2.1.6 Longest Consecutive Sequence

Thursday, January 19, 2017 8:36 PM

```
14 public class LongestConsecutiveSolution
15 {
16     public static List<Integer> inputList = Arrays.asList(1, 2, 3);
17
18     public static void main (String[] args)
19     {
20         Map<Integer, Integer> _aMap = new HashMap<>();
21         Iterator<Integer> i = inputList.iterator();
22         while (i.hasNext())
23             _aMap.put(i.next(), 0);
24         int longest = 0;
25         Iterator<Integer> iKey = _aMap.keySet().iterator();
26         while (iKey.hasNext())
27         {
28             int key = iKey.next();
29
30             if (_aMap.get(key) != 0) //always process the integer as intact as degree 0.
31                 continue;
32
33             int result = calConsecutiveDegree(key, _aMap);
34             longest = longest > result ? longest : result;
35         }
36         System.out.println(longest);
37         System.out.println(_aMap);
38
39     }
40
41     public static int calConsecutiveDegree (int i, Map<Integer, Integer> aMap)
42     {
43         if (!aMap.containsKey(i + 1)) //conquer when the key(i+1) cannot found as a result of no further consecutive.
44         {
45             aMap.put(i, 1);
46             return 1;
47         }
48         else
49         {
50             int degree = 1 + calConsecutiveDegree(i + 1, aMap); //else recursive to its consecutive as i+1.
51             aMap.put(i, degree); //and increase degree by 1 and put into the Map.
52             return degree;
53
54         }
55     }
56 }
```

2.1.8 3Sum

Thursday, January 19, 2017 8:52 PM

```
8  public class ThreeSumSolution
9  {
10     public static void main (String[] args)
11     {
12         List<Integer> inputList = Arrays.asList(-2, -1, 0, 1, 2, 3, 4, 5, 6);
13         System.out.println(calculate3Sum(inputList, 0));
14
15     }
16
17     public static List<List<Integer>> calculate3Sum (List<Integer> input, int target)
18     {
19         List<List<Integer>> results = new ArrayList<>();
20
21         Collections.sort(input);                    //sort integers first
22         int begin = 0, last = input.size() - 1;     //retain a place for k
23
24         for (int i = begin; i < last; ++i)
25         {
26
27             int j = i + 1;                          // the second index will the next to first
28             int k = last;                           // the third index will be the last.
29
30             if (i < last &&
31                 input.get(i) == input.get(i + 1))   // skip the same integer
32                 continue;
33
34             while (j < k)                           //both(j,k) approach pussy
35             {
36                 if (input.get(i) + input.get(j) + input.get(k) < target)  //less then increase j forward to pussy
37                 {
38                     ++j;
39                     while (j < k && input.get(j) == input.get(j + 1)) // skip the same integer
40                         ++j;
41                 }
42                 else if (input.get(i) + input.get(j) + input.get(k) > target) //more then k backward pussy
43                 {
44                     --k;
45                     while (j < k && input.get(k) == input.get(k - 1)) // skip the same integer
46                         --k;
47                 }
48                 else
49                 {
50                     results.add(Arrays.asList(input.get(i), input.get(j), input.get(k))); //meet then add to resultList
51                     ++j;                                                                  //both(j,k) approach
52                     --k;
53                     while (j < k && input.get(j) == input.get(j + 1) // skip the same integer
54                         && input.get(k) == input.get(k - 1))         // skip the same integer
55                         ++j;                                         //then j approach to pussy
56                 }
57             }
58         }
59         return results;
60     }
61 }
62
```

Screen clipping taken: 1/19/2017 9:21 PM

Screen clipping taken: 1/19/2017 9:16 PM

Search on rotated and sorted array

Saturday, January 21, 2017 10:31 AM

```java
public class SearchRotatedSortedArray
{

    public static void main (String[] args)
    {
        int[] inputArray = {4, 5, 6, 7, 8, 1, 2, 3};
        //0, 1, 2, 3, 4, 5, 6, 7
        int result = search(9, inputArray);
        System.out.println(result);
    }

    public static int search (int target, int... nums)
    {
        int middle = 0, first = 0, last = nums.length; //define last as the array length
                                                       //in order to calculate middle.
        while (first != last)
        {
            middle = (first + last) / 2; //the middle is (first+last)/2
            if (nums[middle] == target)
                return middle;

            if (nums[first] < nums[middle])
            {
                if (nums[first] <= target && target < nums[middle]) //target equal or less than first
                    last = middle;
                else
                    first = middle + 1; // change first always to middle + 1
            }
            else
            {
                if (nums[middle] < target && target <= nums[last - 1]) //target equal or more as last
                    first = middle + 1; // change first always to middle + 1
                else
                    last = middle;
            }
        }
        return -1;
    }
}
```

