Linked lists

Ankit Pangasa

Question: What are advantages and disadvantages of linked lists?

Advantages of linked lists in Java:

- Linked lists have constant-time insertion and deletion operation in any position. Arrays require O(n)O(n) time to do the same thing, because you'd have to "shift" all the subsequent items over 1 index.
- Linked lists can continue to expand as long as there is space on the machine. Arrays (in low-level languages) must have their size specified ahead of time.Even if array is dynamic like array list that automatically resize themselves when they run out of space. the operation to resize a dynamic array has a large cost which can make a single insertion unexpectedly expensive.

Disadvantages of linked lists in Java:

> To access or edit an item in a linked list, you have to take O(i)O(i) time to walk from the head of the list to the iith item (unless of course you already have a pointer directly to that item). Arrays have constant-time lookups and edits to the iith item.

Question: How many types of Linked lists exists?

Java Linked lists can be categorized into three types:

Single linked: lists have a single pointer pointing to the next element in the list. The last pointer is empty or points to null, meaning the end of the list.

Doubly linked: lists have two pointers, one pointing to the next element and one pointing to the previous element.

The head node's previous pointer points to null and the tail node's next pointer points to null to signal the end of the list.

Circular linked: lists usually represent buffers. They have no head or tail, and the primary issue is avoiding infinite traversals because of the cycle.

Arrays are oftentimes a better substitute for a circular linked list, using the modulus operator to wrap around.

Question: How to find if linked list has a loop?

Answer: If we maintain two pointers, and we increment one pointer after processing two nodes and other after processing every node.

We are likely to find a situation where both the pointers will be pointing to same node. This will only happen if linked list has loop.

Question: What is difference between Singly Linked List and Doubly Linked List in Java?

The Main difference between singly linked list and doubly linked list is ability to traverse.

In a single linked list, node only points towards next node, and there is no pointer to previous node. Which means you can not traverse back on a singly linked list.

On the other hand doubly linked list maintains two pointers, towards next and previous node, which allows you to navigate in both direction in any linked list.

Question: How to find 3rd element from end in a linked list in one pass?

If we apply a trick of maintaining two pointers and increment other pointer, when first has moved up to 3rd element.

Than when first pointer reaches to the end of linked list, second pointer will be pointing to the 3rd element from last in a linked list.

Question: Which interfaces are implemented by Linked List in Java?

Following interfaces are implemented by Java Linked lists

`Serializable, Queue, List, Cloneable, Collection, Deque, Iterable`

Question: What is the package name for Linked List class in Java?

