

Name – Siddhraj Gupta

UID – 22BCS15225

Section – 608/B

AP LAB ASSIGNMENT 2

1. Print Linked List

The screenshot shows a web browser window with the URL `geeksforgeeks.org/problems/print-linked-list-elements/0`. The page displays the 'Print Linked List' problem, which has been solved successfully. The 'Output Window' shows 'Compilation Results' with a message 'Problem Solved Successfully'. The 'Test Cases Passed' section shows '1112 / 1112'. The 'Attempts: Correct / Total' section shows '2 / 3'. The 'Accuracy: 66%' is also displayed. The 'Time Taken' is '1.4'. A note states: 'You get marks only for the first correct submission if you solve the problem without viewing the full solution.' The 'Solve Next' button is visible. The code editor on the right shows the following Java code:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
```

```
/* Node is defined as
class Node {
    int data;
    Node next;
    Node(int x) {
        data = x;
        next = null;
    }
}*/

Print elements of a linked list on console
Head pointer input could be NULL as well for empty list

class Solution {
    // Function to display the elements of a linked list in same line
    void printList(Node head) {
        // Add code here
        StringBuilder sb = new StringBuilder();
        while(head != null){
            sb.append(head.data).append(" ");
            head = head.next;
        }
        System.out.print(sb.toString().trim());
    }
}
```

2. Remove duplicates from a sorted list

The screenshot shows a web browser window with the URL `geeksforgeeks.org/problems/remove-duplicates-from-a-sorted-list/0`. The page displays the 'Remove duplicates from a sorted list' problem, which has been solved successfully. The 'Runtime' section shows '0 ms' and 'Beats: 100.00%'. The 'Memory' section shows '16.13 MB' and 'Beats: 67.72%'. The 'Code' editor on the right shows the following C++ code:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

```
/* Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x), next(next) {}
};

class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* res = head;
        while (head && head->next) {
            if (head->val == head->next->val) {
                head->next = head->next->next;
            } else {
                head = head->next;
            }
        }
        return res;
    }
};
```

3. R

DescriptionEditorialSolutionsSubmissions

206. Reverse Linked List

EasyTopicsCompanies

Given the `head` of a singly linked list, reverse the list, and return the *reversed list*.

Example 1:

Input: `head = [1,2,3,4,5]`

Output: `[5,4,3,2,1]`

Example 2:

Input: `head = [1,2,3,4,5]`

Output: `[5,4,3,2,1]`

Code

```

C++
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
1 // ListNode(int x, ListNode* next) : val(x), next(next) {}
2 //
3 //
4 class Solution {
5 public:
6     ListNode* reverseList(ListNode* head) {
7         ListNode* node = nullptr;
8
9         while (head != nullptr) {
10             ListNode* temp = head->next;
11             head->next = node;
12             node = head;
13             head = temp;
14         }
15
16         return node;
17     }
18 };

```

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

head = [1,2,3,4,5]

Output

4. Delete middle node of a list

DescriptionEditorialSolutionsSubmissions

2095. Delete the Middle Node of a Linked List

MediumTopicsCompaniesHint

You are given the `head` of a linked list. Delete the **middle node**, and return the *head* of the modified linked list.

The **middle node** of a linked list of size n is the $\lfloor n / 2 \rfloor$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

For $n = 1, 2, 3, 4$, and 5 , the middle nodes are $0, 1, 1, 2$, and 2 , respectively.

Example 1:

Input: `head = [1,3,4,7,1,2,6]`

Output: `[1,3,4,1,2,6]`

Example 2:

Input: `head = [1,2,3,4]`

Output: `[1,2,4]`

Code

```

C++
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
1 class Solution {
2 public:
3     ListNode* deleteMiddle(ListNode* head) {
4         if (!head || !head->next) return nullptr; // If only 1 node, return nullptr
5
6         ListNode* slow = head;
7         ListNode* fast = head;
8         ListNode* prev = nullptr; // To keep track of node before middle
9
10        // Move fast by 2 steps and slow by 1 step
11        while (fast && fast->next) {
12            prev = slow;
13            slow = slow->next;
14            fast = fast->next->next;
15        }
16
17        // Delete middle node
18        prev->next = slow->next;
19        delete slow; // Free memory
20
21        return head;
22    }
23 };

```

Testcase

Test Result

Case 1

Case 2

Case 3

head = [1,3,4,7,1,2,6]

5. Merge two sorted linked lists

DescriptionEditorialSolutionsSubmissions

21. Merge Two Sorted Lists

EasyTopicsCompanies

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted list**. The list should be made by splicing together the nodes of the first two lists.

Return the *head* of the merged list.

Example 1:

Input: `list1 = [1,2,4]`, `list2 = [1,3,4]`

Output: `[1,1,2,3,4,4]`

Example 2:

Input: `list1 = []`, `list2 = []`

Output: `[]`

Example 3:

Input: `list1 = []`, `list2 = [0]`

Output: `[0]`

Code

