



Explore | Expand | Enrich

<Codemithra />TM

<Codemithra />TM



Explore | Expand | Enrich

SDOT

MERGE TWO SORTED LINKED LISTS

Merge Two Sorted Linked Lists

Merge two sorted linked lists and return it as a sorted list. The list should be made by splicing together the nodes of the first two lists.

Input Format

The format for each test case is as follows:

The first line contains an integer n , the length of the first linked list.
The next line contain n integers, the elements of the linked list.
The next line contains an integer m , the length of the second linked list.
The next lines contain m integers, the elements of the second linked list.

Output Format

Output a single line of $(n + m)$ integers consisting all elements of linked lists in sorted order.

Example

Input

3

1 2 4

3

1 3 4

Output

1 1 2 3 4 4

LOGIC:

Input:

Array 1: 1 2 4

Array 2: 1 3 4

Output:

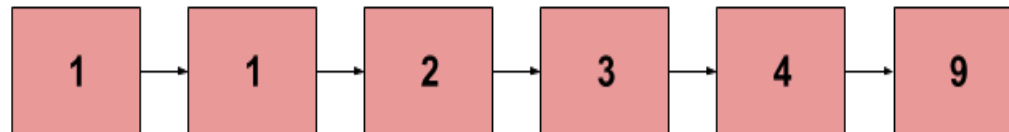
Merged Array: 1 1 2 3 4 4

Logic:

1. Start with two sorted arrays.
2. Initialize pointers for each array, starting at the first element
Pointer for Array 1: 1
Pointer for Array 2: 1
3. Compare the elements at the pointers. Choose the smaller element and append it to the merged array.
Merged Array: 1
Increment pointers for both arrays.
4. Repeat the comparison and appending process until one of the arrays is exhausted
Merged Array: 1 1 2 3
5. Once one array is exhausted, append the remaining elements from the other array to the merged array.
Merged Array: 1 1 2 3 4 4



Before Merging



After Merging

PYTHON CODE

```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def merge_sorted_lists(l1, l2):
    # Dummy node to start the merged list
    dummy = ListNode()
    current = dummy

    # Traverse both lists until one of them
    is exhausted
    while l1 is not None and l2 is not None:
        if l1.value < l2.value:
            current.next = l1
            l1 = l1.next
```

```
else:
    current.next = l2
    l2 = l2.next
    current = current.next

    # If any list is not exhausted, append
    the remaining nodes
    if l1 is not None:
        current.next = l1
    elif l2 is not None:
        current.next = l2

    # Return the merged list starting from
    the dummy node's next
    return dummy.next

# Helper function to create a linked list
from a list of values
def create_linked_list(values):
```

```
if not values:
    return None
head = ListNode(values[0])
current = head
for value in values[1:]:
    current.next = ListNode(value)
    current = current.next
return head

# Helper function to print the linked list
def print_linked_list(head):
    while head is not None:
        print(head.value, end=" ")
        head = head.next
    print()
```

```
# Sample sorted values for the first and second lists
list1_values = [1, 3, 15, 67]
list2_values = [2, 4, 6]

# Create two sorted linked lists
list1 = create_linked_list(list1_values)
list2 = create_linked_list(list2_values)

# Merge the two lists
merged_list = merge_sorted_lists(list1, list2)

# Print the merged list
print("Merged List:")
print_linked_list(merged_list)
```

JAVA CODE

```
import java.io.*;
import java.util.*;
class Node{
    int data;
    Node next;
    Node(int data){
        this.data = data;
        next = null;
    }
}
class LinkedList{
    Node head;
    void add(int data ){
        Node new_node = new
Node(data);
        if(head == null){
            head = new_node;
            return;
        }
    }
}
```

```
Node current = head;
    while(current.next !=null){
        current = current.next;
    }
    current.next = new_node;
}
}
class Solution {
    static Node merge(Node head1, Node
head2){
        // Write your code here
        if(head1==null)
        {
            return head2;
        }
        if(head2==null)
        {
            return head1;
        }
    }
}
```

```
Node ansHead; //NEW HEAD
    if(head1.data<head2.data){
        ansHead=head1;
        head1=head1.next;
    }
    else{
        ansHead=head2;
        head2=head2.next;
    }
    Node temp=ansHead; // after
doing all temp will be at last
position
    while(head1!=null &&
head2!=null){
        if(head1.data<head2.data){
            temp.next=head1; //
ansHead is already decided so we will
traverse from //temp.next
            head1=head1.next;

        }
    }
```

```
    else{
        temp.next=head2;
        head2=head2.next;
    }
    temp=temp.next;
    }
    if(head1!=null){
        temp.next=head1;
    }
    if(head2!=null){
        temp.next=head2;
    }
    return ansHead;
    }
}

public class Main {
    public static void main(String
args[]) {
```

```
Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    LinkedList l1= new
LinkedList();
    for(int i=0;i<n;i++){
        l1.add(sc.nextInt());
    }
    int m = sc.nextInt();
    LinkedList l2 = new
LinkedList();
    for(int i=0;i<m;i++){
        l2.add(sc.nextInt());
    }
```

```
Solution Ob = new Solution();
    Node head = Ob.merge(l1.head,
l2.head);
    while(head != null){
        System.out.print(head.
data + " ");
        head = head.next;
    }
}
```

PALINDROME LIST

Given a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.

Input Format

You will be provided with the linked list's head.

Output Format

Return true if the given list is a palindrome, else false..

Example 1

Input

5

1 2 3 2 1

Output

true

Example 2

Input

6

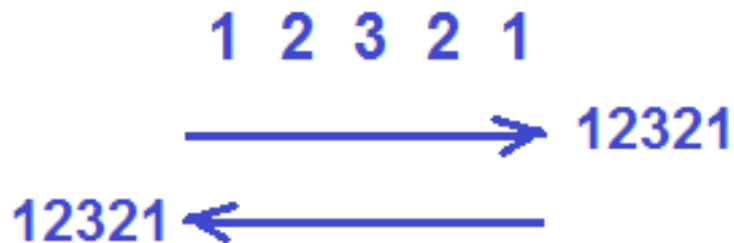
10 20 30 10 50 30

Output:

false

LOGIC

Check Number is Palindrome



"12321" is palindrome because if you read number from front or from back, it remains same.

Example

Input

3

1 2 4

3

1 3 4

Output

1 1 2 3 4 4

PYTHON CODE

```
class ListNode:
    def __init__(self, value=0,
next=None):
        self.value = value
        self.next = next

def is_palindrome(head):
    # Function to reverse a linked
list
    def reverse_list(node):
        prev = None
        while node:
            next_node = node.next
            node.next = prev
            prev = node
            node = next_node
        return prev
```

```
# Function to check if two lists are
equal
def are_lists_equal(list1, list2):
    while list1 and list2:
        if list1.value !=
list2.value:
            return False
        list1 = list1.next
        list2 = list2.next
    return True

# Find the middle of the linked
list
slow = fast = head
while fast and fast.next:
    slow = slow.next
    fast = fast.next.next
```

```
# Reverse the second half of the
linked list
    reversed_second_half =
reverse_list(slow)

    # Compare the first half with the
reversed second half
    return are_lists_equal(head,
reversed_second_half)

# Example usage:
# Get input from the user
n = int(input("Enter the number of
elements in the linked list: "))
values = list(map(int, input("Enter
the elements of the linked list: ").
split()))
```

```
# Create a linked list
head = None
for value in reversed(values):
    head = ListNode(value, head)

# Check if the linked list is a
palindrome
result = is_palindrome(head)

# Print the result
print(result)
```

JAVA CODE


```
import java.util.Scanner;
class ListNode {
    char val;
    ListNode next;
    public ListNode(char val) {
        this.val = val;
    }
}
public class PalindromeList {
    public static boolean
    isPalindrome(ListNode head) {
        if (head == null || head.next
        == null) {
            return true;
        }
        ListNode slow = head, fast =
        head;
        while (fast != null && fast.
        next != null) {
```

```
        slow = slow.next;
            fast = fast.next.next;
        }
        ListNode secondHalf =
        reverseList(slow);
        while (secondHalf != null) {
            if (head.val !=
            secondHalf.val) {
                return false; // The
                list is not a palindrome
            }
            head = head.next;
            secondHalf = secondHalf.
            next;
        }
        return true; // The list is a
        palindrome
    }
}
```

```
private static ListNode
reverseList(ListNode head) {
    ListNode prev = null;
    ListNode current = head;
    ListNode next;
    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.
in);
```

```
System.out.println("Enter the number of
elements in the linked list:");
    int n = scanner.nextInt();
    System.out.println("Enter the
elements of the linked list:");
    ListNode list = new ListNode(scanner.
next().charAt(0));
    ListNode current = list;
    for (int i = 1; i < n; i++) {
        current.next = new
ListNode(scanner.next().charAt(0));
        current = current.next;
    }
    System.out.println("Is the linked
list a palindrome? " + isPalindrome(list));
    scanner.close();
}
}
```

MERGE K SORTED LINKED LISTS

You are given an array of k linkedlists, where each linked list is sorted in ascending order.
Write a Java program to merge all the linked lists into a single linked list and return it.

Input:

An integer ``k`` representing the number of linkedlists.

For each linkedlist:

An integer ``size`` representing the number of elements in the linkedlist.

``size`` integers representing the elements of the linkedlist in sorted order.

Output:

A single line containing the elements of the merged linked list in sorted order.

Example 1:

Input:

Enter the number of linkedlists (k):

2

Enter the size of linkedlist 1:

4

Enter the elements of linkedlist 1:

1 4 5 6

Enter the size of linkedlist 2:

3

Enter the elements of linkedlist 2:

1 2 4 5

Output:

Merged Linked List:

1 1 2 4 4 5 6

Example 2:

Input:

Enter the number of linkedlists (k):

3

Enter the size of linkedlist 1:

3

Enter the elements of linkedlist 1:

1 3 7

Enter the size of linkedlist 2:

2

Enter the elements of linkedlist 2:

2 4

Enter the size of linkedlist 3:

4

Enter the elements of linkedlist 3:

5 6 8 9

Output:

Merged Linked List:

1 2 3 4 5 6 7 8 9

LOGIC

- ❑ Create a priority queue (min-heap) to keep track of the smallest element from each linked list.
- ❑ Insert the first element from each linked list into the priority queue along with the index of the linked list.
- ❑ Pop the smallest element from the priority queue.
- ❑ Append the popped element to the merged linked list.
- ❑ If there is a next element in the linked list from which the element was popped, insert that next element into the priority queue.
- ❑ Repeat steps 3-5 until the priority queue is empty.
- ❑ The merged linked list is now sorted.

PYTHON CODE


```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def merge_k_sorted_lists(lists):
    from heapq import heapify, heappush,
    heappop

    # Create a min heap and heapify it with
    tuples (value, node)
    heap = [(node.value, node) for node in
    lists if node]
    heapify(heap)
    # Dummy node to start the merged list
    dummy = ListNode()
    current = dummy
```

```
# Continue until the heap is not empty
while heap:
    _, min_node = heappop(heap)
    current.next = min_node
    current = current.next

    # Add the next node of the min_node
    to the heap
    if min_node.next:
        heappush(heap, (min_node.next.
        value, min_node.next))

    # Return the merged list starting from
    the dummy node's next
    return dummy.next

# Helper function to create a linked list
from a list of values
def create_linked_list(values):
```

```
if not values:
    return None
head = ListNode(values[0])
current = head
for value in values[1:]:
    current.next = ListNode(value)
    current = current.next
return head
# Helper function to print the linked
list
def print_linked_list(head):
    while head is not None:
        print(head.value, end=" ")
        head = head.next
    print()
# Get input from the user
k = int(input("Enter the number of
linked lists (k): "))
lists = []
```

```
for i in range(k):
    elements = list(map(int,
input(f"Enter the elements of linked
list {i+1}: ").split()))
    lists.
append(create_linked_list(elements))

# Merge the k sorted lists
merged_list =
merge_k_sorted_lists(lists)

# Print the merged list
print("\nMerged Linked List:")
print_linked_list(merged_list)
```

JAVA CODE

```
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Scanner;
class ListNode {
    int val;
    ListNode next;

    public ListNode(int val) {
        this.val = val;
    }
}
public class MergeKSortedArrays {
    public static ListNode
mergeKLists(List<ListNode> lists) {
    if (lists == null || lists.
isEmpty()) {
        return null;
    }
}
```

```
PriorityQueue<ListNode> minHeap = new
PriorityQueue<>((a, b) -> a.val - b.val);
    for (ListNode list : lists) {
        if (list != null) {
            minHeap.offer(list);
        }
    }
    ListNode dummy = new ListNode(0);
    ListNode current = dummy;
    while (!minHeap.isEmpty()) {
        ListNode minNode = minHeap.
poll();

        current.next = minNode;
        current = current.next;

        if (minNode.next != null) {
```

```
minHeap.offer(minNode.next);
        }
    }
    return dummy.next;
}
private static void printLinkedList(ListNode head) {
    while (head != null) {
        System.out.print(head.val + " ");
        head = head.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.
in);
```

```
System.out.println("Enter the number of
linked-lists (k):");
    int k = scanner.nextInt();
    List<ListNode> lists = new
ArrayList<>();
    for (int i = 0; i < k; i++) {
        System.out.println("Enter the
size of linked-list " + (i + 1) + ":");
        int size = scanner.nextInt();
        System.out.println("Enter the
elements of linked-list " + (i + 1) + ":");
        ListNode head = new
ListNode(scanner.nextInt());
        ListNode current = head;
```

```
for (int j = 1; j < size; j++) {  
    current.next = new ListNode(scanner.nextInt());  
    current = current.next;  
}  
lists.add(head);  
}  
ListNode mergedList = mergeKLists(lists);  
System.out.println("Merged Linked List:");  
printLinkedList(mergedList);  
scanner.close();  
}  
}
```

REVERSE K ELEMENTS

Reversing a Linked List Given a linked list and a positive number k , reverse the nodes in groups of k . All the remaining nodes after multiples of k should be left as it is.

Example 1:

Input:

Linked list: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$

k: 3

Output:

Result: $3 \rightarrow 2 \rightarrow 1 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 9 \rightarrow 8 \rightarrow 7$

Example 2:

Input:

Linked list: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$

k: 2

Output:

Result: $2 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 7$

LOGIC

Traverse the linked list and split in groups of k nodes.

Reverse each group of k nodes.

Connect the reversed groups.



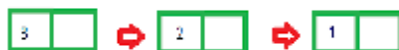
for $k=3$ break linked list in to two part.



first part contain first k elements



second part contain rest of the elements



reverse first part



join both part together

REVERSAL OF
FIRST K NODES
A LINKED LIST

PYTHON CODE

```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def reverse_k_elements(head, k):
    def reverse_group(curr, k):
        prev, temp = None, curr
        count = 0

        # Count the number of nodes in the
group        while temp is not None and count < k:
            next_node = temp.next
            temp.next = prev
            prev = temp
            temp = next_node
            count += 1

        return prev, curr, temp
```

```
# Initialize pointers
dummy = ListNode()
dummy.next = head
current = dummy

# Reverse nodes in groups of k
while current.next is not None:
    prev, start, end =
reverse_group(current.next, k)
    current.next = prev
    start.next = end
    current = start

    return dummy.next

# Helper function to create a linked list
from a list of values
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
```

```
for value in values[1:]:
    current.next = ListNode(value)
    current = current.next
return head

# Helper function to print the linked list
def print_linked_list(head):
    while head is not None:
        print(head.value, end=" ")
        head = head.next
    print()

# Get input from the user
elements = list(map(int, input("Enter the
elements of the linked list: ").split()))
k = int(input("Enter the value of k: "))
```

```
# Create a linked list
head = create_linked_list(elements)

# Reverse nodes in groups of k
result = reverse_k_elements(head, k)

# Print the result
print("\nResult:")
print_linked_list(result)
```

JAVA CODE

```
import java.util.Scanner;
class ListNode {
    int val;
    ListNode next;
    ListNode(int val) {
        this.val = val;
    }
}
public class ReverseInGroups {
    private static ListNode
reverse(ListNode head, int k) {
    ListNode prev = null;
    ListNode current = head;
    ListNode next = null;
    int count = 0;
    while (count < k && current !=
null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
```

```
count++;
    }
    if (next != null) {
        head.next = reverse(next, k);
    }
    return prev;
}
private 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) {
    Scanner scanner = new
Scanner(System.in);
```



```
System.out.println("Enter the linked list values (separated by
space):");
String[] values = scanner.nextLine().split(" ");
ListNode head = new ListNode(Integer.parseInt(values[0]));
ListNode current = head;
for (int i = 1; i < values.length; i++) {
    current.next = new ListNode(Integer.
parseInt(values[i]));
    current = current.next;
}
System.out.println("Enter the value of k:");
int k = scanner.nextInt();
head = reverse(head, k);
System.out.println("Result:");
printList(head);
scanner.close();
}
}
```

REORDER LIST

You are given the head of a singly linked list.

The list can be represented as :

$L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$

Reorder the list to be on the following form:

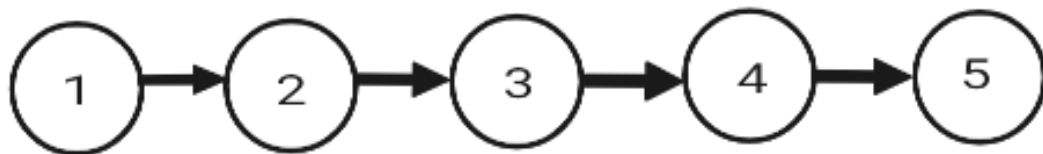
$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$

Examples:

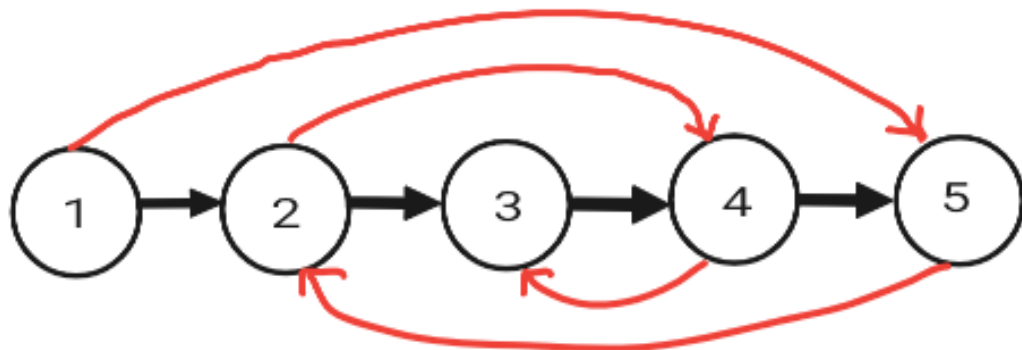
Example 1:

Input: $N = 5$, List = {1, 2, 3, 4, 5}

Input

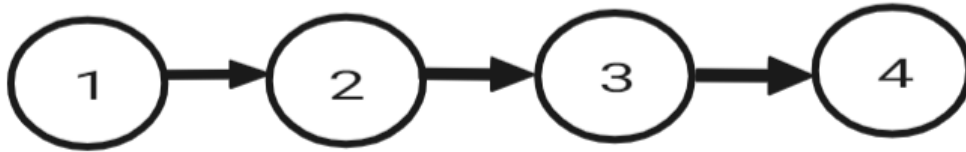


Output:

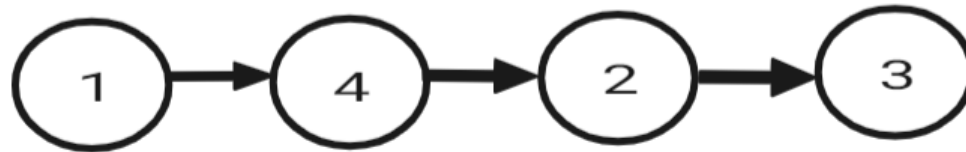


Example 2:

Input: $N = 4$, List = {1, 2, 3, 4}



Output:



PYTHON CODE

```
class ListNode:
    def __init__(self, value=0,
next=None):
        self.value = value
        self.next = next
def reorder_linked_list(head):
    def reverse_list(node):
        prev, current = None, node
        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        return prev
    def merge_lists(list1, list2):
        while list2:
            temp1, temp2 = list1.next,
list2.next
            list1.next, list2.next =
list2, temp1
```

```
list1, list2 = temp1, temp2

    if not head or not head.next:
        return head
    slow = fast = head
    while fast.next and fast.next.
next:
        slow = slow.next
        fast = fast.next.next
    reversed_second_half =
reverse_list(slow.next)
    slow.next = None
    merge_lists(head,
reversed_second_half)
    return head
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
```



```
for value in values[1:]:
    current.next = ListNode(value)
    current = current.next
return head

def print_linked_list(head):
    while head:
        print(head.value, end=" → ")
        head = head.next
    print("None")

# Get input from the user for linked
list values
elements = list(map(int, input("Enter
the elements of the linked list
(space-separated): ").split()))
```

```
# Create a linked list
linked_list =
create_linked_list(elements)

# Reorder the linked list
reorder_linked_list(linked_list)

# Print the result
print("\nReordered Linked List:")
print_linked_list(linked_list)
```

JAVA CODE

```
import java.util.Scanner;
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
    }
}
class Solution {
    ListNode middle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while (fast.next != null && fast.
next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

```
ListNode reverse(ListNode head) {
    ListNode curr = head;
    ListNode prev = null;
    ListNode forw = null;
    while (curr != null) {
        forw = curr.next;
        curr.next = prev;
        prev = curr;
        curr = forw;
    }
    return prev;
}
public void reorderList(ListNode
head) {
    if (head == null || head.next ==
null)
        return;
    ListNode mid = middle(head);
    ListNode k = reverse(mid.next);
```