Screen clipping taken: 1/21/2017 10:31 AM

Grey code

Saturday, January 21, 2017 6:01 PM

方法 2，n 比特的格雷码，可以递归地从 $n-1$ 比特的格雷码生成。如图 §2-5所示。

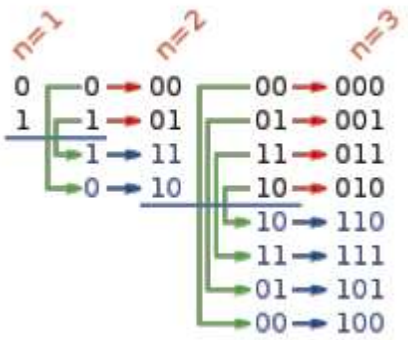

图 2-5 The first few steps of the reflect-and-prefix method.

Screen clipping taken: 1/21/2017 6:01 PM

```java
public class GrayCodeSolution
{

    public static void main (String[] args)
    {
        System.out.println(calculateGrayCodeSequence(3));
        //[0, 1, 3, 2, 6, 7, 5, 4]
    }

    private static List<Integer> calculateGrayCodeSequence (int n)
    {
        List<Integer> result = new ArrayList<>();
        result.add(0); //add initial 0 as the very starter
        for (int i = 0; i < n; ++i)
        {
            int hightest_bit = 1 << i; //cal highest bit for '|' operator for currect cycle.
            for (int j = result.size() - 1; j >= 0; j--) //reverse the order of
            {                                            //the previous result.
                result.add(hightest_bit | result.get(j));
            }
        }
        return result;
    }
}

```

Screen clipping taken: 1/21/2017 6:02 PM

Set matrix zero

Saturday, January 21, 2017 9:37 PM

```
22
23      private static void setZero (char[][] data)
24      {
25          boolean row_zero = false, col_zero = false;
26          int nrow = data.length, ncol = data[0].length;
27
28          for (int j = 0; j < ncol; ++j)
29          {
30              if (Character.getNumericValue(data[0][j]) == 0) // cal if 0th row has '0'
31              {
32                  row_zero = true;
33                  break;
34              }
35          }
36
37          for (int i = 0; i < nrow; ++i)
38          {
39              if (Character.getNumericValue(data[i][0]) == 0) // cal if 0th col has '0'
40              {
41                  col_zero = true;
42                  break;
43              }
44          }
45
46          for (int i = 1; i < nrow; ++i)                  //start with 1th row and col to
47              for (int j = 1; j < ncol; ++j)              // check if '0' occurs
48              {
49                  if (Character.getNumericValue(data[i][j]) == 0)
50                  {
51                      data[0][j] = '0';
52                      data[i][0] = '0';
53                  }
54              }
55          for (int i = 1; i < nrow; ++i)                  //set '0' align with '0's at
56              for (int j = 1; j < ncol; ++j)              //0th row and col.
57              {
58                  if (Character.getNumericValue(data[0][j]) == 0
59                      || Character.getNumericValue(data[i][0]) == 0)
60                  {
61                      data[i][j] = '0';
62                  }
63              }
64          if (row_zero)
65              for (int j = 0; j < ncol; ++j)              //set 0th row '0' if row_zero is true
66                  data[0][j] = '0';
67          if (col_zero)
68              for (int i = 0; i < nrow; ++i)              //set  0th col '0' if col_zero is true
69                  data[i][0] = '0';
70      }
71
```

