LINKED LISTS -2
&
RECURSION/DFS

Question: A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null. Return a deep copy of the list.

YOU HAVE 15 MINUTES

LOOK UP :

Link: https://leetcode.com/problems/copy-list-with-random-pointer/

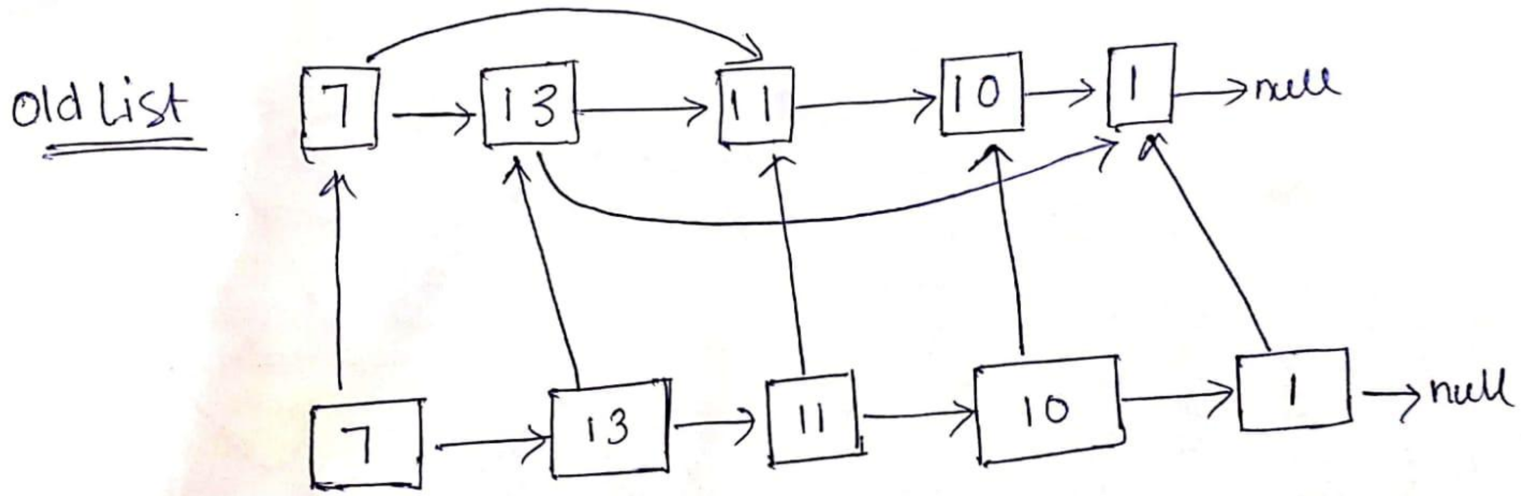
Old List

$$\begin{tikzcd}
\boxed{7} \arrow[r] \arrow[rr, bend left] & \boxed{13} \arrow[r] \arrow[rrr, bend right] & \boxed{11} \arrow[r] & \boxed{10} \arrow[r] & \boxed{1} \arrow[r] & \text{null}
\end{tikzcd}$$

