



<Codemithra />TM

Explore | Expand | Enrich

<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:

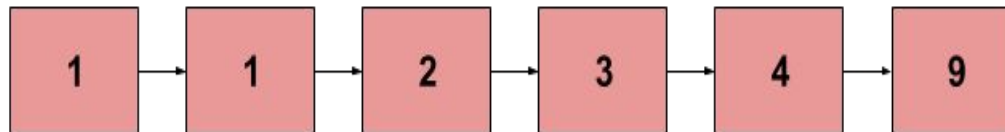
Merged Array: 1 1 2 3 4 4

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



Explore | Expand | Enrich

PYTHON CODE

```
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None
def merge_two_lists(l1, l2):
    dummy = ListNode(0)
    current = dummy
    while l1 and l2:
        if l1.val < l2.val:
            current.next = l1
            l1 = l1.next
        else:
            current.next = l2
            l2 = l2.next
        current = current.next
    if l1:
        current.next = l1
    else:
        current.next = l2
    return dummy.next
def create_linked_list(length, values):
    dummy = ListNode(0)
```

```
        current = dummy
        for val in values:
            current.next = ListNode(val)
            current = current.next
        return dummy.next
if __name__ == "__main__":
    n = int(input())
    values_l1 = list(map(int,
        input().split()))
    l1 = create_linked_list(n, values_l1)

    m = int(input())
    values_l2 = list(map(int,
        input().split()))
    l2 = create_linked_list(m, values_l2)

    merged = merge_two_lists(l1, l2)

    while merged:
        print(merged.val, end=" ")
        merged = merged.next
```

JAVA CODE

```
import java.util.*;
import java.lang.*;
import java.io.*;

class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

class Main {
    public static ListNode
mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(0);
    ListNode current = dummy;

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

```
        current.next = l2;
            l2 = l2.next;
        }
        current = current.next;
    }

    if (l1 != null) {
        current.next = l1;
    } else {
        current.next = l2;
    }

    return dummy.next;
}

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

```
int n = scanner.nextInt();
    ListNode l1 =
createLinkedList(scanner, n);

    int m = scanner.nextInt();
    ListNode l2 =
createLinkedList(scanner, m);

    ListNode merged =
Main.mergeTwoLists(l1, l2);

    while (merged != null) {
        System.out.print(merged.val + "
");
        merged = merged.next;
    }

    scanner.close();
}
```

```
private static ListNode
createLinkedList(Scanner scanner, int length)
{
    ListNode dummy = new ListNode(0);
    ListNode current = dummy;
    for (int i = 0; i < length; i++) {
        int val = scanner.nextInt();
        current.next = new ListNode(val);
        current = current.next;
    }
    return dummy.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

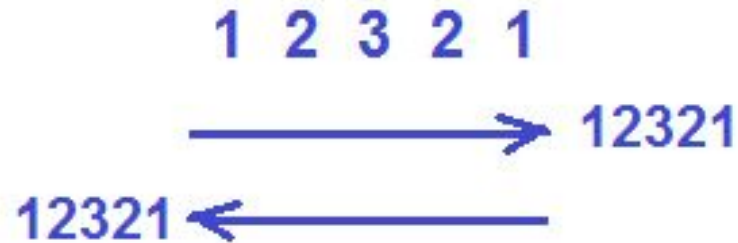
10 20 30 10 50 30

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 Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class Main:
    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
            self.tail = new_node
        self.size += 1
```

```
def reverse_list(self, temp):
    current = temp
    prev_node = None
    next_node = None

    while current:
        next_node = current.next
        current.next = prev_node
        prev_node = current
        current = next_node
    return prev_node

def is_palindrome(self):
    current = self.head
    flag = True
    mid = self.size // 2
    if self.size % 2 == 0 else (self.size + 1) // 2
```

```
for _ in range(1, mid):
    current = current.next

    rev_head =
self.reverse_list(current.next)

    while self.head and rev_head:
        if self.head.data !=
rev_head.data:
            flag = False
            break
        self.head = self.head.next
        rev_head = rev_head.next

    if flag:
        print("true")
    else:
        print("false")
```

```
if __name__ == "__main__":
    s_list = Main()

    numbers_str = input().split()

    for num_str in numbers_str:
        num = int(num_str)
        s_list.add_node(num)

    s_list.is_palindrome()
```

JAVA CODE

```
/* package whatever; // don't place
package name! */

import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main"
only if the class is public. */
class Main
{
    class Node{
        int data;
        Node next;
        public Node(int data){
            this.data = data;
            this.next = null;
        }
    }
}
```

```
public int size;

public Node head = null;
public Node tail = null;

public void addNode(int data){
    Node newNode = new Node(data);

    if(head==null){
        head=newNode;
        tail=newNode;
    }
    else{
        tail.next=newNode;
        tail=newNode;
    }
    size++;
}
```

```
public Node reverseList(Node temp){
    Node current = temp;
    Node prevNode = null,
nextNode=null;
    while(current!=null){
        nextNode = current.next;
        current.next = prevNode;
        prevNode = current;
        current = nextNode;
    }
    return prevNode;
}
public void isPalindrome(){
    Node current = head;
    boolean flag = true;
    int mid =
(size%2==0)?(size/2):((size+1)/2);
```

```
for(int i=1; i<mid; i++){
    current=current.next;
}
Node revHead =
reverseList(current.next);
while(head!=null &&
revHead!=null){
    if(head.data!=revHead.data){
        flag=false;
        break;
    }
    head = head.next;
    revHead = revHead.next;
}if(flag)
    System.out.println("true");
else
    System.out.println("false");
}
```



```
public static void main (String[]
args) throws java.lang.Exception
{
    // your code goes here
    Main sList = new Main();

    Scanner scn = new
Scanner(System.in);

    String input = scn.nextLine();

    String[] numbersStr =
input.split("\\s+");
```

```
for (String numStr : numbersStr) {
    int num =
Integer.parseInt(numStr);
    sList.addNode(num);
}

sList.isPalindrome();
}
}
```

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 Node:
    def __init__(self, key):
        self.data = key
        self.next = None

def print_list(node):
    while node:
        print(node.data, end=" ")
        node = node.next
    print()

def merge_lists(arr):
    result = None
    for node in arr:
        result = merge(result, node)
    return result

def merge(list1, list2):
    dummy = Node(0)
    tail = dummy
```

```
while True:
    if list1 is None:
        tail.next = list2
        break
    if list2 is None:
        tail.next = list1
        break

    if list1.data <= list2.data:
        tail.next = list1
        list1 = list1.next
    else:
        tail.next = list2
        list2 = list2.next
    tail = tail.next

    return dummy.next

if __name__ == "__main__":
    num_lists = int(input())
    lists = []
```



```
for _ in range(num_lists):
    size = int(input())
    elements = list(map(int, input().split()))
    current = None
    for element in elements:
        if not current:
            current = Node(element)
            lists.append(current)
        else:
            current.next = Node(element)
            current = current.next

head = merge_lists(lists)
print_list(head)
```



Explore | Expand | Enrich

JAVA CODE

```
/* package whatever; // don't place package
name! */
import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if
the class is public. */
class Node {
    int data;
    Node next;
    Node(int key) {
        data = key;
        next = null;
    }
}

class Main {
    static Node head;
    static void printList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");

```

```
node = node.next;
        }
        System.out.println();
    }

    static Node mergeLists(Node arr[], int
last) {
    // Implementing merge of two sorted
linked lists
    Node result = null;
    for (int i = 0; i <= last; i++) {
        result = merge(result, arr[i]);
    }
    return result;
}

    static Node merge(Node list1, Node list2)
{
    Node dummy = new Node(0);
    Node tail = dummy;

```

```
while (true) {  
    if (list1 == null) {  
        tail.next = list2;  
        break;  
    }  
    if (list2 == null) {  
        tail.next = list1;  
        break;  
    }  
  
    if (list1.data <= list2.data) {  
        tail.next = list1;  
        list1 = list1.next;  
    } else {  
        tail.next = list2;  
        list2 = list2.next;  
    }  
    tail = tail.next;  
}
```

```
return dummy.next;  
}  
  
public static void main(String[] args)  
throws java.lang.Exception {  
    // Accepting user input  
    Scanner scn = new Scanner(System.in);  
  
    int numLists = scn.nextInt();  
    Node[] lists = new Node[numLists];  
  
    for (int i = 0; i < numLists; i++) {  
        int size = scn.nextInt();  
        Node current = null;  
        for (int j = 0; j < size; j++) {  
            int element = scn.nextInt();
```

```
if (current == null) {
    current = new Node(element);
    lists[i] = current;
} else {
    current.next = new Node(element);
    current = current.next;
}
}

// Merging lists
head = mergeLists(lists, numLists - 1);
printList(head);
}
```

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→2→3→4→5→6→7→8→9

k: 3

Output:

Result: 3→2→1→6→5→4→9→8→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.



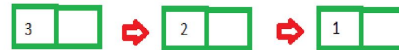
for k=3, break linked list in to two part.



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



Explore | Expand | Enrich

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)
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.*;
import java.lang.*;
import java.io.*;
/* Name of the class has to be "Main"
only if the class is public. */
class Main {
    static class ListNode {
        int val;
        ListNode next;

        ListNode(int val) {
            this.val = val;
        }
    }
    public ListNode head = null;
    public void addNode(int val) {
        ListNode newNode = new ListNode(val);
```

```
        if (head == null) {
            head = newNode;
        } else {
            ListNode temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }
    public static ListNode
reverseKGroup(ListNode head, int k) {
        if (head == null || k == 1)
            return head;
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode start = dummy;
        ListNode end = head;
        int count = 0;
```

```

while (end != null) {
    count++;
    if (count % k == 0) {
        start = reverse(start, end.next);
        end = start.next;
    } else {
        end = end.next;
    }
}
return dummy.next;
}

public static ListNode
reverse(ListNode start, ListNode end) {
    ListNode prev = start;
    ListNode curr = start.next;
    ListNode first = curr;
    while (curr != end) {
        ListNode temp = curr.next;
        curr.next = prev;

```

```

        prev = curr;
        curr = temp;
    }
    start.next = prev;
    first.next = curr;
    return first;
}

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

public static void main(String[]
args) throws java.lang.Exception {
    Main sList = new Main();
    Scanner scn = new
Scanner(System.in);

```



```
String input = scn.nextLine();
    String[] numbersStr = input.split("\\s+");
    for (String numStr : numbersStr) {
        int num = Integer.parseInt(numStr);
        sList.addNode(num);
    }
    int k = scn.nextInt();
    ListNode result = reverseKGroup(sList.head, k);
    printList(result);
}
```



Explore | Expand | Enrich

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:

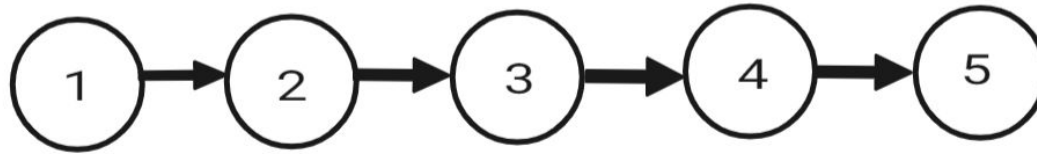
$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$

Examples:

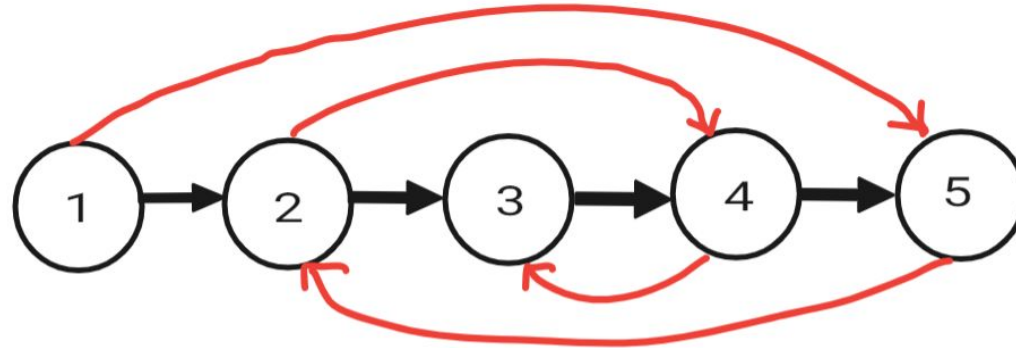
Example 1:

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

Input

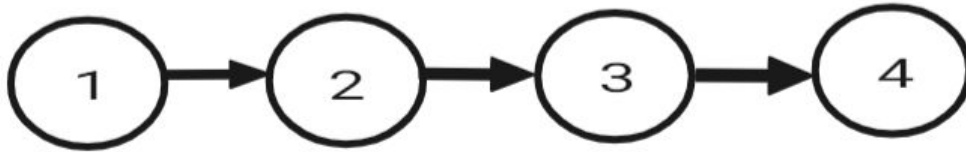


Output:

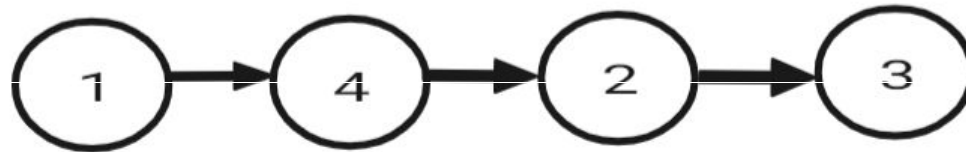


Example 2:

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



Output:





Explore | Expand | Enrich

PYTHON CODE

```
class ListNode:
    def __init__(self, val):
        self.val = val
        self.next = None
def reorderList(head):
    if not head or not head.next:
        return
    # 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
    secondHalf = reverseList(slow.next)
    slow.next = None # Break the linked
list into two halves
    # Merge the two halves
    mergeLists(head, secondHalf)
def reverseList(head):
```

```
prev = None
current = head
while current:
    next_node = current.next
    current.next = prev
    prev = current
    current = next_node
return prev
def mergeLists(first, second):
    while second:
        temp1 = first.next
        temp2 = second.next
        first.next = second
        second.next = temp1
        first = temp1
        second = temp2
def printList(head):
    while head:
        print(head.val, end=" ")
        head = head.next
    print()
```



```
# Input
values = input().split()
head = ListNode(int(values[0]))
current = head
for val in values[1:]:
    current.next = ListNode(int(val))
    current = current.next

# Reorder the list
reorderList(head)

# Output
printList(head)
```



Explore | Expand | Enrich

JAVA CODE

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

    ListNode(int val) {
        this.val = val;
    }
}
public class Main {
    public static void
    reorderList(ListNode head) {
        if (head == null || head.next ==
        null) {
            return;
        }
        // Find the middle of the linked list
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null)
        {
```

```
        slow = slow.next;
            fast = fast.next.next;
        }
        // Reverse the second half of the
        linked list
        ListNode secondHalf =
        reverseList(slow.next);
        slow.next = null; // Break the
        linked list into two halves
        // Merge the two halves
        mergeLists(head, secondHalf);
    }
    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;
}
private static void
mergeLists(ListNode first, ListNode
second) {
    while (second != null) {
        ListNode temp1 = first.next;
        ListNode temp2 = second.next;

        first.next = second;
        second.next = temp1;

        first = temp1;
        second = temp2;
    }
}
```

```
private static void printList(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);
    // Input
    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;
}

// Reorder the list
reorderList(head);

// Output
printList(head);

scanner.close();
}
}
```



Explore | Expand | Enrich

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

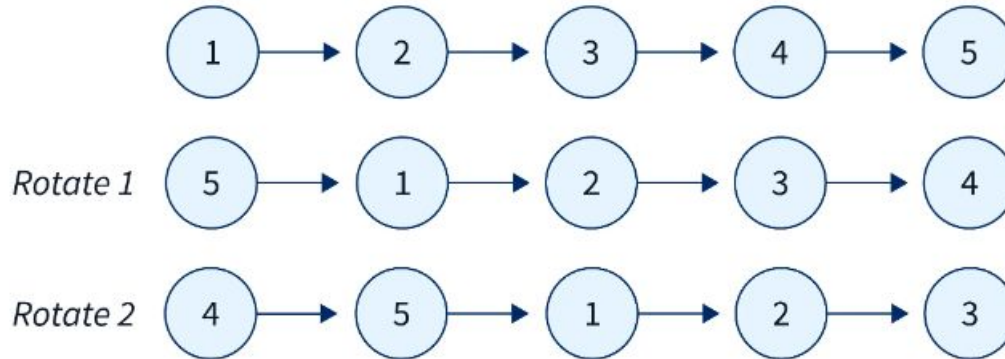
Head: 10->20->30->40->50

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.



Explore | Expand | Enrich

PYTHON CODE

```
class ListNode:
    def __init__(self, val):
        self.val = val
        self.next = None
def rotateRight(head, k):
    if not head or not head.next or k == 0:
        return head
    length = 1
    tail = head
    while tail.next:
        tail = tail.next
        length += 1
    k = k % length
    if k == 0:
        return head
    newTail = head
    for _ in range(length - k - 1):
        newTail = newTail.next
    newHead = newTail.next
    newTail.next = None
    tail.next = head
```

```
return newHead
def printList(head):
    temp = head
    while temp:
        print(temp.val, end=" ")
        temp = temp.next
# Input
numbersStr = input().split()
dummy = ListNode(0)
current = dummy
for numStr in numbersStr:
    num = int(numStr)
    current.next = ListNode(num)
    current = current.next
head = dummy.next
k = 2
head = rotateRight(head, k)
printList(head)
```



Explore | Expand | Enrich

JAVA CODE

```
/* package whatever; // don't place
package name! */
import java.util.*;
import java.lang.*;
import java.io.*;
/* Name of the class has to be "Main"
only if the class is public. */
class ListNode{
    int val;
    ListNode next;
    ListNode(int val){
        this.val = val;
        this.next=null;
    }
}
class Main
{
    public static ListNode
    rotateRight(ListNode head, int k) {
        if (head == null || head.next ==
        null || k == 0)
```

```
return head;
        int length = 1;
        ListNode tail = head;
        while (tail.next != null) {
            tail = tail.next;
            length++;
        }
        k = k % length;
        if (k == 0)
            return head;
        ListNode newTail = head;
        for (int i = 0; i < length - k -
1; i++) {
            newTail = newTail.next;
        }
        ListNode newHead = newTail.next;
        newTail.next = null;
        tail.next = head;
        return newHead;
    }
```

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

public static void main(String[] args)
throws java.lang.Exception {
    Scanner scn = new
Scanner(System.in);

    String input = scn.nextLine();

    String[] numbersStr =
input.split("\\s+");
```

```
ListNode dummy = new ListNode(0);
    ListNode current = dummy;

    for (String numStr : numbersStr)
    {
        int num =
Integer.parseInt(numStr);
        current.next = new
ListNode(num);
        current = current.next;
    }

    ListNode head = dummy.next;

    int k = 2;
    head = rotateRight(head, k);
    printList(head);
}
}
```

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:

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

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.





Explore | Expand | Enrich

PYTHON CODE


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

class Solution:
    def oddEvenList(self, head):
        if not head or not head.next:
            return head

        even_head, even_tail = None, None
        odd_head, odd_tail = None, None

        current = head

        while current:
            if current.val % 2 == 0: # even number
                if not even_head:
                    even_head = current
                    even_tail = current
            else:
                even_tail.next = current
                even_tail = even_tail.next
            else: # odd number
```

```
        if not odd_head:
            odd_head = current
            odd_tail = current
        else:
            odd_tail.next = current
            odd_tail = odd_tail.next
        current = current.next

        if even_tail:
            even_tail.next = odd_head
        else:
            even_head = odd_head

        if odd_tail:
            odd_tail.next = None

        return even_head

def printLinkedList(head):
    while head:
        print(head.val, end=" ")
        head = head.next
    print()
```

```
# Input
input_list = input().split()
head = None
current = None

for val_str in input_list:
    val = int(val_str)
    if not head:
        head = ListNode(val)
        current = head
    else:
        current.next = ListNode(val)
        current = current.next

# Solution
solution = Solution()
result = solution.oddEvenList(head)
printLinkedList(result)
```



Explore | Expand | Enrich

JAVA CODE

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

    public ListNode(int val) {
        this.val = val;
    }
}
class Solution {
    public ListNode oddEvenList(ListNode
head) {
    if (head == null || head.next ==
null) {
        return head;
    }
    ListNode evenHead = null,
evenTail = null;
    ListNode oddHead = null, oddTail
= null;
```

```
ListNode current = head;

    while (current != null) {
        if (current.val % 2 == 0) {
// even number
            if (evenHead == null) {
                evenHead = current;
                evenTail = current;
            } else {
                evenTail.next = current;
                evenTail = evenTail.next;
            }
        } else { // odd number
            if (oddHead == null) {
                oddHead = current;
                oddTail = current;
            } else {
                oddTail.next = current;
                oddTail = oddTail.next;
```

```

    }
        }
        current = current.next;
    }
    if (evenTail != null) {
        evenTail.next = oddHead;
    } else {
        evenHead = oddHead;
    }
    if (oddTail != null) {
        oddTail.next = null;
    }
    return evenHead;
}
}
public class Main {
    public static void main(String[]
args) {

```

```

Scanner scanner = new Scanner(System.in);
    String[] input =
scanner.nextLine().split(" ");
    ListNode head = null;
    ListNode current = null;
    for (String s : input) {
        int val = Integer.parseInt(s);
        if (head == null) {
            head = new ListNode(val);
            current = head;
        } else {
            current.next = new ListNode(val);
            current = current.next;
        }
    }
    Solution solution = new
Solution();

```

```
ListNode result = solution.oddEvenList(head);
    printLinkedList(result);
    scanner.close();
}

private static void printLinkedList(ListNode head) {
    while (head != null) {
        System.out.print(head.val + " ");
        head = head.next;
    }
    System.out.println();
}
}
```

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.