```
mid.next = null;
    ListNode c1 = head;
    ListNode c2 = k;
    ListNode f1 = null;
    ListNode f2 = null;
    while (c1 != null && c2 != null)
    {
        f1 = c1.next;
        f2 = c2.next;
        c1.next = c2;
        c2.next = f1;
        c1 = f1;
        c2 = f2;
    }
}

public class Main {
    public static void main(String[]
args) {
```

```
Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the
linked list values (separated by space):
");
    String[] values = scanner.
nextLine().split(" ");
    ListNode head = new
ListNode(Integer.parseInt(values[0]));
    ListNode current = head;
    for (int i = 1; i < values.
length; i++) {
        current.next = new
ListNode(Integer.parseInt(values[i]));
        current = current.next;
    }
    Solution solution = new
Solution();
```

```
solution.reorderList(head);
    System.out.println("Result:");
    printList(head);
    scanner.close();
}
private static void printList(ListNode head) {
    while (head != null) {
        System.out.print(head.val + " →");
        head = head.next;
    }
    System.out.println("null");
}
}
```

ROTATE LIST

You are given the head of a singly linked list and an integer K, write a program to Rotate the Linked List in a clockwise direction by K positions from the last node.

Example

Input-1

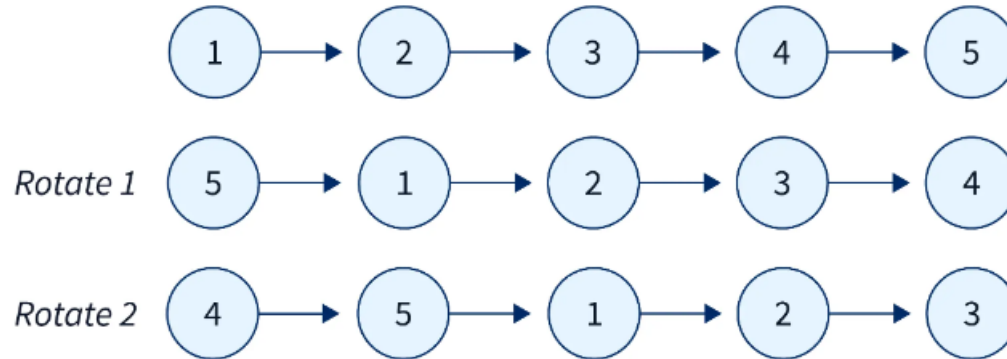
Head: 10->20->30->40->50

K: 2

Output-1

40->50->10->20->30

we have rotated the linked list exactly 2 times. On the First rotation 5 becomes head and 4 becomes tail, and on the second rotation 4 becomes head and 3 becomes the tail node.



LOGIC

- Find the length of the linked list to determine the effective rotation (adjust k if it's greater than the length).
- Iterate k times, each time moving the last node to the front of the list.
- Return the updated head of the linked list after rotations.

PYTHON CODE

```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next
def insert_node(head, val):
    new_node = ListNode(val)
    if head is None:
        return new_node
    temp = head
    while temp.next is not None:
        temp = temp.next
    temp.next = new_node
    return head
def rotate_right(head, k):
    if head is None or head.next is None:
        return head
    length = 0
    temp = head
    while temp is not None:
        length += 1
        temp = temp.next
```

k = k % length # Adjust k if it's greater than the length of the list

```
for _ in range(k):
    temp = head
    while temp.next.next is not None:
        temp = temp.next
    end = temp.next
    temp.next = None
    end.next = head
    head = end
```

return head

```
def print_list(head):
    while head.next is not None:
        print(head.value, end="->")
        head = head.next
    print(head.value)
```

```
# Get input from the user
head = None
print("Enter the linked list values (enter -1 to stop):")
while True:
    val = int(input())
    if val == -1:
        break
    head = insert_node(head, val)

print("Original list:", end=" ")
print_list(head)

# Get the value of k
k = int(input("Enter the value of k:"))

# Rotate the linked list
new_head = rotate_right(head, k)

print(f"After {k} rotations:", end=" ")
print_list(new_head)
```

JAVA CODE

```
import java.util.Scanner;

class Node {
    int num;
    Node next;

    Node(int a) {
        num = a;
        next = null;
    }
}

public class Main {
    static Node insertNode(Node head,
int val) {
        Node newNode = new Node(val);
        if (head == null) {
            head = newNode;
            return head;
        }
    }
}
```

```
Node temp = head;
    while (temp.next != null) temp =
temp.next;
    temp.next = newNode;
    return head;
}
static Node rotateRight(Node head,
int k) {
    if (head == null || head.next ==
null) return head;
    for (int i = 0; i < k; i++) {
        Node temp = head;
        while (temp.next.next !=
null) temp = temp.next;
        Node end = temp.next;
        temp.next = null;
        end.next = head;
        head = end;
    }
    return head;
}
```

```
static void printList(Node head) {
    while (head.next != null) {
        System.out.print(head.num +
        "->");
        head = head.next;
    }
    System.out.println(head.num);
}

public static void main(String
args[]) {
    Scanner scanner = new
Scanner(System.in);
    Node head = null;
    System.out.println("Enter the
linked list values (enter -1 to stop):");
    int val;
    while ((val = scanner.nextInt())
!= -1) {
```

```
head = insertNode(head, val);
    }
    System.out.print("Original list:
");
    printList(head);
    System.out.println("Enter the
value of k:");
    int k = scanner.nextInt();
    Node newHead = rotateRight(head,
k);
    System.out.print("After " + k +
" rotations: ");
    printList(newHead); // list
after rotating nodes
    scanner.close();
}
}
```


ODD EVEN LINKED LIST

Given a linked list containing integer values, segregate the even and odd numbers while maintaining their original order. The even numbers should be placed at the beginning of the linked list, followed by the odd numbers.

Example 1:

1,2,3

Input Linked List:



Output Linked List:



- In our original linked list, we have numbers: $1 \rightarrow 2 \rightarrow 3$.
- We want to separate even numbers from odd numbers and keep the order the same.
- So, first, we find the even number, which is 2. We keep it at the beginning.
- Next, we find the odd numbers, which are 1 and 3. We keep their order and put them after 2.
- Now, our new linked list is segregated: $2 \rightarrow 1 \rightarrow 3$.

Example 2:

2 → 1 → 6 → 4 → 8.

Input linked list



Output linked list



Example 2:

$2 \rightarrow 1 \rightarrow 6 \rightarrow 4 \rightarrow 8.$

In our original linked list, we have numbers: $2 \rightarrow 1 \rightarrow 6 \rightarrow 4 \rightarrow 8.$

We want to separate even numbers from odd numbers and keep the order the same.

So, first, we find the even numbers, which are 2, 6, 4, and 8. We keep them at the beginning in the same order, forming $2 \rightarrow 6 \rightarrow 4 \rightarrow 8.$

Next, we find the odd number, which is 1. We keep it after the even numbers, maintaining its original order, forming $2 \rightarrow 6 \rightarrow 4 \rightarrow 8 \rightarrow 1.$

Finally, our segregated linked list is: $2 \rightarrow 6 \rightarrow 4 \rightarrow 8 \rightarrow 1.$

Input:

Original list:

1 2 3 4 5

Output:

Segregated list (even before odd):

2 4 1 3 5



LOGIC

Edge Cases:

If the linked list is empty or has only one node, no rearrangement is needed.

Initialization:

Create two pointers, odd and even, to track the odd and even nodes separately.

Initialize odd to the head of the linked list.

Initialize even to the second node (if exists) or None if there is no second node.

Rearrangement:

Traverse the linked list using a loop until either odd or even becomes None.

Inside the loop, perform the following steps:

Connect the odd node to the next odd node (if exists).


Connect the even node to the next even node (if exists).

Move odd and even pointers to their respective next odd and even nodes.

Continue this process until the end of the linked list is reached.

Finalization:

Connect the last odd node to the starting node of the even nodes.



PYTHON CODE

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def oddEvenList(head):
    if not head or not head.next:
        return head
    # Separate odd and even nodes
    odd_head = odd = ListNode(0)
    even_head = even = ListNode(0)
    is_odd = True
    while head:
        if is_odd:
            odd.next = head
            odd = odd.next
        else:
            even.next = head
            even = even.next
        is_odd = not is_odd
        head = head.next
    # Append even nodes to the end of odd nodes
    odd.next = even_head.next
    even.next = None
    return odd_head.next
# Helper function to create a linked list from a
list of values
```

```
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head
# Helper function to print the linked list
def print_linked_list(head):
    while head:
        print(head.val, end=" ")
        head = head.next
    print()
# Get input from the user
elements = list(map(int, input("Enter the elements
of the linked list: ").split()))
head = create_linked_list(elements)
# Perform the operation
result = oddEvenList(head)
# Print the result
print("\nResulting Linked List:")
print_linked_list(result)
```

JAVA CODE

```
class Node {
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
class LinkedList {
    Node head;

    public void append(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node current = head;
        while (current.next != null) {
            current = current.next;
        }
    }
}
```

```
    }
    current.next = newNode;
}
public void segregateEvenOdd() {
    if (head == null) {
        System.out.println("The list
is empty.");
        return;
    }
    Node evenHead = null, evenTail =
null;
    Node oddHead = null, oddTail =
null;

    Node current = head;
    while (current != null) {
        int data = current.data;
        if (data % 2 == 0) { // even node
            if (evenHead == null) {
                evenHead = evenTail = current;
            }
        }
    }
}
```

```

        } else {
            evenTail.next = current;
            evenTail = current;
        }
        } else { // odd node
            if (oddHead == null) {
                oddHead = oddTail = current;
            } else {
                oddTail.next =
current;
                oddTail = current;
            }
        }
        current = current.next;
    }
    // Join even and odd lists
    if (evenHead != null) {
        evenTail.next = oddHead;
    }

```

```

    if (oddHead != null) {
        oddTail.next = null;
    }
    head = evenHead != null ? evenHead :
oddHead;
}

    public void printList() {
        Node current = head;
        while (current != null) {
            System.out.print(current.
data + " ");
            current = current.next;
        }
        System.out.println();
    }
}

```

```
public class Main {  
    public static void main(String[] args) {  
        LinkedList list = new LinkedList();  
        list.append(1);  
        list.append(2);  
        list.append(3);  
        list.append(4);  
        list.append(5);  
  
        System.out.println("Original list:");  
        list.printList();  
  
        list.segregateEvenOdd();  
  
        System.out.println("Segregated list (even before odd):");  
        list.printList();  
    }  
}
```

LONGEST VALID PARENTHESES

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring. A valid parentheses substring is defined as a substring that has an equal number of '(' and ')' characters and is correctly closed.

Examples:

Input: "(()"

Output: 2

Explanation: The longest valid parentheses substring is "()".

Input: ")()())"

Output: 4

Explanation: The longest valid parentheses substring is "()()".

LOGIC

1. Use a stack to keep track of the indices of opening parentheses.
2. Initialize the stack with 1.
3. Iterate through the string:

For each opening parenthesis '(', push its index onto the stack.

For each closing parenthesis ')', pop from the stack:

If the stack becomes empty, push the current index onto the stack.

If not empty, update the maximum length by calculating the difference between the current index and the index at the top of the stack.

4. The maximum length is the length of the longest valid parentheses substring.

PYTHON CODE

```
def longest_valid_parentheses(s):
    stack = [-1]
    max_length = 0

    for i in range(len(s)):
        if s[i] == '(':
            stack.append(i)
        else:
            stack.pop()
            if not stack:
                stack.append(i)
            else:
                max_length = max(max_length, i - stack[-1])

    return max_length

# Example usage
s = "(()))"
result = longest_valid_parentheses(s)
print("Length of the longest valid parentheses substring:", result)
```

JAVA CODE

```
import java.util.Stack;
class Main {
    public static void main(String[] args) {
        String s = "(()";
        Stack<Integer> st = new Stack<>();
        int max = 0;
        st.push(-1);
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c == '(') {
                st.push(i);
            } else {
                st.pop();
                if (st.isEmpty()) {
                    st.push(i);
                } else {
                    int len = i - st.peek();
                    max = Math.max(max, len);
                }
            }
        }
        System.out.println("Length of the longest valid parentheses substring: " + max);
    }
}
```

INFIX TO POSTFIX CONVERSION

Implement a Java program that converts an infix expression to a postfix expression using a stack.

Examples:

Convert the infix expression $(a+b)*(c-d)$ to postfix:

Infix: $(a+b)*(c-d)$

Postfix: $ab+cd-*$

Convert the infix expression $a+b*c-d/e$ to postfix:

Infix: $a+b*c-d/e$

Postfix: $abc*+de/-$

LOGIC

- ✓ Initialize an empty stack for operators and an empty list for the postfix expression.
- ✓ Scan the infix expression from left to right.
- ✓ If a character is an operand (letter or digit), add it to the postfix expression.
- ✓ If a character is '(', push it onto the stack.
- ✓ If a character is ')', pop operators from the stack and append to the postfix expression until '(' is encountered. Discard '('.
- ✓ If a character is an operator (+, -, *, /):
- ✓ Pop operators from the stack and append to postfix while the stack is not empty and the top operator has higher or equal precedence.
- ✓ Push the current operator onto the stack.
- ✓ After scanning all characters, pop any remaining operators from the stack and append to the postfix expression.
- ✓ The resulting list is the postfix expression.

PYTHON CODE

```
class InfixToPostfixConverter:
    def __init__(self):
        self.operator_stack = []
        self.postfix_expression = []
    def is_operator(self, ch):
        return ch in ['+', '-', '*', '/']
    def get_precedence(self, operator):
        if operator in ['+', '-']:
            return 1
        elif operator in ['*', '/']:
            return 2
        else:
            return 0
    def infix_to_postfix(self, infix):
        for ch in infix:
            if ch.isalnum():
                self.postfix_expression.append(ch)
            elif ch == '(':
                self.operator_stack.append(ch)
            elif ch == ')':
                while self.operator_stack and
self.operator_stack[-1] != '(':
                    self.postfix_expression.append(self.
operator_stack.pop())
```

```
self.operator_stack.pop() # Pop the '('
            elif self.is_operator(ch):
                while (self.operator_stack and
                    self.get_precedence(ch) <= self.
get_precedence(self.operator_stack[-1])):
                    self.postfix_expression.append(self.
operator_stack.pop())
                    self.operator_stack.append(ch)
            while self.operator_stack:
                self.postfix_expression.append(self.
operator_stack.pop())
            return ''.join(self.postfix_expression)
    def main():
        infix_converter = InfixToPostfixConverter()
        infix_expression = input("Enter infix
expression: ")
        postfix_expression = infix_converter.
infix_to_postfix(infix_expression)
        print(f"Infix: {infix_expression}")
        print(f"Postfix: {postfix_expression}")
    if __name__ == "__main__":
        main()
```

JAVA CODE

```
import java.util.Scanner;
import java.util.Stack;
public class Main {
    private static boolean isOperator(char ch) {
        return ch == '+' || ch == '-' || ch ==
'*' || ch == '/';
    }
    private static int getPrecedence(char
operator) {
        switch (operator) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            default:
                return 0;
        }
    }
    private static String infixToPostfix(String
infix) {
```

```
        StringBuilder postfix = new StringBuilder();
        Stack<Character> operatorStack = new
Stack<>();
        for (char ch : infix.toCharArray()) {
            if (Character.isLetterOrDigit(ch)) {
                postfix.append(ch);
            } else if (ch == '(') {
                operatorStack.push(ch);
            } else if (ch == ')') {
                while (!operatorStack.isEmpty()
&& operatorStack.peek() != '(') {
                    postfix.
append(operatorStack.pop());
                }
                operatorStack.pop(); // Pop the
 '('
            } else if (isOperator(ch)) {
                while (!operatorStack.isEmpty()
&& getPrecedence(ch) <=
getPrecedence(operatorStack.peek())) {
```

```
postfix.append(operatorStack.pop());
    }
    operatorStack.push(ch);
}
}
while (!operatorStack.isEmpty()) {
    postfix.append(operatorStack.pop());
}
return postfix.toString();
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter infix expression 1: ");
    String infix1 = scanner.nextLine();
    String postfix1 = infixToPostfix(infix1);
    System.out.println("Postfix expression 1: " + postfix1);
    scanner.close();
}
}
```

EVALUATE POSTFIX EXPRESSION

You are given a postfix expression, and your task is to evaluate it using a stack. A postfix expression is an arithmetic expression in which the operators come after their operands. For example, the infix expression `"3 + 4"` is written as `"3 4 +"` in postfix notation.

Your goal is to implement a Java program that takes a postfix expression as input, evaluates it using a stack, and outputs the result.

Examples:

Input: "5 3 4 * +"

Output: 17

Explanation: The given postfix expression represents the infix expression $(3 * 4) + 5$, which evaluates to 17.

Input: "7 2 / 4 *"

Output: 14

Explanation: The given postfix expression represents the infix expression $(7 / 2) * 4$, which evaluates to 14.

LOGIC

- ✓ Initialize an empty stack for operands.
- ✓ Scan the postfix expression from left to right.
- ✓ For each character:
- ✓ If it is a digit, push it onto the stack as an operand.
- ✓ If it is an operator (+, -, *, /), pop two operands from the stack, perform the operation and push the result back onto the stack.
- ✓ After scanning the entire expression, the result is the only element left in the stack.

PYTHON CODE

```
class PostfixEvaluator:
    def __init__(self):
        self.operand_stack = []
    def evaluate_postfix(self, expression):
        operators = {'+': lambda x, y: x + y,
                    '-': lambda x, y: x - y,
                    '*': lambda x, y: x * y,
                    '/': lambda x, y: x // y}
# Use // for integer division
        for char in expression.split():
            if char.isdigit():
                self.operand_stack.
append(int(char))
            elif char in operators:
                operand2 = self.operand_stack.
pop()
                operand1 = self.operand_stack.
pop()
```

```
        result = operators[char](operand1,
operand2)
        self.operand_stack.
append(result)
        return self.operand_stack.pop()
def main():
    postfix_evaluator =
PostfixEvaluator()
    postfix_expression = input("Enter
the postfix expression: ")
    result = postfix_evaluator.
evaluate_postfix(postfix_expression)
    print("Result:", result)
if __name__ == "__main__":
    main()
```

JAVA CODE

```
import java.util.Stack;
import java.util.Scanner;
public class Main {
    public static int evaluatePostfix(String
expression) {
        Stack<Integer> stack = new Stack<>();
        for (char c : expression.toCharArray())
        {
            if (Character.isDigit(c)) {
                stack.push(Character.
getNumericValue(c));
            } else {
                int operand2 = stack.pop();
                int operand1 = stack.pop();
                switch (c) {
                    case '+':
                        stack.push(operand1 + operand2);
                        break;
                    case '-':
                        stack.push(operand1 - operand2);
                        break;
```

```
                    case '*':
                        stack.push(operand1 * operand2);
                        break;
                    case '/':
                        stack.push(operand1 / operand2);
                        break;
                }
            }
        }
        return stack.pop();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.
in);
        System.out.print("Enter the postfix
expression: ");
        String postfixExpression = scanner.
nextLine();
        int result =
evaluatePostfix(postfixExpression);
        System.out.println("Result: " + result);
    }
}
```

BASIC CALCULATOR

Design a basic calculator using a stack data structure. The calculator should be able to perform addition, subtraction, multiplication, and division operations. Implement the calculator logic using a stack and provide a simple Java program that takes user input for arithmetic expressions and outputs the result.

Examples:

Input: $3 + 5 * 2$


Output: 13

Input: $8 / 2 - 1$

Output: 3

A decorative graphic in the top-left corner consisting of a cluster of overlapping squares in various colors (red, orange, yellow, green, blue, purple).

LOGIC:

- ✓ Initialize two stacks, one for operands (operands) and another for operators (operators).
 - ✓ Iterate through each character in the input expression from left to right.
 - ✓ If a digit is encountered, convert consecutive digits into a number and push it onto the operands stack.
 - ✓ If an operator (+, -, *, /) is encountered:
 - ✓ Pop operators from the operators stack and operands from the operands stack while the top operator on the stack has equal or higher precedence than the current operator.
 - ✓ Evaluate the popped operator and operands, then push the result back onto the operands stack.
 - ✓ Push the current operator onto the operators stack.
 - ✓ After processing all characters, evaluate any remaining operators and operands on the stacks.
 - ✓ The final result is the only element left on the operands stack.
- 
- A decorative graphic in the bottom-right corner consisting of a cluster of overlapping squares in various colors (red, orange, yellow, green, blue, purple).

