In [1]:

```
# Implement your function below.
def most_frequent(given_list):
    max_count = -1
    max_item = None
    count = {}
    for i in given_list:
        if i not in count:
            count[i] = 1
        else:
            count[i] += 1
        if count[i] > max_count:
            max_count = count[i]
            max_item = i
    return max_item
list5 = [0, -1, 10, 10, -1, 10, -1, -1, -1, 1]
print(most_frequent(list5))
```

```
-1
```

In [6]:

```
def common_elements(A,B):
    p1=0;
    p2=0;
    result=[]
    while (p1<len(A) and p2<len(B)):
        if (A[p1] == B[p2]):
            result.append(A[p1])
            p1 +=1 ;
            p2 +=1;
        elif  A[p1]>B[p2]:
            p2+=1;
        else:
            p1+=1;
    return result;


A=[1,3,4,6,7,9]
B=[1,2,4,5,9,10]
print(common_elements(A,B))
```

```
[1, 4, 9]
```

In [14]:

```
def is_rotation(A,B):
    if len(A)!=len(B):
        return False;
    temp = A[0]
    for i in range(len(B)):
        if temp==B[i]:
            ind = i;
            break;
    if not ind:
        return False;
    for i in range(len(A)):
        if A[i]!=B[(i+ind)%len(A)]:
            return False;
    return True;



A=[1,2,3,4,5,6,7,9]
B=[4,5,6,7,1,2,3,8]
print(is_rotation(A,B))
```

```
False
```

In [41]:

```
def non_repeating(c):
    for ind,i in enumerate(c):
        if i not in c[ind+1:] and i not in c[:ind]:
            return i
    return None
def non_repeating(c):
    charc_count={};
    for c in given_string:
        if c in char_count:
            char_count[c]+=1;
        else:
            char_count[c]=1;
    for c in given_string:
        charc_count[c]==1:
            return c;
    return None


a='aabcccefffb'
print(non_repeating(a))
```

```
-----------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
<ipython-input-41-0359d598dc8c> in <module>()
      8
      9 a='aabccceefffb'
---> 10 print(non_repeating(a))
     11
     12

<ipython-input-41-0359d598dc8c> in non_repeating(c)
      3         if i not in c[ind+1:] and i not in c[:ind]:
      4             return i
----> 5     return Null
      6
      7

NameError: name 'Null' is not defined
```

In [42]:

```
def away(s1,s2):
    if len(s1)>len(s2)+1 or len(s2)>len(s1)+1:
        return False;

    elif s1==s2:
        return True;
    elif len(s1)==len(s2):
        for i in range(len(s1)):
            if s1[i]==s2[i]:
                counter+=1;
            if counter>1:
                return False;
        return True;
    elif len(s1)>len(s2):
        Check(s1,S2);
    elif len(s2)>len(s1):
        Check(s2,S1);

def check(s3,s4):
    i=0;
    count_diff=0;
    while (i<len(s4)):
        if s3[i+count_diff] == s4[i]:
            i+=1;
        else:
            count_diif+=1;
            if count_diff>1:
                return False;
    return True;
```

In [44]:

```
def sweeper(bombs,num_rows,num_cols):

    res = [[0 for k in range(num_cols)] for l in range(num_rows)]
    for i in bombs:
        row_i=i[0];
        col_i=i[1];
        res[row_i][col_i]=-1
        for r in range(row_i-1,row_i+2):
            for c in range(col_i-1,col_i+2):
                if r>=0 and r<num_rows and c>=0 and c<num_cols and res[r
                    res[r][c]+=1;
    return res;

print(sweeper([[0,0],[0,1]],3,4))
```

```
[[-1, -1, 1, 0], [2, 2, 1, 0], [0, 0, 0, 0]]
```

In [45]:

```
# Implement your function below.
def click(field, num_rows, num_cols, given_i, given_j):
    import queue
    to_check = queue.Queue()
    if field[given_i][given_j] == 0:
        field[given_i][given_j] = -2
        to_check.put((given_i, given_j))
    else:
        return field
    while not to_check.empty():
        (current_i, current_j) = to_check.get()
        for i in range(current_i - 1, current_i + 2):
            for j in range(current_j - 1, current_j + 2):
                if (0 <= i < num_rows and 0 <= j < num_cols
                        and field[i][j] == 0):
                    field[i][j] = -2
                    to_check.put((i, j))
    return field


# NOTE: Feel free to use the following function for testing.
# It converts a 2-dimensional array (a list of lists) into
# an easy-to-read string format.
def to_string(given_array):
    list_rows = []
    for row in given_array:
        list_rows.append(str(row))
    return '[' + ',\n '.join(list_rows) + ']'


# NOTE: The following input values will be used for testing your solutio
field1 = [[0, 0, 0, 0, 0],
          [0, 1, 1, 1, 0],
          [0, 1, -1, 1, 0]]

# click(field1, 3, 5, 2, 2) should return:
# [[0, 0, 0, 0, 0],
#  [0, 1, 1, 1, 0],
#  [0, 1, -1, 1, 0]]

# click(field1, 3, 5, 1, 4) should return:
# [[-2, -2, -2, -2, -2],
#  [-2, 1, 1, 1, -2],
#  [-2, 1, -1, 1, -2]]


field2 = [[-1, 1, 0, 0],
          [1, 1, 0, 0],
          [0, 0, 1, 1],
          [0, 0, 1, -1]]

# click(field2, 4, 4, 0, 1) should return:
# [[-1, 1, 0, 0],
#  [1, 1, 0, 0],
#  [0, 0, 1, 1],
#  [0, 0, 1, -1]]
# click(field2, 4, 4, 1, 3) should return:
# [[-1, 1, -2, -2],
#  [1, 1, -2, -2],
#  [-2, -2, 1, 1],
#  [-2, -2, 1, -1]]
```