Explore | Expand | Enrich

PYTHON CODE

```
def longest_valid_parentheses(s):
    stack = [-1]
    max_len = 0
    for i, char in enumerate(s):
        if char == '(':
            stack.append(i)
        else:
            stack.pop()
            if not stack:
                stack.append(i)
            else:
                max_len = max(max_len, i - stack[-1])
    return max_len

def convert_to_string(result):
    return str(result)

if __name__ == "__main__":
    input_str = input()
    result = longest_valid_parentheses(input_str)
    print(convert_to_string(result))
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Main
{
    static int findMaxLen(String str){
        int n = str.length();
        Stack<Integer> stk = new Stack<>();
        stk.push(-1);
        int result = 0;
        for(int i=0; i<n; i++){
            if(str.charAt(i)=='('){
                stk.push(i);
            }
            else {
                if(!stk.empty()){
                    stk.pop();
                }
            }
        }
    }
}
```

```
if(!stk.empty()){
    result=Math.max(result, i-stk.peek());
}

else{
    stk.push(i);
}
}
}
return result;
}

public static void main (String[] args) throws java.lang.Exception
{
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();
    System.out.println(findMaxLen(str));
}
}
```

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.
- 



Explore | Expand | Enrich

PYTHON CODE

```
def infix_to_postfix(infix):
    postfix = []
    stack = []

    for c in infix:
        if c.isalnum():
            postfix.append(c)
        elif c == '(':
            stack.append(c)
        elif c == ')':
            while stack and stack[-1] != '(':
                postfix.append(stack.pop())
            stack.pop() # Remove '(' from stack
        else:
            while stack and precedence(c) <=
precedence(stack[-1]):
                postfix.append(stack.pop())
            stack.append(c)
```

```
        while stack:
            postfix.append(stack.pop())
        return ''.join(postfix)
def precedence(operator):
    if operator in ('+', '-'):
        return 1
    elif operator in ('*', '/'):
        return 2
    return -1
def convert_to_string(result):
    return result
if __name__ == "__main__":
    infix_expression = input()
    result = infix_to_postfix(infix_expression)
    print(convert_to_string(result))
```



Explore | Expand | Enrich

JAVA CODE

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

public class Main {
    public static String infixToPostfix(String infix) {
        StringBuilder postfix = new StringBuilder();
        Stack<Character> stack = new Stack<>();
        for (char c : infix.toCharArray()) {
            if (Character.isLetterOrDigit(c)) {
                postfix.append(c);
            } else if (c == '(') {
                stack.push(c);
            } else if (c == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    postfix.append(stack.pop());
                }
                stack.pop(); // Remove '(' from stack
            } else {
```

```
                while (!stack.isEmpty() && precedence(c) <=
                    precedence(stack.peek())) {
                    postfix.append(stack.pop());
                }
                stack.push(c);
            }
        }
        while (!stack.isEmpty()) {
            postfix.append(stack.pop());
        }
        return postfix.toString();
    }

    private static int precedence(char operator) {
        switch (operator) {
            case '+':
```

```
case '-':  
    return 1;  
case '*':  
case '/':  
    return 2;  
}  
return -1;  
}  
  
public static String convertToString(String result) {  
    return result;  
}
```

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    String infixExpression = scanner.nextLine();  
    String result = infixToPostfix(infixExpression);  
    System.out.println(convertToString(result));  
    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.



Explore | Expand | Enrich

PYTHON CODE

```
def evaluate_postfix(postfix):
    stack = []
    for token in postfix.split():
        if is_operand(token):
            stack.append(int(token))
        else:
            operand2 = stack.pop()
            operand1 = stack.pop()
            if token == "+":
                stack.append(operand1 + operand2)
            elif token == "-":
                stack.append(operand1 - operand2)
            elif token == "*":
                stack.append(operand1 * operand2)
            elif token == "/":
                stack.append(operand1 / operand2)
```

```
    return stack.pop()
def is_operand(token):
    return token.isdigit()
def convert_to_string(result):
    return str(result)
if __name__ == "__main__":
    postfix_expression = input()
    result =
    evaluate_postfix(postfix_expression)
    print(convert_to_string(result))
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.Scanner;
import java.util.Stack;
public class Main {
    public static int evaluatePostfix(String
postfix) {
        Stack<Integer> stack = new Stack<>();
        for (String token :
postfix.split("\\s+")) {
            if (isOperand(token)) {
stack.push(Integer.parseInt(token));
            } else {
                int operand2 = stack.pop();
                int operand1 = stack.pop();
                switch (token) {
                    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();
    }
    private static boolean isOperand(String
token) {
```

```
return token.matches("\\d+");
}
public static String convertToString(int result) {
    return Integer.toString(result);
}
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String postfixExpression = scanner.nextLine();
    int result = evaluatePostfix(postfixExpression);
    System.out.println(convertToString(result));
    scanner.close();
}
}
```


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



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.
- 



Explore | Expand | Enrich

PYTHON CODE

```
def precedence(c):
    if c in ['+', '-']:
        return 1
    elif c in ['*', '/']:
        return 2
    else:
        return 0

def evaluate(operands, operators):
    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)
```

```
def main():
    operands = []
    operators = []
    input_str = input().strip()
    i = 0
    while i < len(input_str):
        c = input_str[i]
        if c.isdigit():
            num = int(c)
            while i + 1 < len(input_str) and input_str[i + 1].isdigit():
                num = num * 10 + int(input_str[i + 1])
                i += 1
            operands.append(num)
        elif c in ['+', '-', '*', '/']:
            while operators and precedence(c) <= precedence(operators[-1]):
                evaluate(operands, operators)
            operators.append(c)
        i += 1
    while operators:
        evaluate(operands, operators)
    print(operands.pop())

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



Explore | Expand | Enrich

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

Implement stack using two queues. You need to complete the push and pop function of stack class. You are given 2 types of queries-

'1' which represents that we need to push an integer into the stack.

'2' which represents that we need to pop the top element from the stack. If there is no top element simply return -1.

Input Format

First line contains q of queries.

Followed by q lines.

Query of type 1 is followed by an integer x to push element in the stack.

Query of type 2 is for pop the top value from the stack and print it.

Output Format

Print the value for pop operations in the query given.

Example 1

Input

5

1 2

1 3

2

1 4

2

Output:

3 4

Example 2

Input

3

2

1 2

2

Output:

-1 2

LOGIC

1. Initialization: Two queues ``q1`` and ``q2`` are initialized as instances of the ``LinkedList`` class, which implements the ``Queue`` interface.
2. Push Operation: When a new element is pushed onto the stack, it is added to ``q1``. If ``q1`` is not empty, the elements of ``q1`` are moved to ``q2`` one by one. Then, the new element is added to ``q1``. Finally, the elements of ``q2`` are moved back to ``q1``. This ensures that the new element is always at the front of the queue, simulating the behavior of a stack.
3. Pop Operation: When an element is popped from the stack, it is simply removed from ``q1`` if ``q1`` is not empty. If ``q1`` is empty, ``-1`` is returned to indicate that the stack is empty.

LOGIC

4. Main Method: In the ``main`` method, the number of queries ``q`` is read from the input. Then, a loop runs ``q`` times to process each query. For each query, the type of operation (``QueryType``) is read from the input. If the operation is a push (``QueryType == 1``), the value to be pushed onto the stack is read from the input and pushed onto the stack using the ``push`` method. If the operation is a pop (``QueryType == 2``), the value popped from the stack is added to the ``ans`` list using the ``pop`` method.

5. Output: Finally, the elements in the ``ans`` list are printed, separated by spaces, to display the result of the pop operations.



Explore | Expand | Enrich

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()}")

```



Explore | Expand | Enrich

JAVA CODE

```
import java.io.*;
import java.util.*;

class Queues {
    Queue<Integer> q1 = new LinkedList<>();
    Queue<Integer> q2 = new LinkedList<>();

    // Function to push an element into stack using two queues.
    void push(int a) {
        // your code here
        if (q1.isEmpty() == true) {
            q1.add(a);
        } else {
            while (!q1.isEmpty()) {
                q2.add(q1.poll());
            }
            q1.add(a);
            while (!q2.isEmpty()) {
                q1.add(q2.poll());
            }
        }
    }
}
```

```
// Function to pop an element from stack using two queues.
int pop() {
    // your code here
    if (!q1.isEmpty()) {
        return q1.poll();
    }
    return -1;
}

}

public class Main {
    public static void main(String args[]) throws IOException {
        Scanner sc = new Scanner(System.in);
        Queues g = new Queues();
        int q = sc.nextInt();
        ArrayList<Integer> ans = new ArrayList<>();
        while (q > 0) {
            int QueryType = sc.nextInt();
            if (QueryType == 1) {
                int a = sc.nextInt();
                g.push(a);
            } else if (QueryType == 2)
```

```
ans.add(g.pop());  
    q--;  
}  
for (int x : ans)  
    System.out.print(x + " ");  
System.out.println();  
}  
}
```

IMPLEMENT QUEUE USING STACKS

Implement Queue using two queue S1 and S2. You need to complete the push and pop function of Queue class. You are given 2 types of query 1 for push an integer into queue and 2 for enqueue the value from the queue and print.

Input Format

First line contains q of queries.

Followed by q lines.

Query of type 1 is followed by an integer x to push element in the queue.

Query of type 2 is for dequeue the last value from the queue and print.

Output Format

Print the value for dequeue operations in the query given.

LOGIC

1. Initialize two stacks, "s1" and "s2", in the "StackQueue" class.
2. Implement the "Push" method to enqueue an element into the queue using two stacks:

Transfer all elements from "s1" to "s2".

Push the new element onto "s1".

Transfer all elements from "s2" back to "s1".
3. Implement the "Pop" method to dequeue an element from the queue:

Check if "s1" is empty. If it is, return "1".

Otherwise, pop the top element from "s1" and return it.

4. In the "main" method, create an instance of "StackQueue" class.
5. Read the number of queries "q" from the input, and loop "q" times:

 For each query, read the query type ("QueryType").

 If the query type is "1", read the value to be pushed ("a") and call the "Push" method.

 If the query type is "2", call the "Pop" method and add the result to the "ans" list.
6. Print the elements of the "ans" list, separated by spaces, to display the result of the pop operations.



Explore | Expand | Enrich

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()
```