Screen clipping taken: 1/21/2017 9:38 PM

Calculate how many binary 1 out of an integer

Saturday, January 28, 2017 9:48 PM

```
5  public class OneInAInteger
6  {
7
8      public static void main (String[] args)
9      {
10         int testData = 111111111;
11         System.out.println(Integer.toBinaryString(testData));
12         System.out.println(calHowMany1InAInteger(testData));
13     }
14
15     private static int calHowMany1InAInteger(Integer data)
16     {
17         int result = 0;
18         while(data != 0)
19         {
20             data = data & (data - 1);      // 1101          1100      1000
21             result++;                      //&1101         &1011     &0111
22         }                                  // 1100          1000      0000
23         return result;
24     }
25 }
26
```

Screen clipping taken: 1/28/2017 9:51 PM

Trapped water

Tuesday, January 31, 2017 10:14 PM

```
3  public class WaterTrapSolution
4  {
5
6      public static void main (String[] args)
7      {
8          int[] inputs = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
9          System.out.println(calculateTrappedWater(inputs));
10
11     }
12
13     //[0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3]   Math.min(left_max,
14     //[3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 1, 0]            right_max)
15     //[0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]  -height
16     //-----------------------------------------------------------
17     //[0, 0, 1, 0, 1, 2, 1, 0, 0, 1, 0, 0]   =trapped_water
18
19     public static int calculateTrappedWater (int... blocks)
20     {
21         int n = blocks.length;
22         int[] left_max = new int[n];      //max water of left side blocks
23         int[] right_max = new int[n];     //max water of right side blocks
24         for (int i = 1; i < n; ++i)
25         {
26             left_max[i] = Math.max(left_max[i - 1], blocks[i - 1]);
27             right_max[n - 1 - i] = Math.max(right_max[n - i], blocks[n - i]);
28         }
29
30         int sum = 0;
31         for (int i = 0; i < n; ++i)
32         {
33             if (blocks[i] < Math.min(left_max[i], right_max[i]))
34                 sum += Math.min(left_max[i], right_max[i]) - blocks[i];
35         }
36         return sum;
37     }
38 }
39
```

Screen clipping taken: 1/31/2017 10:14 PM

Reverse the linked list from m to n

Saturday, February 04, 2017 1:07 PM

2.2.2 Reverse Linked List II

Reverse a linked list from position m to n. Do it in-place and in one-pass.
For example: Given 1->2->3->4->5->nullptr, m = 2 and n = 4,
return 1->4->3->2->5->nullptr.
Note: Given m, n satisfy the following condition: 1 ≤ m ≤ n ≤ length of list.

```
4  public class ReverseLinkedList
5  {
6
7      public static void main (String[] args)
8      {
9          ListNode nd5 = new ListNode(null, 5);
10         ListNode nd4 = new ListNode(nd5, 4);
11         ListNode nd3 = new ListNode(nd4, 3);
12         ListNode nd2 = new ListNode(nd3, 2);
13         ListNode nd1head = new ListNode(nd2, 1);
14         iterateLinkedList(nd1head);
15         reverseLinkedList(nd1head, 2, 5);
16         iterateLinkedList(nd1head);
17     }
18
19     private static void reverseLinkedList (ListNode head, int m, int n)
20     {
21         ListNode prev = new ListNode(head, 0);  //Define a prev node to define a fixed head node
22         for (int i = 0; i < m - 1; ++i)         //to link with the new head of reversed list
23             prev = prev.getNext();
24
25         final ListNode fixHead = prev;          //a fixed head node to linked with the new head of reversed list
26         prev = prev.getNext();                  //prev is the head of reversed list
27         ListNode cur = prev.getNext();          //cur is the tail and then become the new head of revserse list link with the fixed head.
28
29         for(int i = m;i < n; ++i)
30         {
31             prev.setNext(cur.getNext());        //prepare the cur node for moving cur to the next node.
32             cur.setNext(fixHead.getNext());     //inject the cur node as before the head of reverse list.
33             fixHead.setNext(cur);               //always link cur with the fixed head as inject cur as head.
34             cur = prev.getNext();               //move cur as the next tail
35         }
36
37     }
```