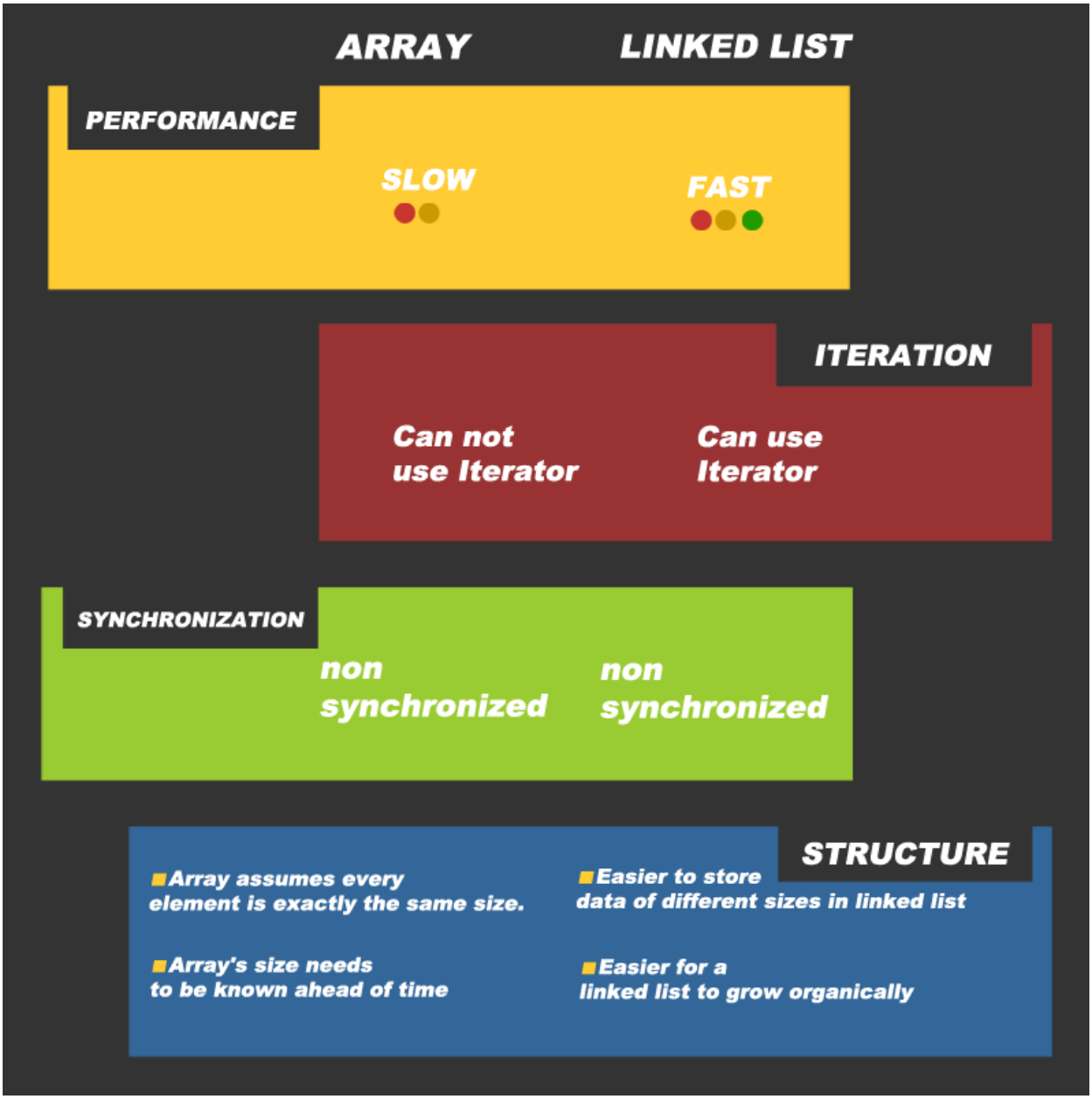
`java.util`

Question: What class is the Parent Class of LinkedList class?

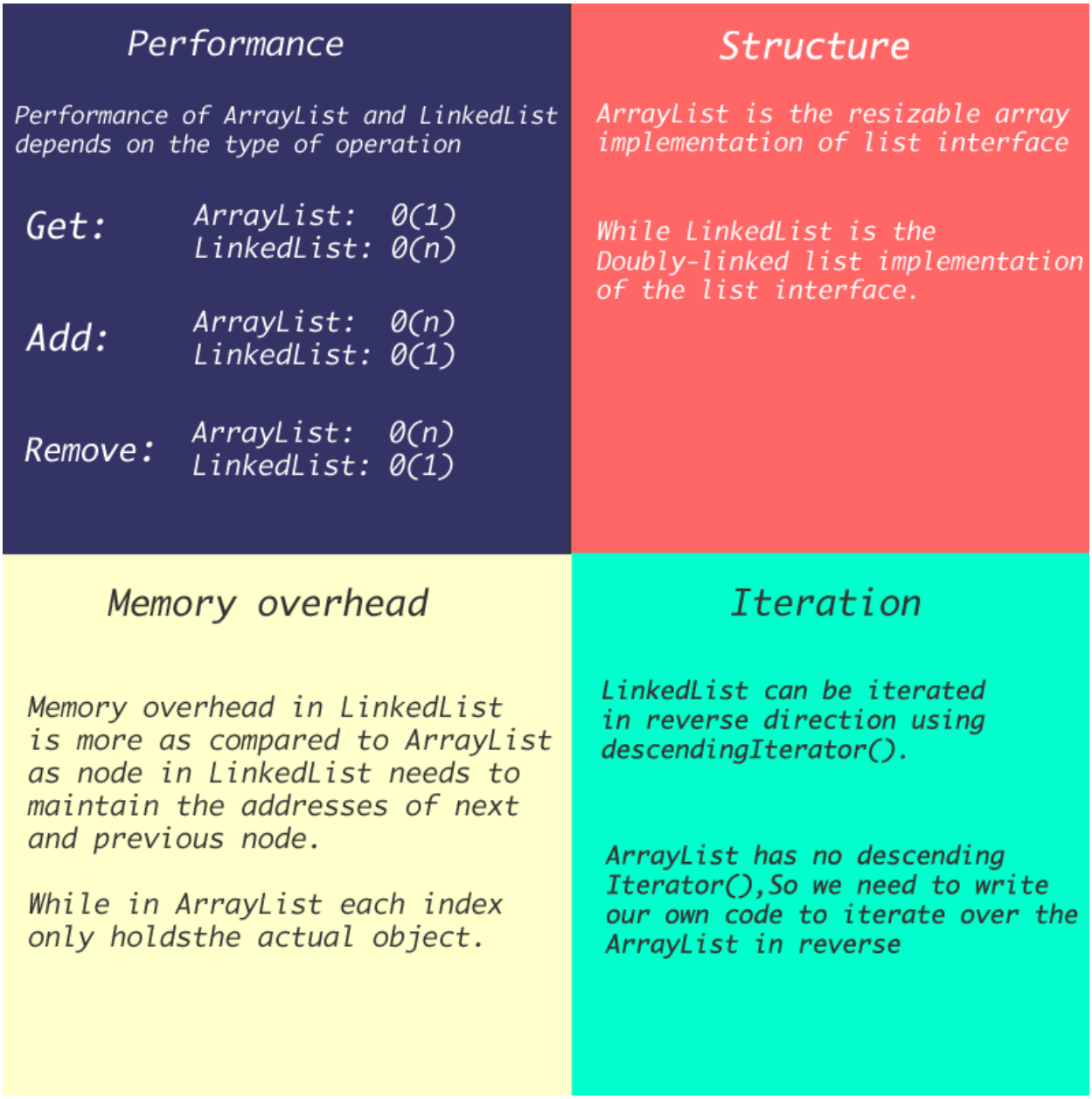
```
java.util.AbstractSequentialList
```

Java Linked List Vs Other Data Structures

Java Linked List Vs Array



Java Linked List Vs Array List



Java Linked List Algorithm Questions

Question: Find union and Intersection of two Linked Lists in Java? [Microsoft]

The Solution:

In order to find intersection of two linked lists, we will create a new linked lists and copy all the elements which are same in both linked lists. Order of elements in new linked list does not matter.

And how exactly we will achieve that ?