Explore | Expand | Enrich

JAVA CODE

```
import java.io.*;
import java.util.*;
class StackQueue {
    Stack<Integer> s1 = new Stack<>();
    Stack<Integer> s2 = new Stack<>();
    // Function to push an element in queue by using 2 stacks.
    void Push(int x) {
        // Write your code here
        while (!s1.isEmpty()) {
            s2.push(s1.pop());
        }
        s1.push(x);
        while (!s2.isEmpty()) {
            s1.push(s2.pop());
        }
    }
    // Function to pop an element from queue by using 2 stacks.
    int Pop() {
        if (s1.isEmpty()) {
            return -1;
        }
    }
}
```



```
        return s1.pop();
    }
}

public class Main {
    public static void main(String args[]) throws IOException {
        Scanner sc = new Scanner(System.in);
        StackQueue s = new StackQueue();
        int q = sc.nextInt();
        ArrayList<Integer> ans = new ArrayList<>();
        while (q > 0) {
            int QueryType = sc.nextInt();
            if (QueryType == 1) {
                int a = sc.nextInt();
                s.Push(a);
            } else if (QueryType == 2)
                ans.add(s.Pop());
            q--;
        }
        for (int x : ans)
            System.out.print(x + " ");
        System.out.println();
    }
}
```

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

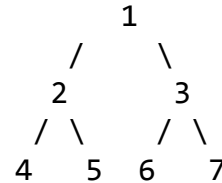
1 2 3 4 5 6 7

Output

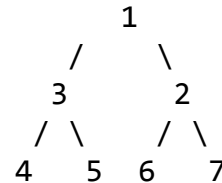
1 3 2 4 5 6 7

Explanation

Original tree was:



After Zig Zag traversal, tree formed would be:



Test cases:

Example 2

Input

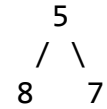
5 8 7

Output

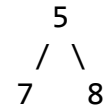
5 7 8

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.



Explore | Expand | Enrich

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)
```



Explore | Expand | Enrich

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);
        }
    }
}
```

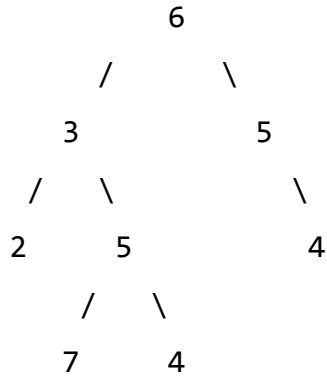
```
if (currentLevel.isEmpty()) {
    leftToRight = !leftToRight;
    Stack<Node> temp =
currentLevel;

        }
    }
}
```

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

$$\text{Answer} = 632 + 6357 + 6354 + 654 = 13997$$

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.



Explore | Expand | Enrich

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 node is None:
            return 0
        val = val * 10 + node.data
        if node.left is None and node.right is None:
            return val
        return self.tree_paths_sum_util(node.left,
val) + self.tree_paths_sum_util(node.right, val)

    def tree_paths_sum(self):
        return self.tree_paths_sum_util(self.root, 0)

def insert_node(root, data):
    if root is None:
        return Node(data)
```

```
queue = [root]
while queue:
    temp = queue.pop(0)

    if temp.left is None:
        temp.left = Node(data)
        break
    else:
        queue.append(temp.left)

    if temp.right is None:
        temp.right = Node(data)
        break
    else:
        queue.append(temp.right)

def main():
    s = input().strip()

    tree = BinaryTree()
    tree.root = Node(int(s[0]))
```

```
index = 1
while index < len(s):
    insert_node(tree.root, int(s[index]))
    index += 1

print(tree.tree_paths_sum())

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



Explore | Expand | Enrich

JAVA CODE

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;
import java.util.Queue;
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() {
    return treePathsSumUtil(root, 0);
}
static void insertNode(Node root, int data) {
    Queue<Node> queue = new LinkedList<>();
    queue.add(root);
```

```
while (!queue.isEmpty()) {  
    Node temp = queue.poll();  
    if (temp.left == null) {  
        temp.left = new Node(data);  
        break;  
    } else {  
        queue.add(temp.left);  
    }  
    if (temp.right == null) {  
        temp.right = new Node(data);  
        break;  
    } else {  
        queue.add(temp.right);  
    }  
}  
  
public static void main(String args[]) throws IOException {
```

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String s = br.readLine();
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(Character.getNumericValue(s.charAt(0)));
    int index = 1;
    while (index < s.length()) {
        insertNode(tree.root, Character.getNumericValue(s.charAt(index)));
        index++;
    }
    System.out.println(tree.treePathsSum());
}
```


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.



Explore | Expand | Enrich

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):
        if not root:
            return []
        result = []
        queue = deque()
        queue.append(root)
        while queue:
            n = len(queue)
            for i in range(n):
                curr = queue.popleft()
                if i == n - 1:
                    result.append(curr.data)
                if curr.left:
                    queue.append(curr.left)
                if curr.right:
                    queue.append(curr.right)
        return result
```

```
def buildTree(s):
    if not s or s[0] == 'N':
        return None

    ip = s.split()
    root = Node(int(ip[0]))
    queue = deque([root])
    i = 1

    while queue and i < len(ip):
        currNode = queue.popleft()

        currVal = ip[i]
        if currVal != 'N':
            currNode.left = Node(int(currVal))
            queue.append(currNode.left)
        i += 1

    if i >= len(ip):
        break
```



```
currVal = ip[i]
    if currVal != 'N':
        currNode.right = Node(int(currVal))
        queue.append(currNode.right)
    i += 1
```

```
    return root
```

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



Explore | Expand | Enrich

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.



Explore | Expand | Enrich

PYTHON CODE

```

from collections import deque
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
class Solution:
    class A:
        def __init__(self):
            self.ans = 0
    @staticmethod
    def height(root, a):
        if not root:
            return 0
        left = Solution.height(root.left, a)
        right = Solution.height(root.right, a)
        a.ans = max(a.ans, left + right + 1)
        return 1 + max(left, right)
    @staticmethod
    def diameter(root):
        if not root:
            return 0

```

```

a = Solution.A()
    Solution.height(root, a)
    return a.ans

def buildTree(s):
    if not s or s[0] == 'N':
        return None
    ip = s.split()
    root = Node(int(ip[0]))
    queue = deque([root])
    i = 1
    while queue and i < len(ip):
        currNode = queue.popleft()
        currVal = ip[i]
        if currVal != 'N':
            currNode.left = Node(int(currVal))
            queue.append(currNode.left)
        i += 1
    if i >= len(ip):
        break
    currVal = ip[i]

```

```
if currVal != 'N':
    currNode.right = Node(int(currVal))
    queue.append(currNode.right)
    i += 1
return root

if __name__ == "__main__":
    s1 = input().strip()
    root1 = buildTree(s1)
    g = Solution()
    print(g.diameter(root1))
```



Explore | Expand | Enrich

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.

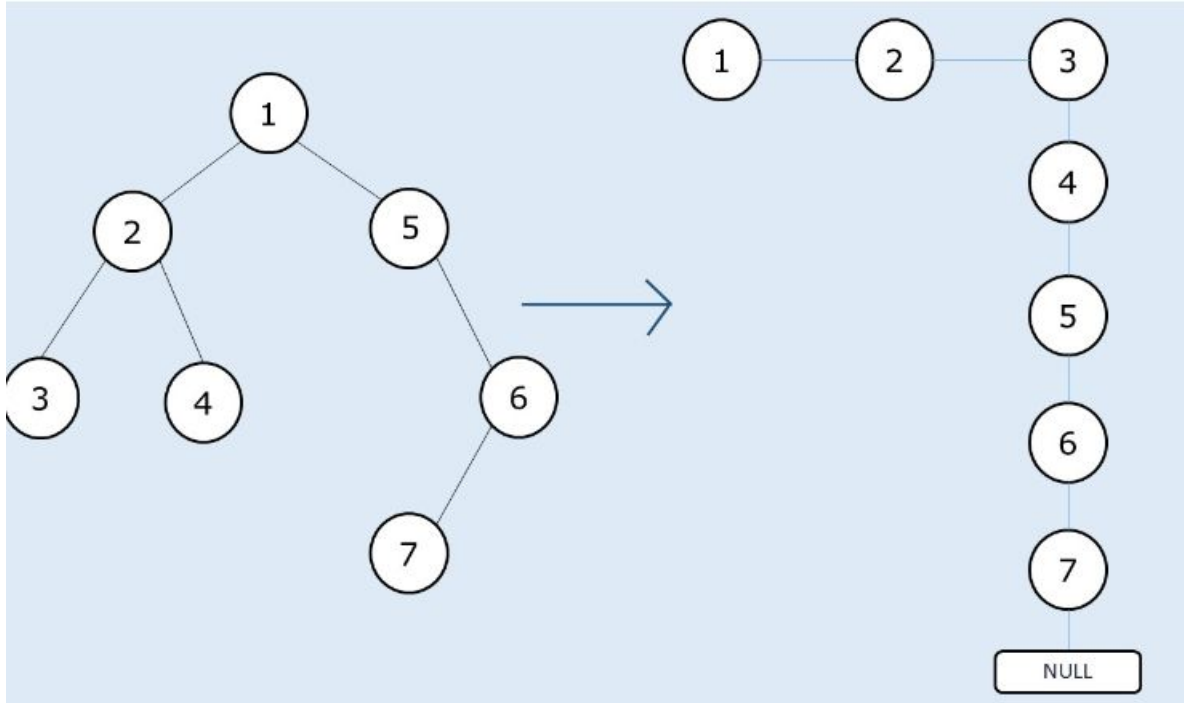
Note:

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

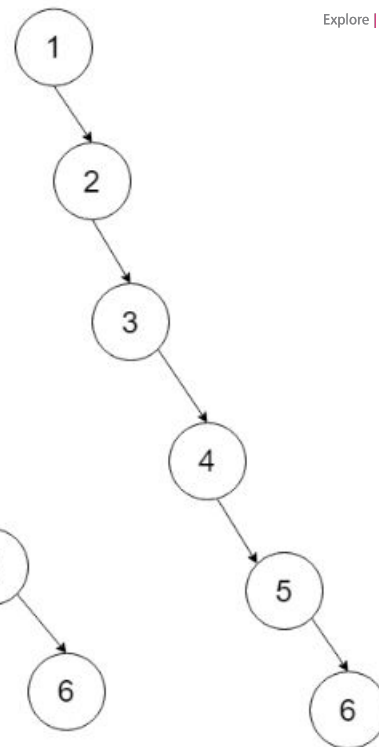
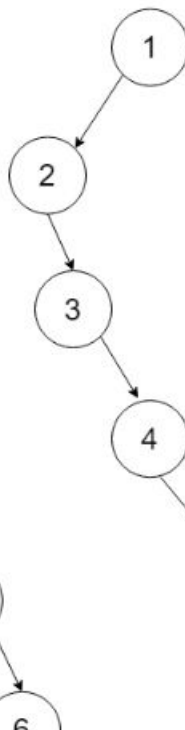
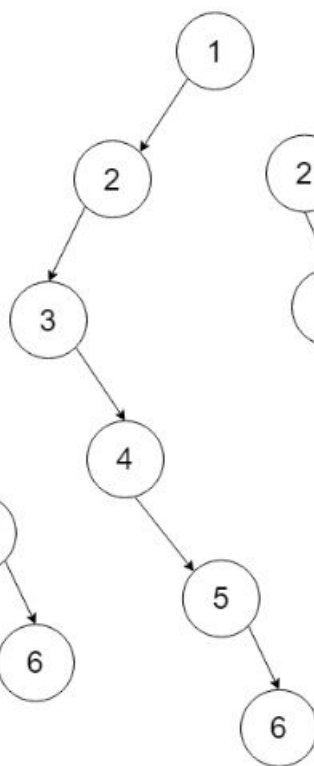
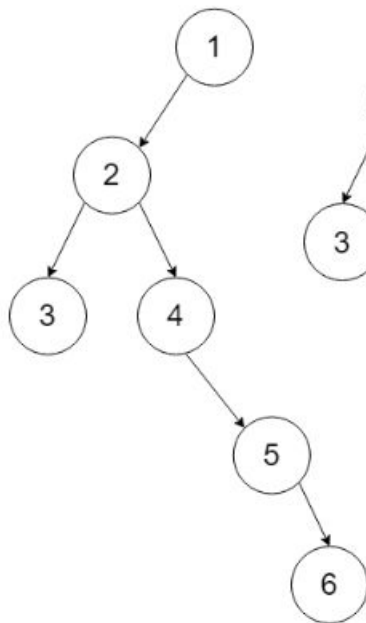
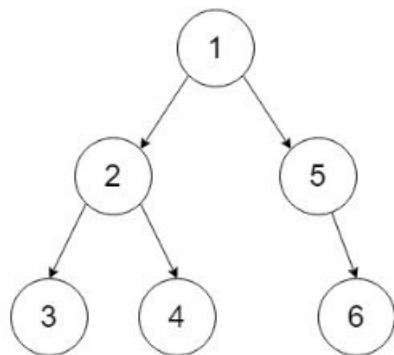
Make the left node as a right node of the root

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.



Explore | Expand | Enrich

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)
```