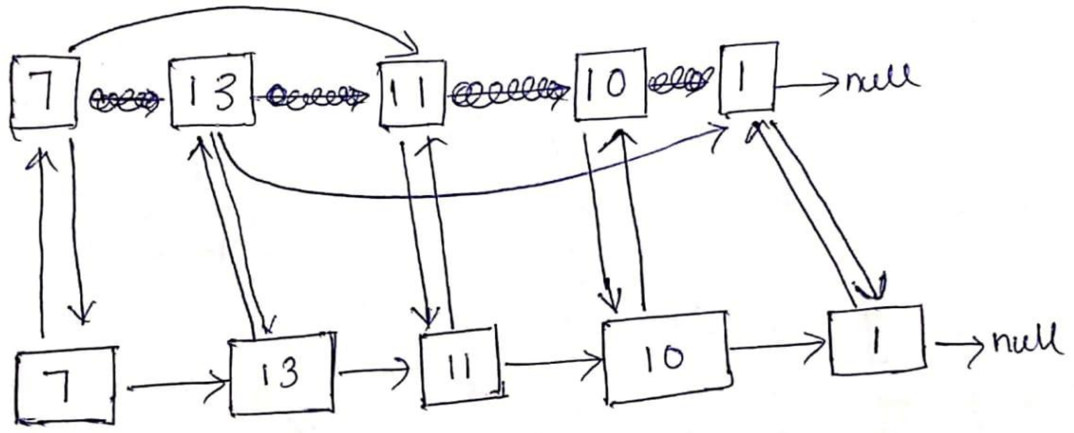
Old List

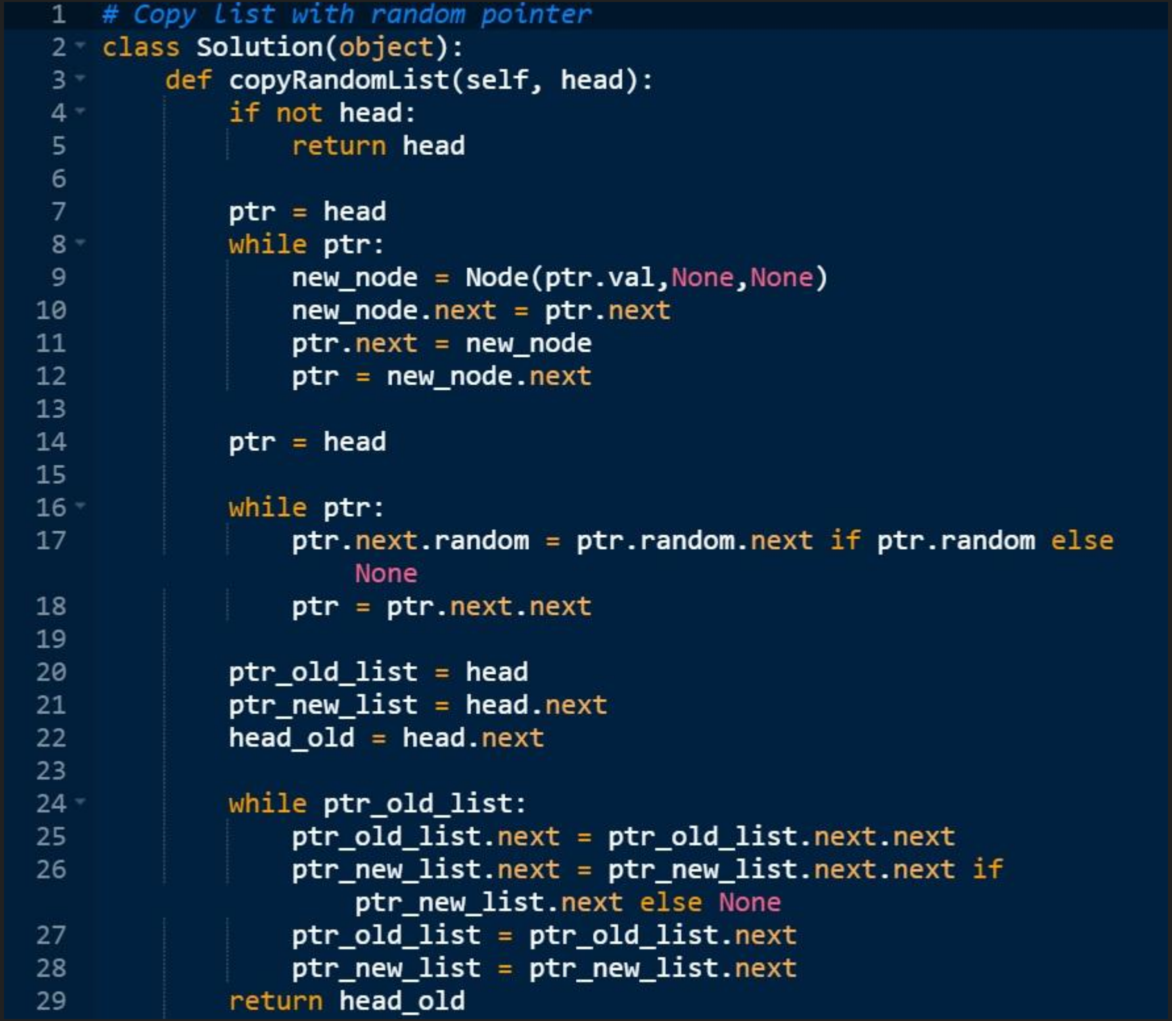
$$\begin{tikzcd}
\boxed{7} \arrow[r] \arrow[rr, bend left] & \boxed{13} \arrow[r] \arrow[rrr, bend right] & \boxed{11} \arrow[r] & \boxed{10} \arrow[r] & \boxed{1} \arrow[r] & \text{null}
\end{tikzcd}$$