PYTHON CODE

```
def calculate(expression):
    operands = []
    operators = []
    def precedence(op):
        if op in {'+', '-'}:
            return 1
        elif op in {'*', '/'}:
            return 2
        else:
            return 0
    def evaluate():
        b = operands.pop()
        a = operands.pop()
        op = operators.pop()
        if op == '+':
            result = a + b
        elif op == '-':
            result = a - b
        elif op == '*':
            result = a * b
        else:
            result = a / b
        operands.append(result)
    i = 0
    while i < len(expression):
```

```
char = expression[i]
    if char.isdigit():
        num = int(char)
        while i + 1 < len(expression) and expression[i + 1].isdigit():
            num = num * 10 + int(expression[i + 1])
            i += 1
        operands.append(num)
    elif char in {'+', '-', '*', '/'}:
        while operators and precedence(char) <= precedence(operators[-1]):
            evaluate()
        operators.append(char)
    i += 1
while operators:
    evaluate()
return operands[0]

# Example Usage:
expression1 = "3 + 5 * 2"
expression2 = "8 / 2 - 1"
result1 = calculate(expression1)
result2 = calculate(expression2)
print(f"Input: {expression1}\nOutput: {result1}")
print(f"Input: {expression2}\nOutput: {result2}")
```

JAVA CODE

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Stack<Integer> operands = new Stack<>();
        Stack<Character> operators = new Stack<>();
        String input = scanner.nextLine();
        for (int i = 0; i < input.length(); i++) {
            char c = input.charAt(i);
            if (Character.isDigit(c)) {
                int num = c - '0';
                while (i + 1 < input.length() && Character.
                    isDigit(input.charAt(i + 1))) {
                    num = num * 10 + (input.charAt(i + 1) - '0');
                    i++;
                }
                operands.push(num);
            } else if (c == '+' || c == '-' || c == '*' || c
                == '/') {
                while (!operators.isEmpty() && precedence(c) <=
                    precedence(operators.peek())) {
                    evaluate(operands, operators);
                }
            }
        }
    }
}
```

```
operators.push(c);
}
}
while (!operators.isEmpty()) {
    evaluate(operands, operators);
}
System.out.println(operands.pop());
}

private static int precedence(char c) {
    if (c == '+' || c == '-') {
        return 1;
    } else if (c == '*' || c == '/') {
        return 2;
    } else {
        return 0;
    }
}

private static void evaluate(Stack<Integer>
    operands, Stack<Character> operators) {
    int b = operands.pop();
}
```

```
int a = operands.pop();
char op = operators.pop();
int result;
if (op == '+') {
    result = a + b;
} else if (op == '-') {
    result = a - b;
} else if (op == '*') {
    result = a * b;
} else {
    result = a / b;
}
operands.push(result);
}
```



IMPLEMENT A STACK USING QUEUES

You are required to implement a stack using queues. A stack is a data structure that follows the Last In, First Out (LIFO) principle, where the last element added to the stack is the first one to be removed.

You need to design a stack that supports the following operations:

`push(x)`: Add an element `x` to the top of the stack.

`pop()`: Remove the element on the top of the stack and return it.

`top()`: Return the element on the top of the stack without removing it.

`isEmpty()`: Return `true` if the stack is empty, `false` otherwise.

Use queues to implement the stack.

Stack Operations:

1. Push
2. Pop
3. Top
4. Is Empty
5. Exit

Enter your choice (1-5): 1

Enter element to push: 12

Element 12 pushed onto the stack.

Stack Operations:

1. Push
2. Pop
3. Top
4. Is Empty
5. Exit

Enter your choice (1-5): 2

Popped element: 12



LOGIC

Initialize two queues (queue1 and queue2).

push(x) operation:

Add the element x to the non-empty queue.

pop() operation:

Move all elements except the last one from the non-empty queue to the empty queue.

Remove and return the last element from the non-empty queue.

top() operation:


Perform pop() to retrieve the top element.

Push the top element back onto the stack.

Return the top element.

isEmpty() operation:

Check if both queues are empty.



PYTHON CODE

```
from queue import Queue
class StackUsingQueues:
    def __init__(self):
        self.queue1 = Queue()
        self.queue2 = Queue()
    def push(self, x):
        # Push the element into the non-empty
        queue
        if not self.queue1.empty():
            self.queue1.put(x)
        else:
            self.queue2.put(x)
    def pop(self):
        # Move elements from the non-empty
        queue to the empty one, except the last one
        if self.is_empty():
            raise RuntimeError("Stack is empty")
        if not self.queue1.empty():
            while self.queue1.qsize() > 1:
                self.queue2.put(self.
                queue1.get())
            return self.queue1.get()
        else:
            while self.queue2.qsize() > 1:
```

```
self.queue1.put(self.queue2.get())
        return self.queue2.get()
    def top(self):
        if self.is_empty():
            raise RuntimeError("Stack is empty")
        top_element = self.pop()
        self.push(top_element) # Push the top
        element back after retrieving it
        return top_element
    def is_empty(self):
        return self.queue1.empty() and self.
        queue2.empty()
    # Example usage:
    stack = StackUsingQueues()
    stack.push(12)
    print("Element 12 pushed onto the stack.")
    try:
        popped_element = stack.pop()
        print(f"Popped element: {popped_element}")
        top_element = stack.top()
        print(f"Top element: {top_element}")
    except RuntimeError as e:
        print(f"Error: {e}")
    print(f"Is stack empty? {stack.is_empty()}")
```

JAVA CODE

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
public class StackUsingQueues {
    Queue<Integer> queue1;
    Queue<Integer> queue2;
    public StackUsingQueues() {
        queue1 = new LinkedList<>();
        queue2 = new LinkedList<>();
    }
    public void push(int x) {
        // Push the element into the non-empty queue
        if (!queue1.isEmpty()) {
            queue1.offer(x);
        } else {
            queue2.offer(x);
        }
    }
    public int pop() {
        // Move elements from the non-empty queue to the empty one, except the last one
        if (queue1.isEmpty() && queue2.isEmpty()) {
```



```
throw new RuntimeException("Stack is empty");
    }
    if (!queue1.isEmpty()) {
        while (queue1.size() > 1) {
            queue2.offer(queue1.poll());
        }
        return queue1.poll();
    } else {
        while (queue2.size() > 1) {
            queue1.offer(queue2.poll());
        }
        return queue2.poll();
    }
}

public int top() {
    int topElement = pop();
    push(topElement); // Push the top element back after retrieving it
    return topElement;
}

public boolean isEmpty() {
    return queue1.isEmpty() && queue2.isEmpty();
}
```

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    StackUsingQueues stack = new StackUsingQueues();
    while (true) {
        System.out.println("\nStack Operations:");
        System.out.println("1. Push");
        System.out.println("2. Pop");
        System.out.println("3. Top");
        System.out.println("4. Is Empty");
        System.out.println("5. Exit");
        System.out.print("Enter your choice (1-5): ");
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                System.out.print("Enter element to push: ");
                int elementToPush = scanner.nextInt();
                stack.push(elementToPush);
                System.out.println("Element " + elementToPush + " pushed onto the stack.");
                break;
```

case 2:

```
try {  
    int poppedElement = stack.pop();  
    System.out.println("Popped element: " + poppedElement);  
} catch (RuntimeException e) {  
    System.out.println("Error: " + e.getMessage());  
}
```

break;

case 3:

```
try {  
    int topElement = stack.top();  
    System.out.println("Top element: " + topElement);  
} catch (RuntimeException e) {  
    System.out.println("Error: " + e.getMessage());  
}
```

break;

case 4:

```
System.out.println("Is stack empty? " + stack.isEmpty());  
break;
```

case 5:

```
        System.out.println("Exiting program.");
        scanner.close();
        System.exit(0);
    default:
        System.out.println("Invalid choice. Please
enter a number between 1 and 5.");
    }
}
}
```

IMPLEMENT QUEUE USING STACKS

You are required to implement a queue data structure using two stacks. The goal is to simulate the functionality of a queue, which follows the First-In-First-Out (FIFO) principle, using two stacks that follow the Last-In-First-Out (LIFO) principle.

Your task is to implement the following operations for the queue:

`enqueue(x)`: Add an element `x` to the back of the queue.

`dequeue()`: Remove and return the front element of the queue. If the queue is empty, return `-1`.

`peek()`: Return the front element of the queue without removing it. If the queue is empty, return `-1`.

`isEmpty()`: Return `true` if the queue is empty, `false` otherwise.

Choose operation:

1. Enqueue
2. Dequeue
3. Check if the queue is empty
4. Exit

1

Enter the element to enqueue:

56

Choose operation:

1. Enqueue
2. Dequeue
3. Check if the queue is empty
4. Exit

1

Enter the element to enqueue:

67

Choose operation:

1. Enqueue
2. Dequeue
3. Check if the queue is empty
4. Exit

3

Is the queue empty? false

Initialization:

Create two stacks, stack1 and stack2.

stack1 is used for the enqueue operation.

stack2 is used for the dequeue operation.

Enqueue Operation:

To enqueue an element, push it onto stack1.

Dequeue Operation:

If stack2 is not empty, pop from stack2 (since it has the front element).


If stack2 is empty:

While stack1 is not empty, pop from stack1 and push onto stack2.

Pop from stack2 (now it has the front element).

Check if Queue is Empty (isEmpty):

Return True if both stack1 and stack2 are empty; otherwise, return False.



PYTHON CODE

```
class QueueUsingStacks:
    def __init__(self):
        self.stack1 = [] # Used for enqueue operation
        self.stack2 = [] # Used for dequeue operation
    def enqueue(self, element):
        # Implement enqueue operation using stack1
        self.stack1.append(element)
    def dequeue(self):
        # Implement dequeue operation using stack2 if it's not empty, otherwise transfer elements
        # from stack1 to stack2
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        # Pop from stack2 to perform dequeue operation
        if self.stack2:
            return self.stack2.pop()
        else:
            # Queue is empty
            print("Queue is empty")
    return -1 # Return a default value to indicate an empty queue
    def is_empty(self):
        # Check if both stacks are empty to determine if the queue is empty
        return not self.stack1 and not self.stack2
def main():
```

```
queue = QueueUsingStacks()
while True:
    print("\nChoose operation:")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Check if the queue is empty")
    print("4. Exit")
    choice = int(input())
    if choice == 1:
        enqueue_element = int(input("Enter the element to enqueue: "))
        queue.enqueue(enqueue_element)
    elif choice == 2:
        dequeued_element = queue.dequeue()
        if dequeued_element != -1:
            print("Dequeued element:", dequeued_element)
    elif choice == 3:
        print("Is the queue empty?", queue.is_empty())
    elif choice == 4:
        break
    else:
        print("Invalid choice. Please enter a valid option.")
if __name__ == "__main__":
    main()
```

JAVA CODE

```
import java.util.Scanner;
import java.util.Stack;
class QueueUsingStacks {
    private Stack<Integer> stack1; // Used for enqueue operation
    private Stack<Integer> stack2; // Used for dequeue operation
    public QueueUsingStacks() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }
    public void enqueue(int element) {
        // Implement enqueue operation using stack1
        stack1.push(element);
    }
    public int dequeue() {
        // Implement dequeue operation using stack2 if it's not empty, otherwise transfer
        elements from stack1 to stack2
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
    }
}
```

```
// Pop from stack2 to perform dequeue operation
    if (!stack2.isEmpty()) {
        return stack2.pop();
    } else {
        // Queue is empty
        System.out.println("Queue is empty");
        return -1; // Return a default value to indicate an empty queue
    }
}

public boolean isEmpty() {
    // Check if both stacks are empty to determine if the queue is empty
    return stack1.isEmpty() && stack2.isEmpty();
}
}

public class Main {
    public static void main(String[] args) {
        QueueUsingStacks queue = new QueueUsingStacks();
        Scanner scanner = new Scanner(System.in);
        while (true) {
```

```
System.out.println("\nChoose operation:");
System.out.println("1. Enqueue");
System.out.println("2. Dequeue");
System.out.println("3. Check if the queue is empty");
System.out.println("4. Exit");
int choice = scanner.nextInt();
switch (choice) {
    case 1:
        System.out.println("Enter the element to enqueue:");
        int enqueueElement = scanner.nextInt();
        queue.enqueue(enqueueElement);
        break;
    case 2:
        int dequeuedElement = queue.dequeue();
        if (dequeuedElement != -1) {
            System.out.println("Dequeued element: " +
dequeuedElement);
```

```
}  
  
        break;  
case 3:  
    System.out.println("Is the queue empty? " + queue.isEmpty());  
    break;  
case 4:  
    System.exit(0);  
    break;  
default:  
    System.out.println("Invalid choice. Please enter a valid  
option.");  
    }  
    }  
}
```


ZIG ZAG LEVEL ORDER TRAVERSAL

Given the root node of a tree, print its nodes in zig zag order, i.e. print the first level left to right, next level right to left, third level left to right and so on.

Note

You need to complete the given function. The input and printing of output will be handled by the driver code.

Input Format

The first line of input contains a string representing the nodes, N is to show null node.

Output Format

For each test case print the nodes of the tree in zig zag traversal.

Test cases:

Example 1

Input

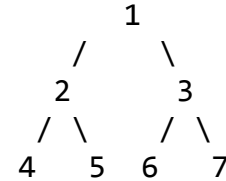
1 2 3 4 5 6 7

Output

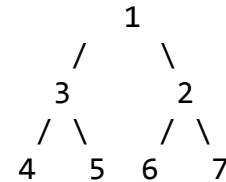
1 3 2 4 5 6 7

Explanation

Original tree was:



After Zig Zag traversal, tree formed would be:



Test cases:

Example 2

Input

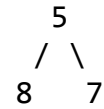
5 8 7

Output

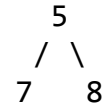
5 7 8

Explanation

Original Tree was:



New tree formed is:



LOGIC

- We use two stacks: `currentLevel` to store nodes at the current level, and `nextLevel` for the next level.
- Start with the root node and push it onto `currentLevel`.
- While `currentLevel` is not empty:
- Pop a node from `currentLevel`.
- Print its data.
- If traversing from left to right, push its left and right children onto `nextLevel` if they exist.
- If traversing from right to left, push its right and left children onto `nextLevel` if they exist.
- If `currentLevel` becomes empty, switch the values of `currentLevel` and `nextLevel`.
- Repeat until all nodes are traversed.

PYTHON CODE

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def zigzag_level_order_traversal(root):
    if not root:
        return
    current_level = []
    next_level = []
    left_to_right = True
    current_level.append(root)
    while current_level:
        node = current_level.pop()
        print(node.data, end=" ")
        if left_to_right:
            if node.left:
                next_level.append(node.left)
            if node.right:
                next_level.append(node.right)
```

```
else:
    if node.right:
        next_level.append(node.right)
    if node.left:
        next_level.append(node.left)
    if not current_level:
        left_to_right = not left_to_right
        current_level, next_level = next_level,
        current_level
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)
zigzag_level_order_traversal(root)
```


JAVA CODE

```
import java.util.LinkedList;
import java.util.Queue;
import java.io.*;
import java.util.*;
class Main {
    static Node buildTree(String str) {
        if (str.length() == 0 || str.
charAt(0) == 'N') {
            return null;
        }
        String ip[] = str.split(" ");
        Node root = new Node(Integer.
parseInt(ip[0]));
        Queue<Node> queue = new
LinkedList<>();
        queue.add(root);
        int i = 1;
        while (queue.size() > 0 && i < ip.
length) {
            Node currNode = queue.peek();
            queue.remove();
```

```
String currVal = ip[i];
            if (!currVal.equals("N")) {
                currNode.left = new
Node(Integer.parseInt(currVal));
                queue.add(currNode.left);
            }
            i++;

            if (i >= ip.length) {
                break;
            }
            currVal = ip[i];
            if (!currVal.equals("N")) {
                currNode.right = new
Node(Integer.parseInt(currVal));
                queue.add(currNode.right);
            }
            i++;
        }
        return root;
    }
}
```

```
public static void main(String[] args) throws
IOException {
    BufferedReader br = new
    BufferedReader(new InputStreamReader(System.
in));
    String s1 = br.readLine();
    Node root1 = buildTree(s1);
    Solution g = new Solution();
    g.binaryTreeZigZagTraversal(root1);
}
}
class Node {
    int data;
    Node left;
    Node right;
    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}
```

```
class Solution {
    public static void
    binaryTreeZigZagTraversal(Node root) {
        if (root == null) {
            return;
        }
        Stack<Node> currentLevel = new
    Stack<>();
        Stack<Node> nextLevel = new Stack<>();
        boolean leftToRight = true;
        currentLevel.push(root);

        while (!currentLevel.isEmpty()) {
            Node node = currentLevel.pop();
            System.out.print(node.data + " ");

            if (leftToRight) {
                if (node.left != null) {
                    nextLevel.push(node.left);
                }
            }
        }
    }
}
```

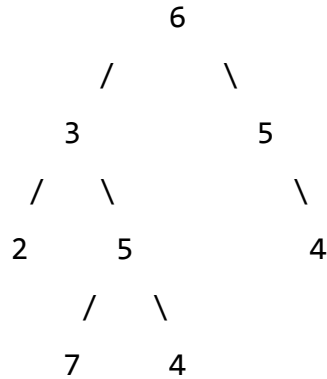
```
    if (node.right != null) {  
        nextLevel.push(node.  
right);  
    }  
    } else {  
        if (node.right != null) {  
            nextLevel.push(node.  
right);  
        }  
        if (node.left != null) {  
            nextLevel.push(node.left);  
        }  
    }  
}
```

```
if (currentLevel.isEmpty()) {  
    leftToRight = !leftToRight;  
    Stack<Node> temp =  
currentLevel;  
    currentLevel = nextLevel;  
    nextLevel = temp;  
}  
}  
}
```

SUM ROOT TO LEAF NODES

Given a binary tree, where every node value is a Digit from 1-9. Find the sum of all the numbers which are formed from root to leaf paths.

For example, consider the following Binary Tree.



There are 4 leaves, hence 4 root to leaf paths:

Path	Number
6->3->2	632
6->3->5->7	6357
6->3->5->4	6354
6->5->4	654

Answer = 632 + 6357 + 6354 + 654 = 13997

LOGIC

Create a Node class to represent the nodes of the binary tree with a value, left, and right children.

Create a BinaryTree class with a method to calculate the sum of all numbers formed from root to leaf paths.

In the sum calculation method:

- If the current node is None, return 0.
- Update the running total by multiplying it by 10 and adding the current node's value.
- If the current node is a leaf, return the updated total.
- Recursively call the method on the left and right children, passing the updated total.
- The result is the sum of all numbers formed from root to leaf paths.

In the main program, create a BinaryTree instance, build the tree, and call the sum calculation method with the root node.

Print the result.

PYTHON CODE

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
class BinaryTree:
    def __init__(self):
        self.root = None
    def tree_paths_sum_util(self, node, val):
        if not node:
            return 0
        val = val * 10 + node.data
        if not node.left and not node.right:
            return val
        return (self.tree_paths_sum_util(node.left,
val) +
                self.tree_paths_sum_util(node.right,
val))
```

```
def tree_paths_sum(self, node):
    return self.tree_paths_sum_util(node, 0)
# Example usage
tree = BinaryTree()
tree.root = Node(6)
tree.root.left = Node(3)
tree.root.right = Node(5)
tree.root.right.right = Node(4)
tree.root.left.left = Node(2)
tree.root.left.right = Node(5)
tree.root.left.right.right = Node(4)
tree.root.left.right.left = Node(7)
print("Sum of all paths is", tree.
tree_paths_sum(tree.root))
```

JAVA CODE

```
class Node {
    int data;
    Node left, right;
    Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {
    Node root;

    int treePathsSumUtil(Node node, int val) {
        if (node == null)
            return 0;
        val = (val * 10 + node.data);
        if (node.left == null && node.right == null)
            return val;
```

```
        return treePathsSumUtil(node.left, val) +
            treePathsSumUtil(node.right, val);
    }

    int treePathsSum(Node node) {
        return treePathsSumUtil(node, 0);
    }
}

public class Main {
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(6);
        tree.root.left = new Node(3);
        tree.root.right = new Node(5);
        tree.root.right.right = new Node(4);
```

```
tree.root.left.left = new Node(2);
    tree.root.left.right = new Node(5);
    tree.root.left.right.right = new Node(4);
    tree.root.left.right.left = new Node(7);
    System.out.print("Sum of all paths is " + tree.treePathsSum(tree.root));
}
}
```

BINARY RIGHT SIDE VIEW

You are given a root pointer to the root of binary tree. You have to print the right view of the tree from top to bottom.

Note

The right view of a tree is the set of nodes that are visible from the right side.

You need to complete the given function. The input and printing of output will be handled by the driver code.

Input Format

The first line contains the number of test cases.

The second line contains a string giving array representation of a tree, if the root has no children give N in input.

Output Format

For each test case print the right view of the binary tree.

Example 1

Input

1

1 2 3

1

/ \

2

3

Output

1 3

Explanation

'1' and '3' are visible from the right side.

Example 2

Input:

1

1 2 3 N N 4

1

/ \

2 3

/

4

Output

1 3 4

Explanation

'1', '3', and '4' are visible from the right side.

LOGIC

1. Initialize an empty queue.
2. Enqueue the root of the tree.
3. While the queue is not empty:
 - Get the size of the current level (``level_size``).
 - Traverse the nodes at the current level:
 - Dequeue a node from the front of the queue.
 - If it is the last node in the level, add its value to the result (rightmost node).
 - Enqueue its left and right children (if they exist).
4. Return the result, which contains the values of the rightmost nodes at each level.

PYTHON CODE

```
from collections import deque
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
class Solution:
    def rightView(self, root):
        result = []
        if not root:
            return result
        queue = deque()
        queue.append(root)
        while queue:
            level_size = len(queue)
            for i in range(level_size):
                current = queue.popleft()
                if i == level_size - 1:
                    result.append(current.data)
                if current.left:
                    queue.append(current.left)
                if current.right:
                    queue.append(current.right)
            return result
def buildTree(arr):
    if not arr or arr[0] == 'N':
        return None
    root = Node(int(arr[0]))
```

```
queue = deque([root])
i = 1
while queue and i < len(arr):
    current_node = queue.popleft()

    if arr[i] != 'N':
        current_node.left = Node(int(arr[i]))
        queue.append(current_node.left)
    i += 1
    if i < len(arr) and arr[i] != 'N':
        current_node.right = Node(int(arr[i]))
        queue.append(current_node.right)
    i += 1
return root

if __name__ == "__main__":
    t = int(input())
    for _ in range(t):
        s = input().split()
        root = buildTree(s)
        tree = Solution()
        arr = tree.rightView(root)
        for x in arr:
            print(x, end=" ")
        print()
```

JAVA CODE

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

class Node {
    int data;
    Node left;
    Node right;

    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}

class Solution {
    ArrayList<Integer> rightView(Node root) {
        ArrayList<Integer> list = new
ArrayList<>();
```

```
if (root == null) {
    return list;
}

Queue<Node> q = new LinkedList<>();
q.add(root);

while (!q.isEmpty()) {
    int n = q.size();
    for (int i = 0; i < n; i++) {
        Node curr = q.poll();
        if (i == n - 1) {
            list.add(curr.data);
        }
        if (curr.left != null) {
            q.add(curr.left);
        }
        if (curr.right != null) {
            q.add(curr.right);
        }
    }
}
return list;
}
```

```
public class Main {
    static Node buildTree(String str) {
        if (str.length() == 0 || str.charAt(0) ==
        'N') {
            return null;
        }

        String[] ip = str.split(" ");
        Node root = new Node(Integer.
        parseInt(ip[0]));
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        int i = 1;

        while (!queue.isEmpty() && i < ip.length)
        {
            Node currNode = queue.poll();

            String currVal = ip[i];
            if (!currVal.equals("N")) {
                currNode.left = new Node(Integer.
                parseInt(currVal));
                queue.add(currNode.left);
            }
        }
    }
}
```

```
        i++;

        if (i >= ip.length)
            break;

        currVal = ip[i];
        if (!currVal.equals("N")) {
            currNode.right = new Node(Integer.
            parseInt(currVal));
            queue.add(currNode.right);
        }
        i++;
    }
    return root;
}

public static void main(String[] args) throws
IOException {
    BufferedReader br = new
    BufferedReader(new InputStreamReader(System.in));
    int t = Integer.parseInt(br.readLine());
```



```
while (t-- > 0) {  
    String s = br.readLine();  
    Node root = buildTree(s);  
    Solution tree = new Solution();  
    ArrayList<Integer> arr = tree.rightView(root);  
    for (int x : arr)  
        System.out.print(x + " ");  
    System.out.println();  
}  
}
```

DIAMETER OF BINARY TREE

Given a root of a binary tree, write a function to get the diameter of the tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

Input Format

You are given a string `s` which describes the nodes of the binary tree. (The first element corresponds to the root, the second is the left child of the root and so on). In the function, you are provided with the root of the binary tree.