Explore | Expand | Enrich

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

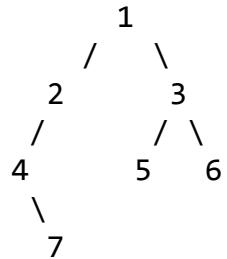
Input

```
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

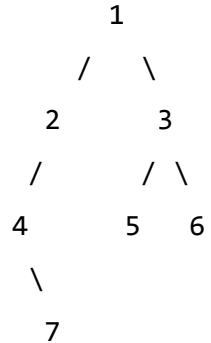
1 2 3 4 -1 5 6 -1 7 -1 -1 -1 -1 -1

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.



Explore | Expand | Enrich

PYTHON CODE

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

def buildTree(arr):
    if not arr or arr[0] == 'N':
        return None

    root = TreeNode(int(arr[0]))
    queue = [root]
    i = 1
    while queue and i < len(arr):
        currNode = queue.pop(0)

        if arr[i] != '-1':
            currNode.left =
TreeNode(int(arr[i]))
            queue.append(currNode.left)
        i += 1
```

```
if i >= len(arr):
    break
    if arr[i] != '-1':
        currNode.right =
TreeNode(int(arr[i]))
        queue.append(currNode.right)
    i += 1

return root

def findLCA(node, n1, n2):
    if not node:
        return None
    if node.data == n1 or node.data == n2:
        return node

    left_lca = findLCA(node.left, n1, n2)
    right_lca = findLCA(node.right, n1, n2)
```

```
if left_lca and right_lca:
    return node
return left_lca if left_lca else right_lca

if __name__ == "__main__":
    s = input().split()
    root = buildTree(s)
    x, y = map(int, input().split())
    ans = findLCA(root, x, y)
    print(ans.data if ans else None)
```



Explore | Expand | Enrich

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 -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.



Explore | Expand | Enrich

PYTHON CODE


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

def build_tree(nodes):
    if not nodes:
        return None

    root = BinaryTreeNode(int(nodes[0]))
    queue = [root]
    index = 1

    while index < len(nodes):
        node = queue.pop(0)

        left_val = nodes[index]
        index += 1
        if left_val != "-1":
            node.left =
BinaryTreeNode(int(left_val))
```

```
queue.append(node.left)

        if index < len(nodes):
            right_val = nodes[index]
            index += 1
            if right_val != "-1":
                node.right =
BinaryTreeNode(int(right_val))
                queue.append(node.right)

    return root

def helper(root, min_val, max_val):
    if root is None:
        return True

    if not min_val <= root.data <= max_val:
        return False

    return helper(root.left, min_val,
root.data - 1) and helper(root.right
root.data + 1, max_val)
```

```
def validate_BST(root):  
    return helper(root, float('-inf'), float('inf'))  
  
if __name__ == "__main__":  
    t = int(input())  
  
    for _ in range(t):  
        nodes = input().split()  
        root = build_tree(nodes)  
        print("true" if validate_BST(root) else "false")
```



Explore | Expand | Enrich

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;

        // Create a List of
        BinaryTreeNode<Integer>
        List<BinaryTreeNode<Integer>> tree
        = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            if
(Integer.parseInt(str_arr[i]) != -1)
                tree.add(new
BinaryTreeNode<>(Integer.parseInt(str_arr[i]))
);

```

```
else
            tree.add(null);
        }

        int i = 0, j = 1;
        while (i < n) {
            if (tree.get(i) != null) {
                tree.get(i).left =
tree.get(j++);
                tree.get(i).right =
tree.get(j++);
            }
            i++;
        }

        // Check if the binary tree is a
        valid BST and print the result

        System.out.println(validateBST(tree.get(0)));
    }
    sc.close();
}
}
```

Kth SMALLEST ELEMENT IN A BST

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

Input:

BST:

```
      2
     / \
    1   3
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`.



Explore | Expand | Enrich

PYTHON CODE

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
def kthSmallest(root, k):
    # Array to store count and result
    result = [0, 0]
    # Initialize the count
    result[0] = 0
    # Recursive function to find the k-th
    # smallest element
    kthSmallestHelper(root, k, result)
    # Return the result
    return result[1]
def kthSmallestHelper(root, k, result):
    # Base case: If root is null or k-th
    # smallest element is found
```

```
    if root is None or result[0] == k:
        return
    # Traverse left subtree
    kthSmallestHelper(root.left, k, result)
    # Increment count
    result[0] += 1
    # If count becomes k, update result
    if result[0] == k:
        result[1] = root.val
        return
    # Traverse right subtree
    kthSmallestHelper(root.right, k, result)
# Create a sample BST
root = TreeNode(1)
root.right = TreeNode(3)
root.right.left = TreeNode(2)
# Set the value of k
k = 3
```

```
# Find the k-th smallest element in the BST  
result = kthSmallest(root, k)  
  
# Output the result  
print(result)
```



Explore | Expand | Enrich

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(1);
        root.right = new TreeNode(3);
        root.right.left = new TreeNode(2);
        // Set the value of k
        int k = 3;
        // Find the k-th smallest element in the BST
        int result = kthSmallest(root, k);
        // Output the result
        System.out.println(result);
    }
}
```

```
public static int kthSmallest(TreeNode root, int k)
{
    // Array to store count and result
    int[] result = new int[2];
    // Initialize the count
    result[0] = 0;
    // Recursive function to find the k-th
smallest element
    kthSmallestHelper(root, k, result);
    // Return the result
    return result[1];
}

private static void kthSmallestHelper(TreeNode
root, int k, int[] result) {
    // Base case: If root is null or k-th smallest
element is found
    if (root == null || result[0] == k) {
        return;
    }
}
```

```
// Traverse left subtree
    kthSmallestHelper(root.left, k, result);
    // Increment count
    result[0]++;
    // If count becomes k, update result
    if (result[0] == k) {
        result[1] = root.val;
        return;
    }
    // Traverse right subtree
    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.

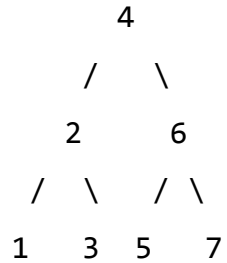
Input: Linked List 1->2->3

Output: A Balanced BST

```
    2
   / \
  1   3
```

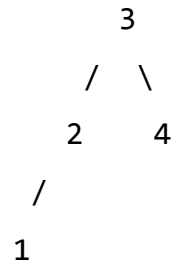
Input: Linked List 1->2->3->4->5->6->7

Output: A Balanced BST



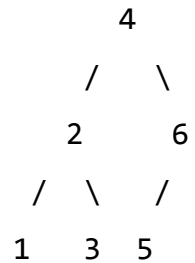
Input: Linked List 1->2->3->4

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.