In [46]:

```
import copy


# Implement your function below.
# n = # rows = # columns in the given 2d array
def rotate(given_array, n):
    rotated = copy.deepcopy(given_array)
    for i in range(n):
        for j in range(n):
            (new_i, new_j) = rotate_sub(i, j, n)
            rotated[new_i][new_j] = given_array[i][j]
    return rotated


def rotate_sub(i, j, n):
    return j, n - 1 - i


# NOTE: Feel free to use the following function for testing.
# It converts a 2-dimensional array (a list of lists) into
# an easy-to-read string format.
def to_string(given_array):
    list_rows = []
    for row in given_array:
        list_rows.append(str(row))
    return '[' + ',\n '.join(list_rows) + ']'


# NOTE: The following input values will be used for testing your solutio
a1 = [[1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]]
# rotate(a1, 3) should return:
# [[7, 4, 1],
#  [8, 5, 2],
#  [9, 6, 3]]

a2 = [[1, 2, 3, 4],
      [5, 6, 7, 8],
      [9, 10, 11, 12],
      [13, 14, 15, 16]]
# rotate(a2, 4) should return:
# [[13, 9, 5, 1],
#  [14, 10, 6, 2],
#  [15, 11, 7, 3],
#  [16, 12, 8, 4]]
```

In [47]:

```
import math


# Implement your function below.
# n = # rows = # columns in the given 2d array
def rotate(given_array, n):
    for i in range(math.ceil(n/2)):
        for j in range(math.floor(n/2)):
            tmp = [-1] * 4
            (current_i, current_j) = (i, j)
            for k in range(4):
                tmp[k] = given_array[current_i][current_j]
                (current_i, current_j) = rotate_sub(current_i, current_j
            for k in range(4):
                given_array[current_i][current_j] = tmp[(k - 1) % 4]
                (current_i, current_j) = rotate_sub(current_i, current_j
    return given_array


def rotate_sub(i, j, n):
    return j, n - 1 - i


# NOTE: Feel free to use the following function for testing.
# It converts a 2-dimensional array (a list of lists) into
# an easy-to-read string format.
def to_string(given_array):
    list_rows = []
    for row in given_array:
        list_rows.append(str(row))
    return '[' + ',\n '.join(list_rows) + ']'


# NOTE: The following input values will be used for testing your solutio
a1 = [[1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]]
# rotate(a1, 3) should return:
# [[7, 4, 1],
#  [8, 5, 2],
#  [9, 6, 3]]

a2 = [[1, 2, 3, 4],
      [5, 6, 7, 8],
      [9, 10, 11, 12],
      [13, 14, 15, 16]]
# rotate(a2, 4) should return:
# [[13, 9, 5, 1],
#  [14, 10, 6, 2],
#  [15, 11, 7, 3],
#  [16, 12, 8, 4]]
```

In [48]:

```
# Use this class to create linked lists.
class Node:
    def __init__(self, value, child=None):
        self.value = value
        self.child = child

    # The string representation of this node.
    # Will be used for testing.
    def __str__(self):
        return str(self.value)


# Implement your function below.
def nth_from_last(head, n):
    left = head
    right = head
    for i in range(n):
        if right == None:
            return None
        right = right.child
    while right:
        right = right.child
        left = left.child
    return left


# NOTE: Feel free to use the following function for testing.
# It converts the given linked list into an easy-to-read string format.
# Example: 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> (None)
def linked_list_to_string(head):
    current = head
    str_list = []
    while current:
        str_list.append(str(current.value))
        current = current.child
    str_list.append('(None)')
    return ' -> '.join(str_list)


# NOTE: The following input values will be used for testing your solutio
current = Node(1)
for i in range(2, 8):
    current = Node(i, current)
head = current
# head = 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> (None)

current2 = Node(4)
for i in range(3, 0, -1):
    current2 = Node(i, current2)
head2 = current2
# head2 = 1 -> 2 -> 3 -> 4 -> (None)


# nth_from_last(head, 1) should return 1.
# nth_from_last(head, 5) should return 5.
# nth_from_last(head2, 2) should return 3.
# nth_from_last(head2, 4) should return 1.
# nth_from_last(head2, 5) should return None.
# nth_from_last(None, 1) should return None.
```