Output Format

Return the diameter of the binary tree.

Example 1

Input

8 2 1 3 N N 5

Output

5

Explanation

The longest path is between 3 and 5. The diameter is 5.

Example 2

Input

1 2 N

Output

2

Explanation

The longest path is between 1 and 2. The diameter is 2.

LOGIC

1. The diameter of a binary tree is the length of the longest path between any two nodes.
2. This path may or may not pass through the root.
3. To find the diameter, we need to find the height of the left and right subtrees for each node.
4. The diameter at a particular node is the sum of the height of the left and right subtrees plus 1 (for the current node).
5. The diameter of the entire tree is the maximum diameter among all nodes.

PYTHON CODE

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
class Solution:
    def __init__(self):
        self.ans = 0
    def height_and_diameter(self, root):
        if not root:
            return 0
        left_height = self.
height_and_diameter(root.left)
        right_height = self.
height_and_diameter(root.right)
        # Update diameter for the current node
        self.ans = max(self.ans, left_height +
right_height + 1)
        # Return height of the current subtree
        return 1 + max(left_height,
right_height)
```

```
def diameter(self, root):
    if not root:
        return 0
self.height_and_diameter(root)
    return self.ans
# Example usage
tree = TreeNode(8)
tree.left = TreeNode(2)
tree.right = TreeNode(1)
tree.left.left = TreeNode(3)
tree.left.right = None
tree.right.left = None
tree.right.right = TreeNode(5)
solution = Solution()
print("Diameter of the binary tree:", solution.
diameter(tree))
```


JAVA CODE

```
import java.util.LinkedList;
import java.util.Queue;
import java.io.*;
import java.util.*;
class Main {
    static Node buildTree(String str) {
        if (str.length() == 0 || str.
charAt(0) == 'N') {
            return null;
        }
        String ip[] = str.split(" ");
        Node root = new Node(Integer.
parseInt(ip[0]));
        Queue<Node> queue = new
LinkedList<>();
        queue.add(root);
        int i = 1;
        while (queue.size() > 0 && i < ip.
length) {
            Node currNode = queue.peek();
```

```
queue.remove();
            String currVal = ip[i];
            if (!currVal.equals("N")) {
                currNode.left = new Node(Integer.
parseInt(currVal));
                queue.add(currNode.left);
            }
            i++;
            if (i >= ip.length) break;
            currVal = ip[i];
            if (!currVal.equals("N")) {
                currNode.right = new Node(Integer.
parseInt(currVal));
                queue.add(currNode.right);
            }
            i++;
        }
        return root;
    }
    public static void main(String[] args) throws
IOException {
```

```

BufferedReader br =new BufferedReader(new
InputStreamReader(System.in));
    String s1 = br.readLine();
    Node root1 = buildTree(s1);
    Solution g = new Solution();
    System.out.println(g.diameter(root1));
}
}
class Node {
    int data;
    Node left;
    Node right;
    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}
class A
{
    int ans = 0;
}

```

```

class Solution {
    static int height(Node root, A a) {
        if (root == null) {
            return 0;
        }
        int left = height(root.left, a);
        int right = height(root.right, a);
        a.ans = Math.max(a.ans, left + right + 1);
        return 1 + Math.max(left, right);
    }
    public static int diameter(Node root) {
        if (root == null) {
            return 0;
        }
        A a = new A();
        height(root, a);
        return a.ans;
    }
}

```

FLATTEN BINARY TREE TO LINKED LIST

Flatten Binary Tree To Linked List. Write a program that flattens a given binary tree to a linked list.

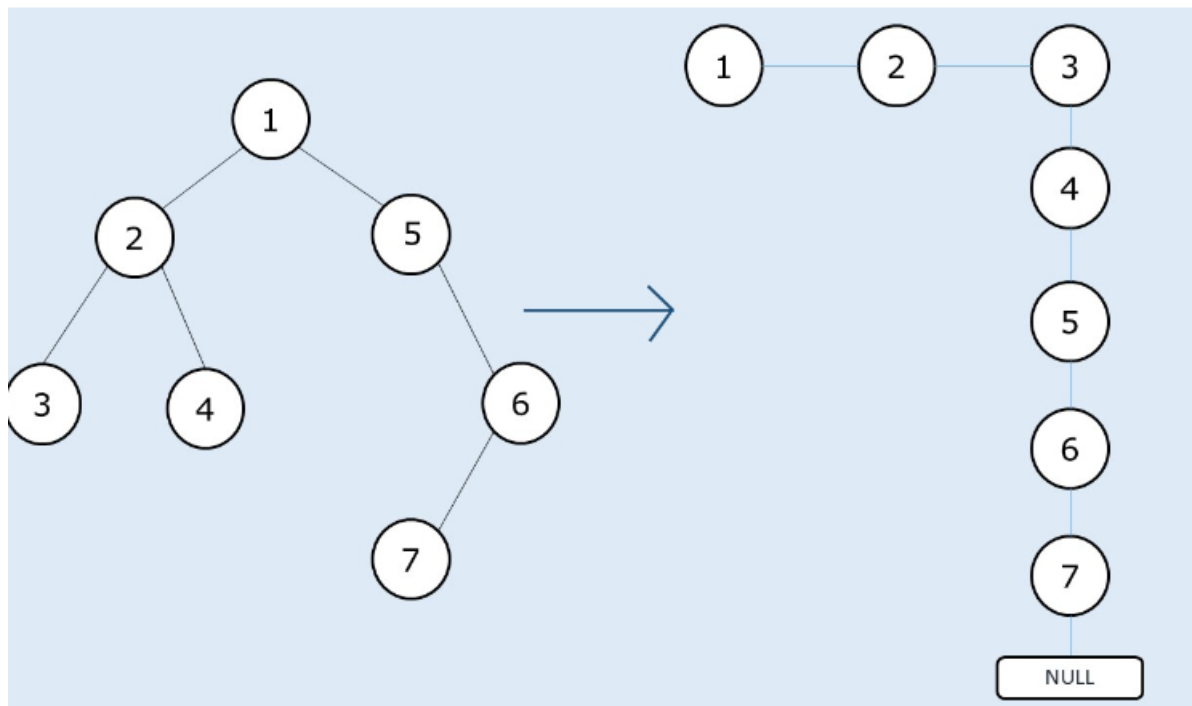
Note:

The sequence of nodes in the linked list should be the same as that of the preorder traversal of the binary tree.

The linked list nodes are the same binary tree nodes. You are not allowed to create extra nodes.

The right child of a node points to the next node of the linked list whereas the left child points to NULL.

Example



Reverse Preorder traversal

Root's left tree became the right tree

New right tree's rightmost node points to root's right tree

Solution steps–

Process right sub-tree

Process left sub-tree

Process root

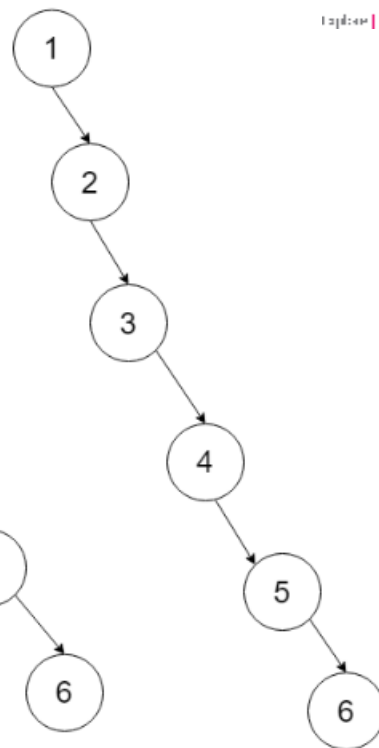
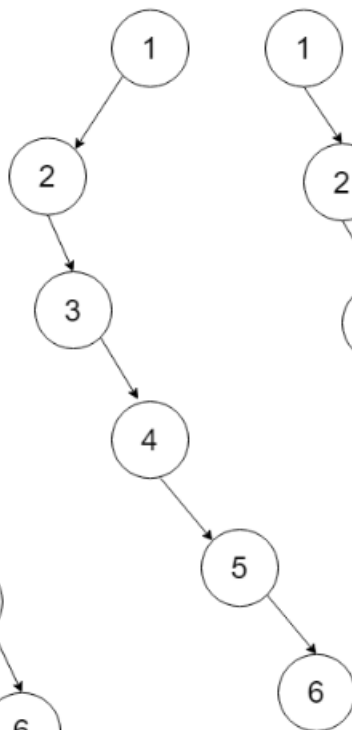
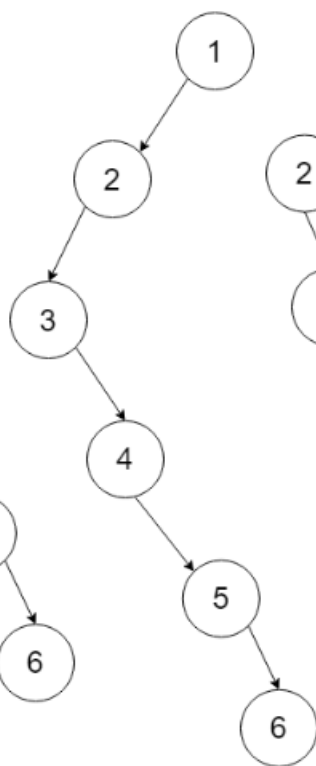
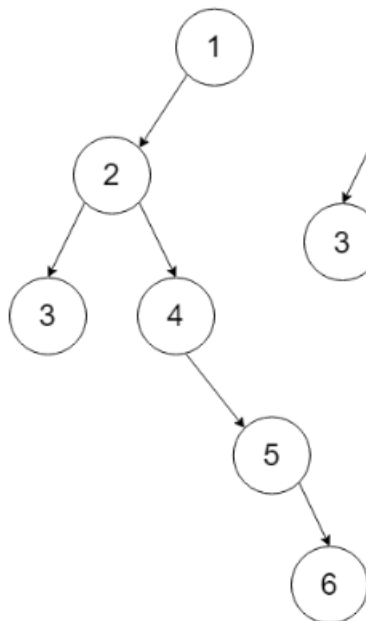
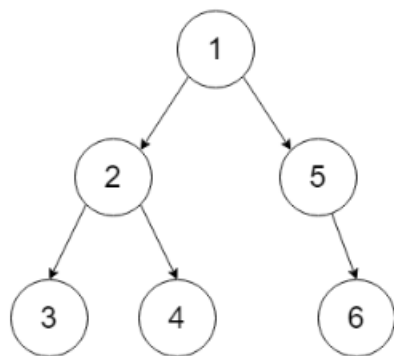
Make the left node as a right node of the root

Set right node as a right node of the new right sub-tree

Set left node to NULL

Intermediate steps

Input



LOGIC

1. Start at the root of the tree.
2. For each node in the tree:
 - a. If the node has a left child:
 - * Find the rightmost node in the left subtree.
 - * Move the right subtree of the current node to the right of the rightmost node in the left subtree.
 - * Set the left subtree as the new right subtree.
 - * Set the left child to null.
 - b. Move to the next node using the right pointer.

This process rearranges the connections in the tree, effectively turning it into a linked list. The linked list retains the order of nodes as if traversing the tree in a preorder fashion.

PYTHON CODE

```
class TreeNode:
    def __init__(self, value):
        self.val = value
        self.left = None
        self.right = None
def flatten(root):
    if not root:
        return
    current = root
    while current:
        if current.left:
            # Find the rightmost node in the left subtree
            rightmost = current.left
            while rightmost.right:
                rightmost = rightmost.right
            # Move the right subtree of the current node to the rightmost node in the
            left subtree
            rightmost.right = current.right
            # Set the left subtree as the new right subtree
            current.right = current.left
```

```
# Set the left child to null
    current.left = None
# Move to the next node in the modified tree
    current = current.right
def print_linked_list(root):
    while root:
        print(root.val, '*>', end=' ')
        root = root.right
    print('null')
# Example usage:
# Constructing a sample binary tree
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(5)
root.left.left = TreeNode(3)
root.left.right = TreeNode(4)
root.right.right = TreeNode(6)
# Flatten the binary tree to a linked list
flatten(root)
# Print the linked list
print_linked_list(root)
```

JAVA CODE

```
class TreeNode {
    int val;
    TreeNode left, right;

    public TreeNode(int value) {
        val = value;
        left = right = null;
    }
}

public class FlattenBinaryTreeToLinkedList {
    public static void flatten(TreeNode root) {
        if (root == null) {
            return;
        }

        TreeNode current = root;
        while (current != null) {
            if (current.left != null) {
                // Find the rightmost node in the left subtree
```

```
TreeNode rightmost = current.left;
    while (rightmost.right != null) {
        rightmost = rightmost.right;
    }

    // Move the right subtree of the current node to the
    rightmost node in the left subtree
    rightmost.right = current.right;

    // Set the left subtree as the new right subtree
    current.right = current.left;

    // Set the left child to null
    current.left = null;
}

// Move to the next node in the modified tree
current = current.right;
} }
```

```
public static void printLinkedList(TreeNode root) {
    while (root != null) {
        System.out.print(root.val + " -> ");
        root = root.right;
    }
    System.out.println("null");
}

public static void main(String[] args) {
    // Example usage:
    // Constructing a sample binary tree
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(5);
    root.left.left = new TreeNode(3);
    root.left.right = new TreeNode(4);
    root.right.right = new TreeNode(6);
    // Flatten the binary tree to a linked list
    flatten(root);
    // Print the linked list
    printLinkedList(root);
}
}
```


LOWEST COMMON ANCESTOR

Given the root node of a tree, whose nodes have their values in the range of integers. You are given two nodes x , y from the tree. You have to print the lowest common ancestor of these nodes.

Lowest common ancestor of two nodes x , y in a tree or directed acyclic graph is the lowest node that has both nodes x , y as its descendants.

Your task is to complete the function `findLCA` which receives the root of the tree, x , y as its parameters and returns the LCA of these values.

Input Format:

The first line contains the values of the nodes of the tree in the level order form.

The second line contains two integers separated by space which denotes the nodes x and y.

Output Format:

Print the LCA of the given nodes in a single line.

Example 1

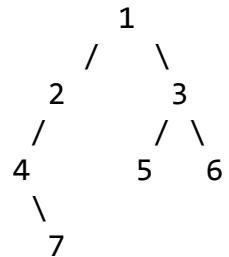
Input

```
1 2 3 4 -1 5 6 -1 7 -1 -1 -1 -1 -1  
7 5
```

Output

1

Explanation



The root of the tree is the deepest node which contains both the nodes 7 and 5 as its descendants, hence 1 is the answer.

Example 2

Input

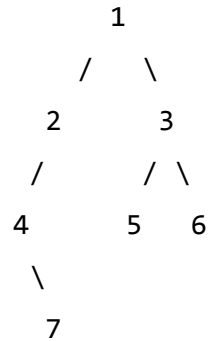
1 2 3 4 -1 5 6 -1 7 -1 -1 -1 -1 -1

4 2

Output

2

Explanation



The node with value 2 of the tree is the deepest node which contains both the nodes 4 and 2 as its descendants, hence 2 is the answer.

LOGIC

1. Perform a recursive traversal of the tree.
2. If the current node is one of the given nodes (``n1`` or ``n2``), return the current node.
3. Recursively search for the LCA in the left and right subtrees.
4. If both left and right subtrees return non-null values, the current node is the LCA.
5. Return the LCA found during the traversal.

PYTHON CODE

```
class Node:
    def __init__(self, key):
        self.data = key
        self.left = None
        self.right = None
def build_tree(arr):
    if not arr or arr[0] == "N":
        return None
    root = Node(int(arr[0]))
    queue = [root]
    i = 1
    while queue and i < len(arr):
        curr_node = queue.pop(0)
        curr_val = arr[i]
        if curr_val != "-1":
            curr_node.left =
Node(int(curr_val))
            queue.append(curr_node.left)
        i += 1
    if i >= len(arr):
        break
    curr_val = arr[i]
    if curr_val != "-1":
```

```
curr_node.right = Node(int(curr_val))
        queue.append(curr_node.right)
        i += 1
    return root
def find_lca(root, n1, n2):
    if root is None:
        return None
    if root.data == n1 or root.data == n2:
        return root
    left_lca = find_lca(root.left, n1, n2)
    right_lca = find_lca(root.right, n1, n2)
    if left_lca is not None and right_lca is
not None:
        return root
    return left_lca if left_lca is not None
else right_lca
# Input
s = input().split()
root = build_tree(s)
x, y = map(int, input().split())
# Find LCA
ans = find_lca(root, x, y)
print(ans.data if ans else "N")
```


JAVA CODE

```
import java.util.LinkedList;
import java.util.Queue;
import java.io.*;
import java.util.*;
class Main {
    static Node buildTree(String str) {
        if (str.length() == 0 || str.
charAt(0) == 'N') {
            return null;
        }
        String ip[] = str.split(" ");
        Node root = new Node(Integer.
parseInt(ip[0]));
        Queue<Node> queue = new
LinkedList<>();
        queue.add(root);
        int i = 1;
        while (queue.size() > 0 && i < ip.length) {
            Node currNode = queue.peek();
```

```
queue.remove();
            String currVal = ip[i];
            if (!currVal.equals("-1")) {
                currNode.left = new
Node(Integer.parseInt(currVal));
                queue.add(currNode.left);
            }
            i++;
            if (i >= ip.length) break;
            currVal = ip[i];
            if (!currVal.equals("-1")) {
                currNode.right = new
Node(Integer.parseInt(currVal));
                queue.add(currNode.right);
            }
            i++;
        }

        return root;
    }
}
```

```

public static void main(String[] args)
throws IOException {
    Scanner sc = new Scanner(System.in);
    String s = sc.nextLine();
    Node root = buildTree(s);
    int x = sc.nextInt();
    int y = sc.nextInt();
    Solution g = new Solution();
    Node ans = g.findLCA(root,x,y);
    System.out.println(ans.data);
}
}
class Node {
    int data;
    Node left;
    Node right;
    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}

```

```

class Solution {
    public static Node findLCA(Node node,int
n1,int n2) {
        if (node == null)
            return null;

        if (node.data == n1 || node.data ==
n2)
            return node;
        Node left_lca = findLCA(node.left,
n1, n2);
        Node right_lca = findLCA(node.right,
n1, n2);
        if (left_lca != null && right_lca !=
null)
            return node;
        return (left_lca != null) ? left_lca :
right_lca;
    }
}

```

VALIDATE BINARY SEARCH TREE

Given a binary tree with N number of nodes, check if that input tree is BST (Binary Search Tree) or not. If yes, print true, print false otherwise. A binary search tree (BST) is a binary tree data structure which has the following properties.

- The left subtree of a node contains only nodes with data less than the node's data.
- The right subtree of a node contains only nodes with data greater than the node's data.
- Both the left and right subtrees must also be binary search trees.

Input Format

The first line contains an Integer 't', which denotes the number of test cases or queries to be run. Then the test cases follow.

The first line of input contains the elements of the tree in the level order form separated by a single space.

If any node does not have a left or right child, take -1 in its place.

Output Format

For each test case, print true if the binary tree is a BST, else print false.

Output for every test case will be denoted in a separate line.

Example 1

Input

1

3 1 5 -1 2 -1 -1 -1 -1

Output

true

Explanation

Level 1: For node 3 all the nodes in the left subtree (1,2) are less than 3 and all the nodes in the right subtree (5) are greater than 3.

Level 2: For node 1: The left subtree is empty and all the nodes in the right subtree (2) are greater than 1.

For node 5: Both right and left subtrees are empty.

Level 3: For node 2, both right and left subtrees are empty. Because all the nodes follow the property of a binary search tree, the function should return true.

Example 2

Input

1

3 2 5 1 4 -1 -1 -1 -1 -1

Output

false

Explanation

For the root node, all the nodes in the right subtree (5) are greater than 3. But node with data 4 in the left subtree of node 3 is greater than 3, this does not satisfy the condition for the binary search tree. Hence, the function should return false.

LOGIC

1. Traverse the binary tree in a recursive manner.
2. At each node, check whether its data lies within the range (min_val, max_val).
3. For the left subtree, the maximum value is updated to the current node's data, and for the right subtree, the minimum value is updated.
4. Continue the traversal until all nodes are checked, and return True if all nodes satisfy the BST conditions, otherwise False.

PYTHON CODE