Explore | Expand | Enrich

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)
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.*;
class Main {
    static LNode head;
    static class LNode {
        int data;
        LNode next, prev;

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

```
TNode(int d) {
    data = d;
    left = right = null;
}
}
static TNode sortedListToBST() {
    int n = countNodes(head);
    return sortedListToBSTRecur(n);
}
static 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;
}
static int countNodes(LNode head) {
    int count = 0;
    LNode temp = head;
    while (temp != null) {
        temp = temp.next;
        count++;
    }
    return count;
}
```

```
static void push(int new_data) {
    LNode new_node = new LNode(new_data);
    new_node.next = null;
    if (head == null) {
        new_node.prev = null;
        head = new_node;
        return;
    }
    LNode last = head;
    while (last.next != null)
        last = last.next;
    last.next = new_node;
    new_node.prev = last;
}
static void printList(LNode node) {
```

```

while (node != null) {
    System.out.print(node.data + " ");
    node = node.next;
}
static 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) {
    Scanner scanner = new
Scanner(System.in);

```

```

String input = scanner.nextLine().trim();
// String input = "1 2 3 4 5 6 7";
scanner.close();
String[] values = input.split("\\s+");
Main llist = new Main();
for (String value : values) {
    llist.push(Integer.parseInt(value));
}
TNode root = llist.sortedListToBST();
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.

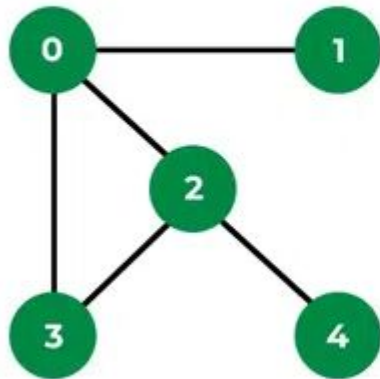
Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure

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

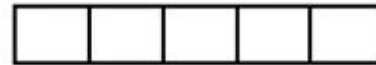
1. Visited
2. Not Visited

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

Step1: Initially stack and visited arrays are empty.



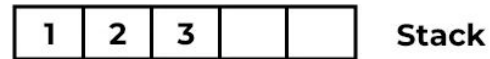
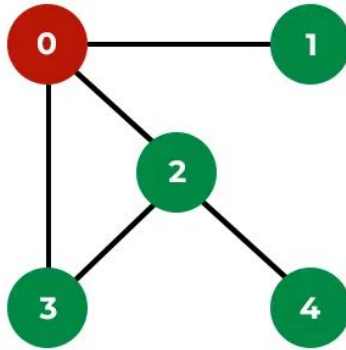
Visited



Stack

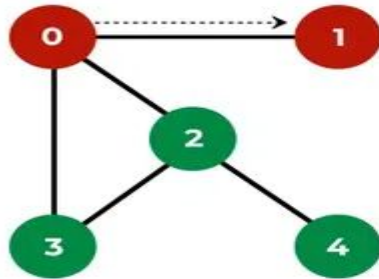
DFS on Graph

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.



0	1			
---	---	--	--	--

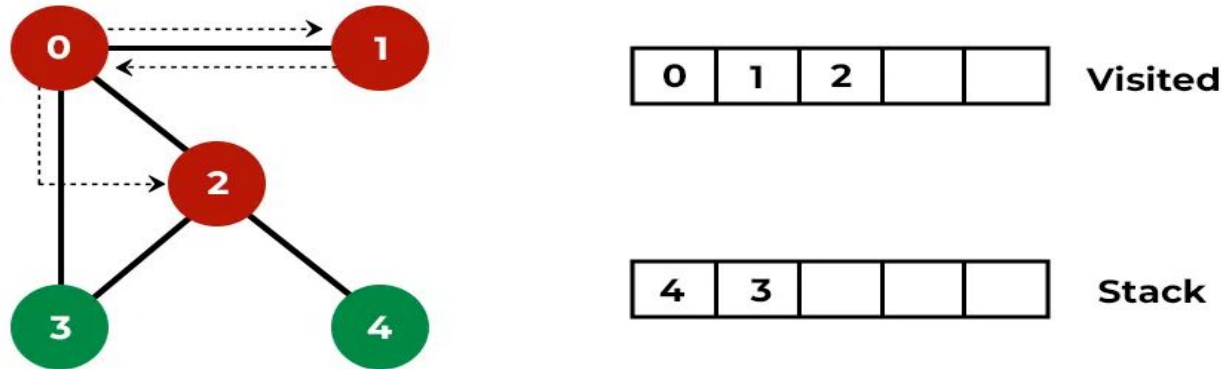
Visited

2	3			
---	---	--	--	--

Stack

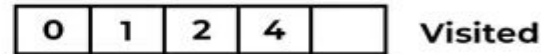
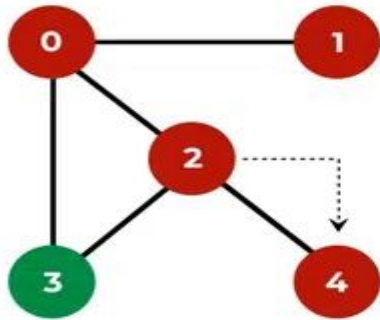
DFS on Graph

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

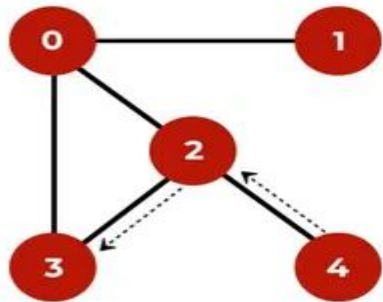
Step 5: Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



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

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)
}
```




Explore | Expand | Enrich

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)
```



Explore | Expand | Enrich

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:

- Choose the vertex ``u`` with the minimum key value from the set of vertices not yet included in the MST (``mstSet[]`` is ``false``).
- Include ``u`` in the MST by setting ``mstSet[u]`` to ``true``.
- 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.



Explore | Expand | Enrich

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)
```



Explore | Expand | Enrich

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();  
}  
}
```



Explore | Expand | Enrich

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.

Test Case 1

Input:-

3 1,3 2,1 0,0 4,0 5,4 6

2

Output:-

7

Test Case 2

Input:-

0 1,0 2,0 3

5

Output:-

3

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.



Explore | Expand | Enrich

PYTHON CODE

class Solution:

```
def __init__(self):
```

```
    self.ans = 0
```

```
def minimum_fuel_cost(self, roads, seats):
```

```
    adj = [[] for _ in range(len(roads) + 1)]
```

```
    self.ans = 0
```

```
    for a in roads:
```

```
        adj[a[0]].append(a[1])
```

```
        adj[a[1]].append(a[0])
```

```
    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 += (people + seats - 1) // seats
```

```
            return people
```

```
if __name__ == "__main__":
```

```
    solution = Solution()
```

```
    road_input = input().split(",")
```

```
    roads = [list(map(int, pair.split())) for
```

```
pair in road_input]
```

```
    seats_input = input()
```

```
    seats = int(seats_input)
```

```
    result = solution.minimum_fuel_cost(roads,
```

```
seats)
```

```
    print(result)
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.ArrayList;
import java.util.Scanner;
public class Main {
    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) {  
    Main solution = new Main();  
    Scanner scanner = new Scanner(System.in);  
    String roadInput = scanner.nextLine();  
    // Split the input string and convert it  
into a 2D array  
    String[] roadPairs = roadInput.split(",");  
    int numberOfRoads = roadPairs.length;  
    int[][] roads = new int[numberOfRoads][2];  
    for (int i = 0; i < numberOfRoads; i++) {  
        String[] pair =  
roadPairs[i].trim().split(" ");  
        roads[i][0] = Integer.parseInt(pair[0]);  
        roads[i][1] = Integer.parseInt(pair[1]);  
    }  
}
```

```
// Input the number of seats as a string and convert  
to an integer  
        //System.out.print("Enter the number of  
seats: ");  
        String seatsInput = scanner.nextLine();  
        int seats = Integer.parseInt(seatsInput);  
        long result =  
solution.minimumFuelCost(roads, seats);  
        System.out.println(String.valueOf(result));  
        // Close the scanner  
        scanner.close();  
    }}
```

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 two lines contain 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.

Test Case 1

Input:-

3

4

1 0 1 0

1 1 0 0

0 0 1 1

Output:-

3

Test Case 2

Input:-

3

4

1 1 0 0

0 0 0 1

0 1 1 0

Output:-

2

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.



Explore | Expand | Enrich

PYTHON CODE

class Solution:

```

    def __init__(self):
self.dirs = [(0, -1), (-1, 0), (0, 1), (1, 0)]
    def toString(self, r, c):
        return str(r) + " " + str(c)
    def dfs(self, grid, x0, y0, i, j, v):
        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(self.toString(i - x0, j - y0))
        for dx, dy in self.dirs:
            self.dfs(grid, x0, y0, i + dx, j +
dy, v)

```

def countDistinctIslands(self, grid):

```

    if not grid or not grid[0]:
        return 0
    coordinates = set()
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == 1:
                v = []
                self.dfs(grid, i, j, i, j,
v)
                coordinates.add(tuple(v))
    return len(coordinates)

if __name__ == "__main__":

```

```
n = int(input())
m = int(input())
grid = []
for _ in range(n):
    row_input = list(map(int, input().split()))
    grid.append(row_input)
solution = Solution()
ans = solution.countDistinctIslands(grid)
print(ans)
```



Explore | Expand | Enrich

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);
        //System.out.print("Enter the number of rows
(N): ");

```

```

int n = Integer.parseInt(sc.nextLine());
//System.out.print("Enter the number of
columns (M): ");
int m = Integer.parseInt(sc.nextLine());
int[][] grid = new int[n][m];
//System.out.println("Enter the grid
elements (1 for land, 0 for water): ");
for (int i = 0; i < n; i++) {
    String[] rowInput =
sc.nextLine().split(" ");
    for (int j = 0; j < m; j++) {
        grid[i][j] =
Integer.parseInt(rowInput[j]);
    }
}
Solution ob = new Solution();

```

```
int ans = ob.countDistinctIslands(grid);  
    System.out.println(ans);  
    // Close the scanner  
    sc.close();  
}  
}
```


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 next line 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.

Input:-

5

4

1 0

2 1

3 2

4 3

Output:-

1

Test Case 2

Input:-

4

3

1 2

2 3

3 1

Output:-

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.