Well there are two different ways

Algorithm 1

Create a new linked list called result and Initialize as NULL. Traverse list1 and look for its each element in list2, if the element is present in list2, then add the element to result. **Time complexity:** Time Complexity for this algorithm is **O(mn) m is the number of elements in first list n is the number of elements in second list**

Algorithm 2:

The third algorithm for the implementation of this problem is very simple, here are steps

1. Create a new **HashMap.**
2. Traverse **second linked list** and store data of each node in a map as a key and false (Boolean) as a value.
3. Traverse **first linked list,** check if data (data stored in the node) is present in a map (as a key) and its value is false. if it is present then we create a new node with that data, add it to the new linked list and update its value to true in map. Otherwise continue.

Algorithm3**:**

In this method we will use merge sort.

Following are the steps to be followed to get intersection of lists.

1) Sort the first Linked List using merge sort. This step takes O(mLogm) time.2) Sort the second Linked List using merge sort. This step takes O(nLogn) time. 3) Linearly scan both sorted lists to get the union and intersection. This step takes O(m + n) time.

Time complexity:

Time complexity of this method is O(mLogm + nLogn) which is better than method 1's time complexity.

Code Implementation Algorithm 1:

```
// Java program to find union and intersection of two unsorted linked lists
class LinkedList
```

```
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    void getIntersection(Node head1, Node head2)
    {
        Node result = null;
        Node t1 = head1;

        // Traverse list1 and search each element of it in list2.
        // If the element is present in list 2, then insert the
        // element to result
        while (t1 != null)
        {
            if (isPresent(head2, t1.data))
                push(t1.data);
            t1 = t1.next;
        }
    }

    /* Utility function to print list */
    void printList()
    {
        Node temp = head;
        while(temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }

    /*  Inserts a node at start of linked list */
    void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }
```

```
    /* A utilty function that returns true if data is present
       in linked list  else return false */
    boolean isPresent (Node head, int data)
    {
        Node t = head;
        while (t != null)
        {
            if (t.data == data)
                return true;
            t = t.next;
        }
        return false;
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist1 = new LinkedList();
        LinkedList llist2 = new LinkedList();
        LinkedList unin = new LinkedList();
        LinkedList intersecn = new LinkedList();

        /*create a linked lits 10->15->5->20 */
        llist1.push(20);
        llist1.push(4);
        llist1.push(15);
        llist1.push(10);

        /*create a linked lits 8->4->2->10 */
        llist2.push(10);
        llist2.push(2);
        llist2.push(4);
        llist2.push(8);

        intersecn.getIntersection(llist1.head, llist2.head);
        unin.getUnion(llist1.head, llist2.head);

        System.out.println("First List is");
        llist1.printList();

        System.out.println("Second List is");
        llist2.printList();

        System.out.println("Intersection List is");
        intersecn.printList();
    }
}
```

Code Implementation Algorithm 2:

1) Create a class **Node**, this class will be a node of our linked list.

```
 public class Node {

  private String data;
  private Node next;

  public String getData() {
    return data;
  }

  public void setData(String data) {
    this.data = data;
  }

  public Node getNext() {
    return next;
  }

  public void setNext(Node next) {
    this.next = next;
  }

}
```

2) Create a class CreateLinkedList, this class has a method createList which takes string array as an argument and return linked list.

```
public class CreateLinkedList {

  public Node createList(String[] dataArr){
    Node listHead = null;
    Node node = null;
    Node tempList = null;
    for(String data : dataArr){
      node = new Node();
      node.setData(data);
      if(listHead == null){
        listHead = node;
        tempList = node;
      }else{
        tempList.setNext(node);
        tempList = node;
      }
    }
    return listHead;
  }

}
```

3) Create a class IntersectionLinkedList, this class has a method intersection which takes two linked list as an argument and return intersection linked list.

```
 import java.util.HashMap;
import java.util.Map;

public class IntersectionLinkedList {

   private Node intersectionListHeadNode;

   private Map<String,Boolean> getAllDataOfList(Node list){
     Map<String,Boolean> uniqueMap = new HashMap<String,Boolean>();
     while(list != null){
       uniqueMap.put(list.getData(),false);
       list = list.getNext();
     }
     return uniqueMap;
   }

   public Node intersection(Node firstList, Node secondList){
     Map<String,Boolean> uniqueMap = getAllDataOfList(secondList);
     addList(firstList,uniqueMap);
     return this.intersectionListHeadNode;
   }

   private void addList(Node sourceList,Map<String,Boolean> uniqueMap){
     Node tempNode = null;
     Node tempList = null;
     while(sourceList != null){
       if(uniqueMap.containsKey(sourceList.getData()) &&
!uniqueMap.get(sourceList.getData())){
         uniqueMap.put(sourceList.getData(), true);
         tempNode = createNode(sourceList.getData());
         if(tempList == null){
           tempList = tempNode;
           this.intersectionListHeadNode = tempList;
         }else{
           tempList.setNext(tempNode);
           tempList = tempNode;
         }
       }
       sourceList = sourceList.getNext();
     }
   }

   private Node createNode(String data){
     Node node = new Node();
     node.setData(data);
     return node;
   }
}
```

4) Create a class Main, this class will be our demo class.

```
public class Main {

   public static void main(String[] args) {
     CreateLinkedList createLinkedList = new CreateLinkedList();
     String[] firstListData = {"a","c","c","d"};
     Node firstList = createLinkedList.createList(firstListData);
     String[] secondListData = {"c","d","e","a"};
     Node secondList = createLinkedList.createList(secondListData);

     IntersectionLinkedList intersectionLinkedList = new IntersectionLinkedList();
     Node intersectionList = intersectionLinkedList.intersection(firstList,
secondList);

     while(intersectionList != null){
       System.out.println(intersectionList.getData());
       intersectionList = intersectionList.getNext();
     }
   }
}
```

Result:

```
a
c
d
```

Find Union of two linkedLists

You have given two linked list, create a linked list containing all the elements of the given two linked list excluding duplicates. i.e. create union of two given linked list. Order of elements in new linked list does not matter.