```

C++
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
1 //
2 //
3 class Solution {
4 public:
5     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
6         ListNode* curr1 = list1;
7         ListNode* curr2 = list2;
8         ListNode* dummyNode = new ListNode(-1);
9         ListNode* temp = dummyNode;
10
11        while (curr1 && curr2) {
12            if (curr1->val < curr2->val) {
13                temp->next = curr1;
14                temp = curr1;
15                curr1 = curr1->next;
16            }
17            else {
18                temp->next = curr2;
19                temp = curr2;
20                curr2 = curr2->next;
21            }
22        }
23
24        if (curr1) temp->next = curr1;
25        if (curr2) temp->next = curr2;
26        return dummyNode->next;
27    }
28 };

```

Testcase

Test Result

list1 = [1,2,4]

list2 = [1,3,4]

Output [1,1,2,3,4,4]

Expected [1,1,2,3,4,4]

141. Linked List Cycle

Easy

Topics

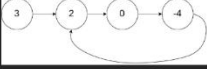
Companies

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**


Return `true` if there is a cycle in the linked list. Otherwise, return `false`.

Example 1:




Input: `head = [3,2,0,-4]`, `pos = 1`
Output: `true`
Explanation: There is a cycle in the Linked list, where the tail connects to the 1st node (0-indexed).

Example 2:



Input: `head = [1,2]`, `pos = 0`
Output: `true`
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:



Input: `head = [1]`, `pos = -1`
Output: `false`

Code

```

1  * int val;
2  * ListNode *next;
3  * ListNode(int x) : val(x), next(NULL) {}
4  * };
5  */
6  class Solution {
7  public:
8      bool hasCycle(ListNode *head) {
9          ListNode *fast = head;
10         ListNode *slow = head;
11         while(fast != NULL && fast->next != NULL)
12         {
13             fast = fast->next->next;
14             slow = slow->next;
15             if(fast == slow)
16                 return true;
17         }
18         return false;
19     }
20 };

```

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

`head =`
`[3,2,0,-4]`

`pos =`
`1`

7. Rotate a list

Problem List

Description

Editorial

Solutions

Submissions

← All Solutions

Making the List Circular

Breaking the Cycle at the Correct Position

Code

Python3

```

1  # Definition for singly-linked list.
2  # class ListNode:
3  #     def __init__(self, val=0, next=None):
4  #         self.val = val
5  #         self.next = next
6  class Solution:
7      def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
8          if head==None or head.next==None or k==0:
9              return head
10         l=1
11         curr=head
12         while curr.next:
13             curr=curr.next
14             l+=1
15         r=k%l
16         k=l-r
17         curr.next=head
18         while k>0:
19             curr=curr.next
20             k-=1
21         head=curr.next
22         curr.next = None
23         return head

```

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Input

`head =`
`[1,2,3,4,5]`

`k =`
`2`

8. Sort List

Problem List

Run

Submit

Description

Editorial

Solutions

Submissions

148. Sort List

Medium Topics Companies

Given the `head` of a linked list, return the list after sorting it in **ascending order**.

Example 1:

Input: `head = [4,2,1,3]`
Output: `[1,2,3,4]`

Example 2:

Input: `head = [-1,5,3,4,0]`
Output: `[-1,0,3,4,5]`

Example 3:

Input: `head = []`
Output: `[]`

12.2K

111

123 Online

Code

C++

Auto

```

11 class Solution {
12 public:
13     ListNode* sortList(ListNode* head) {
14         if (!head || !head->next) return head;
15
16         // Find the middle using slow and fast pointers
17         ListNode* slow = head;
18         ListNode* fast = head->next;
19         while (fast && fast->next) {
20             slow = slow->next;
21             fast = fast->next->next;
22         }
23
24         ListNode* mid = slow->next;
25         slow->next = nullptr;
26
27         // Recursively split and merge
28         ListNode* left = sortList(head);
29         ListNode* right = sortList(mid);
30
31         return merge(left, right);
32     }
33
34     ListNode* merge(ListNode* l1, ListNode* l2) {
35         ListNode dummy(0);
36         ListNode* tail = &dummy;
37         while (l1 && l2) {
38             if (l1->val < l2->val) {
39                 tail->next = l1;
40                 l1 = l1->next;
41             } else {
42                 tail->next = l2;
43                 l2 = l2->next;
44             }
45             tail = tail->next;
46         }
47         tail->next = l1 ? l1 : l2;
48         return dummy->next;
49     }
50 };

```

Saved

Ln 50, Col 27

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

`head =`
`[4,2,1,3]`

Output

`[1,2,3,4]`

9. Merge k sorted lists

Problem List

Run

Submit

Description

Editorial

Solutions

Submissions

23. Merge k Sorted Lists

Hard Topics Companies

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: `lists = [[1,4,5],[1,3,4],[2,6]]`
Output: `[1,1,2,3,4,4,5,6]`
Explanation: The linked-lists are:

```

1->4->5,
1->3->4,
2->6

```

merging them into one sorted list:
1->1->2->3->4->4->5->6

Example 2:

Input: `lists = []`
Output: `[]`

Example 3:

Input: `lists = [[]]`
Output: `[]`

Constraints:

- $k == \text{lists.length}$
- $0 \leq k \leq 10^4$
- $0 \leq \text{lists[i].length} \leq 500$
- $-10^4 \leq \text{lists[i][j]} \leq 10^4$

20.1K

253

247 Online

Code

C++

Auto

```

30     return merge(left, right);
31 }
32
33 ListNode* merge(ListNode* l1, ListNode* l2) {
34     ListNode* dummy = new ListNode(0);
35     ListNode* curr = dummy;
36
37     while (l1 && l2) {
38         if (l1->val < l2->val) {
39             curr->next = l1;
40             l1 = l1->next;
41         } else {
42             curr->next = l2;
43             l2 = l2->next;
44         }
45         curr = curr->next;
46     }
47     curr->next = l1 ? l1 : l2;
48     return dummy->next;
49 }
50
51 };
52

```

Saved

Ln 50, Col 28

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

`lists =`
`[[1,4,5],[1,3,4],[2,6]]`

Output

`[1,1,2,3,4,4,5,6]`