

Lecture 7: Jun. 14, 2015

*Lecturer: Anwar Mamat***Disclaimer:** *These notes may be distributed outside this class only with the permission of the Instructor.*

7.1 Singly Linked List

A linked list is a data structure consisting of a group of nodes which together represent a sequence. Each node is composed of a data and a reference (in other words, a link) to the next node in the sequence. A Node class usually look like this:

Listing 1: Singly Linked List Node

```

1 class Node<E> {
2     public E data;
3     public Node<E> next;
4     Node(E item){
5         data = item;
6     }
7 }
```

Usually Node class is nested inside the LinkedList class, and members of Node are private.

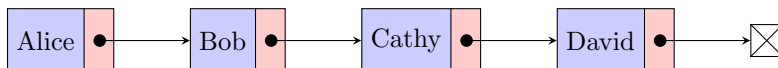
7.1.1 Create a simple linked list

Now, let us create a simple linked list.

```

1 Node<String> n1 = new Node(" Alice" );
2 Node<String> n2 = new Node("Bob" );
3 Node<String> n3 = new Node(" Cathy" );
4 Node<String> n4 = new Node(" David" );
5 n1.next = n2;
6 n2.next = n3;
7 n3.next = n4;
```

This linked list represents this:



7.1.2 Display the Linked List

We can display all the linked list:

```

1 Node<String> current = first;
2 while( current != null){
3     System.out.println( current.data );
4     current = current.next;
5 }

```

7.1.3 Insert a node

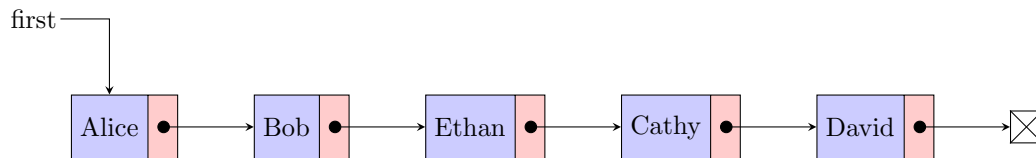
Now, let us insert a node between “Bob” and “Cathy”.

```

1 Node<String> n5 = new Node("Ethan");
2 n5.next = n2.next;
3 n2.next = n5;
4 //use "first" to reference the first node of the list.
5 Node<String> first = n1;

```

This linked list represents this:



7.1.4 Delete a node

7.1.4.1 Delete the first node

To delete the first node, we can simply move “first” to next node.

```

1 first = first.next;

```

7.1.4.2 Delete other nodes

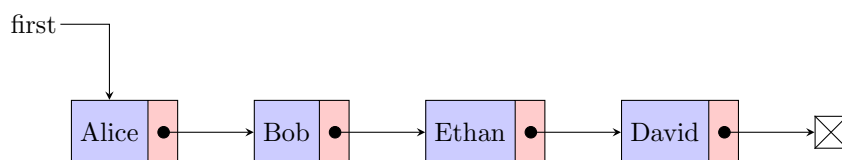
In order to delete a Node, we have to know the parent of the node. Assume “parent” references the node “Ethan”, to delete the node “Cathy” reference by “current”, we can do this:

```

1 parent.next = current.next;

```

No, we have:



7.2 Linked List Class

```

1  /**
2   *  The Bag class represents a collection of generic items.
3   *  It supports insertion and iterating over the items in arbitrary order.
4   */
5
6  import java.util.ArrayList;
7  import java.util.Iterator;
8
9  public class LinkedBag<E> extends Comparable<E>> implements Iterable<E>
10 {
11     protected int N; //number of items in the bag
12     private Node<E> first; //beginning of bag
13
14     // helper linked list class
15     private class Node<E> {
16         private E data;
17         private Node<E> next;
18         Node(E item){
19             data = item;
20         }
21     }
22
23     /**
24     *  Initializes an empty bag.
25     */
26     public LinkedBag() {
27         first = null;
28         N = 0;
29     }
30
31     /**
32     *  Returns an iterator that iterates through the items in the bag
33     *  @return an iterator that iterates through the items in the bag
34     */
35     public Iterator<E> iterator() {
36         return new BagIterator(first);
37     }
38
39     /**
40     *  The iterator implementation
41     */
42     private class BagIterator implements Iterator<E> {
43         private Node<E> current = null;
44         public BagIterator(Node<E> first) {
45             current = first;
46         }
47         public boolean hasNext() { return current != null;}
48         public void remove() { System.out.println("to_be_implemented."); }
49     }
50
51     public E next() {
52         if (!hasNext()) {return null;}
53     }
54 }

```

```

49         E item = current.data;
50         current = current.next;
51         System.out.println("work");
52         return item;
53     }
54 }
55 }
56
57 /**
58  * Adds the item to this bag.
59  * @param item the item to add to this bag
60  */
61 public void insert(E item) {
62     Node<E> oldfirst = first;
63     first = new Node<E>(item);
64     first.next = oldfirst;
65     N++;
66 }
67
68 /**
69  * Returns an item by index
70  * @param index is the item index
71  */
72 public E get(int index)
73 {
74     Node<E> current = first;
75     int i = 0;
76     while(current != null && i < index){
77         current = current.next;
78         i++;
79     }
80     if(current != null){
81         return current.data;
82     }else{
83         return null;
84     }
85 }
86 /**
87  * Deletes an item
88  * @param item is the item to be deleted
89  * @return true if item is deleted. false otherwise
90  */
91 public boolean remove(E item)
92 {
93     Node<E> current = first;
94     Node<E> parent = first;
95     while(current != null){
96         if(current.data.equals(item)){
97             if(current == first){
98                 first = first.next; //remove first node
99             }else{