Algorithm 1:

Create a new linked list called result and Initialize as NULL. Traverse list1 and add all of its elements to the result.

Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

Time complexity:

Time Complexity for this algorithm is **O(mn) m is the number of elements in first listn is the number of elements in second list**. **Algorithm 2:**
The algorithm for the implementation of this problem is very simple, here are steps :-

1. Create a new **HashSet.**
2. Traverse **first linked list.**
3. Check data (data stored in the node) is present in a set or not, if it is not present then we create a new node with that data, add it to the new linked list and to the set. Otherwise continue.
4. Repeat step 2 and step 3 for**second linked list**.

Code Implementation Algorithm 1:

Rest of the code is same as algorithm implementation 1 for intersection of linked lists

```
void getUnion(Node head1, Node head2)
 {
     Node t1 = head1, t2 = head2;

     //insert all elements of list1 in the result
     while (t1 != null)
     {
         push(t1.data);
         t1 = t1.next;
     }

     // insert those elements of list2 that are not present
     while (t2 != null)
     {
         if (!isPresent(head, t2.data))
             push(t2.data);
         t2 = t2.next;
     }
 }
```

Code Implementation Algorithm 2:

Rest of the code is same as above implementation for algorithm 2 in intersection of linked lists

```
public class UnionLinkedList {

  private Node unionListHeadNode;

  public Node union(Node firstList, Node secondList){
    Set uniqueSet = new HashSet();
    Node unionListLastNode = addList(firstList,null,uniqueSet);
    unionListLastNode = addList(secondList,unionListLastNode,uniqueSet);
    return this.unionListHeadNode;
  }

  private Node addList(Node sourceList,Node unionListLastNode,Set uniqueSet){
    Node tempNode = null;
    Node tempList = unionListLastNode;
    while(sourceList != null){
      if(!uniqueSet.contains(sourceList.getData())){
        uniqueSet.add(sourceList.getData());
        tempNode = createNode(sourceList.getData());
        if(tempList == null){
          tempList = tempNode;
          this.unionListHeadNode = tempList;
        }else{
          tempList.setNext(tempNode);
          tempList = tempNode;
        }
      }
      sourceList = sourceList.getNext();
    }
    return tempList;
  }

  private Node createNode(String data){
    Node node = new Node();
    node.setData(data);
    return node;
  }

}
```

Question: How to find middle element of linked list in one pass? [Google]

The Problem

Question: How would you find the middle element of a linked list in one pass/loop?

This is one of the starter question in most of technical interviews. so make sure you understand the concept and implementation correctly.

The Solution

Middle element of a linked list can be found by using two pointers, which will move like this

- **Pointer 1:** Incrementing one at each iteration
- **Pointer 2:** Incrementing at every second iteration

When first pointer will point at end of Linked List, second pointer will be pointing at middle node of Linked List.

Pseudo Code

```
Node current = LinkedListHead;
int length = 0;
Node middle = LinkedListHead;
while(current.next() != null){
length++;
if(length % 2 == 0) {
middle = middle.next();
}
current = current.next();
}
return middle;
```

Complete Solution

```
public class MiddleOfList {
public String findMiddleOfList() {
LinkedListNode tail = new LinkedListNode("data5", null);
LinkedListNode node4 = new LinkedListNode("data4", tail);
LinkedListNode node3 = new LinkedListNode("data3", node4);
LinkedListNode node2 = new LinkedListNode("data2", node3);
LinkedListNode head = new LinkedListNode("data1", node2);
return findMiddle(head);
}
private String findMiddle(LinkedListNode head) {
LinkedListNode current = head;
int length = 0;
LinkedListNode middle = head;
while(current.nextNode != null){
length++;
if(length % 2 == 0) {
middle = middle.nextNode;
}
current = current.nextNode;
}
return middle.data;
}
private class LinkedListNode {
public String data = null;
public LinkedListNode nextNode = null;
public LinkedListNode(String data, LinkedListNode nextNode) {
this.data = data;
this.nextNode = nextNode;
}
}
}
```

Question: How to remove Duplicates from a Linked List in Java? [Telephonic]

The Solution

Write a program which can go through a linked list and remove all the duplicate values, For example if the linked list is 12->11->12->21->41->43->21 then our program should convert the list to 12->11->21->41->43.

There are two approaches to achieve this

Algorithm 1:

Simplest way to achieve this is by using two loops. **Outer loop** is used to pick the elements one by one and **inner loop** compares the picked element with rest of the elements.

Performance:

Time Complexity of this solution in terms of big'0 notation is **O(n^2)**

Algorithm 2:

- Create a Hash Table
- Take two pointers, prevNode and CurrNode.
- PrevNode will point to the head of the linked list and currNode will point to the head.next.