In [49]:

```python
# Use this class to create binary trees.
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def __str__(self):
        return str(self.value)


# Implement your function below.
def is_bst(node, lower_lim=None, upper_lim=None):
    if lower_lim and node.value < lower_lim:
        return False
    if upper_lim and upper_lim < node.value:
        return False

    is_left_bst = True
    is_right_bst = True
    if node.left:
        is_left_bst = is_bst(node.left, lower_lim, node.value)
    if is_left_bst and node.right:
        is_right_bst = is_bst(node.right, node.value, upper_lim)
    return is_left_bst and is_right_bst


# A function for creating a tree.
# Input:
# - mapping: a node-to-node mapping that shows how the tree should be co
# - head_value: the value that will be used for the head ndoe
# Output:
# - The head node of the resulting tree
def create_tree(mapping, head_value):
    head = Node(head_value)
    nodes = {head_value: head}
    for key, value in mapping.items():
        nodes[value[0]] = Node(value[0])
        nodes[value[1]] = Node(value[1])
    for key, value in mapping.items():
        nodes[key].left = nodes[value[0]]
        nodes[key].right = nodes[value[1]]
    return head


# NOTE: The following values will be used for testing your solution.

# The mapping we're going to use for constructing a tree.
# {0: [1, 2]} means that 0's left child is 1, and its right
# child is 2.
mapping1 = {0: [1, 2], 1: [3, 4], 2: [5, 6]}
mapping2 = {3: [1, 4], 1: [0, 2], 4: [5, 6]}
mapping3 = {3: [1, 5], 1: [0, 2], 5: [4, 6]}
mapping4 = {3: [1, 5], 1: [0, 4]}
head1 = create_tree(mapping1, 0)
# This tree is:
```

```
#  head1 = 0
#        /   \
#       1     2
#      /\    / \
#     3  4  5   6
head2 = create_tree(mapping2, 3)
# This tree is:
#  head2 = 3
#        /   \
#       1     4
#      /\    / \
#     0  2  5   6
head3 = create_tree(mapping3, 3)
# This tree is:
#  head3 = 3
#        /   \
#       1     5
#      /\    / \
#     0  2  4   6
head4 = create_tree(mapping4, 3)
# This tree is:
#  head4 = 3
#        /   \
#       1     5
#      /\
#     0  4

# is_bst(head1) should return False
# is_bst(head2) should return False
# is_bst(head3) should return True
# is_bst(head4) should return False
```

In [ ]:

```
# Use this class to create binary trees.
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

    def __str__(self):
        return str(self.value)

    # Overriding the equality operator.
    # This will be used for testing your solution.
    def __eq__(self, other):
        if type(other) is type(self):
            return self.value == other.value
        return False


# Implement your function below.
def lca(root, j, k):
    path_to_j = path_to_x(root, j)
    path_to_k = path_to_x(root, k)

    lca_to_return = None
    while path_to_j and path_to_k:
        j_pop = path_to_j.pop()
        k_pop = path_to_k.pop()
        if j_pop is k_pop:
            lca_to_return = j_pop
        else:
            break
    return lca_to_return


def path_to_x(root, x):
    if root is None:
        return None
    if root.value == x:
        return [root]
    left_path = path_to_x(root.left, x)
    if left_path is not None:
        left_path.append(root)
        return left_path
    right_path = path_to_x(root.right, x)
    if right_path is not None:
        right_path.append(root)
        return right_path
    return None

# A function for creating a tree.
# Input:
# - mapping: a node-to-node mapping that shows how the tree should be co
# - head_value: the value that will be used for the head ndoe
# Output:
# - The head node of the resulting tree
def create_tree(mapping, head_value):
    head = Node(head_value)
    nodes = {head_value: head}
    for key, value in mapping.items():
        nodes[value[0]] = Node(value[0])
        nodes[value[1]] = Node(value[1])
    for key, value in mapping.items():
        nodes[key].left = nodes[value[0]]
        nodes[key].right = nodes[value[1]]
    return head


# NOTE: The following values will be used for testing your solution.

# The mapping we're going to use for constructing a tree.
# {0: [1, 2]} means that 0's left child is 1, and its right
# child is 2.
mapping1 = {0: [1, 2], 1: [3, 4], 2: [5, 6]}
head1 = create_tree(mapping1, 0)
# This tree is:
# head1 = 0
#        / \
#       1   2
#      /\   /\
#     3  4 5  6


mapping2 = {5: [1, 4], 1: [3, 8], 4: [9, 2], 3: [6, 7]}
head2 = create_tree(mapping2, 5)
# This tree is:
#  head2 = 5
#        /   \
#       1     4
#      /\    / \
#     3  8  9  2
#    /\
#   6  7


# lca(head1, 1, 5) should return 0
# lca(head1, 3, 1) should return 1
# lca(head1, 1, 4) should return 1
# lca(head1, 0, 5) should return 0
# lca(head2, 4, 7) should return 5
# lca(head2, 3, 3) should return 3
# lca(head2, 8, 7) should return 1
# lca(head2, 3, 0) should return None (0 does not exist in the tree)
```