Screen clipping taken: 2/4/2017 1:08 PM

2.2.4 Remove Duplicates from Sorted List

Sunday, February 05, 2017 11:27 AM

```java
import util.ListNode;

//Given a sorted linked list, delete all duplicates such that each element appear only once.
//For example,
//Given 1->1->2, return 1->2.
//Given 1->1->2->3->3, return 1->2->3.

public class RemoveDuplicatesLinkedList
{

    public static void main (String[] args)
    {
        ListNode nd0 = null;
        ListNode nd1 = new ListNode(null, 1);
        ListNode nd2 = new ListNode(nd1, 2);
        ListNode nd3 = new ListNode(nd2, 2);
        ListNode nd4 = new ListNode(nd3, 3);
        ListNode nd5 = new ListNode(nd4, 3);
        ListNode nd6 = new ListNode(nd5, 4);

        ListNode.iterateListNodes(nd4);
        removeDuplicates(nd4);
        ListNode.iterateListNodes(nd4);
    }

    private static void removeDuplicates (ListNode head)
    {
        if (head == null)
            return;
        for (ListNode prev = head, cur = prev.getNext(); cur != null; cur = cur.getNext()) //define prev and cur as prev's next node
        {
            if (prev.getValue() == cur.getValue())
            {
                prev.setNext(cur.getNext());  //if found duplicates, then  prev's next set as cur's next
            }
            else
            {
                prev = cur;                   //if not found, then assign cur to prev
            }
        }
    }
}

```

Screen clipping taken: 2/5/2017 10:08 PM

Screen clipping taken: 2/5/2017 11:29 AM

2.2.5 Remove Duplicates from Sorted List II

Sunday, February 05, 2017 10:07 PM

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.
For example,
Given 1->2->3->3->4->4->5, return 1->2->5.
Given 1->1->1->2->3, return 2->3.

```
5  public class RemoveDuplicatesLinkedList
6  {
7
8      public static void main (String[] args)
9      {
10         ListNode nd0 = null;
11         ListNode nd1 = new ListNode(null, 1);
12         ListNode nd2 = new ListNode(nd1, 2);
13         ListNode nd3 = new ListNode(nd2, 2);
14         ListNode nd4 = new ListNode(nd3, 3);
15         ListNode nd5 = new ListNode(nd4, 3);
16         ListNode nd6 = new ListNode(nd5, 4);
17
18         ListNode.iterateListNodes(nd6);
19         removeDuplicates(nd6);
20         ListNode.iterateListNodes(nd6);
21
22     }
23
24     private static ListNode removeDuplicates (ListNode head)
25     {
26         if (head == null || head.getNext() == null) //conquer occurs as isNull(head.getNext())
27             return head;
28         if (head.getValue() == head.getNext().getValue()) //the duplicates found as a next
29         {
30             while (head.getValue() == head.getNext().getValue())
31             {
32                 head.setNext(head.getNext().getNext()); //remove duplicates
33             }
34             return removeDuplicates(head.getNext()); //get rid of the source of duplicates
35         }
36         else
37         {
38             ListNode next = removeDuplicates(head.getNext());
39             head.setNext(next);                          //head with a next of no-duplicates.
40             return head;
41         }
42     }
43 }
44
```

2.2.6 rotate linked list

Thursday, February 09, 2017 10:37 PM

```
//Given a list, rotate the list to the right by k places, where k is non-negative.
//For example: Given 1->2->3->4->5->nullptr and k = 2, return 4->5->1->2->3->nullptr.

import util.ListNode;
public class RotateLinkedList
{

    public static void main (String[] args)
    {
        ListNode nd5 = new ListNode(null, 5);
        ListNode nd4 = new ListNode(nd5, 4);
        ListNode nd3 = new ListNode(nd4, 3);
        ListNode nd2 = new ListNode(nd3, 2);
        ListNode nd1head = new ListNode(nd2, 1);
        ListNode.iterateListNodes(nd1head);
        ListNode.iterateListNodes(rotate(nd1head, 2));
    }

    private static ListNode rotate (ListNode head, int k)
    {
        int length = 1;
        ListNode node = head;                                       //always move the node
        for (;node.getNext() != null; node = node.getNext(), ++length) //break the iteration when the node steps at the tail.
        {                                                           //so the check is the node.getNext()
                                                                    //calculate the list's length
        }
        node.setNext(head);                                        //tail.setNext(head)    tail -> head

        int offset = length - k%length;
        for (int i = 0; i < offset; ++i)                           //node at the tail continues moving (length - k) steps
        {                                                          //until reach the new tail
            node = node.getNext();
        }
        head = node.getNext();                                     // the new head is the new tail's next.
        node.setNext(null);                                        // set new tail's next = null
        return head;
    }

}