- Now navigate through the linked list.
- Check every node data is present in the HashTable.
- if yes then delete that node using prevNode and currNode.
- If No, then insert that node data into the linked list
- Return the head of the list

Performance:

Time Complexity of this solution in terms of big'0 notation is**O(n)**

Code implementation of Algorithm 1:

```
// Java program to remove duplicates from unsorted linked list

class LinkedList {

    static Node head;

    static class Node {

        int data;
        Node next;

        Node(int d) {
            data = d;
            next = null;
        }
    }

    /* Function to remove duplicates from a unsorted linked list */
    void remove_duplicates() {
        Node ptr1 = null, ptr2 = null, dup = null;
        ptr1 = head;

        /* Pick elements one by one */
        while (ptr1 != null && ptr1.next != null) {
            ptr2 = ptr1;

            /* Compare the picked element with rest of the elements */
            while (ptr2.next != null) {

                /* If duplicate then delete it */
                if (ptr1.data == ptr2.next.data) {

                    /* sequence of steps is important here */
```

```
                    dup = ptr2.next;
                    ptr2.next = ptr2.next.next;
                    System.gc();
                } else /* This is tricky */ {
                    ptr2 = ptr2.next;
                }
            }
            ptr1 = ptr1.next;
        }
    }

    void printList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.head = new Node(10);
        list.head.next = new Node(12);
        list.head.next.next = new Node(11);
        list.head.next.next.next = new Node(11);
        list.head.next.next.next.next = new Node(12);
        list.head.next.next.next.next.next = new Node(11);
        list.head.next.next.next.next.next.next = new Node(10);

        System.out.println("Linked List before removing duplicates ");
        list.printList(head);

        list.remove_duplicates();
        System.out.println("");
        System.out.println("Linked List after removing duplicates");
        list.printList(head);

    }
}
```

Code implementation of Algorithm 2:

```
import java.util.HashMap;

public class RemoveDuplicates {

    public Node removeDup(Node head){
        HashMap<Integer, Integer> ht = new HashMap<Integer, Integer>();
        if(head==null){
            return null;
        }
        Node currNode = head.next;
        Node prevNode = head;
        Node temp; //keeping it so that last node would be eligible for garbage
collection
        ht.put(head.data, 1);
```

```
        while(currNode!=null){
            int data = currNode.data;
            if(ht.containsKey(data)){
                prevNode.next = currNode.next;
                temp= currNode;
                currNode = currNode.next;
                temp.next = null;
            }else{
                ht.put(data, 1);
                prevNode = currNode;
                currNode = currNode.next;
            }
        } return head;
    }
    public void display(Node head){
        Node n=head;
        while(n!=null){
            System.out.print("->" + n.data);
            n=n.next;
        }
    }
    public static void main(String args[]){
        Node n = new Node(2);
        n.next = new Node(2);
        n.next.next = new Node(2);
        n.next.next.next = new Node(3);
        n.next.next.next.next = new Node(4);
        n.next.next.next.next.next = new Node(4);
        n.next.next.next.next.next.next = new Node(2);
        System.out.print("Original List : ");
        RD rm = new RD();
        rm.display(n);
        System.out.print("n Updated List: ");
        Node x =rm.removeDup(n);
        rm.display(x);
    }
}
class Node{
    int data;
    Node next;
    public Node(int data){
        this.data = data;
        next = null;
    }
}
```

Question: Reverse a Linked List using Recursion. [Accenture]

The solution:

There are two ways to reverse a given linked list

Algorithm 1 : by using iteration

- Create 3 nodes, currNode, PrevNode and nextNode.
- Initialize them as currNode = head; nextNode = **null**;prevNode = **null**;
- Now keep reversing the pointers one by one till currNode!=null.

```
   while(currNode!=null){
     nextNode = currNode.next;
     currNode.next = prevNode;
     prevNode = currNode;
     currNode = nextNode;
}
```

Algorithm 2 : by using recursion

- Take 3 nodes as Node ptrOne,Node ptrTwo, Node prevNode
- Initialize them as ptrOne = head; ptrTwo=head.next, prevNode = null.
- Call reverseRecursion(head,head.next,null)
- Reverse the ptrOne and ptrTwo
- Make a recursive call for **reverseRecursion(ptrOne.next,ptrTwo.next,null)**