Explore | Expand | Enrich

PYTHON CODE

class Solution:

```
def canFinish(self, n, prerequisites):
```

```
    G = [[] for _ in range(n)]
```

```
    degree = [0] * n
```

```
    bfs = []
```

```
    for e in prerequisites:
```

```
        G[e[1]].append(e[0])
```

```
        degree[e[0]] += 1
```

```
    for i in range(n):
```

```
        if degree[i] == 0:
```

```
            bfs.append(i)
```

```
    for i in range(len(bfs)):
```

```
        for j in G[bfs[i]]:
```

```
            degree[j] -= 1
```

```
            if degree[j] == 0:
```

```
                bfs.append(j)
```

```
    return 1 if len(bfs) == n else 0
```

```
if __name__ == "__main__":
```

```
    N = int(input())
```

```
    M = int(input())
```

```
    prerequisites = []
```

```
    for _ in range(M):
```

```
        input_values = list(map(int,
```

```
input().split()))
```

```
        prerequisites.append(input_values)
```

```
    obj = Solution()
```

```
    print(obj.canFinish(N, prerequisites))
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.ArrayList;
import java.util.Scanner;
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<>();
        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);
        }
    }
}
return bfs.size() == n ? 1 : 0;
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        //System.out.print("Enter the number of courses (N): ");
        int N = Integer.parseInt(sc.nextLine());
        //System.out.print("Enter the size of prerequisites array (M): ");
```

```
int M = Integer.parseInt(sc.nextLine());
int[][] prerequisites = new int[M][2];
//System.out.println("Enter the prerequisites array elements (ai bi): ");
for (int i = 0; i < M; i++) {
    String[] input = sc.nextLine().split(" ");
    prerequisites[i][0] = Integer.parseInt(input[0]);
    prerequisites[i][1] = Integer.parseInt(input[1]);
}
Solution obj = new Solution();
System.out.println(obj.canFinish(N, prerequisites));
// Close the scanner
sc.close();
}
}
```

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 Case 1

Input:-

23

Output:-

ad ae af bd be bf cd ce cf

Test Case 2

Input:-

234

Output:-

adg adh adi aeg aeh aei afg afh afi bdg bdh bdi beg beh bei bfg bfh bfi cdg
cdh cdi ceg ceh cei cfg cfh cfi

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Main
{
private static final String[] mapping = {
    "",
    "", "abc","def",
    "ghi", "jkl", "mno",
    "pqrs", "tuv", "wxyz"
};
};
```

```
public static List<String>
letterCombinations(String digits) {
    List<String> result = new
ArrayList<>();
    if (digits == null ||
digits.length() == 0) {
        return result;
    }
    backtrack(result, "", digits, 0);
    return result;
}
```



```
private static void
backtrack(List<String> result, String
combination, String digits, int index) {
    if (index == digits.length()) {
        result.add(combination);
    } else {
        char digit =
digits.charAt(index);
        String letters =
mapping[digit - '0'];
        for (int i = 0; i <
letters.length(); i++) {
            backtrack(result,
combination + letters.charAt(i), digits,
index + 1);
        }
    }
}
```

```
public static void main (String[] args)
throws java.lang.Exception
{
    Scanner scanner = new
Scanner(System.in);
    String digits =
scanner.nextLine();
    scanner.close();
    List<String> combinations =
letterCombinations(digits);
    for (String combination :
combinations) {
        System.out.print(combination
+ " ");
    }
}
```

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.



Explore | Expand | Enrich

PYTHON CODE

```
def possible_words(s, ans, result):
    if len(s) == 0:
        result.add(ans)
        return
    key = keypad[int(s[0])]
    for char in key:
        possible_words(s[1:], ans + char, result)

if __name__ == "__main__":
    keypad = ["", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"]
    s = input()
    result = set()
    possible_words(s, "", result)
    for combination in sorted(result):
        print(combination, end=" ")
```



Explore | Expand | Enrich

JAVA CODE

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.

Test Cases:

Input:-

1 2 3

Output:-

1 2 3

1 3 2

2 1 3

2 3 1

3 2 1

3 1 2

Input:-

1 2 -2

Output:-

1 2 -2

1 -2 2

2 1 -2

2 -2 1

-2 2 1

-2 1 2

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.



Explore | Expand | Enrich

PYTHON CODE

```
def generate_permutations(nums, left, right):  
    if left == right:  
        # Base case: Print the current permutation  
        print_array(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]  
def print_array(nums):  
    print(' '.join(nums))  
if __name__ == "__main__":  
    nums = input().split()  
    n = len(nums)  
    generate_permutations(nums, 0, n - 1)
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner scanner = new
Scanner(System.in);

        String input = scanner.nextLine();
        String[] elements = input.split(" ");
        int n = elements.length;
        int[] nums = new int[n];
        for (int i = 0; i < n; i++) {
            try {
                nums[i] = Integer.parseInt(elements[i]);
            } catch (NumberFormatException e) {
                scanner.close();
            }
        }
    }
}
```

```
return;

        }

    }

    //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) {
        printArray(nums);
    } else {
```

```
for (int i = left; i <= right; i++) {
    swap(nums, left, i);
    generatePermutations(nums, left + 1,
right);
    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.



Explore | Expand | Enrich

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)
```




Explore | Expand | Enrich

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.

Test Cases

Input:-

3

Output:-

((()))

((()))

(()) ()

() (())

() () ()

Input:-

2

Output:-

(())

()()

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.



Explore | Expand | Enrich

PYTHON CODE

```
def generateParenthesis(n):
    def generateParenthesisHelper(openCount,
closeCount, current, result):
        if len(current) == 2 * n:
            result.append(current)
            return
        if openCount < n:
            generateParenthesisHelper(openCount + 1,
closeCount, current + "(", result)
        if closeCount < openCount:
            generateParenthesisHelper(openCount,
closeCount + 1, current + ")", result)
```

```
result = []
    generateParenthesisHelper(0, 0, "", result)
    return result

if __name__ == "__main__":
    n = int(input())
    combinations = generateParenthesis(n)

    for combination in combinations:
        print(combination)
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class Main {
    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);  
    String input = scanner.nextLine();  
    try {  
        int n = Integer.parseInt(input);  
        List<String> combinations = generateParenthesis(n);  
        //System.out.println("Combinations of well-formed parentheses for n = " + n + ":");  
        for (String combination : combinations) {  
            System.out.println(combination);  
        }  
    } catch (NumberFormatException e) {  
        System.out.println("Invalid input. Please enter a valid integer.");  
    }  
    scanner.close();  
}
```

LONGEST HAPPY PREFIX

A string is called a happy prefix if it 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.



Explore | Expand | Enrich

PYTHON CODE

```
def longest_happy_prefix(s):  
    n = len(s)  
    for i in range(n - 1, 0, -1):  
        if s[:i] == s[-i:]:  
            return s[:i]  
    return ""  
  
if __name__ == "__main__":  
    input_string = input()  
    print(longest_happy_prefix(input_string))
```



Explore | Expand | Enrich





Explore | Expand | Enrich

JAVA CODE

```
import java.util.Scanner;

public class Main {
    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) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        System.out.println(longestHappyPrefix(input));
    }
}
```

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.



Explore | Expand | Enrich

PYTHON CODE

```
class Solution:
    def longestSubstring(self, s: str) -> int:
        chars = {}
        left = 0
        right = 0
        res = 0
        while right < len(s):
            r = s[right]
            chars[r] = chars.get(r, 0) + 1
            while chars[r] > 1:
                l = s[left]
                chars[l] -= 1
                left += 1
```

```
        res = max(res, right - left + 1)
        right += 1
        return res

if __name__ == "__main__":
    s = input()
    ob = Solution()
    ans = ob.longestSubstring(s)
    print(ans)
```



Explore | Expand | Enrich

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’





Explore | Expand | Enrich

PYTHON CODE

```
def longest_palindrome(s: str) -> str:
    if len(s) < 1:
        return ""
    start, end = 0, 0
    for i in range(len(s)):
        len1 = expand_around_center(s, i, i)
        len2 = expand_around_center(s, i, i + 1)
        length = max(len1, len2)
        if length > end - start:
            start = i - (length - 1) // 2
            end = i + length // 2
    return s[start:end + 1]
```

```
def expand_around_center(s: str, left: int, right: int)
-> int:
    L, R = left, right
    while L >= 0 and R < len(s) and s[L] == s[R]:
        L -= 1
        R += 1
    return R - L - 1

if __name__ == "__main__":
    input_str = input()
    longest_palindrome_str =
    longest_palindrome(input_str)
    print(longest_palindrome_str)
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new
Scanner(System.in);
        String input = scanner.nextLine();
        String longestPalindrome =
longestPalindrome(input);
        System.out.println(longestPalindrome);
    }
    public static String longestPalindrome(String s)
{
    if (s == null || s.length() < 1)
return "";
    int start = 0, end = 0;
```

```
for (int i = 0; i < s.length(); i++) {
    int len1 = expandAroundCenter(s, i, i);
    int len2 = expandAroundCenter(s, i, i + 1);
    int len = Math.max(len1, len2);
    if (len > end - start) {
        start = i - (len - 1) / 2;
        end = i + len / 2;
    }
}
return s.substring(start, end + 1);
}

private static int expandAroundCenter(String
s, int left, int right) {
    int L = left, R = right;
```

```
while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {  
    L--;  
    R++;  
}  
return R - L - 1;  
}  
}
```




Explore | Expand | Enrich

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: "cbabc"

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

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."



Explore | Expand | Enrich

PYTHON CODE

```
def shortest_palindrome(s):
    if not s:
        return ""
    i = 0
    for j in range(len(s) - 1, -1, -1):
        if s[i] == s[j]:
            i += 1
    if i == len(s):
        return s
    suffix = s[i:]
    prefix = suffix[::-1]
    middle = shortest_palindrome(s[:i])
    return prefix + middle + suffix
def main():
    input_str = input()
    result = shortest_palindrome(input_str)
    print(result)
if __name__ == "__main__":
    main()
```



Explore | Expand | Enrich

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);
    String input = scanner.nextLine();
    String result = shortestPalindrome(input);
    System.out.println(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.



Explore | Expand | Enrich

PYTHON CODE

```
def main():
    nums_str = input().split()
    nums_divide_str = input().split()
    nums = list(map(int, nums_str))
    nums_divide = list(map(int, nums_divide_str))
    print(min_operations(nums, nums_divide))

def min_operations(nums, nums_divide):
    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 numsDivide
    min_op = 0
```



```
for num in nums:
    if num > smallest:
        min_op += 1
return str(min_op)
```

```
def gcd(a, b):
    while b > 0:
        tmp = a
        a = b
        b = tmp % b
    return a
```

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



Explore | Expand | Enrich

JAVA CODE

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] numsStr =
scanner.nextLine().split(" ");
        String[] numsDivideStr =
scanner.nextLine().split(" ");
        int[] nums = new int[numsStr.length];
        int[] numsDivide = new
int[numsDivideStr.length];
        for (int i = 0; i < numsStr.length; i++) {
            nums[i] = Integer.parseInt(numsStr[i]);
        }
        for (int i = 0; i < numsDivideStr.length;
i++) {
```

```
        numsDivide[i] = Integer.parseInt(numsDivideStr[i]);
    }
    System.out.println(minOperations(nums,
numsDivide));
}
    public static String 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 Integer.toString(minOp);  
}
```

```
private static int gcd(int a, int b) {  
    while (b > 0) {  
        int tmp = a;  
        a = b;  
        b = tmp % b;  
    }  
    return a;  
}
```

FIND PEAK INDEX 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
```



Explore | Expand | Enrich

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()
```



Explore | Expand | Enrich

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.



Explore | Expand | Enrich

PYTHON CODE


```
def main():
    test_cases = int(input("Enter the number
of test cases: "))
    while test_cases > 0:
        S = input("Enter the string: ")
        start = 0
        end = 0
        count = 0
        hash_array = [0] * 26
        while end < len(S):
            hash_array[ord(S[end]) -
ord('a')] += 1

            while (
                hash_array[ord('a') - ord('a')] > 0
and hash_array[ord('b') - ord('a')] > 0
and hash_array[ord('c') - ord('a')] > 0
            ):

```

```
        count += len(S) - end
        hash_array[ord(S[start]) - ord('a')] -= 1
        start += 1
        end += 1

        print(count)
        test_cases -= 1

if __name__ == "__main__":
    main()

```



Explore | Expand | Enrich

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;
    int[] hash = new int[26];
    while (end < S.length()) {
        hash[S.charAt(end) - 'a']++;
    }
}
```

```
while (hash['a' - 'a'] > 0 && hash['b' - 'a'] > 0 && hash['c' - 'a'] > 0) {  
    count += S.length() - end;  
    hash[S.charAt(start) - 'a']--;  
    start++;  
}  
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.

