CSCI 112
Monday, July 26, 2010

Linked Lists

Using a Linked List to Create a Stack and Queue

```
class Stack<T> extends LinkedList{
        public Stack{ super(); }
        public void push(T data){ super.insert(0, data); }
        public T pop(){ return super.remove(0); }
        public T peek{ return super.get(0); }
}
```

A Stack is not a Linked List:

- A Linked List provides additional functionality that a Stack would not be able to do

```
class Stack<T> {
   private LinkedList<T> elements;

   public Stack() { elements = new LinkedList<T>(); }
   public boolean isEmpty() { return elements.size() == 0; }
   public boolean isFull() { return false; }
   public int size() { return elements.size(); }
   public void push(T data) { elements.insert(0, data); }
   public T pop() { return elements.remove(0); }
   public T peak() { return elements.get(0); }
}
```

Doubly-Linked Lists

Let's review the basic interpretation of a Node:

- A data field.
- A node reference pointing to the next data node (or this is null if there is no next data field)
- A double linked list requires an additional field to the idea of a Node: one node reference pointing to the element that should go before this node, or null if there is no previous node.

The major downside to a doubly linked list is that insertions and deletions (which are already complicated in the singly linked list implementation) are now even more complicated.

The upside to a doubly linked list is speed. In addition to having a Node link always point to the front of the list (or “head”), there is also a Node link pointing to the back of the list (or “tail”). If you have a N elements (where N is a very large number) in a singly linked list, it takes N operations to get the value of the last node. In a doubly linked list, the operation time is O(1) to grab the last element.

For most algorithms requiring a set of data that grows, the operations of insertion and removal happen on the end extremities. By having two variables “head” and “tail”, we can make insertion and removal on the list ends have a run time of O(1). For insertion and removal operations that happen somewhere in the middle of this list, this still requires an O(N) algorithm.

Circular Doubly Linked Lists

In a Circular Doubly Linked List, the first node's previous field (typically set to null) is set to the last node. Likewise, the last node's next field (typically null) is set to the first node. The result is a list with no definitive head or tail.

Node Lists

– A variation on Linked Lists designed for speed

Node.java

```
class Node<T>{
	private Node<T> prev;
	private Node<T> next;
	private T data;

	public Node(Node<T> prev, Node<T> next, T data){
		this.prev = prev;
		this.next = next;
		this.data = data;
	}

	public T get(){
		if( prev == null && next == null ){
			System.out.println( "Error: element does not exit." );
			System.exit(1);
		}
		return data;
	}
	public Node<T> getNext(){
		return next;
	}
	public Node<T> getPrev(){
		return prev;
	}
	public void setNext(Node<T> next){
		this.next = next;
	}
	public void SetPrev(Node<T> prev){
		this.prev = prev;
	}
	public void set(T data){
		this.data = data;
	}
}
```

Node.java

```

class NodeList<T>{
	private Node<T> head;
	private Node<T> tail;
	private int size;

	public NodeList(){
		size = 0;
		head = new Node<T>(null, null, null);
		tail = new Node<T>(null, null, null);
		tail.setPrev(head);
		head.setNext(tail);
	}

	public Node<T> first(){
		return head;
	}
	public Node<T> last(){
		return tail;
	}
	public int size(){
		return size;
	}
	public int isEmpty{
		return size == 0;
	}
	public Node<T> prev(Node<T> p){
		return p.getPrev();
	}
	public Node<T> next(Node<T> n){
		return n.getNext();
	}

	//update methods

	//set -> sets data in a node
	//addFirst -> sets a new node at the beginning
		// of the list. the user is responsible for making their own node
	//addLast -> sets new node as last element
	//addBefore -> takes two arguments: a node in the list and a node not
```

```
		// in the list.  The node not in the list is added before the node
		// in the list.
	//addAfter -> takes two arguments: a node in the list and a node not
		// in the list.  The node not in the list is added after the node
		// in the list.
	//remove -> remove a node
```

Array Lists

- An array list is also known as a vector
- whereas Linked Lists are memory efficient, they are slow
- An array list is fast, but has poor memory usage.
- An array list is implemented with a standard array

Theory Behind and Array List

- The perceived size to the user
- The actual size known to the data structure

Three Fields

- size (perceived size)(starts at 0)
- real-size (actual size of an internal array)(starts at 2)
- an array of elements

ArrayList.java

```
class ArrayList<T>{
	private int size;
	private int real_size;
	private T[] elements;

	public ArrayList(){
		size = 0;
		real_size = 2;
		elements = (T[])new Object[real_size];
	}

	private void resizeArray(){
		real_size *= 2;
		T[] newArray = (T[])new Object[real_size];
		for( int i = 0; i < elements.length; i++ ){
			newArray[i] = elements [i];
		}
		elements = newArray;
	}
	public boolean isEmpty(){
		return size == 0;
	}
	public int size(){
		return size;
	}
	public T get(int pos){
		if( pos < 0 || pos >= size ){
			System.out.println( "Error: Array Out Of Bounds: " + pos );
			System.exit(1);
		}
		return elements[pos];
	}
	public void set(int pos, T data){
		if( pos < 0 || pos >= size ){
			System.out.println( "Error: Array Out Of Bounds: " + pos );
			System.exit(1);
		}
		elements[pos] = data;
	}
	public T remove(int pos){
		if( pos < 0 || pos >= size ){
			System.out.println( "Error: Array Out Of Bounds: " + pos );
			System.exit(1);
		}
		T temp = elements[pos];
		for( int i = pos; i < size-1; i++ ){
			elements[i] = elements[i+1];
		}
		size--;
		return temp;
	}
	public void insert(T data, int pos){
		if( pos < 0 || pos > size ){
			System.out.println( "Error: Array Out Of Bounds: " + pos );
			System.exit(1);
		}
		if( size = real_size ){
			resizeArray();
		}
		for( int i = size-1; i >= pos; i-- ){
			elements[i+1] = elements[i];
		}
		elements[pos] = data;
		size++;
	}
}
```