Full implementation:

```
public class ReverseLinkedList {
 public static void main (String[] args) throws java.lang.Exception
 {
  LinkedListT a = new LinkedListT();
  a.addAtBegin(5);
  a.addAtBegin(10);
  a.addAtBegin(15);
  a.addAtBegin(20);
  a.addAtBegin(25);
  a.addAtBegin(30);
//  System.out.print("Original Link List 1 : ");
  a.display(a.head);
  a.reverseIterative(a.head);
  LinkedListT b = new LinkedListT();
  b.addAtBegin(31);
  b.addAtBegin(32);
  b.addAtBegin(33);
  b.addAtBegin(34);
  b.addAtBegin(35);
  b.addAtBegin(36);
  System.out.println("");
  System.out.println("____________________");
  System.out.print("Original Link List 2 : ");
  b.display(b.head);
  b.reverseRecursion(b.head,b.head.next,null);
  System.out.println("");
  //b.display(x);
 }
}
class Node{
 public int data;
```

```
 public Node next;
 public Node(int data){
  this.data = data;
  this.next = null;
 }
}
class LinkedListT{
 public Node head;
 public LinkedListT(){
  head=null;
 }

 public void addAtBegin(int data){
  Node n = new Node(data);
  n.next = head;
  head = n;
 }
public class ReverseLinkedList {
 public static void main (String[] args) throws java.lang.Exception
 {
  LinkedListT a = new LinkedListT();
  a.addAtBegin(5);
  a.addAtBegin(10);
  a.addAtBegin(15);
  a.addAtBegin(20);
  a.addAtBegin(25);
  a.addAtBegin(30);
//  System.out.print("Original Link List 1 : ");
  a.display(a.head);
  a.reverseIterative(a.head);
  LinkedListT b = new LinkedListT();
  b.addAtBegin(31);
  b.addAtBegin(32);
  b.addAtBegin(33);
  b.addAtBegin(34);
  b.addAtBegin(35);
  b.addAtBegin(36);
  System.out.println("");
  System.out.println("_____________________");
  System.out.print("Original Link List 2 : ");
  b.display(b.head);
  b.reverseRecursion(b.head,b.head.next,null);
  System.out.println("");
  //b.display(x);
 }
}
class Node{
 public int data;
 public Node next;
 public Node(int data){
  this.data = data;
  this.next = null;
 }
}
class LinkedListT{
 public Node head;
 public LinkedListT(){
```

```
 head=null;
}

public void addAtBegin(int data){
 Node n = new Node(data);
 n.next = head;
 head = n;
}
public void reverseIterative(Node head){
 Node currNode = head;
 Node nextNode = null;
 Node prevNode = null;

 while(currNode!=null){
  nextNode = currNode.next;
  currNode.next = prevNode;
  prevNode = currNode;
  currNode = nextNode;
 }
 head = prevNode;
 System.out.println("n Reverse Through Iteration");
 display(head);
}

public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
 if(ptrTwo!=null){
   if(ptrTwo.next!=null){
    Node t1 = ptrTwo;
    Node t2 = ptrTwo.next;
    ptrOne.next = prevNode;
    prevNode = ptrOne;
    reverseRecursion(t1,t2, prevNode);
   }
   else{
    ptrTwo.next = ptrOne;
    ptrOne.next = prevNode;
    System.out.println("n Reverse Through Recursion");
    display(ptrTwo);
   }
 }
 else if(ptrOne!=null){
  System.out.println("n Reverse Through Recursion");
  display(ptrOne);
 }
}

public void display(Node head){
 //
 Node currNode = head;
 while(currNode!=null){
  System.out.print("->" + currNode.data);
  currNode=currNode.next;
 }
}
}
 public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
```

```
  if(ptrTwo!=null){
    if(ptrTwo.next!=null){
     Node t1 = ptrTwo;
     Node t2 = ptrTwo.next;
     ptrOne.next = prevNode;
     prevNode = ptrOne;
     reverseRecursion(t1,t2, prevNode);
    }
    else{
     ptrTwo.next = ptrOne;
     ptrOne.next = prevNode;
     System.out.println("n Reverse Through Recursion");
     display(ptrTwo);
    }
  }
  else if(ptrOne!=null){
   System.out.println("n Reverse Through Recursion");
   display(ptrOne);
  }
 }

 public void display(Node head){
  //
  Node currNode = head;
  while(currNode!=null){
   System.out.print("->" + currNode.data);
   currNode=currNode.next;
  }
 }
}
```

Output:

```
->30->25->20->15->10->5
Reverse Through Iteration
->5->10->15->20->25->30
_____________________
Original Link List 2 : ->36->35->34->33->32->31
Reverse Through Recursion
->31->32->33->34->35->36
```

Question: Move last node to front in Java linked list. [Teleph. Interviews]

Analysis:

```
Traverse the list till last node. Use two pointers: one to store the address of
last node and other for address of second last node. After the end of loop do
following operations.
i) Make second last as last (secLast->next = NULL).
ii) Set next of last as head (last->next = *head_ref).
iii) Make last as head ( *head_ref = last)
```

Full Implementation:

```
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    void moveToFront()
    {
        /* If linked list is empty or it contains only
           one node then simply return. */
           if(head == null || head.next == null)
              return;

        /* Initialize second last and last pointers */
        Node secLast = null;
        Node last = head;

        /* After this loop secLast contains address of
           second last  node and last contains address of
           last node in Linked List */
        while (last.next != null)
        {
           secLast = last;
           last = last.next;
        }

        /* Set the next of second last as null */
        secLast.next = null;

        /* Set the next of last as head */
        last.next = head;

        /* Change head to point to last node. */
        head = last;
    }


    /* Utility functions */

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);
```

```
        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* Function to print linked list */
    void printList()
    {
        Node temp = head;
        while(temp != null)
        {
           System.out.print(temp.data+" ");
           temp = temp.next;
        }
        System.out.println();
    }

     /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();
        /* Constructed Linked List is 1->2->3->4->5->null */
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.println("Linked List before moving last to front ");
        llist.printList();

        llist.moveToFront();

        System.out.println("Linked List after moving last to front ");
        llist.printList();
    }
}
```

OutPut:

```
Linked list before moving last to front
1 2 3 4 5
Linked list after removing last to front
5 1 2 3
```

Performance:

```
Time Complexity: O(n) where n is the number of nodes in the given Linked List.
```

Question: Reverse a singly Linked List in Java?

