Data Structures [R1UC308B]

Module-V: Linked lists

Dr. Subhash Chandra Gupta

GALGOTIAS UNIVERSITY
School of Computer Science and Engineering
Plat No 2, Sector 17A, Yamuna Expressway
Greater Noida, Uttar Pradesh - 203201

July 31, 2025

Contents

Introduction to Linked lists

- A linked list is a linear data structure that consists of a series of nodes connected by pointers or references.
- Each node contains data and a pointer/reference to the next node in the list.
- Unlike arrays, linked lists allow for efficient insertion or removal of elements from any position in the list, as the nodes are not stored contiguously in memory.

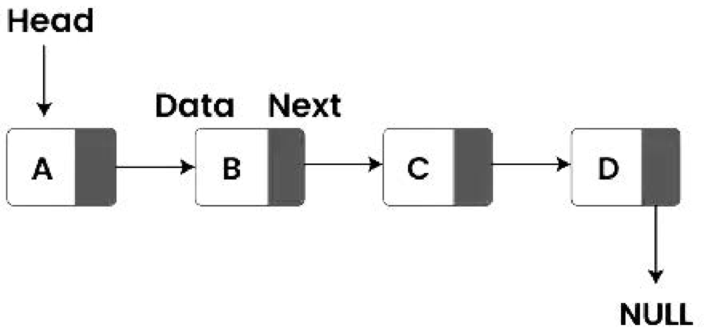


Figure: Link List representation

Comparison of Linked List vs Arrays

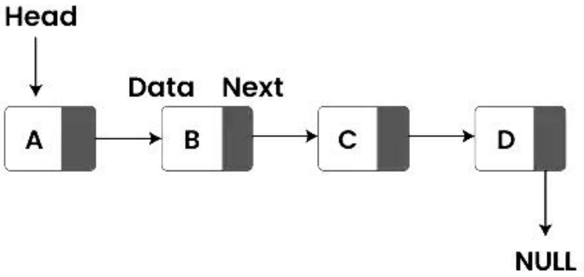
	Array	Linked List
Data Structure	Contiguous	Non-contiguous
Memory Allocation	Typically allocated to the whole array	Typically allocated one by one to individual elements
Insertion/ Deletion	Inefficient	Efficient
Access	Random and Fast	Sequential and Slow

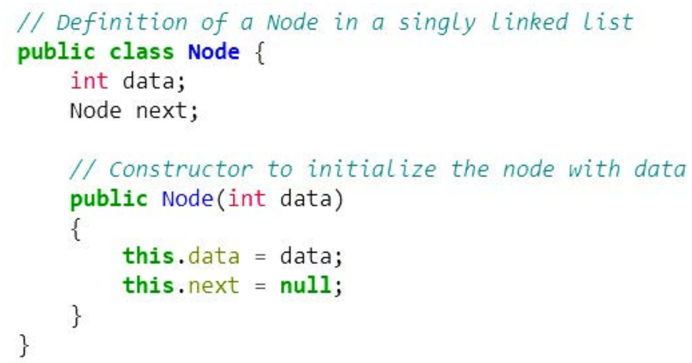
Types of Linked List

- Singly Linked List
- Doubly Linked List
- Circular Linked List
- Circular Doubly Linked List
- Header Linked List
- Multilevel Linked List
- Generalized Linked List

Singly Linked Lists

- It is a collection of nodes where each node contains a data field and a reference (link) to the next node in the sequence.
- The last node in the list points to null, indicating the end of the list.


Head
Data
Next
A
B
C
D
NULL

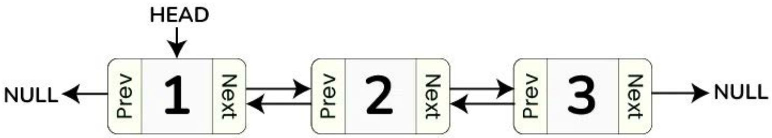


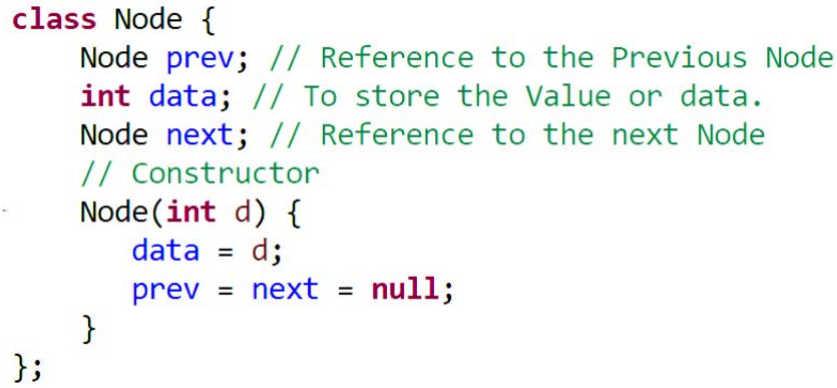
```
// Definition of a Node in a singly linked list
public class Node {
    int data;
    Node next;

    // Constructor to initialize the node with data
    public Node(int data)
    {
        this.data = data;
        this.next = null;
    }
}
```