```
class BinaryTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def validate_bst(root, min_val=float('-inf'),
max_val=float('inf')):
    if not root:
        return True
    if not min_val <= root.data <= max_val:
        return False
    left_search = validate_bst(root.left,
min_val, root.data)
    right_search = validate_bst(root.right,
root.data, max_val)

    return left_search and right_search
```

```
# Input handling
t = int(input())
for _ in range(t):
    elements = list(map(int, input().split()))
    n = len(elements)
    tree = [BinaryTreeNode(elements[i]) if
elements[i] != -1 else None for i in range(n)]
    i, j = 0, 1
    while i < n:
        if tree[i] is not None:
            tree[i].left = tree[j]
            j += 1
            tree[i].right = tree[j]
            j += 1
        i += 1
    # Check if the tree is a valid BST
    print(validate_bst(tree[0]))
```

JAVA CODE

```
import java.util.*;
class BinaryTreeNode<T> {
    public T data;
    public BinaryTreeNode<T> left;
    public BinaryTreeNode<T> right;

    BinaryTreeNode(T data) {
        this.data = data;
        left = null;
        right = null;
    }
}

public class Main {

    public static boolean
    helper(BinaryTreeNode<Integer> root, int min,
    int max)
    {
```

```
// An empty tree is a BST
    if (root == null)
    {
        return true;
    }

    // If this node violates the min/max
    constraint
    if ((root.data <= min) || (root.data
    >= max))
    {
        return false;
    }

    boolean leftSearch = helper(root.left,
    min, root.data);
    boolean rightSearch = helper(root.
    right, root.data, max);

    return leftSearch & rightSearch;
}
```

```

public static boolean
validateBST(BinaryTreeNode<Integer> root) {
    return helper(root, Integer.MIN_VALUE,
Integer.MAX_VALUE);
}

```

```

public static void main(String[] args)
throws Throwable {
    Scanner sc = new Scanner(System.in);
    int t = sc.nextInt();
    sc.nextLine();
    while(t>0)
    {
        t--;
        String str=sc.nextLine();
        String[] str_arr = str.split(" ");
        int n = str_arr.length;
        BinaryTreeNode<Integer>[] tree =
new BinaryTreeNode[n];
        for(int i=0;i<n;i++)
        {

```

```

if(Integer.parseInt(str_arr[i])!=-1)
            tree[i] = new
BinaryTreeNode(Integer.parseInt(str_arr[i]));
            else
                tree[i] = null;
        }
        int i=0,j=1;
        while(i<n)
        {
            if(tree[i] != null)
            {
                tree[i].left = tree[j];
                j++;
                tree[i].right = tree[j];
                j++;
            }
            i++;
        }
        System.out.
println(validateBST(tree[0]));
    }
}

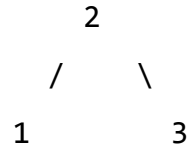
```

Kth SMALLEST ELEMENT IN A BST

Given a binary search tree (BST) and an integer k , find k -th smallest element.

Input:

BST:



k=3

Output: 3

The 3rd smallest element is 3.

Notes

Input Format: There are two arguments in the input. First one is the root of the BST and second one is an integer k.

Output: Return an integer, the k-th smallest element of the BST.

LOGIC

1. Perform an in-order traversal of the BST.
2. During the traversal, maintain a count of visited nodes (`result[0]`).
3. When the count equals `k`, store the value of the current node as the k-th smallest element (`result[1]`).
4. Continue the traversal until the count reaches `k`.

PYTHON CODE

```
class TreeNode:
    def __init__(self, val=0, left=None,
right=None):
        self.val = val
        self.left = left
        self.right = right

def kth_smallest(root, k):
    result = [0, 0]
    kth_smallest_helper(root, k, result)
    return result[1]
def kth_smallest_helper(root, k, result):
    if root is None:
        return
    kth_smallest_helper(root.left, k, result)

    # Visit the current node
    result[0] += 1
    if result[0] == k:
        result[1] = root.val
        return
```

```
kth_smallest_helper(root.right, k,
result)
```

```
# Create a sample BST
root = TreeNode(2)
root.left = TreeNode(1)
root.right = TreeNode(3)
```

```
# Set the value of k
k = 3
```

```
# Find the k-th smallest element in
the BST
result = kth_smallest(root, k)
print(f"The {k}-th smallest element is:
{result}")
```

JAVA CODE

```
class TreeNode {
    int val;
    TreeNode left, right;

    public TreeNode(int val) {
        this.val = val;
        this.left = this.right = null;
    }
}

public class Main {

    public static void main(String[] args) {
        // Create a sample BST
        TreeNode root = new TreeNode(2);
        root.left = new TreeNode(1);
        root.right = new TreeNode(3);

        // Set the value of k
        int k = 3;
```

```
// Find the k-th smallest element in the BST
        int result = kthSmallest(root, k);

        System.out.println("The " + k + "-th
smallest element is: " + result);
    }

    public static int kthSmallest(TreeNode
root, int k) {
        int[] result = new int[2];
        kthSmallestHelper(root, k, result);
        return result[1];
    }

    private static void
kthSmallestHelper(TreeNode root, int k, int[]
result) {
        if (root == null) {
            return;
        }
```

```
kthSmallestHelper(root.left, k, result);

// Visit the current node
if (++result[0] == k) {
    result[1] = root.val;
    return;
}

kthSmallestHelper(root.right, k, result);
}
```

CONVERT SORTED LIST TO BST

Given a Singly Linked List which has data members sorted in ascending order. Construct a
Balanced Binary Search Tree which has same data members as the given Linked List.

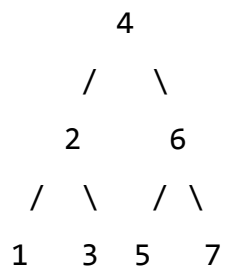
Input: Linked List 1->2->3

Output: A Balanced BST

```
    2
   / \
  1   3
```

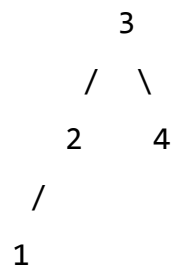
Input: Linked List 1->2->3->4->5->6->7

Output: A Balanced BST



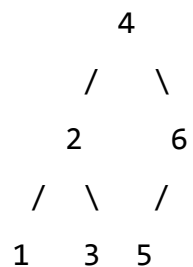
Input: Linked List 1->2->3->4

Output: A Balanced BST



Input: Linked List 1->2->3->4->5->6

Output: A Balanced BST



LOGIC

Count the Number of Nodes:

A helper function `countNodes` is used to count the number of nodes in the linked list.

Recursive Construction of BST:

The main function `sortedListToBST` calculates the number of nodes (n) in the linked list and calls the recursive function `sortedListToBSTRecur` with this count.

Base Case:

If n is less than or equal to 0, return None (base case).

Recursive Calls:

Recursively construct the left subtree (left) by calling `sortedListToBSTRecur` on the first half of the linked list.

Create the root of the current subtree with the data of the current head node.

Move the head pointer to the next node in the linked list.

Recursively construct the right subtree (right) by calling `sortedListToBSTRecur` on the second half of the linked list.

Return the Root:

Return the root of the current subtree.

Linked List Modification:

The head pointer is modified during the process to simulate traversing the linked list.

Print the Pre-order Traversal

The `preOrder` function is used to print the pre-order traversal of the constructed BST.

Driver Code:

The `push` function is used to add elements to the linked list, and the main code initializes the linked list with values, constructs the BST, and prints the pre-order traversal.

PYTHON CODE


```
class LinkedList:
    def __init__(self):
        self.head = None
    class LNode:
        def __init__(self, data):
            self.data = data
            self.next = None
            self.prev = None
    class TNode:
        def __init__(self, data):
            self.data = data
            self.left = None
            self.right = None
    def sortedListToBST(self):
        n = self.countNodes(self.head)
        return self.sortedListToBSTRecur(n)
```

```
def sortedListToBSTRecur(self, n):
    if n <= 0:
        return None
    left = self.sortedListToBSTRecur(n // 2)
    root = self.TNode(self.head.data)
    root.left = left
    self.head = self.head.next
    root.right = self.sortedListToBSTRecur(n -
n // 2 - 1)
    return root
def countNodes(self, head):
    count = 0
    temp = head
    while temp is not None:
        temp = temp.next
        count += 1
    return count
```

```
def push(self, new_data):
    new_node = self.LNode(new_data)
    new_node.prev = None
    new_node.next = self.head
    if self.head is not None:
        self.head.prev = new_node
    self.head = new_node

def printList(self, node):
    while node is not None:
        print(node.data, end=" ")
        node = node.next

def preOrder(self, node):
    if node is None:
        return
    print(node.data, end=" ")
    self.preOrder(node.left)
    self.preOrder(node.right)
```

```
if __name__ == "__main__":
    llist = LinkedList()
    llist.push(7)
    llist.push(6)
    llist.push(5)
    llist.push(4)
    llist.push(3)
    llist.push(2)
    llist.push(1)
    print("Given Linked List ")
    llist.printList(llist.head)
    root = llist.sortedListToBST()
    print("\nPre-Order Traversal of constructed BST ")
    llist.preOrder(root)
```

JAVA CODE

```
class LinkedList {  
    static LNode head;  
    class LNode {  
        int data;  
        LNode next, prev;  
        LNode(int d) {  
            data = d;  
            next = prev = null;  
        }  
    }  
    class TNode {  
        int data;  
        TNode left, right;  
        TNode(int d) {  
            data = d;  
            left = right = null;  
        }  
    }  
}
```

```
TNode sortedListToBST() {  
    int n = countNodes(head);  
    return sortedListToBSTRecur(n);  
}  
  
TNode sortedListToBSTRecur(int n) {  
    if (n <= 0)  
        return null;  
  
    TNode left = sortedListToBSTRecur(n / 2);  
    TNode root = new TNode(head.data);  
    root.left = left;  
    head = head.next;
```

```
root.right = sortedListToBSTRecur(n - n / 2 -
1);

return root;
}

int countNodes(LNode head) {
    int count = 0;
    LNode temp = head;
    while (temp != null) {
        temp = temp.next;
        count++;
    }
    return count;
}

void push(int new_data) {
    LNode new_node = new LNode(new_data);
```

```
new_node.prev = null;
new_node.next = head;
if (head != null)
    head.prev = new_node;
head = new_node;
}

void printList(LNode node) {
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
}

void preOrder(TNode node) {
    if (node == null)
        return;
```

```
System.out.print(node.data + " ");
    preOrder(node.left);
    preOrder(node.right);
}
public static void main(String[] args) {
    LinkedList llist = new LinkedList();
    llist.push(7);
    llist.push(6);
    llist.push(5);
    llist.push(4);
    llist.push(3);
    llist.push(2);
```

```
llist.push(1);
System.out.println("Given Linked List ");
    llist.printList(head);
TNode root = llist.sortedListToBST();
    System.out.println("");
    System.out.println("Pre-Order Traversal
of constructed BST ");
    llist.preOrder(root);
}
}
```

DEPTH-FIRST SEARCH

You are given a graph represented as an adjacency list. Implement the Depth-First Search (DFS) algorithm to traverse the graph and return the order in which the nodes are visited.

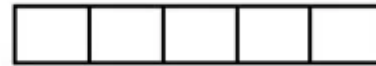
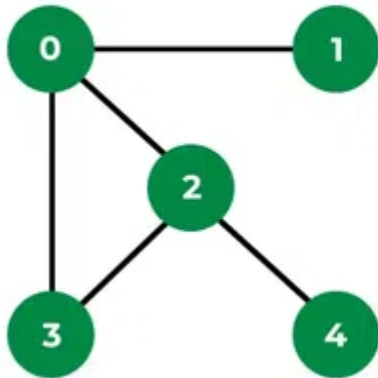
Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure

A standard DFS implementation puts each vertex of the graph into one of two categories:

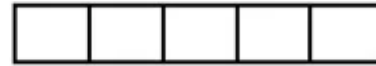
1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

Step1: Initially stack and visited arrays are empty.



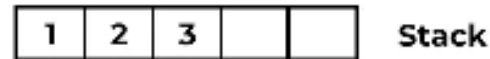
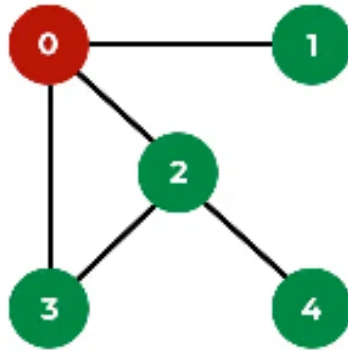
Visited



Stack

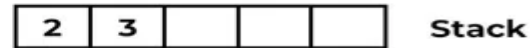
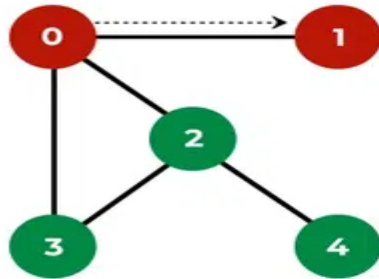
DFS on Graph

Step 2: Visit 0 and put its adjacent nodes which are not visited yet into the stack.



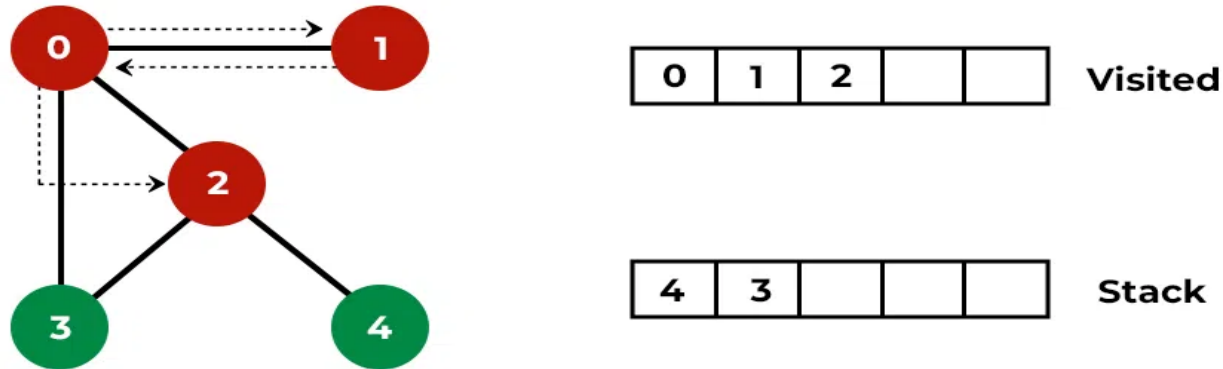
DFS on Graph

Step 3: Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



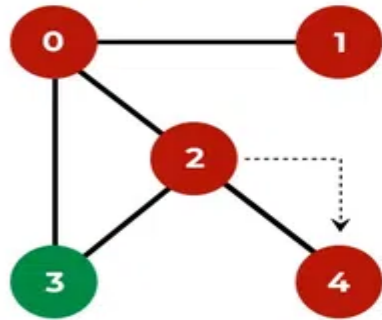
DFS on Graph

Step 4: Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited (i.e, 3, 4) in the stack.



DFS on Graph

Step 5: Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



0	1	2	4	
---	---	---	---	--

Visited

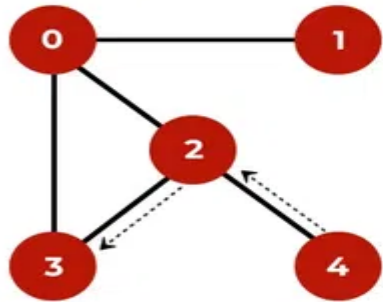
3				
---	--	--	--	--

Stack

DFS on Graph

Step 6: Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.

Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.



0	1	2	4	3
---	---	---	---	---

Visited

--	--	--	--	--

Stack

DFS on Graph

LOGIC

```
DFS(G, u)
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G, v)

init() {
    For each u ∈ G
        u.visited = false
    For each u ∈ G
        DFS(G, u)
}
```


PYTHON CODE

```
from collections import defaultdict

class Graph:
    def __init__(self, vertices):
        self.adjLists = defaultdict(list)
        self.visited = [False] * vertices
    def addEdge(self, src, dest):
        self.adjLists[src].append(dest)
    def DFS(self, vertex):
        self.visited[vertex] = True
        print(vertex, end=" ")
```

```
for adj in self.adjLists[vertex]:
    if not self.visited[adj]:
        self.DFS(adj)

if __name__ == "__main__":
    g = Graph(4)
    g.addEdge(0, 1)
    g.addEdge(0, 2)
    g.addEdge(1, 2)
    g.addEdge(2, 3)
    print("Following is Depth First Traversal")
    g.DFS(2)
```

JAVA CODE

```
import java.util.*;

class Graph {
    private LinkedList<Integer> adjLists[];
    private boolean visited[];
    // Graph creation
    Graph(int vertices) {
        adjLists = new LinkedList[vertices];
        visited = new boolean[vertices];
        for (int i = 0; i < vertices; i++)
            adjLists[i] = new
LinkedList<Integer>();
    }

    // Add edges
```

```
void addEdge(int src, int dest) {
    adjLists[src].add(dest);
} // DFS algorithm
void DFS(int vertex) {
    visited[vertex] = true;
    System.out.print(vertex + " ");
    Iterator<Integer> ite =
adjLists[vertex].listIterator();
    while (ite.hasNext()) {
        int adj = ite.next();
        if (!visited[adj])
            DFS(adj);
    } }
```

```
public static void main(String args[]) {  
    Graph g = new Graph(4);  
  
    g.addEdge(0, 1);  
    g.addEdge(0, 2);  
    g.addEdge(1, 2);  
    g.addEdge(2, 3);  
  
    System.out.println("Following is Depth First Traversal");  
  
    g.DFS(2);  
}  
}
```

PRIM'S ALGORITHM

Given a weighted graph, we have to find the minimum spanning tree (MST) of that graph using Prim's algorithm. Print the final weight of the MST.

Input Format

The first line contains one integer v representing the number of nodes.

Next lines contains a $v \times v$ matrix representing the graph. $\text{matrix}[i][j]$ represents the value of the weight between the i th node and the j th node. If there is no edge, the value is 0.

Output Format

Print the final weight of the MST.

Example 1

Input

```
5
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
```

Output

16

Explanation

Edge Weight

```
0 - 1      2
1 - 2      3
0 - 3      6
1 - 4      5
```

Total sum = 16

Example 2

Input

```
5
0 4 2 0 0
4 0 1 3 0
2 1 0 7 2
0 3 7 0 5
0 0 2 5 0
```

Output

8
Explanation

Edge	Weight
2 - 1	1
0 - 2	2
1 - 3	3
2 - 4	2

Total sum = 8

LOGIC

1. Initialize:

Create an array ``key[]`` to store the key values of vertices, initially set to ``INFINITY`` except for the first vertex, which is set to 0.

Create an array ``parent[]`` to store the parent (or the vertex that leads to the minimum key value) for each vertex.

Create a boolean array ``mstSet[]`` to represent whether a vertex is included in the MST or not.

Initialize all keys as ``INFINITY``, set the first key to 0, and set all elements in ``mstSet[]`` as ``false``.

2. Select Vertices:

Repeat the following until all vertices are included in the MST:

a. Choose the vertex ``u`` with the minimum key value from the set of vertices not yet included in the MST (``mstSet[]`` is ``false``).

b. Include ``u`` in the MST by setting ``mstSet[u]`` to ``true``.

c. Update the key values of all adjacent vertices of ``u`` if they are not included in the MST and the weight of the edge (``graph[u][v]``) is less than the current key value of the vertex ``v``.

3. Print MST:

Print the sum of the key values of all vertices in the MST. This sum represents the weight of the minimum spanning tree.

This logic using adjacency lists to represent the graph. It initializes key values, updates them during the algorithm's execution, and prints the final weight of the MST.

PYTHON CODE

```
import sys

def minKey(key, mstSet, V):
    min_val = sys.maxsize
    min_index = -1
    for v in range(V):
        if not mstSet[v] and key[v] < min_val:
            min_val = key[v]
            min_index = v
    return min_index

def printMST(parent, graph):
    V = len(graph)
    total_sum = 0
    for i in range(1, V):
        total_sum += graph[i][parent[i]]
```

```
print(total_sum)

def primMST(graph):
    V = len(graph)
    parent = [-1] * V
    key = [sys.maxsize] * V
    mstSet = [False] * V
    key[0] = 0
    parent[0] = -1
    for _ in range(V - 1):
        u = minKey(key, mstSet, V)
        mstSet[u] = True
        for v in range(V):
            if graph[u][v] != 0 and not
mstSet[v] and graph[u][v] < key[v]:
```

```
parent[v] = u  
key[v] = graph[u][v]  
  
printMST(parent, graph)  
  
# Input  
V = int(input())  
graph = [list(map(int, input().split())) for _ in range(V)]  
  
# Execute Prim's Algorithm  
primMST(graph)
```

JAVA CODE

```
import java.util.*;
class Solution{
    static int minKey(int key[], boolean
mstSet[], int V) {
    int min = Integer.MAX_VALUE, min_index
= -1;
    for (int v = 0; v < V; v++) {
        if (!mstSet[v] && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}
```

```
static void printMST(int parent[],
List<List<Integer>> graph) {
    int V = graph.size();
    int sum = 0;
    for (int i = 1; i < V; i++) {
        sum += graph.get(i).get(parent[i]);
    }
    System.out.println(sum);
}
static void primMST(List<List<Integer>>
graph) {
    int V = graph.size();
    int parent[] = new int[V];
    int key[] = new int[V];
```