```

Screen clipping taken: 2/9/2017 10:41 PM

2.2.9 Reverse Nodes in k-Group

Saturday, February 11, 2017 10:40 PM

```java
import util.ListNode;

//Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.
//If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.
//You may not alter the values in the nodes, only nodes itself may be changed.
//Only constant memory is allowed.
//For example, Given this linked list: 1->2->3->4->5
//For k = 2, you should return: 2->1->4->3->5
//For k = 3, you should return: 3->2->1->4->5

public class ReverseNodeKGroup extends ListNode
{

    static ListNode head = ListNode.nd1head;
    public static void main (String[] args)
    {
        ListNode.iterateListNodes(head);
        ListNode.iterateListNodes(reverse(null, head, 2));

    }

    private static ListNode reverse (ListNode priorHead, ListNode head, int k)
    {
        ListNode node = head;
        for (int i = 0; i < k; ++i)
        {
            if (node == null)
                return head; // return head if the k-group's length < k.
            node = node.getNext();
        }
        ListNode fixedHead = priorHead;
        if (fixedHead == null)
            fixedHead = new ListNode(head, -1); //set head as fixedHead's next.
        ListNode prev = head;
        ListNode cur = prev.getNext();

        for (int i = 1; i < k; ++i) // k-group is moving k-1 steps towards tail.
        {
            prev.setNext(cur.getNext());
            cur.setNext(fixedHead.getNext());
            fixedHead.setNext(cur);
            cur = prev.getNext(); //move cur to the next node as prev's next and fixedHead's next wont change.
        }

        ListNode result = reverse(prev, cur, k);
        prev.setNext(result);       //link prior tail with head of next.
        return fixedHead.getNext(); // return the head node of k-group nodes fixedHead's next.
    }

}

```

Screen clipping taken: 2/11/2017 10:44 PM

13.2 Maximum Subarray

Monday, March 13, 2017 8:53 PM

```java
public class MaximumContinues
{

    public static void main (String[] args)
    {
        int[] inputData = new int[] {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        System.out.println(maximum(inputData));
    }
    private static int maximum (int... inputArray)
    {
        int inter = 0, result = Integer.MIN_VALUE; //pre-set result as negative infinite.

        for (int i = 0; i < inputArray.length; ++i)
        {
            inter = Math.max(inter + inputArray[i], inputArray[i]); //if (inter <= 0), then reset inputArray[i] as sub-array.
            result = Math.max(result, inter);    // result always keep the maximum sum.
        }
        return result;
    }

}
```

Screen clipping taken: 3/13/2017 9:07 PM

最大连续子序列和，非常经典的题。

当我们从头到尾遍历这个数组的时候，对于数组里的一个整数，它有几种选择呢？它只有两种选择：1、加入之前的 SubArray；2. 自己另起一个 SubArray。那什么时候会出现这两种情况呢？

如果之前 SubArray 的总体和大于 0 的话，我们认为其对后续结果是有贡献的。这种情况下我们选择加入之前的 SubArray

如果之前 SubArray 的总体和为 0 或者小于 0 的话，我们认为其对后续结果是没有贡献，甚至是有害的（小于 0 时）。这种情况下我们选择以这个数字开始，另起一个 SubArray。

设状态为 `f[j]`，表示以 `S[j]` 结尾的最大连续子序列和，则状态转移方程如下：

$$f[j] = \max\{f[j-1] + S[j], S[j]\}, \text{其中} 1 \le j \le n$$
$$target = \max\{f[j]\}, \text{其中} 1 \le j \le n$$

解释如下：

- 情况一，S[j] 不独立，与前面的某些数组成一个连续子序列，则最大连续子序列和为 $f[j-1]+S[j]$。
- 情况二，S[j] 独立划分成为一段，即连续子序列仅包含一个数 S[j]，则最大连续子序列和为 $S[j]$。