The solution:

There are two ways to reverse a given linked list

Algorithm 1 : by using iteration

- Create 3 nodes, currNode, PrevNode and nextNode.
- Initialize them as currNode = head; nextNode = **null**;prevNode = **null**;
- Now keep reversing the pointers one by one till currNode!=null.

```
   while(currNode!=null){
     nextNode = currNode.next;
     currNode.next = prevNode;
     prevNode = currNode;
     currNode = nextNode;
}
```

Algorithm 2 : by using recursion

- Take 3 nodes as Node ptrOne,Node ptrTwo, Node prevNode
- Initialize them as ptrOne = head; ptrTwo=head.next, prevNode = null.
- Call reverseRecursion(head,head.next,null)
- Reverse the ptrOne and ptrTwo
- Make a recursive call for **reverseRecursion(ptrOne.next,ptrTwo.next,null)**

Full implementation:

```
public class ReverseLinkedList {
 public static void main (String[] args) throws java.lang.Exception
 {
  LinkedListT a = new LinkedListT();
  a.addAtBegin(5);
  a.addAtBegin(10);
  a.addAtBegin(15);
  a.addAtBegin(20);
  a.addAtBegin(25);
  a.addAtBegin(30);
//  System.out.print("Original Link List 1 : ");
  a.display(a.head);
  a.reverseIterative(a.head);
  LinkedListT b = new LinkedListT();
  b.addAtBegin(31);
  b.addAtBegin(32);
  b.addAtBegin(33);
  b.addAtBegin(34);
  b.addAtBegin(35);
  b.addAtBegin(36);
  System.out.println("");
  System.out.println("____________________");
  System.out.print("Original Link List 2 : ");
  b.display(b.head);
```

```
  b.reverseRecursion(b.head,b.head.next,null);
  System.out.println("");
  //b.display(x);
 }
}
class Node{
 public int data;
 public Node next;
 public Node(int data){
  this.data = data;
  this.next = null;
 }
}
class LinkedListT{
 public Node head;
 public LinkedListT(){
  head=null;
 }

 public void addAtBegin(int data){
  Node n = new Node(data);
  n.next = head;
  head = n;
 }
public class ReverseLinkedList {
 public static void main (String[] args) throws java.lang.Exception
 {
  LinkedListT a = new LinkedListT();
  a.addAtBegin(5);
  a.addAtBegin(10);
  a.addAtBegin(15);
  a.addAtBegin(20);
  a.addAtBegin(25);
  a.addAtBegin(30);
//  System.out.print("Original Link List 1 : ");
  a.display(a.head);
  a.reverseIterative(a.head);
  LinkedListT b = new LinkedListT();
  b.addAtBegin(31);
  b.addAtBegin(32);
  b.addAtBegin(33);
  b.addAtBegin(34);
  b.addAtBegin(35);
  b.addAtBegin(36);
  System.out.println("");
  System.out.println("_____________________");
  System.out.print("Original Link List 2 : ");
  b.display(b.head);
  b.reverseRecursion(b.head,b.head.next,null);
  System.out.println("");
  //b.display(x);
 }
}
class Node{
 public int data;
 public Node next;
 public Node(int data){
```

```
  this.data = data;
  this.next = null;
 }
}
class LinkedListT{
 public Node head;
 public LinkedListT(){
  head=null;
 }

 public void addAtBegin(int data){
  Node n = new Node(data);
  n.next = head;
  head = n;
 }
 public void reverseIterative(Node head){
  Node currNode = head;
  Node nextNode = null;
  Node prevNode = null;

  while(currNode!=null){
   nextNode = currNode.next;
   currNode.next = prevNode;
   prevNode = currNode;
   currNode = nextNode;
  }
  head = prevNode;
  System.out.println("n Reverse Through Iteration");
  display(head);
 }

 public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
  if(ptrTwo!=null){
    if(ptrTwo.next!=null){
     Node t1 = ptrTwo;
     Node t2 = ptrTwo.next;
     ptrOne.next = prevNode;
     prevNode = ptrOne;
     reverseRecursion(t1,t2, prevNode);
    }
    else{
     ptrTwo.next = ptrOne;
     ptrOne.next = prevNode;
     System.out.println("n Reverse Through Recursion");
     display(ptrTwo);
    }
  }
  else if(ptrOne!=null){
   System.out.println("n Reverse Through Recursion");
   display(ptrOne);
  }
 }

 public void display(Node head){
  //
  Node currNode = head;
```

```
  while(currNode!=null){
   System.out.print("->" + currNode.data);
   currNode=currNode.next;
  }
 }
}
 public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
  if(ptrTwo!=null){
    if(ptrTwo.next!=null){
     Node t1 = ptrTwo;
     Node t2 = ptrTwo.next;
     ptrOne.next = prevNode;
     prevNode = ptrOne;
     reverseRecursion(t1,t2, prevNode);
    }
    else{
     ptrTwo.next = ptrOne;
     ptrOne.next = prevNode;
     System.out.println("n Reverse Through Recursion");
     display(ptrTwo);
    }
  }
  else if(ptrOne!=null){
   System.out.println("n Reverse Through Recursion");
   display(ptrOne);
  }
 }

 public void display(Node head){
  //
  Node currNode = head;
  while(currNode!=null){
   System.out.print("->" + currNode.data);
   currNode=currNode.next;
  }
 }
}
```

Output:

```
->30->25->20->15->10->5
Reverse Through Iteration
->5->10->15->20->25->30
_____________________
Original Link List 2 : ->36->35->34->33->32->31
Reverse Through Recursion
->31->32->33->34->35->36
```

Question: Insert nodes into a Linked list in a sorted fashion in Java. [Telephonic]

How will you add a new node in sorted manner in a linked list?