Figure: Singly Linked Lists

Doubly Linked List

- A doubly linked list is a data structure that consists of a set of nodes, each of which contains a value and two pointers, one pointing to the previous node in the list and one pointing to the next node in the list.
- This allows for efficient traversal of the list in both directions, making it suitable for applications where frequent insertions and deletions are required.





```
class Node {
    Node prev; // Reference to the Previous Node
    int data; // To store the Value or data.
    Node next; // Reference to the next Node
    // Constructor
    Node(int d) {
        data = d;
        prev = next = null;
    }
};
```

Figure: Doubly Linked List

Circular Linked Lists

- A circular linked list is a special type of linked list where all the nodes are connected to form a circle.
- Unlike a regular linked list, which ends with a node pointing to NULL, the last node in a circular linked list points back to the first node.
- This means that you can keep traversing the list without ever reaching a NULL value.

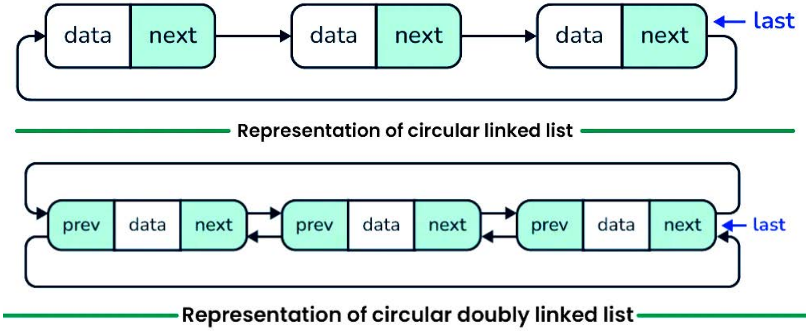

data
next
data
next
data
next
last
Representation of circular linked list
prev
data
next
prev
data
next
prev
data
next
last
Representation of circular doubly linked list

Figure: Doubly Linked List

Operations on a Linked List

- Traversing
- Searching
- Insertion
  - Insert at the beginning
  - Insert at the end
  - Insert at a specific position
- Deletion
  - Delete from the beginning
  - Delete from the end
  - Delete a specific node

Application of Linked List

- Implementing stacks and queues using linked lists.
- Using linked lists to handle collisions in hash tables.
- Representing graphs using linked lists.
- Allocating and deallocating memory dynamically.
- Addition/Multiplication of two polynomials using Linked List

Addition of two polynomials using Linked List

```
//Addition of two polynomials using Linked List
import java.lang.Math;

class Polynomial
{
public static NodeP addPolynomial(NodeP p1, NodeP p2)
{
NodeP a = p1, b = p2;
NodeP newHead = new NodeP(0, 0),c = newHead;

while (a != null || b != null)
{
if (a == null)
{
c.next = b;
break;
}
```

```
else if (b == null)
{
c.next = a;
break;
}

else if (a.pow == b.pow)
{
c.next = new NodeP(a.coeff +
b.coeff, a.pow);
a = a.next;
b = b.next;
}

else if (a.pow > b.pow)
{
c.next = new NodeP(a.coeff,
a.pow);
a = a.next;
}

else if (a.pow < b.pow)
{
c.next = new NodeP(b.coeff,
b.pow);
b = b.next;
}
c = c.next;
}
return newHead.next;
}
}

// Utilities for Linked List
// Nodes
class NodeP
{
int coeff;
int pow;
NodeP next;
NodeP(int a, int b)
{
coeff = a;
pow = b;
next = null;
}
}

// Linked List main class
public class PolyAddition
{
```

```
//Function To Display The Linked list
static void printList(NodeP trav) {
/*while (trav.next != null) {
System.out.print( trav.coeff + "x^" + trav.pow + " + ");
trav = trav.next;
}
System.out.print( trav.coeff +"\n");
*/

if(trav != null)
if(trav.pow==0)
System.out.print(trav.coeff);
else
System.out.print(trav.coeff + "x^" +  trav.pow);
else return;
while (trav.next != null)
{
```

```
trav = trav.next;
if(trav.coeff!=0) {
if(trav.coeff>0)
System.out.print(" + ");
else
System.out.print(" - ");
if(trav.pow==0)
System.out.print(Math.abs(trav.coeff));
else
System.out.print(Math.abs(trav.coeff) + "x^" +  trav.pow);
}
}
System.out.println();
}


public static void main(String args[])
```

```
{
NodeP start1 = null, cur1 = null,
start2 = null, cur2 = null;
int[] list1_coeff = {5, 5, 2};
int[] list1_pow = {2, 1, 0};
int n = list1_coeff.length;

int i = 0;
while (n-- > 0)
{
int a = list1_coeff[i];
int b = list1_pow[i];
NodeP ptr = new NodeP(a, b);
if (start1 == null)
{
start1 = ptr;
cur1 = ptr;
}
else
{
cur1.next = ptr;
cur1 = ptr;
}
i++;
}

int[] list2_coeff = {-5, -5};
int[] list2_pow = {1, 0};
n = list2_coeff.length;

i = 0;
while (n-- > 0)
{
int a = list2_coeff[i];
int b = list2_pow[i];

NodeP ptr = new NodeP(a, b);

if (start2 == null)
{
start2 = ptr;
cur2 = ptr;
}
else
{
cur2.next = ptr;
cur2 = ptr;
}
i++;
}
```

```
Polynomial obj = new Polynomial();
NodeP sum = obj.addPolynomial(start1, start2);

NodeP trav = start1;
System.out.println("1st Polynomial:");
printList(start1);

System.out.println("2nd Polynomial:");
printList(start2);

System.out.println("Addition of Polynomial:");
printList(sum);

}
}
```

