Алгоритмы и структуры данных. Обучение в записи

Задание 1. Удаление дубликатов в односвязном списке

Напишите метод, который удаляет все дубликаты из односвязного списка.

Пример:

Input: 1 -> 2 -> 3 -> 2 -> 4 -> 1

Output: 1 -> 2 -> 3 -> 4

```
import java.util.HashSet;


class ListNode {
   int val;
   ListNode next;
   ListNode(int val) { this.val = val; }
}


public class RemoveDuplicates {
   public static void removeDuplicates(ListNode head) {
       // ваша реализация
   }


   public static void printList(ListNode head) {
       while (head != null) {
           System.out.print(head.val + " -> ");
           head = head.next;
       }
```

```
        System.out.println("null");
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(2);
        head.next.next.next.next = new ListNode(4);
        head.next.next.next.next.next = new ListNode(1);

        System.out.println("Before removing duplicates:");
        printList(head);

        removeDuplicates(head);

        System.out.println("After removing duplicates:");
        printList(head);
    }
}
```

Подсказка № 1

Создайте вспомогательную структуру данных для хранения значений узлов, которые вы уже встретили. Для этого удобно использовать структуру данных `HashSet`, так как она эффективно поддерживает проверку наличия элемента.

Подсказка № 2

Используйте указатель `current`, который будет двигаться по списку. Этот указатель начнется с головы списка и будет перемещаться до тех пор, пока не достигнет конца.

Подсказка № 3

Для каждого узла списка проверяйте, содержится ли его значение в множестве `HashSet`. Если значение уже встречалось, это означает, что узел является дубликатом, и его нужно удалить, пропустив этот узел. Для этого измените указатель `next` текущего узла.

Подсказка № 4

Если значение узла еще не встречалось, добавьте его в множество `HashSet` и продолжайте двигаться дальше по списку.

Подсказка № 5

Обратите внимание, что если список пуст или состоит из одного элемента, то в нем нет дубликатов, и никаких изменений делать не нужно. Проверьте это условие перед началом основной работы с циклом.

Эталонное решение:

```
import java.util.HashSet;


class ListNode {
   int val;
   ListNode next;
   ListNode(int val) { this.val = val; }
}


public class RemoveDuplicates {
   public static void removeDuplicates(ListNode head) {
       if (head == null) return;


       HashSet<Integer> seen = new HashSet<>();
       ListNode current = head;
```

```
        seen.add(current.val);

        while (current.next != null) {
            if (seen.contains(current.next.val)) {
                current.next = current.next.next; // Удаляем дубликат
            } else {
                seen.add(current.next.val);
                current = current.next;
            }
        }
    }

    public static void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " -> ");
            head = head.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(2);
        head.next.next.next.next = new ListNode(4);
        head.next.next.next.next.next = new ListNode(1);
```

```
        System.out.println("Before removing duplicates:");
        printList(head);

        removeDuplicates(head);

        System.out.println("After removing duplicates:");
        printList(head);
    }
}
```

Задача 2. Поиск среднего элемента в односвязном списке

Реализуйте метод, который находит средний элемент односвязного списка за один проход по нему.

Пример:
Input: 1 -> 2 -> 3 -> 4 -> 5

Output: 3

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) { this.val = val; }
}

public class FindMiddle {
    public static ListNode findMiddle(ListNode head) {
```

```
        // ваша реализация
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(4);
        head.next.next.next.next = new ListNode(5);

        ListNode middle = findMiddle(head);
        System.out.println("Middle element: " + middle.val);
    }
}
```

Подсказка № 1

Используйте два указателя: один медленный (`slow`), который будет двигаться по одному узлу за раз, и один быстрый (`fast`), который будет двигаться по два узла за раз. Это позволит вам найти средний элемент за один проход по списку.

Подсказка № 2

Начните с того, что оба указателя (`slow` и `fast`) будут указывать на первый элемент списка (голову списка). Цикл продолжается до тех пор, пока быстрый указатель не достигнет конца списка или его следующего элемента.

Подсказка № 3

В каждом шаге цикла перемещайте медленный указатель на один узел вперед, а быстрый — на два узла вперед. Это позволит медленному указателю оказаться в середине списка, когда быстрый достигнет конца.

Подсказка № 4

Убедитесь, что ваша функция правильно обрабатывает случай пустого списка (когда `head` равен `null`). В таком случае метод должен возвращать `null`, чтобы предотвратить ошибки.

Подсказка № 5

После завершения цикла медленный указатель (`slow`) будет указывать на средний элемент списка. Верните этот узел как результат работы функции.

Эталонное решение:

