

Data Structures and Algorithms I

Linked Lists

Dr. Tim Margush
University of Akron
© 2008

Goals

- Be able to describe and implement a linked list
- Know the efficiency of linked list operations
- Understand the relative advantages of linked, doubly-linked, and circular-linked lists

© 2008 Dr. Tim Margush

Linked vs Array

| | |
|---|--|
| <ul style="list-style-type: none">• Elements scattered through memory<ul style="list-style-type: none">– Easily expands• Not random access<ul style="list-style-type: none">– Cannot calculate the location of an item• Easy to insert and remove | <ul style="list-style-type: none">• Elements in consecutive memory locations<ul style="list-style-type: none">– Requires one large and fixed size block• Random access• Inefficient to insert and remove |
|---|--|

© 2008 Dr. Tim Margush

Linked List Structure

- Build from Nodes
 - Each Node has
 - A data value
 - A reference to another Node
- One Node is the head of the list
 - All nodes are accessed starting from the head
- The last node (end of list) is identified by a null reference in its next field


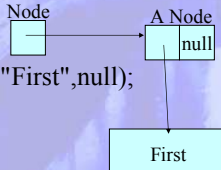
© 2008 Dr. Tim Margush

A Basic Node

```
public class<E> Node{  
    private E data;  
    private Node<E> next;  
    public Node(E d, Node<E> n){  
        data = d;  
        next = n;  
    }  
}
```

© 2008 Dr. Tim Margush

Two Simple Lists

- `Node<String> empty = null;`

- `Node<String> single = new Node<String>("First",null);`


© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    public String toString(){  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    public LinkedList(LinkedList<? extends E> a){  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    public Object[] toArray(){  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    public <T> T[] toArray(T[] t){  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    public void addToEnd(E e){  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    public E removeFromEnd(E e){  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E> {  
    Node<E> head = null;  
    public E get(int i) {  
  
  
  
  
  
  
  
  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E> {  
    Node<E> head = null;  
    public void set(int i, E e) {  
  
  
  
  
  
  
  
  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E> {  
    Node<E> head = null;  
    public void add(int i, E e) {  
  
  
  
  
  
  
  
  
  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E> {  
    Node<E> head = null;  
    public E remove(int i) {  
  
  
  
  
  
  
  
  
  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E> {  
    Node<E> head = null;  
    public int indexOf(E e) {  
  
  
  
  
  
  
  
  
  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E> {  
    Node<E> head = null;  
    public boolean remove(E e) {  
  
  
  
  
  
  
  
  
  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    public Iterator<E> iterator() {  
  
  
  
  
  
  
  
  
  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    private class LLIterator<E>{  
        private Node<E> current = head;  
        public boolean hasNext(){  
  
  
  
  
  
  
  
  
  
        }  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{  
    Node<E> head = null;  
    private class LLIterator<E>{  
        private Node<E> current = head;  
        public E next(){  
  
  
  
  
  
  
  
  
  
        }  
    }  
}
```

© 2008 Dr. Tim Margush

A LinkedList Class

```
class LinkedList<E>{
    Node<E> head = null;
    private class LLIterator<E>{
        private Node<E> theNext = head;

        public void remove(){

        }
    }
}
```

© 2008 Dr. Tim Margush

Improvements on Linked Lists

- Head and Tail Pointer
- List Header Node
- Doubly-Linked List
- Circular Lists
 - You can treat these separately, but the improvements are only noticed if some or all are employed at the same time

© 2008 Dr. Tim Margush

Head and Tail Pointer

- ```
class LLHT {
 Node<E> head=null, tail=null;
```
- References to first and last nodes are both maintained
  - Allows efficient implementation of getLast() and addToEnd()
    - Unfortunately removeFromEnd is not helped

© 2008 Dr. Tim Margush

---

---

---

---

---

---

---

---

## List Header Node

```
class LLH{
 Node<E> header =
 new Node<E>(null, null);
```

- Eliminates special case for operations at the front of the list
- Simplifies iterator implementation as every node in the list has a previous node

© 2008 Dr. Tim Margush

---

---

---

---

---

---

---

---

## Doubly-Linked List

```
class Node<E>{
 Node<E> next, prev;
```

- Allows traversal in either direction
  - Simplifies access to previous for inserts and removes
  - Combined with tail, allows efficient removeFromEnd

© 2008 Dr. Tim Margush

---

---

---

---

---

---

---

---

## Circular-Linked List

- Last node references first
  - Eliminates all null references
  - Combined with list header node, eliminates all special cases
  - Combined also with double-linked makes insert and remove at either end equally efficient with no special cases

© 2008 Dr. Tim Margush

---

---

---

---

---

---

---

---

## Efficiency of Linked Lists

- Single-linked vs Arrays
  - Slightly more memory than a full array
  - No random access
  - Insert and remove is very efficient once the location has been found
    - Adds/Removes from both ends can be made very efficient
      - Arrays are efficient for adding/removing at one end only
  - Size expands as needed

© 2008 Dr. Tim Margush

---

---

---

---

---

---

---

---

## Summary

- Linked list structure provides
  - Efficient sequential access
  - Efficient add and remove
    - Once the insertion point is located
  - Efficient memory use
    - No large contiguous blocks required
  - Terrible random access

© 2008 Dr. Tim Margush

---

---

---

---

---

---

---

---