Data Structures: Linked Lists

Hussein Alarag

January 28, 2026

Differences between Linked Lists

There are three primary types of linked lists, each distinguished by how the nodes are connected and how the memory is managed.

1. Singly Linked List

Each node contains data and a single pointer to the **next** node. The last node points to *NULL*.

- **Pros:** Lowest memory overhead (one pointer per node); simple to implement.
- **Cons:** Unidirectional traversal only; finding the previous node requires $O(n)$ time.
- **Uses:** Implementation of Stacks; simple hash table chaining.

2. Circular Linked List

Similar to a singly linked list, but the last node points back to the **first** node, forming a loop.

- **Pros:** Any node can be a starting point for full traversal; ideal for repeated looping.
- **Cons:** Risk of infinite loops if termination logic is incorrect; slightly more complex management.
- **Uses:** Round-Robin Scheduling in OS; multiplayer games (turn cycling).

3. Doubly Linked List

Each node contains data and **two pointers**: one to the next node and one to the previous node.

- **Pros:** Bidirectional traversal; easier deletion of a node (direct access to the predecessor).
- **Cons:** Higher memory consumption (two pointers per node); more pointer updates during insertion/deletion.
- **Uses:** Web browser history (Back/Forward); Undo/Redo functionality in editors.

Question 2: Remove a Random Element from an Array

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        Random rand = new Random();
        int indexToRemove = rand.nextInt(arr.length);

        int[] newArr = new int[arr.length - 1];
        for (int i = 0, k = 0; i < arr.length; i++) {
            if (i == indexToRemove) continue;
            newArr[k++] = arr[i];
        }
    }
}
```

Question 4: Reverse an Array

```java
public static void reverseArray(int[] arr) {
    int start = 0, end = arr.length - 1;
    while (start < end) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

Question 6: Rotate Linked List Right by k

```java
public Node rotateRight(Node head, int k) {
    if (head == null || head.next == null || k == 0) return head;
    Node last = head;
    int length = 1;
    while (last.next != null) {
        last = last.next;
        length++;
    }
    last.next = head;
    k = k % length;
    for (int i = 0; i < length - k; i++) last = last.next;
    head = last.next;
    last.next = null;
```

```
    return head;
}
```

Question 8: Find Index of Data Value

```
public int findIndex(Node head, int value) {
    Node current = head;
    int index = 0;
    while (current != null) {
        if (current.data == value) return index;
        current = current.next;
        index++;
    }
    return -1;
}
```

Question 10: Remove Duplicates from DLL

```
public void removeDuplicates(DLLNode head) {
    DLLNode current = head;
    while (current != null) {
        DLLNode runner = current.next;
        while (runner != null) {
            if (runner.data == current.data) {
                runner.prev.next = runner.next;
                if (runner.next != null) runner.next.prev =
   runner.prev;
            }
            runner = runner.next;
        }
        current = current.next;
    }
}
```

Question 12: Search in Doubly Linked List

```
public boolean searchDLL(DLLNode head, int key) {
    DLLNode temp = head;
    while (temp != null) {
        if (temp.data == key) return true;
        temp = temp.next;
    }
    return false;
}
```

Question 14: Delete Node from Circular Linked List

```
public Node deleteAtPosition(Node head, int pos) {
    if (head == null) return null;
    if (pos == 0) {
        Node last = head;
        while (last.next != head) last = last.next;
        if (head == last) return null;
        last.next = head.next;
        return head.next;
    }
    Node temp = head;
    for (int i = 0; i < pos - 1; i++) temp = temp.next;
    temp.next = temp.next.next;
    return head;
}
```

Question 16: Split Circular Linked List into Two Halves

```
public void splitList(Node head) {
    if (head == null) return;
    Node slow = head, fast = head;
    while (fast.next != head && fast.next.next != head) {
        fast = fast.next.next;
        slow = slow.next;
    }
    Node head1 = head;
    Node head2 = slow.next;
    Node temp = head2;
    while (temp.next != head) temp = temp.next;
    temp.next = head2;
    slow.next = head1;
}
```