```
class ListNode {
   int val;
   ListNode next;
   ListNode(int val) { this.val = val; }
}

public class FindMiddle {
   public static ListNode findMiddle(ListNode head) {
       if (head == null) return null;

       ListNode slow = head;
       ListNode fast = head;

       while (fast != null && fast.next != null) {
           slow = slow.next;
           fast = fast.next.next;
       }

       return slow;
   }
```

```
    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(3);
        head.next.next.next = new ListNode(4);
        head.next.next.next.next = new ListNode(5);

        ListNode middle = findMiddle(head);
        System.out.println("Middle element: " + middle.val);
    }
}
```

Задача 3. Слияние двух отсортированных односвязных списков

Реализуйте метод, который сливает два отсортированных односвязных списка в один отсортированный список.

Пример:

Input: 1 -> 3 -> 5 и 2 -> 4 -> 6

Output: 1 -> 2 -> 3 -> 4 -> 5 -> 6

```
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) { this.val = val; }
}
```

```java
public class MergeSortedLists {

    public static ListNode mergeTwoLists(ListNode l1, ListNode
l2) {

        // ваша реализация


    }


    public static void printList(ListNode head) {

        while (head != null) {

            System.out.print(head.val + " -> ");

            head = head.next;

        }

        System.out.println("null");

    }


    public static void main(String[] args) {

        ListNode l1 = new ListNode(1);

        l1.next = new ListNode(3);

        l1.next.next = new ListNode(5);


        ListNode l2 = new ListNode(2);

        l2.next = new ListNode(4);

        l2.next.next = new ListNode(6);


        ListNode mergedList = mergeTwoLists(l1, l2);
```

```
        System.out.println("Merged List:");

        printList(mergedList);

    }

}
```

Подсказка № 1

Создайте временный узел, который будет служить отправной точкой для нового объединенного списка. Используйте указатель (`current`) для отслеживания текущего положения в новом списке.

Подсказка № 2

Итерируйтесь по обоим входным спискам одновременно. Сравните значения текущих узлов из обоих списков и добавьте узел с меньшим значением в новый список. После добавления узла переместите указатель на следующий узел в списке, из которого был выбран элемент.

Подсказка № 3

После того как один из списков станет пустым, присоедините оставшийся элемент другого списка к новому объединенному списку. Поскольку оба списка отсортированы, оставшийся список уже будет отсортирован.

Подсказка № 4

Не забудьте обработать случай, когда один или оба из входных списков пусты. Если один из списков пуст, просто верните второй список как результат.

Подсказка № 5

Используйте временный узел (например, `dummy`) для упрощения кода и управления началом нового списка. Это поможет вам избежать необходимости проверки на пустоту списка в конце работы.

Эталонное решение:

```
class ListNode {

    int val;

    ListNode next;

    ListNode(int val) { this.val = val; }
```

```
}


public class MergeSortedLists {
    public static ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(-1); // Временный узел
        ListNode current = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                current.next = l1;
                l1 = l1.next;
            } else {
                current.next = l2;
                l2 = l2.next;
            }
            current = current.next;
        }

        // Добавляем оставшиеся элементы одного из списков
        current.next = (l1 != null) ? l1 : l2;

        return dummy.next;
    }

    public static void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " -> ");
```

```
            head = head.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        ListNode l1 = new ListNode(1);
        l1.next = new ListNode(3);
        l1.next.next = new ListNode(5);

        ListNode l2 = new ListNode(2);
        l2.next = new ListNode(4);
        l2.next.next = new ListNode(6);

        ListNode mergedList = mergeTwoLists(l1, l2);

        System.out.println("Merged List:");
        printList(mergedList);
    }
}
```