$$\begin{tikzcd}
\boxed{7} \arrow[r] & \boxed{13} \arrow[r] & \boxed{11} \arrow[r] & \boxed{10} \arrow[r] & \boxed{1} \arrow[r] & \text{null}
\end{tikzcd}$$


Old List
7
13
11
10
1
null
7
13
11
10
1
null

Old List



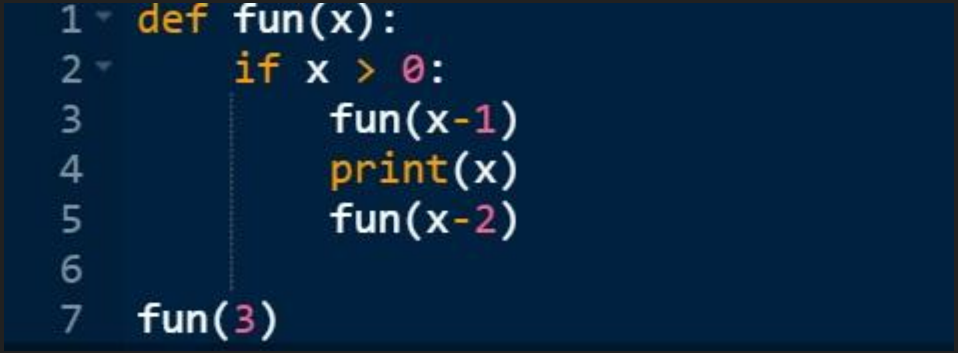


```python
# Copy list with random pointer
class Solution(object):
    def copyRandomList(self, head):
        if not head:
            return head

        ptr = head
        while ptr:
            new_node = Node(ptr.val,None,None)
            new_node.next = ptr.next
            ptr.next = new_node
            ptr = new_node.next

        ptr = head

        while ptr:
            ptr.next.random = ptr.random.next if ptr.random else
                None
            ptr = ptr.next.next

        ptr_old_list = head
        ptr_new_list = head.next
        head_old = head.next

        while ptr_old_list:
            ptr_old_list.next = ptr_old_list.next.next
            ptr_new_list.next = ptr_new_list.next.next if
                ptr_new_list.next else None
            ptr_old_list = ptr_old_list.next
            ptr_new_list = ptr_new_list.next
        return head_old
```