Multiplication of two polynomials using Linked List

```
//Multiplication of two polynomials using Linked List
import java.lang.Math;

class PolyMultiplication
{
//Node structure containing pow
//and coefficient of variable
static class Node {
int coeff, pow;
Node next;
};

//Function add a new node at the end of list
static Node addnode(Node start, int coeff, int pow){
// Create a new node
Node newnode = new Node();
newnode.coeff = coeff;
newnode.pow = pow;
newnode.next = null;

// If linked list is empty
if (start == null)
return newnode;
// If linked list has nodes
Node ptr = start;
while (ptr.next != null)
ptr = ptr.next;
ptr.next = newnode;
return start;
}

//Function To Display The Linked list
static void printList( Node trav) {
if(trav != null)
if(trav.pow==0)
System.out.print(trav.coeff);
else
System.out.print(trav.coeff + "x^" +  trav.pow);
else return;
while (trav.next != null)
{
trav = trav.next;
if(trav.coeff!=0) {
if(trav.coeff>0)
System.out.print(" + ");
else
System.out.print(" - ");
if(trav.pow==0)
System.out.print(Math.abs(trav.coeff));
else
System.out.print(Math.abs(trav.coeff) + "x^" +  trav.pow);
```

```
}
}
System.out.println();
}
//Function to add coefficients of
//two elements having same pow
static void removeDuplicates(Node start) {
Node ptr1, ptr2, dup;
ptr1 = start;
//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 power of two elements are same
if (ptr1.pow == ptr2.next.pow) {
```

```
// Add their coefficients and put it in 1st element
ptr1.coeff = ptr1.coeff + ptr2.next.coeff;
dup = ptr2.next;
ptr2.next = ptr2.next.next;
}
else
ptr2 = ptr2.next;
}
ptr1 = ptr1.next;
}
}
//Function two Multiply two polynomial Numbers
static Node multiply(Node poly1, Node poly2, Node poly3) {
// Create two pointer and store the
// address of 1st and 2nd polynomials
Node ptr1, ptr2;
ptr1 = poly1;
```

```
ptr2 = poly2;
while (ptr1 != null) {
while (ptr2 != null) {
int coeff, pow;
// Multiply the coefficient of both
// polynomials and store it in coeff
coeff = ptr1.coeff * ptr2.coeff;
// Add the power of both polynomials
// and store it in pow
pow = ptr1.pow + ptr2.pow;
// Invoke addnode function to create
// a newnode by passing three parameters
poly3 = addnode(poly3, coeff, pow);
// move the pointer of 2nd polynomial
// two get its next term
ptr2 = ptr2.next;
}
// Move the 2nd pointer to the
// starting point of 2nd polynomial
ptr2 = poly2;
// move the pointer of 1st polynomial
ptr1 = ptr1.next;
}
// this function will be invoke to add
// the coefficient of the elements
// having same power from the resultant linked list
removeDuplicates(poly3);
return poly3;
}

public static void main(String args[]) {
Node poly1 = null, poly2 = null, poly3 = null;
// Creation of 1st Polynomial: 3x^2 + 5x^1 + 6
```

```
poly1 = addnode(poly1, -3, 2);
poly1 = addnode(poly1, 5, 1);
poly1 = addnode(poly1, -6, 0);
// Creation of 2nd polynomial: 6x^1 + 8
poly2 = addnode(poly2, 6, 1);
poly2 = addnode(poly2, 8, 0);
// Displaying 1st polynomial
System.out.print("1st Polynomial: ");
printList(poly1);
// Displaying 2nd polynomial
System.out.print("2nd Polynomial: ");
printList(poly2);
// calling multiply function
poly3 = multiply(poly1, poly2, poly3);
// Displaying Resultant Polynomial
System.out.print( "Resultant Polynomial: ");
printList(poly3);
```

```
    }
    }
```

Thank you

Please send your feedback or any queries to subhash.chandra@galgotiasuniversity.edu.in