In [4]:
```
# write a function that will return the number of times A must be
#stated such that B is a substring of the repeated A. If B can never
# be a substring, return -1.
def repeatedStringMatch(A, B):
    q = (len(B) - 1) // len(A) + 1
    print(q)
    for i in range(2):
        if B in A * (q+i):
            return q+i
    return -1

a='abcd';
b='cdabcdabcdab';
```

In [5]:
```
print(repeatedStringMatch(a,b))
```

```
3
4
```

In [7]:
```
def minDepth(root):
    if not root:
        return 0;
    if not root.left and not root.right:
        return 1
    elif not root.left:
        return minDepth(root.right)+1;
    elif not root.right:
        return minDepth(root.left)+1;
    else:
        return min(minDepth(root.left),minDepth(root.right))+1;
```

In [17]:
```
def maxDepth(root):
    if not root:
        return 0;
    if not root.left and not root.right:
        return 1
    elif not root.left:
        return maxDepth(root.right)+1;
    elif not root.right:
        return maxDepth(root.left)+1;
    else:
        return max(maxDepth(root.left),maxDepth(root.right))+1;
```

In [ ]:

In [9]:
```
L = [('def', 10), ('abc', 15), ('il', 12),('ghi', 9),('p', 20),('tyi', 8
b=[]
for i in L:
    if (i[1])>=10:
        b.append(i[0]);
print(b)
```

```
['def', 'abc', 'il', 'p']
```

In [14]:
```
def find_kth(k, arr):
    if k == 1:
        return max(arr)
    m = max(arr)
    new_arr = list(filter(lambda a: a != m, arr))
    print(new_arr)
    return(find_kth(k-1, new_arr))

l=[2,3,-1,5,4,-2,20,30,-1,80,18,22];
print(find_kth(5,l))
```

```
[2, 3, -1, 5, 4, -2, 20, 30, -1, 18, 22]
[2, 3, -1, 5, 4, -2, 20, -1, 18, 22]
[2, 3, -1, 5, 4, -2, 20, -1, 18]
[2, 3, -1, 5, 4, -2, -1, 18]
18
```

In [ ]:
```
def minJumps(arr, n):
    jumps = [0 for i in range(n)]


    jumps[0] = 0

    # Find the minimum number of
    # jumps to reach arr[i] from
    # arr[0] and assign this
    # value to jumps[i]
    for i in range(1, n):
        jumps[i] = float('inf')
        for j in range(i):
            if (i <= j + arr[j]) and (jumps[j] != float('inf')):
                jumps[i] = min(jumps[i], jumps[j] + 1)
                break
    return jumps

a=[2,3,1,1,2,4,2,0,1,1];
print(minJumps(a,))
```

In [12]:
```
def occurrence(a,b):
    result=0;
    for i in range(len(a)):
        if a[i]==b:
            result+=1;
    return result;
a=[1, 2, 2, 2, 2, 3, 4, 7 ,8 ,8];
print(occurrence(a,8))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call l
ast)
<ipython-input-12-8dbce4831e6c> in <module>()
     18
     19 a=[2,3,1,1,2,4,2,0,1,1];
---> 20 print(minJumps(a,))

TypeError: minJumps() missing 1 required positional argument: 'n'
```

In [20]:
```
def histo(a):


    b=a
    enter=1;
    finalResult=[]
    finalValue=[]
    while (enter):
        res = 0;
        temp=[];
        l=range(len(b));
        num = b[l[0]];
        for ind,val in enumerate(b):
            if val==num:
                res+=1;
            else:
                temp.append(b[ind]);
        finalResult.append(res)
        finalValue.append(num)
        b=temp;
        if not b:
            enter=0;
    dictionary = dict(zip(finalValue, finalResult))
    return dictionary

a=[1, 2, 2, 2, 2, 3, 4, 7 ,8 ,8];
print(histo(a))
```

```
{1: 1, 2: 4, 3: 1, 4: 1, 7: 1, 8: 2}
```

In [22]:
```

```

```
{0: 1, -1: 5, 10: 3, 1: 1}
```

In [ ]:
```
def powerOfFive(a):
    res=0;
    while(a>=5):
        if (a%5==0):
            res+=1;
            a=a//5;
        else:
            return "No";
    return res;

print(powerOfFive(625))
```

In [16]:
```
def removeSpace(a):
    res='';
    temp=0;
    for i in range(len(a)):
        if a[i]!=' ':
            if temp==1:
                res=res+' ';
                temp=0;
            res=res+a[i];
        else:
            temp=1;
    return res
a='I   live   on     earth  ';
print(removeSpace(a))
```

```
{1: 1, 2: 4, 3: 1, 4: 1, 7: 1, 8: 2}
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
def toBit(n):
    l=[];

    while (n>0):
        print(n)
        l.append(n%2);
        n=n//2;
    l.append(n%2)
    return l

print(toBit(2))
```

In [ ]:
```
def countBit(n):
    l=[];
    count=0;
    while (n>1):

        l.append(n%2);
        if (n%2)==1:
            count+=1;
        n=n//2;
    if (n==1):
        count+=1;
    return count

#print(countBit(1))


def bitDifference(a):
    res=0;
    for i in range(len(a)):
        #print(a[i])
        for j in range(len(a)):
            #if (a[i]!=a[j]):
            res=res+(abs(countBit(a[i])-countBit(a[j])));
            print(abs(countBit(a[i])-countBit(a[j])))

    return res;
a=[1,3,5]
print(bitDifference(a))
```

In [ ]:

In [ ]:

In [ ]:

In [18]:
```
def medianStream(a):
    sortedArray=a;
    #sortedArray = sortArray(a);
    size = len(sortedArray);
    if size%2==0:

        return sum(sortedArray[size//2-1:size//2+1])/2;
    else:
        return (sortedArray[size//2])
a=[1,2,3,4,5,6]
print(medianStream(a))
```

```
3.5
```

In [ ]:
```
def total_length(node):
    if not node:
        return 0;
    lengthLeft = total_length(node.left)
    lengthRight = total_length(node.right)
    if node.left and node.left.value == node.value:
        left_arrow == lengthLeft + 1;
    if node.right and node.right.value == node.value:
        right_arrow == lengthRight + 1;
    return max(right_arrow,left_arrow);
```