```

boolean mstSet[] = new boolean[V];
    for (int i = 0; i < V; i++) {
        key[i] = Integer.MAX_VALUE;
        mstSet[i] = false;
    }
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V - 1;
count++) {
        int u = minKey(key, mstSet, V);
        mstSet[u] = true;
        for (int v = 0; v < V; v++) {
            if (graph.get(u).get(v) != 0 &&
!mstSet[v] && graph.get(u).get(v) < key[v]) {

```

```

parent[v] = u;
        key[v] = graph.get(u).
get(v);
        }
    }
    printMST(parent, graph);
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int V = sc.nextInt();
        List<List<Integer>> graph = new
ArrayList<>();

```

```
for (int i = 0; i < V; i++) {  
    List<Integer> temp = new ArrayList<>(V);  
    for (int j = 0; j < V; j++) {  
        temp.add(sc.nextInt());  
    }  
    graph.add(temp);  
}  
Solution.primMST(graph);  
sc.close();  
}  
}
```

MINIMUM FUEL COST TO REPORT TO THE CAPITAL

There is a tree (i.e., a connected, undirected graph with no cycles) structure country network consisting of n cities numbered from 0 to $n - 1$ and exactly $n - 1$ roads. The capital city is city 0 . You are given a 2D integer array `roads` where `roads[i] = [ai, bi]` denotes that there exists a bidirectional road connecting cities a_i and b_i .

There is a meeting for the representatives of each city. The meeting is in the capital city. There is a car in each city. You are given an integer `seats` that indicates the number of seats in each car.

A representative can use the car in their city to travel or change the car and ride with another representative. The cost of traveling between two cities is one liter of fuel.

Return the minimum number of liters of fuel to reach the capital city.

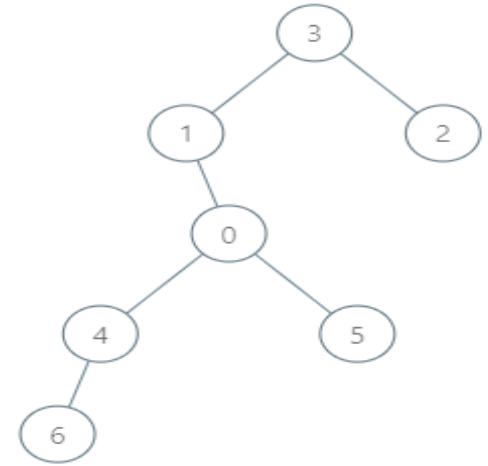
Example 1:

Input: roads = `[[3,1],[3,2],[1,0],[0,4],[0,5],[4,6]]`, seats = 2

Output: 7

Explanation:

- Representative2 goes directly to city 3 with 1 liter of fuel.
- Representative2 and representative3 go together to city 1 with 1 liter of fuel.
- Representative2 and representative3 go together to the capital with 1 liter of fuel.
- Representative1 goes directly to the capital with 1 liter of fuel.
- Representative5 goes directly to the capital with 1 liter of fuel.
- Representative6 goes directly to city 4 with 1 liter of fuel.
- Representative4 and representative6 go together to the capital with 1 liter of fuel.
- It costs 7 liters of fuel at minimum. It can be proven that 7 is the minimum number of liters of fuel needed.



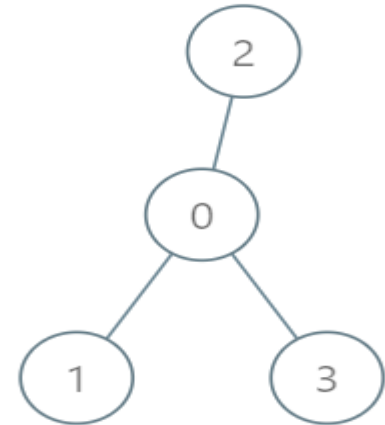
Example 2:

Input: roads = `[[0,1],[0,2],[0,3]]`, seats = 5

Output: 3

Explanation:

- Representative1 goes directly to the capital with 1 liter of fuel.
- Representative2 goes directly to the capital with 1 liter of fuel.
- Representative3 goes directly to the capital with 1 liter of fuel.
- It costs 3 liters of fuel at minimum. It can be proven that 3 is the minimum number of liters of fuel needed.



LOGIC

Initialization:

An ArrayList `adj` is created to represent an adjacency list for a graph. It's used to store the connections between different nodes (roads).

Graph Construction:

The `roads` array is used to construct an undirected graph (adjacency list). Each road connection is added to the `adj` list.

Recursive DFS (Depth-First Search):

The `solve` function is a recursive DFS function that explores the graph, calculating the number of people in each subtree.

The base case is when a leaf node is reached, i.e., a node with only one connection.

Fuel Cost Calculation:

For each node (except the root), the fuel cost is calculated based on the number of people in the subtree and the number of seats available. The cost is added to the global variable ans.

Main Function:

The minimumFuelCost function initializes the adj list, calls the solve function to calculate the fuel cost, and returns the final result.

Example Usage:

An example is provided where roads are defined, and the minimum fuel cost is calculated for a given number of seats.

PYTHON CODE

```
from math import ceil
class Solution:
    def __init__(self):
        self.ans = 0
    def minimum_fuel_cost(self, roads, seats):
        adj = [[] for _ in range(len(roads) + 1)]
        n = len(roads) + 1
        self.ans = 0
        for a, b in roads:
            adj[a].append(b)
            adj[b].append(a)
        self.solve(adj, seats, 0, -1)
        return self.ans
    def solve(self, adj, seats, src, parent):
        people = 1
```

```
        for i in adj[src]:
            if i != parent:
                people += self.solve(adj, seats, i, src)
            if src != 0:
                self.ans += ceil(people / seats)
        return people
# Example Usage
solution = Solution()
roads = [[3, 1], [3, 2], [1, 0], [0, 4], [0, 5],
[4, 6]]
seats = 2
result = solution.minimum_fuel_cost(roads,
seats)
print(result)
```

JAVA CODE

```
import java.util.ArrayList;

public class Solution {
    private long ans = 0L;

    public long minimumFuelCost(int[][] roads, int
seats) {
        ArrayList<ArrayList<Integer>> adj = new
ArrayList<>();
        int n = roads.length + 1;
        ans = 0L;
        for (int i = 0; i < n; i++) {
            adj.add(new ArrayList<>());
        }
        for (int[] a : roads) {
            adj.get(a[0]).add(a[1]);
            adj.get(a[1]).add(a[0]);
        }
    }
}
```

```
solve(adj, seats, 0, -1);
        return ans;
    }

    private long
solve(ArrayList<ArrayList<Integer>> adj, int seats,
int src, int parent) {
        long people = 1L;
        for (int i : adj.get(src)) {
            if (i != parent) {
                people += solve(adj, seats, i, src);
            }
        }
        if (src != 0) {
            ans += (long) Math.ceil((double) people
/ seats);
        }
    }
}
```

```
return people;
}
public static void main(String[] args) {
    Solution solution = new Solution();
    int[][] roads = {{3, 1}, {3, 2}, {1, 0}, {0, 4}, {0, 5}, {4, 6}};
    int seats = 2;
    long result = solution.minimumFuelCost(roads, seats);
    System.out.println(result);
}
}
```

NUMBER OF ISLANDS

You are given a 2D matrix grid of size $n * m$. You have to find the number of distinct islands where a group of connected 1s (horizontally or vertically) forms an island. Two islands are considered to be distinct if and only if one island is not equal to another (rotated or reflected islands are not equal).

Input Format

The first line contains two integers value of N and M .

Next N line contains M boolean values where 1 denotes land and 0 denotes water.

Output Format

Print total number of distinct islands.

Example 1

Input

```
3 4
1 1 0 0
0 0 0 1
1 1 1 0
```

Output

3

Explanation

There are only 3 distinct islands.

```
1 1 in row 1
1 in row 2
down right
1 1 1 in row 3
```


Example 2

Input

```
3 4
1 1 0 0
0 0 0 1
0 1 1 0
```

Output

2

Explanation

There are 3 islands once again, but island in row 1 and row 3 are not distinct, hence only 2 distinct islands.

LOGIC

Initialize Variables:

Define the directions to move (up, left, down, right).

Initialize a set to store the distinct islands' coordinates.

DFS Function:

Implement a DFS function that explores the connected land cells of an island.

Mark visited cells as -1 to indicate they have been processed.

Traverse the Grid:

Iterate through each cell in the grid.

If the cell is part of an unexplored island (grid value is 1), initiate DFS from that cell.

Coordinate Transformation:

Convert the island coordinates to a tuple and add it to the set.

Count Distinct Islands:

The size of the set represents the count of distinct islands.

PYTHON CODE

```
def count_distinct_islands(grid):
    def dfs(x0, y0, i, j, v):
        nonlocal grid
        rows, cols = len(grid), len(grid[0])
        if i < 0 or i >= rows or j < 0 or j >=
cols or grid[i][j] <= 0:
            return
        grid[i][j] *= -1
        v.append((i - x0, j - y0))
        for k in range(4):
            dfs(x0, y0, i + dirs[k][0], j +
dirs[k][1], v)
        rows, cols = len(grid), len(grid[0])
        coordinates = set()
        for i in range(rows):
```

```
for j in range(cols):
            if grid[i][j] != 1:
                continue
            v = []
            dfs(i, j, i, j, v)
            coordinates.add(tuple(v))
        return len(coordinates)

# Input handling
n, m = map(int, input().split())
grid = [list(map(int, input().split())) for _
in range(n)]
dirs = [[0, -1], [-1, 0], [0, 1], [1, 0]]
# Call the function to count distinct islands
ans = count_distinct_islands(grid)
print(ans)
```

JAVA CODE

```
import java.util.*;
class Solution {
    static int[][] dirs = {{0, -1}, {-1, 0}, {0, 1},
    {1, 0}};
    private static String toString(int r, int c) {
        return Integer.toString(r) + " " + Integer.
toString(c);
    }
    private static void dfs(int[][] grid, int x0,
int y0, int i, int j, ArrayList<String> v) {
        int rows = grid.length, cols = grid[0].
length;
        if (i < 0 || i >= rows || j < 0 || j >=
cols || grid[i][j] <= 0)
            return;
        grid[i][j] *= -1;
        v.add(toString(i - x0, j - y0));
```

```
for (int k = 0; k < 4; k++) {
        dfs(grid, x0, y0, i + dirs[k][0], j +
dirs[k][1], v);
    }
}
public static int countDistinctIslands(int[][]
grid) {
    int rows = grid.length;
    if (rows == 0)
        return 0;
    int cols = grid[0].length;
    if (cols == 0)
        return 0;
    HashSet<ArrayList<String>> coordinates =
new HashSet<>();
    for (int i = 0; i < rows; ++i) {
```

```
for (int j = 0; j < cols; ++j) {
    if (grid[i][j] != 1)
        continue;
    ArrayList<String> v = new
ArrayList<>();
    dfs(grid, i, j, i, j, v);
    coordinates.add(v);
}
}
return coordinates.size();
}
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
int n = sc.nextInt();
int m = sc.nextInt();
int[][] grid = new int[n][m];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        grid[i][j] = sc.nextInt();
    }
}
Solution ob = new Solution();
int ans = ob.countDistinctIslands(grid);
System.out.println(ans);
}
}
```

COURSE SCHEDULE

You are given a number N , the number of courses you have to take labeled from 0 to $N-1$. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you must take course bi first if you want to take course ai .

eg: `[2,4]` means take course 4 before you can take course 2.

Input Format

The First line of input contain two integers N denoting number of people and M denoting size of `prerequisites` array.

Next line contains two integer each denoting ai and bi .

Output Format

print 1 if it is possible to finish all the courses else print 0.

Example 1

Input

4 3

1 2

1 3

1 0

Output

1

Explanation

We need to take course 2,3 and 0 before taking course 1. Since no conflict is there, we can take it.

Example 2

Input

4 3

1 2

2 3

3 1

Output

0

Now, let's analyze the prerequisites:

Course 2 (index 1) must be taken before course 1 (index 0).

Course 3 (index 2) must be taken before course 2 (index 1).

Course 1 (index 0) must be taken before course 3 (index 2).

If we visualize this as a graph, it forms a cycle: 1 -> 2 -> 3 -> 1. This cycle indicates a circular dependency, and it is not possible to finish all the courses without violating the prerequisites. In this case, the output should be 0.

LOGIC

1. Build a graph and calculate in-degrees for each course.
2. Initialize a set with courses having no prerequisites.
3. Perform BFS by removing courses with no prerequisites, updating in-degrees, and adding new courses with no prerequisites.
4. If all courses are taken (sum of in-degrees is 0), return 1; otherwise, return 0.
5. The result indicates whether it is possible to finish all courses based on the given prerequisites.

PYTHON CODE

```
from collections import defaultdict
class Solution:
    def canFinish(self, n, prerequisites):
        G = defaultdict(list)
        degree = [0] * n
        for e in prerequisites:
            G[e[1]].append(e[0])
            degree[e[0]] += 1
        no_prerequisites = set(i for i in
range(n) if degree[i] == 0)
        while no_prerequisites:
            course = no_prerequisites.pop()
            for neighbor in G[course]:
                degree[neighbor] -= 1
```

```
if degree[neighbor] == 0:
            no_prerequisites.add(neighbor)
            return int(sum(degree) == 0)
# Input handling
N, M = map(int, input().split())
prerequisites = [list(map(int, input().
split())) for _ in range(M)]
# Call the solution class
Obj = Solution()
print(Obj.canFinish(N, prerequisites))
```

JAVA CODE

```
import java.util.*;
class Solution {
public int canFinish(int n, int[][] prerequisites) {
    ArrayList<Integer>[] G = new ArrayList[n];
    int[] degree = new int[n];
    ArrayList<Integer> bfs = new ArrayList();
    for (int i = 0; i < n; ++i) G[i] = new ArrayList<Integer>();
    for (int[] e : prerequisites) {
        G[e[1]].add(e[0]);
        degree[e[0]]++;
    }
    for (int i = 0; i < n; ++i) if (degree[i] == 0) bfs.add(i);
    for (int i = 0; i < bfs.size(); ++i)
        for (int j: G[bfs.get(i)])
            if (--degree[j] == 0) bfs.add(j);
    if(bfs.size() == n)
```



```
        return 1;
    }
    else
    {
        return 0;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N= sc.nextInt();
        int M= sc.nextInt();
        int prerequisites[][] = new int[M][2];
        for(int i=0; i<M; i++){
            for(int j=0; j<2; j++){
                prerequisites[i][j]= sc.nextInt();
            }
        }
        Solution Obj = new Solution();
        System.out.println(Obj.canFinish(N,prerequisites));
    }
}
```

LETTER COMBINATIONS OF A PHONE NUMBER

Given a string containing digits from 2-9 inclusive, print all possible letter combinations that the number could represent. Print the answer in sorted order. A mapping of digit to letters (just like on the telephone buttons) is given below.

Note

1 does not map to any letters.

2 : abc

3 : def

4 : ghi

5 : jkl

6 : mno

7 : pqrs

8 : tuv

9 : wxyz

Input Format

The first line of input contains a string of digits.

Output Format

Print all possible letter combinations that the number could represent, separated by spaces.

Print the answer in sorted order.

Test cases:

Example 1

Input

23

Output

ad ae af bd be bf cd ce cf

Explanation

2 maps to any of a,b,c whereas
3 maps to any of d,e,f. Hence
9 possible combinations.

Example 2

Input

2

Output

a b c

Explanation

2 maps to a, b, c.

LOGIC

Base Case:

If the input string `s` is empty, print the current combination (`ans`).
This is the stopping condition for the recursion.

Recursive Step:

Get the mapping (key) of the first digit in the input string `s`.

For each character in the mapping:

Recursively call the function with the remaining digits (`s[1:]`) and the updated combination (`ans + char`).

The recursion will continue until the base case is reached.

Mapping (keypad) Explanation:

The keypad array is used to map each digit to the corresponding letters on a telephone keypad.

For example, `keypad[2]` corresponds to "abc," `keypad[3]` corresponds to "def," and so on.

PYTHON CODE

```
def possible_words(s, ans):
    # Base Case
    if not s:
        print(ans)
        return

    # Get the mapping of the first digit
    key = keypad[int(s[0])]

    # Recursive Step
    for char in key:
        # Recursively call the function with the remaining digits and updated combination
        possible_words(s[1:], ans + char)

# Corrected initialization of keypad
keypad = ["", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"]

# Input handling
s = input("Enter a string of digits: ")

# Call the function to print possible letter combinations
print("Possible letter combinations:")
possible_words(s, "")
```


JAVA CODE

```
import java.util.Scanner;
```

```
public class Main{
```

```
    static String[] keypad={"", "", "abc", "def",  
    "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
```

```
static void possibleWords(String s,String ans){
```

```
    if(s.length()==0){
```

```
        System.out.println(ans);
```

```
        return;
```

```
    }
```

```
    String key= keypad[s.charAt(0)-48];
```

```
    for(int i=0;i<key.length();i++){
```

```
        possibleWords(s.substring(1),ans+key.
```

```
charAt(i));
```

```
    }
```

```
}
```

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

```
    Scanner scan=new Scanner(System.in);
```

```
    String s=scan.next();
```

```
    possibleWords(s,"");
```

```
}
```

```
}
```

PERMUTATIONS

Problem Statement: Generating Permutations using Backtracking

Given a set of distinct integers, write a program to generate all possible permutations of the elements in the set.

Example:

Suppose the input set is $\{1, 2, 3\}$.

The program should output the following permutations:

[1, 2, 3]

[1, 3, 2]

[2, 1, 3]

[2, 3, 1]

[3, 1, 2]

[3, 2, 1]

LOGIC

Base Case:

- If the left index is equal to the right index, print the current permutation.

Recursion:

- Iterate through each element from the left index to the right index.
- Swap the current element with the element at index 'i'.
- Recursively generate permutations for the remaining elements.
- Backtrack by undoing the swap to restore the original order.

PYTHON CODE

```
def generate_permutations(nums, left, right):
    if left == right:
        # Base case: Print the current permutation
        print(nums)
    else:
        for i in range(left, right + 1):
            # Swap the current element with the
            element at index 'i'
            nums[left], nums[i] = nums[i],
            nums[left]

            # Recursively generate permutations for
            the remaining elements
            generate_permutations(nums, left + 1,
            right)
```

```
# Backtrack: Undo the swap to restore the original
order
            nums[left], nums[i] = nums[i],
            nums[left]

# Input handling
n = int(input("Enter the number of elements in the
set: "))
nums = list(map(int, input("Enter the elements of
the set: ").split()))

# Call the function to generate permutations
print("Permutations of the set:")
generate_permutations(nums, 0, n - 1)
```


JAVA CODE

```
import java.util.Scanner;

public class PermutationsBacktracking {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.
in);

        // Get input from the user
        System.out.print("Enter the number of
elements in the set: ");

        int n = scanner.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements
of the set:");

        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }
    }
}
```

```
System.out.println("Permutations of the set:");
        generatePermutations(nums, 0, n - 1);
        scanner.close();
    }

    private static void
generatePermutations(int[] nums, int left, int
right) {
        if (left == right) {
            // Base case: Print the current
permutation

            printArray(nums);
        } else {
            for (int i = left; i <= right; i++)
        {
            }
        }
    }
}
```

```
// Swap the current element with the element at  
index 'i'
```

```
    swap(nums, left, i);
```

```
    // Recursively generate  
    permutations for the remaining elements  
    generatePermutations(nums, left  
+ 1, right);
```

```
    // Backtrack: Undo the swap to  
    restore the original order
```

```
    swap(nums, left, i);
```

```
    }
```

```
}}
```

```
private static void swap(int[] nums, int i, int  
j) {
```

```
    int temp = nums[i];
```

```
    nums[i] = nums[j];
```

```
    nums[j] = temp;
```

```
}
```

```
private static void printArray(int[] nums) {
```

```
    for (int num : nums) {
```

```
        System.out.print(num + " ");
```

```
    }
```

```
    System.out.println();
```

```
}
```

```
}
```

COMBINATION SUM

Given an array of distinct integers `nums` and a target integer `target`, return a list of all unique combinations of `nums` where the chosen numbers sum to `target`. You may return the combinations in any order.

The same number may be chosen from `nums` an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

Input Format

Input is managed for you. (You are given an array `nums` and target `target` in the `combinationSum()` function).

Output Format

Output is managed for you. (You can return the possible valid combinations in any order. The combinations will be automatically printed in sorted order).

Example 1

Input

4 16

6 2 7 5

Output

2 2 2 2 2 2 2 2

2 2 2 2 2 6

2 2 2 5 5

2 2 5 7

2 2 6 6

2 7 7

5 5 6

Explanation

Here all these combinations have sum equal to 16.

(2 2 2 2 2 2 2 2)

(2 2 2 2 2 6)

(2 2 2 5 5)

(2 2 5 7)

(2 2 6 6)

(2 7 7)

(5 5 6)

Example 2

Input

3 5

1 2 3

Output

1 1 1 1 1

1 1 1 2

1 1 3

1 2 2

2 3

Explanation

Here all these combinations have sum equal to 5.

(1 1 1 1 1)

(1 1 1 2)

(1 1 3)

(1 2 2)

(2 3)

LOGIC

- Sort the array nums to handle duplicates and for easier comparison later.
- Use a backtracking function to explore all possible combinations, keeping track of the current combination in the tempList.
- If the current combination sums up to the target, add it to the result list.
- Recursively call the backtracking function for each element in the array, allowing duplicates to be reused.
- Sort the result list and its sublists for proper ordering, and print the unique combinations.

PYTHON CODE

```
class Solution:
    def combinationSum(self, nums, target):
        nums.sort()
        result = []
        self.backtrack(result, [], nums,
target, 0)
        return result

    def backtrack(self, result, tempList, nums,
remain, start):
        if remain < 0:
            return
        elif remain == 0:
            result.append(tempList.copy())
        else:
            for i in range(start, len(nums)):
                tempList.append(nums[i])
                self.backtrack(result,
tempList, nums, remain - nums[i], i)
                tempList.pop()
```