The solution:

In order to add a new element we will use following algorithm

1. If Linked list is empty then make the node as head and return it.
2. If value of the node to be inserted is smaller than value of head node then insert the node at start and make it head.
3. If value is larger then the head node then Iterate through the linked list and compare value of each node with the value that you want to insert
4. if value > currentNodeValue then add the value after the current node

Full Code implementation

```
// Java Program to insert in a sorted list
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    /* function to insert a new_node in a list. */
    void sortedInsert(Node new_node)
    {
         Node current;

         /* Special case for head node */
         if (head == null || head.data >= new_node.data)
         {
            new_node.next = head;
            head = new_node;
         }
         else {

            /* Locate the node before point of insertion. */
            current = head;

            while (current.next != null &&
                   current.next.data < new_node.data)
                  current = current.next;

            new_node.next = current.next;
            current.next = new_node;
         }
     }

                  /*Utility functions*/
```

```
    /* Function to create a node */
    Node newNode(int data)
    {
       Node x = new Node(data);
       return x;
    }

     /* Function to print linked list */
     void printList()
     {
         Node temp = head;
         while (temp != null)
         {
            System.out.print(temp.data+" ");
            temp = temp.next;
         }
     }

     /* Drier function to test above methods */
     public static void main(String args[])
     {
         LinkedList llist = new LinkedList();
         Node new_node;
         new_node = llist.newNode(5);
         llist.sortedInsert(new_node);
         new_node = llist.newNode(10);
         llist.sortedInsert(new_node);
         new_node = llist.newNode(7);
         llist.sortedInsert(new_node);
         new_node = llist.newNode(3);
         llist.sortedInsert(new_node);
         new_node = llist.newNode(1);
         llist.sortedInsert(new_node);
         new_node = llist.newNode(9);
         llist.sortedInsert(new_node);
         System.out.println("Created Linked List");
         llist.printList();
     }
}
```

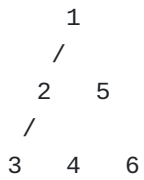
Question: Flatten A Binary Tree to Linked List (In-place). [Google]

How will you flat a binary tree to linked list?

The Solution:

This is a tricky question often asked in google phone interview. here is the solution

For example we have a following binary tree



The flattened tree should look like:

How we will achieve that goal?

Go down through the left, when right is not null, push right to stack.

Complete code:

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void flatten(TreeNode root) {
        Stack stack = new Stack();
        TreeNode p = root;

        while(p != null || !stack.empty()){

            if(p.right != null){
                stack.push(p.right);
            }

            if(p.left != null){
                p.right = p.left;
                p.left = null;
            }else if(!stack.empty()){
                TreeNode temp = stack.pop();
                p.right=temp;
            }

            p = p.right;
        }
    }
}
```

Keys to Interview Success:

You will be asked questions about the linked lists in almost all of your technical interviews. Google is particularly notorious in asking linked lists questions in their telephonic interviews.

Part of this has to do is, with the underlying nature of the linked lists structure. It offers a great flexibility for the developer to modify the linked lists basic functionality based on your requirements.

Few examples are Queues and stack. Make sure you have gone through all the basic concepts and most important programs. Knowing linked lists thoroughly will definitely increase your chances of success in interviews.

INTRODUCTION TO
LINKED LISTS

TOP 5 LINKED LIST QUESTIONS

1- Reverse a linked list
2- Performance
3- Comparisons with other DS
4- Remove duplicates
5- Advantage & Disadvantages

STRUCTURE

Java LinkedList is the Doubly-linked list implementation of the list interface

PERFORMANCE OF LINKED LISTS

	Single Linked list	Doubly Linked list
Access :	O(n) Slow	O(n) Slow
Search :	O(n) Slow	O(n) Slow
Insertion:	O(1) Fast	O(1) Fast
Deletion:	O(1) Fast	O(1) Fast

ADVANTAGES

- Linked lists have constant-time insertion and deletion operation in any position
- Linked lists can continue to expand as long as there is space on the machine

DISADVANTAGES

To access or edit an item in a linked list, you have to take O(i)O(i) time to walk from the head of the list to the iith item (unless of course you already have a pointer directly to that item). Arrays have constant-time lookups and edits to the iith item.

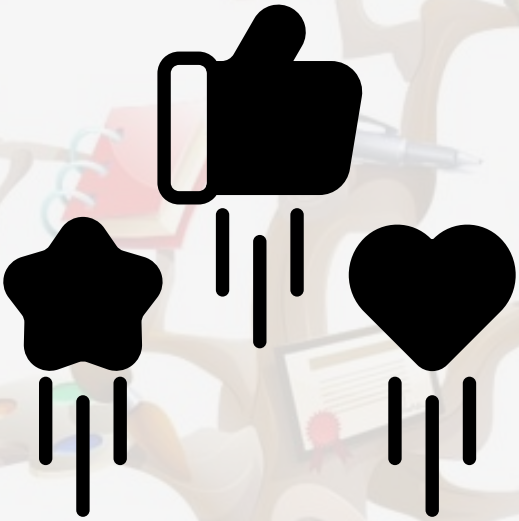
http://www.codespaghetti.com

About The Author:

References:

- https://en.wikipedia.org/wiki/Linked_list
- https://www.amazon.com/Cracking-Coding-Interview-Programming-Questions/dp/098478280X
- https://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html
- https://www.quora.com/
- http://cslibrary.stanford.edu/103/LinkedListBasics.pdf

Loved the content



Ankit Pangasa

WORLD OF TECHIES
EST 2023