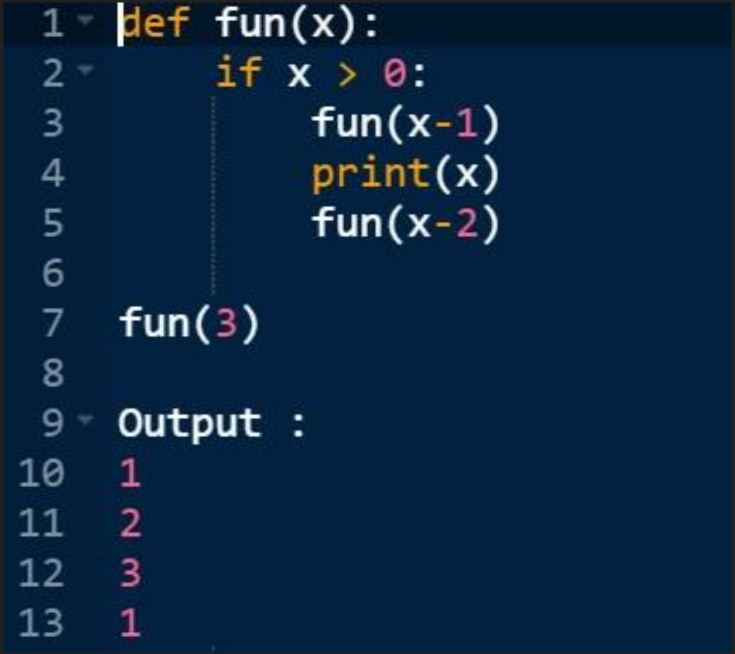
Time Complexity - O(N)
Space Complexity - O(1)

RECURSION

PREDICT THE OUTPUT

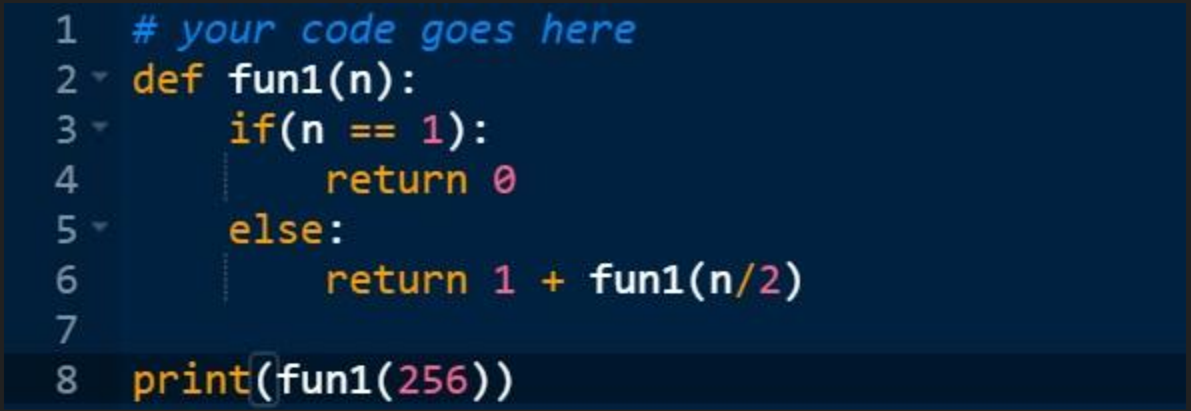


```python
def fun(x):
    if x > 0:
        fun(x-1)
        print(x)
        fun(x-2)

fun(3)
```

