1)$ time.

Node List ADT

- The Node List ADT models a sequence of positions storing arbitrary objects.
- It establishes a before/after relation between positions.
- **Methods**
  - Generic methods: `size()`, `isEmpty()`
  - Accessor methods
    - `first()`, `last()`
    - `prev(p)`, `next(p)`
  - Modifier methods:
    - `set(p, e)`
    - `addBefore(p, e)`, `addAfter(p, e)`,
    - `addFirst(e)`, `addLast(e)`
    - `remove(p)`
Doubly Linked List

• A doubly linked list provides a natural implementation of the Node List ADT.
• Nodes implement position and store:
  ▪ Element
  ▪ Link to the previous node
  ▪ Link to the next node
• Special trailer and header nodes

Insertion

• Insertion operation
  ▪ We visualize `insertAfter(p, X)`, which returns position q.

**Algorithm** `addAfter(p, e)`:  
  Create a new node v  
  v.setElement(e)  
  v.setPrev(p)  
  v.setNext(p.getNext())  
  (p.getNext()).setPrev(v)  
  p.setNext(v)  
  return v  
  //link v to its predecessor  
  //link v to its successor  
  //link p’s old successor to v  
  //link p to its new successor, v  
  //the position for the element e
Deletion

- Deletion in doubly linked list
  - We visualize $\text{remove}(p)$, where $p = \text{last()}$.

Deletion Algorithm

**Algorithm** remove$(p)$:

```java
int t = p.element // a temporary variable to hold the return value
(p.getPrev()).setNext(p.getNext()) // linking out p
(p getNext()).setPrev(p.getPrev())
p.setPrev(null) // invalidating the position p
p.setNext(null)
return t
```