```

```

100         parent.next = current.next; //remove non-first node
101     }
102     return true;
103 }
104     parent = current;
105     current = current.next;
106 }
107     return false;
108 }
109
110 /**
111  * Is this bag empty?
112  * @return true if this bag is empty; false otherwise
113  */
114 public boolean isEmpty() {
115     return first == null;
116 }
117
118 /**
119  * Returns the number of items in this bag.
120  * @return the number of items in this bag
121  */
122 public int size() {
123     return N;
124 }
125
126 /**
127  * if the bag contains a given item?
128  * @return true if bag contains the item. false otherwise
129  */
130 public boolean contains(E item)
131 {
132     Node<E> current = first;
133     while(current != null){
134         if(current.data.equals(item)) return true;
135         current = current.next;
136     }
137     return false;
138 }
139 }
140 }

```

7.2.1 Test the Linked List

```

1 /**
2  * test Linked List Bag
3  */
4 public class LinkedBagUnitTest {
5     public static void main(String[] args){

```

```

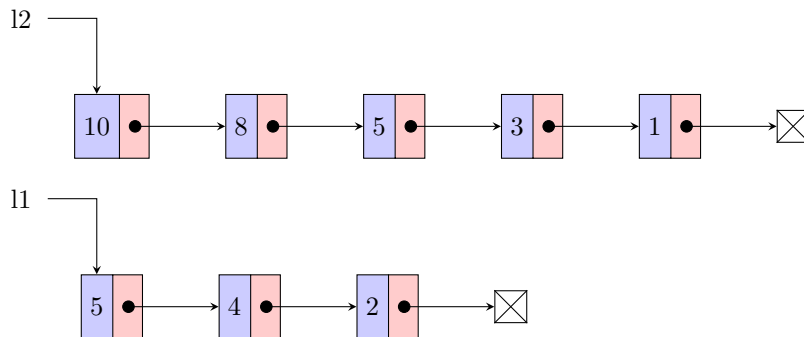
6      LinkedBag<Integer> bag = new LinkedBag();
7      for(int i=1; i <= 3; i++){
8          bag.insert(i);
9      }
10     System.out.println("Size="+bag.size());
11     if(bag.contains(3)){
12         System.out.println("Bag_contains_3");
13     }else{
14         System.out.println("Not_Found");
15     }
16     //print all items using iterator
17     for(Integer i:bag){
18         System.out.print(i + ",");
19     }
20     //print all items using get method, which is not efficient.
21     System.out.println("\n_all_items");
22     for(int i = 0; i < bag.size(); i++){
23         System.out.print(bag.get(i)+" ,");
24     }
25 }
26 }

```

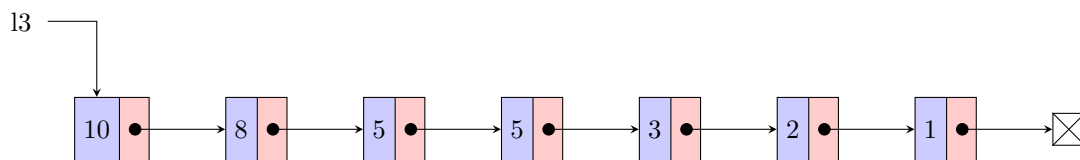
7.3 Code example

7.3.1 Merge two sorted linked list into one

We have two sorted linked lists *list1* and *list2*.



We want to generate the list:



Here is the code that takes two lists as input, and merges them into one list. This function takes $O(n1 + n2)$ time to merge two lists of size $n1$ and $n2$.

```
1 public static Node merge(Node l1 , Node l2){
2
3     //if one list is empty, return the other list
4     if(l1 == null){
5         return l2;
6     }
7     if(l2 == null){
8         return l1;
9     }
10    //if both lists are not empty
11    Node c1 = l1;
12    Node c2 = l2;
13    Node m = null;
14    //pick the larger node from l1 and l2.
15    if(c1.data > c2.data){
16        m = c1;
17        c1 = c1.next;
18    }else{
19        m = c2;
20        c2 = c2.next;
21    }
22    /**
23     *      walk through l1 and l2, every time pick the larger node.
24     *      comparison only occurs at the head of two lists.
25     */
26    Node c3 = m;
27    while(c1 != null && c2 != null){
28        if(c1.data > c2.data){
29            c3.next = c1;
30            c1 = c1.next;
31            c3 = c3.next;
32        }else{
33            c3.next = c2;
34            c2 = c2.next;
35            c3 = c3.next;
36        }
37    }
38
39    if(c1 != null){
40        c3.next = c1;
41    }else{
42        c3.next = c2;
43    }
44
45    return m;
46 }
```