Test Cases:

Input:-

3 0 2 0 4

Output:-

7

Input:-

1 0 2 1 0 1

Output:-

2

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.



Explore | Expand | Enrich

PYTHON CODE

```
def trap_rain_water(heights):  
    n = len(heights)  
    if n <= 2:  
        return 0 # No water can be trapped with less than 3 bars  
  
    left_max = [0] * n  
    right_max = [0] * n  
  
    # Calculate the maximum height of bar to the left of each bar  
    left_max[0] = heights[0]  
    for i in range(1, n):  
        left_max[i] = max(left_max[i - 1], heights[i])  
  
    # Calculate the maximum height of bar to the right of each bar  
    right_max[n - 1] = heights[n - 1]  
    for i in range(n - 2, -1, -1):  
        right_max[i] = max(right_max[i + 1], heights[i])
```

```
water_trapped = 0
    # Calculate the amount of water trapped at each bar
    for i in range(n):
        min_height = min(left_max[i], right_max[i])
        water_trapped += min_height - heights[i]

    return water_trapped

if __name__ == "__main__":
    heights = list(map(int, input().split()))

    # Compute the amount of water trapped after raining
    trapped_water = trap_rain_water(heights)

    # Output the result
    print(trapped_water)
```



Explore | Expand | Enrich

JAVA CODE

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

```
public class Main {
```

```
    public static int trapRainWater(int[] heights) {
```

```
        int n = heights.length;
```

```
        if (n <= 2) {
```

```
            return 0;
```

```
        }
```

```
        int[] leftMax = new int[n];
```

```
        int[] rightMax = new int[n];
```

```
        leftMax[0] = heights[0];
```

```
        for (int i = 1; i < n; i++) {
```

```
            leftMax[i] = Math.max(leftMax[i - 1],
```

```
heights[i]);
```

```
        }
```

```
        rightMax[n - 1] = heights[n - 1];
```

```
        for (int i = n - 2; i >= 0; i--) {
```

```
            rightMax[i] = Math.max(rightMax[i + 1],
```

```
heights[i]);
```

```
        }
```

```
        int waterTrapped = 0;
```

```
        for (int i = 0; i < n; i++) {
```

```
int minHeight = Math.min(leftMax[i], rightMax[i]);
```

```
            waterTrapped += minHeight - heights[i];
```

```
        }
```

```
        return waterTrapped;
```

```
    }
```

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

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

```
String[] input = scanner.nextLine().split(" ");
```

```
        int n = input.length;
```

```
        int[] heights = new int[n];
```

```
for (int i = 0; i < n; i++) {  
    heights[i] = Integer.parseInt(input[i]);  
}  
  
// Compute the amount of water trapped after raining  
int trappedWater = trapRainWater(heights);  
  
// Output the result  
System.out.println(trappedWater);  
  
scanner.close();  
}  
}
```



Explore | Expand | Enrich

SPIRAL MATRIX

Print a given matrix in spiral form.

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

Test Cases:

Input :-

3

3

1 2 3

4 5 6

7 8 9

output:-

1 2 3 6 9 8 7 4 5

input:-

3

4

1 2 3 4

5 6 7 8

9 10 11 12

output:-

1 2 3 4 8 12 11 10 9 5 6 7

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.



Explore | Expand | Enrich

PYTHON CODE

```
def print_spiral(matrix, rows, cols):
    top, bottom, left, right = 0, rows - 1, 0, cols - 1

    while top <= bottom and left <= right:
        # Print top row
        for i in range(left, right + 1):
            print(matrix[top][i], end=" ")
        top += 1

        # Print right column
        for i in range(top, bottom + 1):
            print(matrix[i][right], end=" ")
        right -= 1

        # Print bottom row
        if top <= bottom:
            for i in range(right, left - 1, -1):
                print(matrix[bottom][i], end=" ")
            bottom -= 1
```

```
# Print left column
    if left <= right:
        for i in range(bottom, top - 1, -1):
            print(matrix[i][left], end=" ")
            left += 1

if __name__ == "__main__":
    rows = int(input())
    cols = int(input())
    matrix = []

    # Input matrix elements
    for _ in range(rows):
        row = list(map(int, input().split()))
        matrix.append(row)

    # Print spiral elements
    print_spiral(matrix, rows, cols)
```



Explore | Expand | Enrich

JAVA CODE


```
import java.util.Scanner;

public class Main {
    public static void
    printSpiral(int[][] matrix, int rows, int
    cols) {
        int top = 0, bottom = rows - 1,
        left = 0, right = cols - 1;

        while (top <= bottom && left <=
        right) {
            // Print top row
            for (int i = left; i <=
            right; i++) {
                System.out.print(matrix[top][i] + " ");
            }
            top++;
        }
    }
}
```

```
// Print right column
        for (int i = top; i <=
        bottom; i++) {
            System.out.print(matrix[i][right] + "
            ");
        }
        right--;

        // Print bottom row
        if (top <= bottom) {
            for (int i = right; i
            >= left; i--) {
                System.out.print(matrix[bottom][i] + "
                ");
            }
            bottom--;
        }
    }
}
```

```

        // Print left column
        if (left <= right) {
            for (int i = bottom; i >=
top; i--) {
                System.out.print(matrix[i][left] + " ");
            }
            left++;
        }
    }

    public static void main(String[]
args) {
        Scanner scanner = new
Scanner(System.in);
        int rows = scanner.nextInt();
        int cols = scanner.nextInt();

```

```

int[][] matrix = new int[rows][cols];
        // Input matrix elements
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols;
j++) {
                matrix[i][j] =
scanner.nextInt();
            }
        }
        scanner.close();

        // Print spiral elements
        printSpiral(matrix, rows,
cols);
    }
}

```

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.



Explore | Expand | Enrich

PYTHON CODE

```
def knapSack(N, W, wt, val):  
    # Initialize a 2D array to store the results of subproblems  
    dp = [[-1 for _ in range(W + 1)] for _ in range(N + 1)]  
    # Helper function to calculate the maximum value  
    def knapSackHelper(N, W):  
        # Base case: If there are no items or the knapsack has no capacity, the  
value is 0  
        if N == 0 or W == 0:  
            return 0  
        # If the result is already calculated, return it  
        if dp[N][W] != -1:  
            return dp[N][W]  
        # If the weight of the current item is less than or equal to the  
remaining capacity,  
        # we can either include it or exclude it  
        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]

# Call the helper function with the initial values of N and W
return knapSackHelper(N, W)


# Read the input values
n, W = map(int, input().split())
weights = list(map(int, input().split()))
values = list(map(int, input().split()))

# Call the knapSack function and print the result
result = knapSack(n, W, weights, values)
print(result)
```



Explore | Expand | Enrich

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.

Test Cases:

Input:-

10,22,9,33,21,50,41,60,80

Output:-

6

Input: -

3,10,2,1,20

Output: -

3

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.



Explore | Expand | Enrich

PYTHON CODE

```
def find_longest_increasing_subsequence(sequence):
    if not sequence:
        return 0

    n = len(sequence)
    dp = [1] * n

    for i in range(1, n):
        for j in range(i):
            if sequence[i] > sequence[j]:
                dp[i] = max(dp[i], dp[j] + 1)

    return max(dp)

if __name__ == "__main__":
    sequence = list(map(int, input().split(",")))
    longest_increasing_subsequence_length =
    find_longest_increasing_subsequence(sequence)
    print(longest_increasing_subsequence_length)
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new
Scanner(System.in);
        String[] input =
scanner.nextLine().split(",");
        int[] sequence = new int[input.length];
        for (int i = 0; i < input.length; i++) {
            sequence[i] = Integer.parseInt(input[i]);
        }
        int longestIncreasingSubsequenceLength =
findLongestIncreasingSubsequence(sequence);

System.out.println(longestIncreasingSubsequenceL
ength);
    }
    public static int
findLongestIncreasingSubsequence(int[] sequence)
{
```

```
    if (sequence == null || sequence.length == 0) {
        return 0;
    }
    int n = sequence.length;
    int[] dp = new int[n];
    Arrays.fill(dp, 1);
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (sequence[i] > sequence[j]) {
                dp[i] = Math.max(dp[i], dp[j] + 1);
            }
        }
    }
    int max = 0;
    for (int length : dp) {
        max = Math.max(max, length);
    }

    return max;
}
}
```

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]`.



Explore | Expand | Enrich

PYTHON CODE

```
def is_match(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]
word = input()
pattern = input()
if is_match(word, pattern):
    print("true")
else:
    print("false")
```



Explore | Expand | Enrich

JAVA CODE

```
import java.util.Scanner;

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) {
    Scanner scanner = new
Scanner(System.in);
    String word = scanner.nextLine();
    String pattern = scanner.nextLine();
    if (isMatch(word, pattern)) {
        System.out.println("true");
    } else {
        System.out.println("false");
    }
    scanner.close();
}
```



Explore | Expand | Enrich

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:

Input

1,2,3,1

Output

4

Input

2,7,9,3,1

Output

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.



Explore | Expand | Enrich

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[n - 1]
```

```
if __name__ == "__main__":  
    # Input from the user  
    nums = list(map(int, input().split(',')))  
  
    # Calculate the maximum amount of money that can be robbed  
    solution = Solution()  
    maxMoney = solution.rob(nums)  
  
    # Output the result  
    print(maxMoney)
```



Explore | Expand | Enrich

JAVA CODE


```
import java.util.Scanner;
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];
    }
}
```

```
public class Main {  
    public static void main(String[] args) {  
        // Input from the user  
        Scanner scanner = new Scanner(System.in);  
        String[] input = scanner.nextLine().split(",");  
        int n = input.length;  
        int[] nums = new int[n];  
        for (int i = 0; i < n; i++) {  
            nums[i] = Integer.parseInt(input[i]);  
        }  
        scanner.close();  
        Solution solution = new Solution();  
        int maxMoney = solution.rob(nums);  
        System.out.print(maxMoney);  
    }  
}
```

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[i1][j1]`.

If characters are different, `dp[i][j] = 1 + min(dp[i][j1], dp[i1][j], dp[i1][j1])`.

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.



Explore | Expand | Enrich

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))
```



Explore | Expand | Enrich

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