Screen clipping taken: 3/13/2017 8:57 PM

Merge two sorted lists

Friday, March 24, 2017 9:06 PM

```
4
5  public class MergeTwoSortedLists extends ListNode
6  {
7
8      public static void main (String[] args)
9      {
10         //ListNode.iterateListNodes(merge(ListNode.nd5, ListNode.nd1head));
11         //ListNode.iterateListNodes(merge(ListNode.nd5, ListNode.nd6));
12         ListNode.iterateListNodes(merge(null, null));
13     }
14
15     public static ListNode merge(ListNode list1, ListNode list2)
16     {
17         if(list1 == null)                                    //conquer and then return
18             return list2;
19         if(list2 == null)                                    //conquer and then return
20             return list1;
21         ListNode newHead = null;
22         if (list1.getValue() < list2.getValue())
23         {
24             newHead = list1;
25             newHead.setNext(merge(list1.getNext(), list2)); //divide and then merge
26         }
27
28         if (list1.getValue() >= list2.getValue())
29         {
30             newHead = list2;
31             newHead.setNext(merge(list1, list2.getNext())); //divide and then merge
32         }
33         return newHead;
34     }
35
36 }
37
```

Screen clipping taken: 3/24/2017 9:07 PM

Swap two adjacent nodes

Saturday, March 25, 2017 10:32 AM

```
4
5  public class SwapAdjacentNodes extends ListNode
6  {
7      static ListNode head = nd1head;
8      public static void main (String[] args)
9      {
10
11         ListNode.iterateListNodes(head);
12         ListNode.iterateListNodes(swap(null, head));
13     }
14
15     private static ListNode swap (ListNode prevHead, ListNode head)
16     {
17         if(head == null || head.getNext() == null)
18             return null;
19
20         ListNode dummy =  prevHead != null ? prevHead : new ListNode(head, -1);
21
22         ListNode next = head.getNext();
23
24         head.setNext(next.getNext());  //p -> prev ->head
25         next.setNext(head);            //c -> cur  ->next
26         dummy.setNext(next);           //h -> fixHead -> dummy
27                                        //c cur = prev.getNext()
28
29         swap(head, head.getNext());    //head is the dummy as prevHead after the swap
30
31         return next;                   //return the new head of next (or cur)
32     }
33
34 }
35
```

Screen clipping taken: 3/25/2017 10:39 AM

Complex linked list copy ComplexListNode with a sibling pointer

Saturday, April 08, 2017 8:22 PM

```
3  public class ComplexLinkListNode
4  {
5      public ComplexLinkListNode ()
6      {
7          _value = 0;
8          _next = null;
9          _sibling = null;
10     }
11
12     protected ComplexLinkListNode (int value, ComplexLinkListNode next, ComplexLinkListNode sibling)
13     {
14         _value = value;
15         _next = next;
16         _sibling = sibling;
17     }
```

Screen clipping taken: 4/8/2017 8:24 PM

```
10     public static void main (String[] args)
11     {
12         ComplexLinkListNode.iterate(nodeHead);
13         createClone();
14         connectSibling();
15         System.out.println("====================");
16         ComplexLinkListNode.iterate(finalizeClone());
17     }
18
19     public static void createClone()
20     {
21         ComplexLinkListNode node = nodeHead;
22         while (node != null)
23         {
24             ComplexLinkListNode clone = new ComplexLinkListNode(); //create a clone node as placeholder
25             clone.setValue(node.getValue());
26             clone.setNext(node.getNext());
27
28             node.setNext(clone);
29             node = clone.getNext();
30         }
31     }
32
33     public static void connectSibling ()
34     {
35         ComplexLinkListNode node = nodeHead;
36         while (node != null)
37         {
38             ComplexLinkListNode clone = node.getNext();
39             clone.setSibling(node.getSibling() != null ? node.getSibling().getNext() : null);//check if original node without sibling.
40             node = clone.getNext();
41         }
42     }
43
44     public static ComplexLinkListNode finalizeClone ()
45     {
46         ComplexLinkListNode node = nodeHead;
47         ComplexLinkListNode cloneHead = null, clone = null; //prepare cloneHead and clone if original is null.
48         if (node == null)
49             return cloneHead;
50
51         cloneHead = node.getNext();
52         clone = node.getNext();
53         while (node != null)
54         {
55             node.setNext(clone.getNext());
56             node = node.getNext();
57             if (node != null)
58             {
59             clone.setNext(node.getNext());
60                 clone = clone.getNext();
61             }
62             else  clone.setNext(null);  //node reaches the end, then set clone last's sibling null.
63         }
64         return cloneHead;
65     }
66
67 }
```

Screen clipping taken: 4/8/2017 8:24 PM