```
# Input handling
n, target = map(int, input().split())
nums = list(map(int, input().split()))

# Call the solution class
ob = Solution()
ans = ob.combinationSum(nums, target)

# Sort the result
ans.sort(key=lambda x: (len(x), x))

# Print the result
for combination in ans:
    print(*combination)
```

JAVA CODE

```
import java.util.*;
class Solution {

    public List<List<Integer>>
    combinationSum(int[] nums, int target) {
        List<List<Integer>> list = new
        ArrayList<>();
        Arrays.sort(nums);
        backtrack(list, new ArrayList<>(), nums,
        target, 0);
        return list;
    }

    private void backtrack(List<List<Integer>>
    list, List<Integer> tempList, int [] nums, int
    remain, int start){
        if(remain < 0) return;
        else if(remain == 0){
            list.add(new ArrayList<>(tempList));
        }
        else{
            for(int i = start; i < nums.length;
            i++){
```

```
        tempList.add(nums[i]);
        backtrack(list, tempList, nums, remain -
        nums[i], i); // not i + 1 because we can reuse
        same elements
        tempList.remove(tempList.size() -
        1);
        }
    }
}
}

public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int target = sc.nextInt();
        int []nums = new int[n];
        for(int i = 0 ; i < n ; ++i){
            nums[i] = sc.nextInt();
        }
    }
}
```

```
Solution ob = new Solution();
List<List<Integer>> ans = ob.combinationSum(nums,target);
for(int i = 0 ; i < ans.size() ; ++i){
    Collections.sort(ans.get(i));
}
Collections.sort(ans, (o1, o2) -> {
int m = Math.min(o1.size(), o2.size());
    for (int i = 0; i < m; i++) {
        if (o1.get(i) == o2.get(i)){
            continue;
        }else{
            return o1.get(i) - o2.get(i);
        }
    }
    return 1;
});
for (int i = 0; i < ans.size (); i++)
{
    for (int j = 0; j < ans.get(i).size (); j++)
    {
        System.out.print(ans.get(i).get(j)+" ");
    }
    System.out.println();
}}
```


GENERATE PARENTHESES

Given a positive integer n , write a function to generate all combinations of well-formed parentheses. The goal is to generate all possible combinations of parentheses such that they are balanced.

A well-formed parentheses string is defined as follows:

The empty string is well-formed.

If "X" is a well-formed parentheses string, then "(X)" is also well-formed.

If "X" and "Y" are well-formed parentheses strings, then "XY" is also well-formed.

Example

`n = 3`, the function should return the following combinations:

```
[ "((()))", "(()())", "()(())", "())()", "())()"]
```

Your task is to implement the solution in Java using backtracking and take the input `n` from the user.

LOGIC

1. Start with an empty string.
2. If the count of open parentheses is less than n , add an open parenthesis and recursively call the function.
3. If the count of close parentheses is less than the count of open parentheses, add a close parenthesis and recursively call the function.
4. If the length of the current string is equal to $2 * n$, add it to the result.
5. Repeat these steps recursively, exploring all possible combinations.

PYTHON CODE

```
def generate_parenthesis(n):
    result = []
    def generate_parenthesis_helper(open_count,
close_count, current):
        nonlocal result
        if len(current) == 2 * n:
            result.append(current)
            return
        if open_count < n:
            generate_parenthesis_helper(open_count +
1, close_count, current + "(")
        if close_count < open_count:
            generate_parenthesis_helper(open_count,
close_count + 1, current + ")")
```

```
generate_parenthesis_helper(0, 0, "")
    return result
# Get input from the user
n = int(input("Enter the value of n: "))
combinations = generate_parenthesis(n)

print(f"Combinations of well-formed parentheses
for n = {n}:")
for combination in combinations:
    print(combination)
```

JAVA CODE

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class GenerateParentheses {

    public static List<String>
generateParenthesis(int n) {
    List<String> result = new ArrayList<>();
    generateParenthesisHelper(n, 0, 0, "",
result);
    return result;
}

private static void
generateParenthesisHelper(int n, int openCount,
int closeCount, String current, List<String>
result) {
```

```
    if (current.length() == 2 * n) {
        result.add(current);
        return;
    }
    if (openCount < n) {
        generateParenthesisHelper(n,
openCount + 1, closeCount, current + "(",
result);
    }
    if (closeCount < openCount) {
        generateParenthesisHelper(n,
openCount, closeCount + 1, current + ")",
result);
    }
}
```



```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    System.out.print("Enter the value of n: ");  
    int n = scanner.nextInt();  
    List<String> combinations = generateParenthesis(n);  
    System.out.println("Combinations of well-formed parentheses for n = " + n + ":");  
    for (String combination : combinations) {  
        System.out.println(combination);  
    }  
}
```

LONGEST HAPPY PREFIX

A string is called a happy prefix if is a non-empty prefix which is also a suffix (excluding itself).

Given a string s . Return the longest happy prefix of s .

Return an empty string if no such prefix exists.

Example 1:

Input: `s = "level"`

Output: `"l"`

Explanation: `s` contains 4 prefix excluding itself (`"l"`, `"le"`, `"lev"`, `"leve"`), and suffix (`"l"`, `"el"`, `"vel"`, `"evel"`). The largest prefix which is also suffix is given by `"l"`.

Example 2:

Input: $s = \text{"ababab"}$

Output: "abab"

Explanation: "abab" is the largest prefix which is also suffix. They can overlap in the original string.

LOGIC

1. Prefix and Suffix Matching:

- Iterate through the string from left to right.
- Keep track of the length of the matching prefix and suffix.
- Whenever a matching character is found, increment the length.

2. Check for Happy Prefix:

- If at any point the current matching prefix length is equal to the length of the string minus one, it means we have a happy prefix.
- The reason is that a happy prefix cannot include the entire string, so the length of the matching prefix should be less than the length of the string.

PYTHON CODE

```
def longest_happy_prefix(s):  
    n = len(s)  
    # Compute the prefix function using KMP  
    algorithm  
    prefix_function = [0] * n  
    j = 0  
    for i in range(1, n):  
        while j > 0 and s[i] != s[j]:  
            j = prefix_function[j - 1]  
        if s[i] == s[j]:  
            j += 1  
        prefix_function[i] = j  
    # The length of the longest happy prefix is  
    given by the last value in the prefix function  
    length = prefix_function[-1]
```

```
# Return the longest happy prefix  
    return s[:length]  
  
# Get input from the user  
input_str = input("Enter a string: ")  
  
# Find and print the longest happy prefix  
result = longest_happy_prefix(input_str)  
print("Longest Happy Prefix:", result)
```


JAVA CODE

```
public class LongestHappyPrefix {  
    public static String  
    longestHappyPrefix(String s) {  
        int n = s.length();  
        int[] lps = new int[n];  
        int len = 0;  
        for (int i = 1; i < n; ) {  
            if (s.charAt(i) == s.charAt(len)) {  
                lps[i++] = ++len;  
            } else {  
                if (len != 0) {  
                    len = lps[len - 1];  
                } else {  
                    lps[i++] = 0;  
                }  
            }  
        }  
    }  
}
```

```
}  
  
}  
  
int happyPrefixLength = lps[n - 1];  
return happyPrefixLength > 0 ? s.  
substring(0, happyPrefixLength) : "";  
}  
  
public static void main(String[] args) {  
    System.out.  
println(longestHappyPrefix("level"));  
}  
}
```

LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS

You are given a string `s`. Your task is to find the length of the longest substring that contains each character at most once.

A substring is a contiguous sequence of characters within a string.

Input Format

First line contains the string `s`.

Output Format

Complete the function `longestSubstring()` where you return the required integer.

Example 1

Input

xyzxyzyy

Output

3

Explanation

The answer is "xyz ", with the length of 3

Example 2

Input

xxxxxx

Output

1

LOGIC

- ✓ We use a sliding window approach to find the longest substring without repeating characters.
- ✓ Maintain two pointers, start and end, representing the current substring.
- ✓ Use a dictionary (`char_index_map`) to keep track of the last index of each character encountered.
- ✓ If a character is already in the current substring, update the start index to the next index of the previous occurrence of that character.
- ✓ Update the last index of the current character in the `char_index_map`.
- ✓ Update the length of the current substring and keep track of the maximum length encountered.

PYTHON CODE


```
def longestSubstring(s):  
    char_index_map = {} # To store the last  
    index of each character  
    start = 0 # Start index of the current  
    substring  
    max_length = 0 # Length of the longest  
    substring  
    for end in range(len(s)):  
        if s[end] in char_index_map and  
char_index_map[s[end]] >= start:  
            # If the character is already in  
the current substring, update the start index  
            start = char_index_map[s[end]] + 1  
        # Update the last index of the current  
character
```

```
char_index_map[s[end]] = end  
        # Update the length of the current  
substring  
        max_length = max(max_length, end -  
start + 1)  
    return max_length  
input_str2 = "xxxxxx"  
result2 = longestSubstring(input_str2)  
print("Output:", result2) # Output: 1
```

JAVA CODE

```
import java.util.*;

class Solution {
    public int longestSubstring(String s) {
        Map<Character, Integer> chars = new
HashMap();
        int left = 0;
        int right = 0;
        int res = 0;
        while (right < s.length()) {
            char r = s.charAt(right);
            chars.put(r, chars.getOrDefault(r,0) + 1);
            while (chars.get(r) > 1) {
                char l = s.charAt(left);
                chars.put(l, chars.get(l) - 1);
                left++;
            }
        }
```

```
res = Math.max(res, right - left + 1);
            right++;
        }
        return res;
    }}

public class Main {
    public static void main (String[] args)
throws java.lang.Exception {
        Scanner sc=new Scanner(System.in);
        String s = sc.nextLine();
        Solution ob = new Solution();
        int ans=ob.longestSubstring(s);
        System.out.println(ans);
    }
}
```

LONGEST PALINDROMIC SUBSTRING

Given a string s , find the longest palindromic substring in s .

Example:

Input: "babad"

Output: "bab"

Note: "aba" is also a valid answer.

Input: "cbbd"

Output: "bb"

“Madam”



Palindrome string

Palindrome string remain the same whether
written forwards or backwards

‘d’



‘a d a’



‘m a d a m’



PYTHON CODE

```
def longest_palindromic_substring(s):  
    def expand_around_center(left, right):  
        while left >= 0 and right < len(s) and  
s[left] == s[right]:  
            left -= 1  
            right += 1  
        return left + 1, right - 1  
    start, end = 0, 0  
    for i in range(len(s)):  
        left1, right1 = expand_around_center(i,  
i) # Odd-length palindrome  
        left2, right2 = expand_around_center(i,  
i + 1) # Even-length palindrome
```

```
        if right1 - left1 > end - start:  
            start, end = left1, right1  
        if right2 - left2 > end - start:  
            start, end = left2, right2  
  
    return s[start:end + 1]  
# Get input from the user  
input_str = input("Enter a string: ")  
# Find and print the longest palindromic  
substring  
result =  
longest_palindromic_substring(input_str)  
print("Longest Palindromic Substring:", result)
```


JAVA CODE

```
import java.util.Scanner;

class LongestPalindromicSubstring {
    private static String
findLongestPalindromicSubstring(String input) {
    if(input.isEmpty()) {
        return "";
    }
    int n = input.length();
    int longestSoFar = 0, startIndex = 0,
endIndex = 0;
    boolean[][] palindrom = new boolean[n][n];
    for(int j = 0; j < n; j++) {
        palindrom[j][j] = true;
        for(int i = 0; i < j; i++) {
            if(input.charAt(i) == input.charAt(j)
&& (j-i <= 2 || palindrom[i+1][j-1])) {
```

```
palindrom[i][j] = true;
            if(j-i+1 > longestSoFar) {
                longestSoFar = j-i+1;
                startIndex = i;
                endIndex = j;
            } } } }

        return input.substring(startIndex,
endIndex+1);
    }

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        String input = keyboard.next();    System.
out.
println(findLongestPalindromicSubstring(input));
    }
}
```

SHORTEST PALINDROME

Problem Statement:

Given a string, find the shortest palindrome that can be obtained by adding characters in front of it.

Example 1:

Input: "race"

Output: "ecarace"

Explanation: By adding "eca" in front of "race," we get the shortest palindrome "ecarace."

Example 2:

Input: "abc"

Output: "cba"

Explanation: By adding "cba" in front of "abc," we get the shortest palindrome "cbaabc."

Example 3:

Input: "level"

Output: "level"

Explanation: The given string "level" is already a palindrome, so no additional characters are needed.

Input: "abc" Output: "cbaabc"

LOGIC

- Iterate from the end of the string, considering each prefix.

Start from the end of the string "abc."

Consider each prefix, trying to find the longest palindrome.

Consider the prefix "cba" from "abc."

- The prefix "cba" is a palindrome.

Reverse the remaining suffix "abc" and append it to the original string.

- Reverse "abc" to get "cba."

Append the reversed suffix to the original string.

Result: "cbaabc"

So, by adding the palindrome "cba" in front of the original string "abc" and then appending the reversed suffix "abc," we get the shortest palindrome "cbaabc."

PYTHON CODE

```
def shortest_palindrome(s):  
    def is_palindrome(string):  
        return string == string[::-1]  
    for i in range(len(s), 0, -1):  
        prefix = s[:i]  
        if is_palindrome(prefix):  
            suffix = s[i:]  
            return suffix[::-1] + s  
    return s  
  
# Get input from the user  
input_str = input("Enter a string: ")  
  
# Find and print the shortest palindrome  
result = shortest_palindrome(input_str)  
print("Shortest Palindrome:", result)
```


JAVA CODE

```
import java.util.Scanner;

public class Main {

    public static String shortestPalindrome(String s) {
        if (s == null || s.isEmpty()) {
            return "";
        }
        int i = 0;
        for (int j = s.length() - 1; j >= 0; j--) {
            if (s.charAt(i) == s.charAt(j)) {
                i++;
            }
        }
        if (i == s.length()) {
            return s;
        }
        String suffix = s.substring(i);
        String prefix = new StringBuilder(suffix).reverse().toString();
```

```
String middle = shortestPalindrome(s.substring(0, i));

    return prefix + middle + suffix;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String input = scanner.nextLine();
    String result = shortestPalindrome(input);
    System.out.println("Shortest Palindrome: " + result);
    scanner.close();
}
}
```

MINIMUM DELETIONS TO MAKE ARRAYS DIVISIBLE

You are given two positive integer arrays `nums` and `numsDivide`. You can delete any number of elements from `nums`.

Return the minimum number of deletions such that the smallest element in `nums` divides all the elements of `numsDivide`. If this is not possible, return `-1`.

Note that an integer `x` divides `y` if `y % x == 0`.

Example 1:

Input: `nums = [2,3,2,4,3]`, `numsDivide = [9,6,9,3,15]`

Output: 2

Explanation:

The smallest element in `[2,3,2,4,3]` is 2, which does not divide all the elements of `numsDivide`.

We use 2 deletions to delete the elements in `nums` that are equal to 2 which makes `nums = [3,4,3]`.

The smallest element in `[3,4,3]` is 3, which divides all the elements of `numsDivide`.

It can be shown that 2 is the minimum number of deletions needed.

Example 2:

Input: `nums = [4,3,6]`, `numsDivide = [8,2,6,10]`

Output: -1

Explanation:

We want the smallest element in `nums` to divide all the elements of `numsDivide`.

There is no way to delete elements from `nums` to allow this.

LOGIC

1. Find the greatest common divisor (gcd) of elements in `numsDivide`.
2. Identify the smallest element in `nums` that is a divisor of the gcd.
3. If such an element is found, count the number of elements in `nums` that are different from this smallest divisor.
4. Return the count obtained in step 3 as the minimum number of deletions.
5. This logic ensures that the smallest divisor is selected to minimize the deletions needed for the array to satisfy the given conditions.

PYTHON CODE

```
def min_operations(nums, nums_divide):
    def gcd(a, b):
        while b > 0:
            a, b = b, a % b
        return a
    g = nums_divide[0]
    for i in nums_divide:
        g = gcd(g, i)
    smallest = float('inf')
    for num in nums:
        if g % num == 0:
            smallest = min(smallest, num)
    if smallest == float('inf'):
        return -1 # No element in nums can divide all elements in nums_divide
    min_op = sum(1 for num in nums if num == smallest)
    return min_op
nums1 = [2, 3, 2, 4, 3]
nums_divide1 = [9, 6, 9, 3, 15]
print(min_operations(nums1, nums_divide1)) # Output: 2
```

JAVA CODE

```
public class Main {  
    public static int minOperations(int[] nums,  
int[] numsDivide) {  
        int g = numsDivide[0];  
        for (int i : numsDivide) {  
            g = gcd(g, i);  
        }  
        int smallest = Integer.MAX_VALUE;  
        for (int num : nums) {  
            if (g % num == 0) {  
                smallest = Math.min(smallest, num);  
            }  
        }  
        if (smallest == Integer.MAX_VALUE) {  
            return -1; // No element in nums can  
divide all elements in numsDivide  
        }  
    }  
}
```

```
int minOp = 0;  
    for (int num : nums) {  
        if (num > smallest) {  
            ++minOp;  
        }  
    }  
    return minOp;  
}  
private static int gcd(int a, int b) {  
    while (b > 0) {  
        int tmp = a;  
        a = b;  
        b = tmp % b;  
    }  
    return a;  
}
```

```
public static void main(String[] args) {  
    // Example 1  
    int[] nums1 = {2, 3, 2, 4, 3};  
    int[] numsDivide1 = {9, 6, 9, 3, 15};  
    System.out.println(minOperations(nums1, numsDivide1)); // Output: 2  
}  
}
```

FIND IN MOUNTAIN ARRAY

An array is said to be a mountain array if it satisfies the following conditions:

The length of the given array is should be greater or equal to 3 i.e. $LENGTH \geq 3$.

There must be only one peak in the array or the largest element in the array.

The array must follows the condition: $ARRAY[0] < ARRAY[1] < \dots < ARRAY[i-1] < ARRAY[i] > ARRAY[i+1] > \dots > ARRAY[length-1]$

The task is to find the peak index of the mountain array.

Suppose we have given the input $[60, 20, 90, 110, 10]$.

The output will be 3. Because the largest element in the array is 110 whose index is 3.

LOGIC

```
def find_peak_index(arr):  
    left, right = 0, len(arr) - 1  
  
    while left < right:  
        mid = (left + right) // 2  
  
        if arr[mid] > arr[mid + 1]:  
            right = mid  
        else:  
            left = mid + 1  
  
    return left
```


PYTHON CODE

```
def find_peak_index(arr):
    left, right = 0, len(arr) - 1
    while left < right:
        mid = left + (right - left) // 2
        if arr[mid] < arr[mid + 1]:
            left = mid + 1
        else:
            right = mid
    # At the end, left and right will be equal
    return left

def main():
    n = int(input("Enter the length of the array: "))
    arr = list(map(int, input("Enter the elements of the array separated by spaces: ").
split()))
    peak_index = find_peak_index(arr)
    print("The peak index is:", peak_index)

if __name__ == "__main__":
    main()
```

JAVA CODE

```
import java.util.Scanner;

public class Main {
    public static int findPeakIndex(int[] arr) {
        int left = 0;
        int right = arr.length - 1;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] < arr[mid + 1]) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }
        return left; // or right, they are equal at
the end
    }
    public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the length
of the array: ");
    int n = 0;
    while (!scanner.hasNextInt()) {
        System.out.println("Invalid input.
Please enter a valid integer.");
        scanner.next(); // consume the invalid
input
    }
    n = scanner.nextInt();
    System.out.print("Enter the elements of the
array separated by spaces: ");
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        while (!scanner.hasNextInt()) {
```

```
System.out.println("Invalid input. Please enter a valid integer.");
    scanner.next(); // consume the invalid input
}
arr[i] = scanner.nextInt();
}
int peakIndex = findPeakIndex(arr);
System.out.println("The peak index is: " + peakIndex);
scanner.close();
}
}
```

NUMBER OF SUBSTRING CONTAINS ALL THREE CHARACTERS

Print the number of substrings containing all three characters i.e. a,b,c at least once in a given string.

Test Case

Input:

1

acbaa

Output:

5

Explanation:

The substrings containing at least one occurrence of the characters a, b and c are acb, acba, acbaa, cba and cbaa.

LOGIC

1. initialize pointers start and end to define a substring.
2. Iterate through the string using these pointers.
3. Use an array hash_count to count occurrences of each character in the substring.
4. Check if the substring contains at least one occurrence of each of 'a', 'b', and 'c'.
5. If yes, update the count with the number of substrings that can be formed with the remaining characters.
6. Move the pointers accordingly.
7. Print the count for each test case.

PYTHON CODE

```
def countSubstringsWithABC():
    test_cases = int(input("Enter the number of test cases: "))
    while test_cases > 0:
        S = input("Enter the string: ")
        start, end, count = 0, 0, 0
        if len(S) < 3:
            count = 0
        else:
            while end < len(S):
                hash_count = [0] * 26
                for i in range(start, end):
                    hash_count[ord(S[i]) - ord('a')] += 1
                if all(count > 0 for count in hash_count[:3]):
                    count += len(S) - end + 1
                    start += 1
                else:
                    end += 1
            print(count)
            test_cases -= 1
# Example usage:
countSubstringsWithABC()
```

JAVA CODE

```
import java.util.Scanner;

public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int test_cases = 0;
        while (true) {
            try {
                System.out.println("Enter the number of
test cases:");
                test_cases = sc.nextInt();
                break;
            } catch (java.util.InputMismatchException e){
                System.out.println("Invalid
input. Please enter a valid integer.");
                sc.nextLine();
            } }
}
```

```
while (test_cases != 0) {
    System.out.println("Enter the string:
");

    String S = sc.next();
    int start = 0, end = 0, count = 0;
    if (S.length() < 3)
        count = 0;
    else {
        while (end < S.length()) {
            int hash[] = new int[26];
            for (int i = start; i < end; i++) {
                hash[S.charAt(i) - 'a']++;
            }

            if (hash[0] > 0 && hash[1] > 0 && hash[2] >
0) {
```

```
count += S.length() - end + 1;
        start++;
    } else {
        end++;
    }
}
}

System.out.println(count);
test_cases--;
}
}
}
```

TRAPPING RAIN WATER

Trapping Rain Water

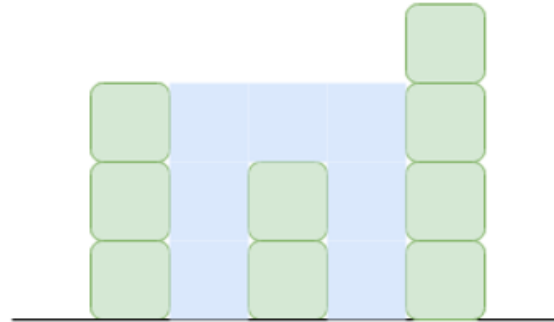
Given with n non-negative integers representing an elevation map where the width of each bar is 1, we need to compute how much water it is able to trap after raining.

`arr[] = {3, 0, 2, 0, 4}.`

Three units of water can be stored in two indexes 1 and 3, and one unit of water at index 2.

Water stored in each index = $0 + 3 + 1 + 3 + 0 = 7$

Explanation : trap “3 units” of water between 3 and 2, “1 unit” on top of bar 2 and “3 units” between 2 and 4.



LOGIC

1. Iterate Through Bars:

Iterate through each bar from the second to the secondtolast bar

2. Find Left and Right Boundaries:

For each bar at index `i``, find the maximum height on its left and right sides.

3. Calculate Trapped Water:

Determine the minimum height between the left and right boundaries.

Subtract the height of the current bar at index `i``.

Add the result to the total trapped water.

4. Return Result:

The total trapped water is the final result.

PYTHON CODE

```
def maxWater(arr, n):  
    res = 0  
    for i in range(1, n - 1):  
        left = arr[i]  
        for j in range(i):  
            left = max(left, arr[j])  
        right = arr[i]  
        for j in range(i + 1, n):  
            right = max(right, arr[j])  
        res += min(left, right) - arr[i]  
    return res  
  
# Example usage:  
arr = [1, 0, 2, 1, 0, 1]  
n = len(arr)  
print(maxWater(arr, n))
```

JAVA CODE

```
public class Main {  
    public static int maxWater(int[] arr, int n) {  
        for (int i = 1; i < n - 1; i++) {  
            int left = arr[i];  
            for (int j = 0; j < i; j++) {  
                left = Math.max(left, arr[j]);  
            }  
            int right = arr[i];  
            for (int j = i + 1; j < n; j++) {  
                right = Math.max(right, arr[j]);  
            }  
        }  
    }  
}
```

```
        res += Math.min(left, right) - arr[i];  
    }  
    return res;  
}  
  
public static void main(String[] args) {  
    int[] arr = { 1, 0, 2, 1, 0, 1};  
    int n = arr.length;  
    System.out.print(maxWater(arr, n));  
}
```

SPIRAL MATRIX

Print a given matrix in spiral form.

Given a 2D array, print it in spiral form. Refer the following examples.

Example 1:

Input:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Output:

1 2 3 4 8 12 16 15 14 13 9 5 6 7
11 10

Example 1:

Input:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Output:

1 2 3 4 5 6 12 18 17 16 15 14
13 7 8 9 10 11

LOGIC

1. Initialization:

Initialize four variables: ``k`` for the starting row, ``l`` for the starting column, ``m`` for the ending row, and ``n`` for the ending column.

2. Spiral Traversal:

While ``k`` is less than ``m`` and ``l`` is less than ``n``, do the following:

Print the elements of the top row from index ``l`` to ``n1``.

Increment ``k``.

Print the elements of the rightmost column from index ``k`` to ``m1``.

Decrement ``n``.

If ``k`` is still less than ``m``, print the elements of the bottom row from index ``n1`` to ``l``.

Decrement ``m``.

If ``l`` is still less than ``n``, print the elements of the leftmost column from index ``m1`` to ``k``.

Increment ``l``.

3. Repeat Until Completion:

Repeat the above steps until all elements are printed.

This approach ensures that the matrix is traversed in a spiral manner, starting from the outer layer and moving towards the center.

PYTHON CODE

```
def spiralPrint(a):
    k, l, m, n = 0, 0, len(a), len(a[0])
    while k < m and l < n:
        for i in range(l, n):
            print(a[k][i], end=" ")
        k += 1
        for i in range(k, m):
            print(a[i][n - 1], end=" ")
        n -= 1
        if k < m:
            for i in range(n - 1, l - 1, -1):
                print(a[m - 1][i], end=" ")
            m -= 1
        if l < n:
            for i in range(m - 1, k - 1, -1):
                print(a[i][l], end=" ")
            l += 1
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16]
]
spiralPrint(matrix)
```

JAVA CODE

```
import java.util.Scanner;

class Main {
    static void spiralPrint(int m, int n,
int a[][]) {
        int i, k = 0, l = 0;
        while (k < m && l < n) {
            for (i = l; i < n; ++i) {
                System.out.print(a[k][i] + "
");
            }
            k++;
            for (i = k; i < m; ++i) {
                System.out.print(a[i][n - 1] +
" ");
            }
            n--;
            if (k < m) {
                for (i = n - 1; i >= l; --i) {
```

```
System.out.print(a[m - 1][i] + " ");
            }
            m--;
        }
        if (l < n) {
            for (i = m - 1; i >= k; --i) {
                System.out.print(a[i][l] + "
");
            }
            l++;
        }
    }

    public static void main(String[]
args) {
        Scanner scanner = new
Scanner(System.in);
        System.out.print("Enter the number
of rows: ");
```

```
int R = scanner.nextInt();
System.out.print("Enter the number of columns: ");
int C = scanner.nextInt();
int a[][] = new int[R][C];
System.out.println("Enter the matrix elements:");
for (int i = 0; i < R; i++) {
    for (int j = 0; j < C; j++) {
        a[i][j] = scanner.nextInt();
    }
}
scanner.close();
spiralPrint(R, C, a);
}
```

0-1 KNAPSACK ALGORITHM

You are given N items that cannot be broken. Each item has a weight and value associated with it.

You also have a KnapSack of capacity W .

Find the maximum value of items you can collect in the KnapSack so that the total weight does not exceed W .

Input Format:

First line contains the values N and W .

Second line contains N integers denoting the weights.

Last line contains N integers denoting the values.

Output Format

Print the maximum value that can be collected with total weight less than or equal to W .

Example 1

Input

3 5

1 2 3

1 5 3

Output

8

Explanation

We can choose item number 2 and 3 to get the value as $5+3(8)$ and total weight as 5 which is less than or equal to $W(5)$.

Example 2

Input

4 10

5 4 6 3

10 40 30 50

Output

90

Explanation

We can choose item number 2 and 4 to get the value as $40+50(90)$ and total weight as 7 which is less than or equal to $W(10)$.

LOGIC

1. Prepare a Table:

Imagine a table where rows represent the items you can choose, and columns represent the capacity of your knapsack.

2. Initialize the Table:

Start by filling the first row and first column with zeros, indicating that with no items or no capacity, the value is zero.

3. Consider Each Item:

Go through each item one by one.

For each item, think about whether it's better to include it in the knapsack or not.

4. Decision Making:

If adding the current item doesn't exceed the capacity, compare the value of including it with the value without it. Choose the maximum.

If adding the item exceeds the capacity, skip it.

5. Build Up the Table:

Keep going through all items and capacities, making decisions and updating the table.

6. Final Answer:

The value in the last cell of the table represents the maximum value you can achieve with the given items and knapsack capacity.


7. Return the Result:

That final value is your answer - it's the maximum value you can get without overloading your knapsack. In essence, it's like deciding which items to pack into a knapsack of limited capacity to maximize the total value.

PYTHON CODE

```
def knapSack(N, W, wt, val):
    dp = [[-1 for _ in range(W + 1)] for _ in range(N + 1)]
    def knapSackHelper(N, W):
        if N == 0 or W == 0:
            return 0
        if dp[N][W] != -1:
            return dp[N][W]
        if wt[N - 1] <= W:
            dp[N][W] = max(val[N - 1] + knapSackHelper(N - 1, W - wt[N - 1]),
                           knapSackHelper(N - 1, W))
            return dp[N][W]
        else:
            dp[N][W] = knapSackHelper(N - 1, W)
            return dp[N][W]
    return knapSackHelper(N, W)
n, W = map(int, input().split())
weights = list(map(int, input().split()))
values = list(map(int, input().split()))
result = knapSack(n, W, weights, values)
print(result)
```


JAVA CODE



```

import java.util.*;
class solution
{
    static int knapSack(int N, int W, int[] wt,
int[] val, int[][] dp) {
        if (N == 0 || W == 0)
            return 0;
        if (dp[N][W] != -1)
            return dp[N][W];
        if (wt[N - 1] <= W)
            return dp[N][W] = Math.max(val[N -
1] + knapSack(N - 1, W - wt[N - 1], wt, val,
dp),
            knapSack(N - 1, W, wt, val, dp));
        return dp[N][W] = knapSack(N - 1, W, wt, val,
dp);
    }
    public int knapSack(int N, int W, int[] wt,
int[] val) {
        int dp[][] = new int[N + 1][W + 1];
        for (int i = 0; i <= N; i++) {
            for (int j = 0; j <= W; j++) {
                dp[i][j] = -1;
            }
        }
    }
}

```

```

        return knapSack(N, W, wt, val, dp);
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n, W;
        n = sc.nextInt();
        W = sc.nextInt();
        int[] wt = new int[n];
        int[] val = new int[n];
        for (int i = 0; i < n; i++)
            wt[i] = sc.nextInt();
        for (int i = 0; i < n; i++)
            val[i] = sc.nextInt();
        solution
        Obj = new solution
        ();
        int result = Obj.knapSack(n, W, wt, val);
        System.out.println(result);
        sc.close();
    }
}

```



NUMBER OF LONGEST INCREASING SUBSEQUENCES

A Longest Increasing Subsequence (LIS) is a subsequence of a given sequence of numbers (not necessarily contiguous) in which the elements are in strictly increasing order. In other words, the Longest Increasing Subsequence problem asks for the length of the longest subsequence such that all elements of the subsequence are sorted in ascending order.

Consider the input sequence: [10, 22, 9, 33, 21, 50, 41, 60, 80]

The Longest Increasing Subsequence in this case is: [10, 22, 33, 50, 60, 80]

EXPLANATION

We start with the first element, 10, and consider it as the first element of a potential increasing subsequence.

Move to the next element, 22. It's greater than 10, so we include it in the potential subsequence.

Move to the next element, 9. It's less than 22, so we can't include it in the current subsequence. We skip it.

Move to the next element, 33. It's greater than 22, so we include it in the potential subsequence.

EXPLANATION

Move to the next element, 21. It's less than 33, so we skip it.

Move to the next element, 50. It's greater than 33, so we include it in the potential subsequence.

Move to the next element, 41. It's less than 50, so we skip it.

Move to the next element, 60. It's greater than 50, so we include it in the potential subsequence.

Move to the next element, 80. It's greater than 60, so we include it in the potential subsequence.

The final Longest Increasing Subsequence is [10, 22, 33, 50, 60, 80] with a length of 6.

LOGIC

1. Initialize:

Create an array `lis` of length `n` filled with 1s.

2. Dynamic Programming:

Iterate through each element in the array (index `i` from 1 to `n`).

For each element, compare it with previous elements.

If current element is greater than the previous one and can extend the LIS, update `lis[i]`.

3. Result:

Return the maximum value in the `lis` array, representing the length of the Longest Increasing Subsequence.

PYTHON CODE

```
def lis(arr, n):  
    lis = [1] * n  
    max_length = 0  
  
    for i in range(1, n):  
        for j in range(i):  
            if arr[i] > arr[j] and lis[i] < lis[j] + 1:  
                lis[i] = lis[j] + 1  
  
    for i in range(n):  
        max_length = max(max_length, lis[i])  
  
    return max_length  
  
# Example usage:  
arr = [10, 22, 33, 50, 60, 80]  
n = len(arr)  
print(lis(arr, n))
```

JAVA CODE

```
class Main {
    static int lis(int arr[], int n)
    {
        int lis[] = new int[n];
        int i, j, max = 0;
        for (i = 0; i < n; i++)
            lis[i] = 1;
        for (i = 1; i < n; i++)
            for (j = 0; j < i; j++)
                if (arr[i] > arr[j] && lis[i] < lis[j] + 1)
                    lis[i] = lis[j] + 1;
        for (i = 0; i < n; i++)
            if (max < lis[i])
```

```
        max = lis[i];
        return max;
    }
    public static void main(String args[])
    {
        int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
        int n = arr.length;
        System.out.println(lis(arr, n));
    }
}
```

WILDCARD PATTERN MATCHING

You are given a pattern string containing letters and wildcard characters. The wildcard character `*` can match any sequence of characters (including an empty sequence), and the wildcard character `?` can match any single character. Your task is to implement a function that determines whether a given input string matches the provided pattern.

Here are the rules for wildcard matching:

The wildcard character `*` can match any sequence of characters (including an empty sequence).

The wildcard character `?` can match any single character.

For example:

The pattern `"h*t"` matches strings like `"hat," "hot," "hut,"` etc.

The pattern `"c?t"` matches strings like `"cat," "cot," "cut,"` etc.

The pattern `"ab*d"` matches strings like `"abd," "abcd," "abbd,"` etc.

Write a function `isMatch(pattern: str, input_str: str) -> bool` that returns `True` if the input string matches the pattern, and `False` otherwise.

LOGIC

1. Initialize 2D Array:

Create a 2D array `T` of size `(n+1) x (m+1)`.

2. Base Case Initialization:

Set `T[0][0] = True`.

For each `j` from 1 to `m`, if pattern at `j1` is '*', set `T[0][j] = T[0][j1]`.

3. Fill 2D Array:

Iterate through each `i` and `j`.

If pattern at `j1` is '*', update `T[i][j] = T[i1][j] or T[i][j1]`.

If pattern at `j1` is '?' or matches word at `i1`, set `T[i][j] = T[i1][j1]`.

4. Result:

Return `T[n][m]`.

PYTHON CODE

```
def isMatch(word, pattern):
    n = len(word)
    m = len(pattern)
    T = [[False] * (m + 1) for _ in range(n + 1)]
    T[0][0] = True
    for j in range(1, m + 1):
        if pattern[j - 1] == '*':
            T[0][j] = T[0][j - 1]
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if pattern[j - 1] == '*':
                T[i][j] = T[i - 1][j] or T[i][j - 1]
            elif pattern[j - 1] == '?' or word[i - 1] == pattern[j - 1]:
                T[i][j] = T[i - 1][j - 1]
    return T[n][m]

# Example usage:
word = "xyxzzxy"
pattern = "x*****x?"
if isMatch(word, pattern):
    print("Match")
else:
    print("No Match")
```

JAVA CODE

```

public class Main {
    public static boolean isMatch(String word, String
pattern) {
        int n = word.length();
        int m = pattern.length();
        boolean[][] T = new boolean[n + 1][m + 1];
        T[0][0] = true;
        for (int j = 1; j <= m; j++) {
            if (pattern.charAt(j - 1) == '*') {
                T[0][j] = T[0][j - 1];
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if (pattern.charAt(j - 1) == '*') {
                    T[i][j] = T[i - 1][j] || T[i][j - 1];
                } else if (pattern.charAt(j - 1) == '?' ||
                    word.charAt(i - 1) == pattern.charAt(j -
1)) {

```

```

                    T[i][j] = T[i - 1][j - 1];
                }
            }
        }
        return T[n][m];
    }
    public static void main(String[] args) {
        String word = "xyxzzxy";
        String pattern = "x***x?";
        if (isMatch(word, pattern)) {
            System.out.print("Match");
        } else {
            System.out.print("No Match");
        }
    }
}

```

HOUSE ROBBER

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night. Given an integer array `nums` representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

Test cases:

Example 1:

Input: `nums = [1,2,3,1]`

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = $1 + 3 = 4$.

Test cases:

Example 2:

Input: `nums = [2,7,9,3,1]`

Output: 12

Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = $2 + 9 + 1 = 12$.

LOGIC

1. Base Cases:

If no houses, no money can be robbed.

If only one house, rob the money in that house.

2. Dynamic Programming:

Create a list `dp` to store max robbed amounts.

3. Recurrence Relation:

To calculate max amount at each house, choose the maximum between:

Amount robbed without current house.

Amount robbed with the current house, plus the amount two houses ago.

4. Initialization:

Initialize first two values in `dp`.

5. Iterative Update:

Iterate through houses, updating `dp` based on the recurrence relation.

6. Result:

Result is the maximum amount in the `dp` list.

This method ensures choosing the best option at each house, either by skipping it or considering it, to maximize the total amount robbed.

PYTHON CODE

```
class Solution:
    def rob(self, nums):
        n = len(nums)
        if n == 0:
            return 0
        if n == 1:
            return nums[0]
        dp = [0] * n
        dp[0] = nums[0]
        dp[1] = max(nums[0], nums[1])
        for i in range(2, n):
            dp[i] = max(dp[i - 1], dp[i - 2] + nums[i])
        return dp[-1]

solution = Solution()
nums1 = [1, 2, 3, 1]
print("Maximum amount you can rob:", solution.rob(nums1)) # Output: 4
nums2 = [2, 7, 9, 3, 1]
print("Maximum amount you can rob:", solution.rob(nums2)) # Output: 12
```

JAVA CODE

```
class Solution {
    public int rob(int[] nums) {
        final int n = nums.length;
        if (n == 0)
            return 0;
        if (n == 1)
            return nums[0];
        // dp[i] := max money of robbing nums[0..i]
        int[] dp = new int[n];
        dp[0] = nums[0];
        dp[1] = Math.max(nums[0], nums[1]);
        for (int i = 2; i < n; ++i)
            dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i]);
        return dp[n - 1];
    }
}
```

EDIT DISTANCE

Given two strings `s1` and `s2`. Return the minimum number of operations required to convert `s1` to `s2`. The possible operations are permitted:

Insert a character at any position of the string.

Remove any character from the string.

Replace any character from the string with any other character.

Input Format

Input consists of two lines, strings `s1` and `s2`

Output Format

Print the minimum number of operations required to convert `s1` to `s2`.

Example 1

Input

horse

ros

Output

3

Explanation

horse -> rorse (replace 'h' with 'r')

rorse -> rose (remove 'r')

rose -> ros (remove 'e')

Example 2

Input

intention

execution

Output

5

Explanation

intention -> inention (remove 't')

inention -> enention (replace 'i' with 'e')

enention -> exention (replace 'n' with 'x')

exention -> exection (replace 'n' with 'c')

exection -> execution (insert 'u')

LOGIC

1. Initialization:

Create a 2D array `dp` with dimensions `(len(s1) + 1) x (len(s2) + 1)`.

Initialize the first row and column with indices, representing the cost of insertions and removals.

2. Dynamic Programming:

Iterate through characters in both strings.

If characters are equal, $dp[i][j] = dp[i-1][j-1]$.

If characters are different, $dp[i][j] = 1 + \min(dp[i][j-1], dp[i-1][j], dp[i-1][j-1])$.

3. Return Result:

Return $dp[len(s1)][len(s2)]$.

This simplified version retains the essence of the logic, focusing on the minimum operations for insertion, removal, or replacement.

PYTHON CODE

```
def edit_distance(s1, s2):
    dp = [[-1 for _ in range(len(s2) + 1)] for _ in range(len(s1) + 1)]
    return rec(s1, s2, len(s1), len(s2), dp)
def rec(s, t, x, y, dp):
    if x == 0:
        return y
    if y == 0:
        return x
    if dp[x][y] != -1:
        return dp[x][y]
    if s[x - 1] == t[y - 1]:
        dp[x][y] = rec(s, t, x - 1, y - 1, dp)
    else:
        dp[x][y] = min(
            1 + rec(s, t, x, y - 1, dp),
            min(1 + rec(s, t, x - 1, y, dp), 1 + rec(s, t, x - 1, y - 1, dp))
        )
    return dp[x][y]
s1 = input()
s2 = input()
print(edit_distance(s1, s2))
```

JAVA CODE

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s1 = sc.next(), s2 = sc.next();
        sc.close();
        System.out.println(editDistance(s1, s2));
    }
    static int[][] dp;
    public static int editDistance(String s1,
String s2) {
        dp = new int[s1.length() + 1][s2.length()
+ 1];
        for (int[] d : dp) Arrays.fill(d, -1);
        return rec(s1, s2, s1.length(),
s2.length());
    }
}
```

```
public static int rec(String s, String t, int
x, int y) {
    if (x == 0) return y;
    if (y == 0) return x;
    if (dp[x][y] != -1) return dp[x][y];
    if (s.charAt(x - 1) == t.charAt(y - 1))
dp[x][y] =
        rec(s, t, x - 1, y - 1); else dp[x][y] =
        Math.min(
            1 + rec(s, t, x, y - 1),
            Math.min(1 + rec(s, t, x - 1, y), 1 +
rec(s, t, x - 1, y - 1))
        );
    return dp[x][y];
}
}
```



/ethnuscodemithra



Ethnus
Codemithra



/ethnus



/code_mithra

<Codemithra />TM



<https://learn.codemithra.com>



Explore | Expand | Enrich



codemithra@ethnus.com



+91 7815 095
095



+91 9019 921 340