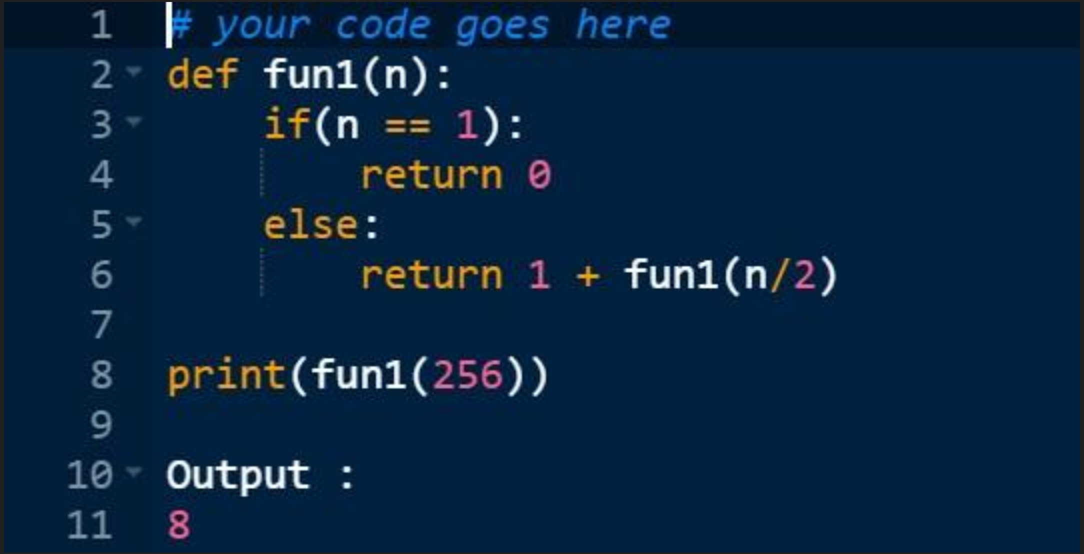


```
def fun(x):
    if x > 0:
        fun(x-1)
        print(x)
        fun(x-2)

fun(3)

Output :
1
2
3
1
```

PREDICT THE OUTPUT



```python
# your code goes here
def fun1(n):
    if(n == 1):
        return 0
    else:
        return 1 + fun1(n/2)

print(fun1(256))
```



```
# your code goes here
def fun1(n):
    if(n == 1):
        return 0
    else:
        return 1 + fun1(n/2)

print(fun1(256))

Output :
8
```

Question: Flood fill problem.

YOU HAVE 15 MINUTES

https://leetcode.com/problems/flood-fill/

1	1	1
1	1	0
0	0	0

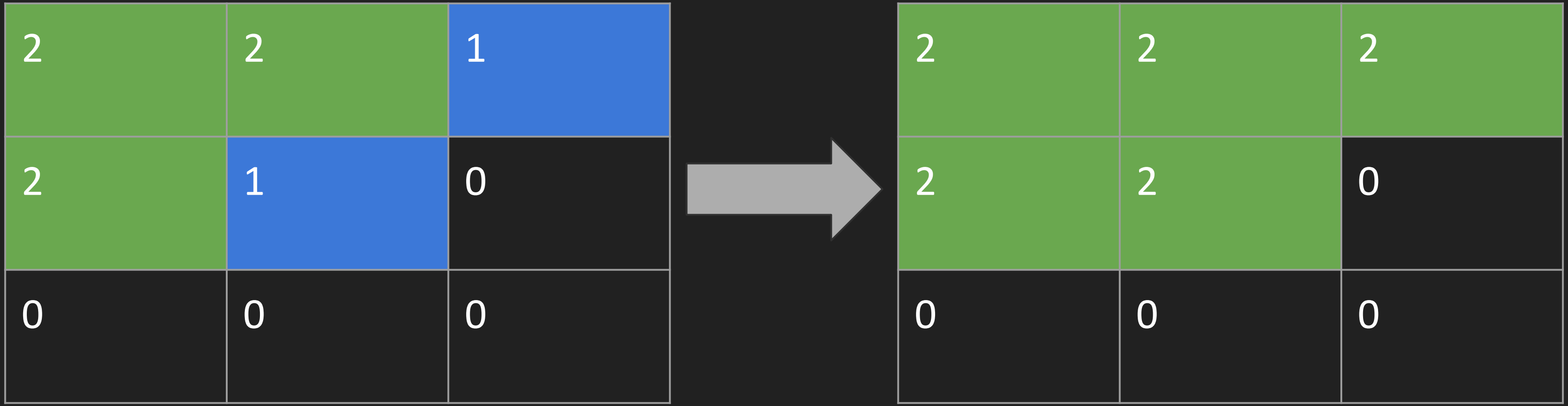
$\Rightarrow$

2	2	2
2	2	0
0	0	0

1	1	1
1	1	0
0	0	0

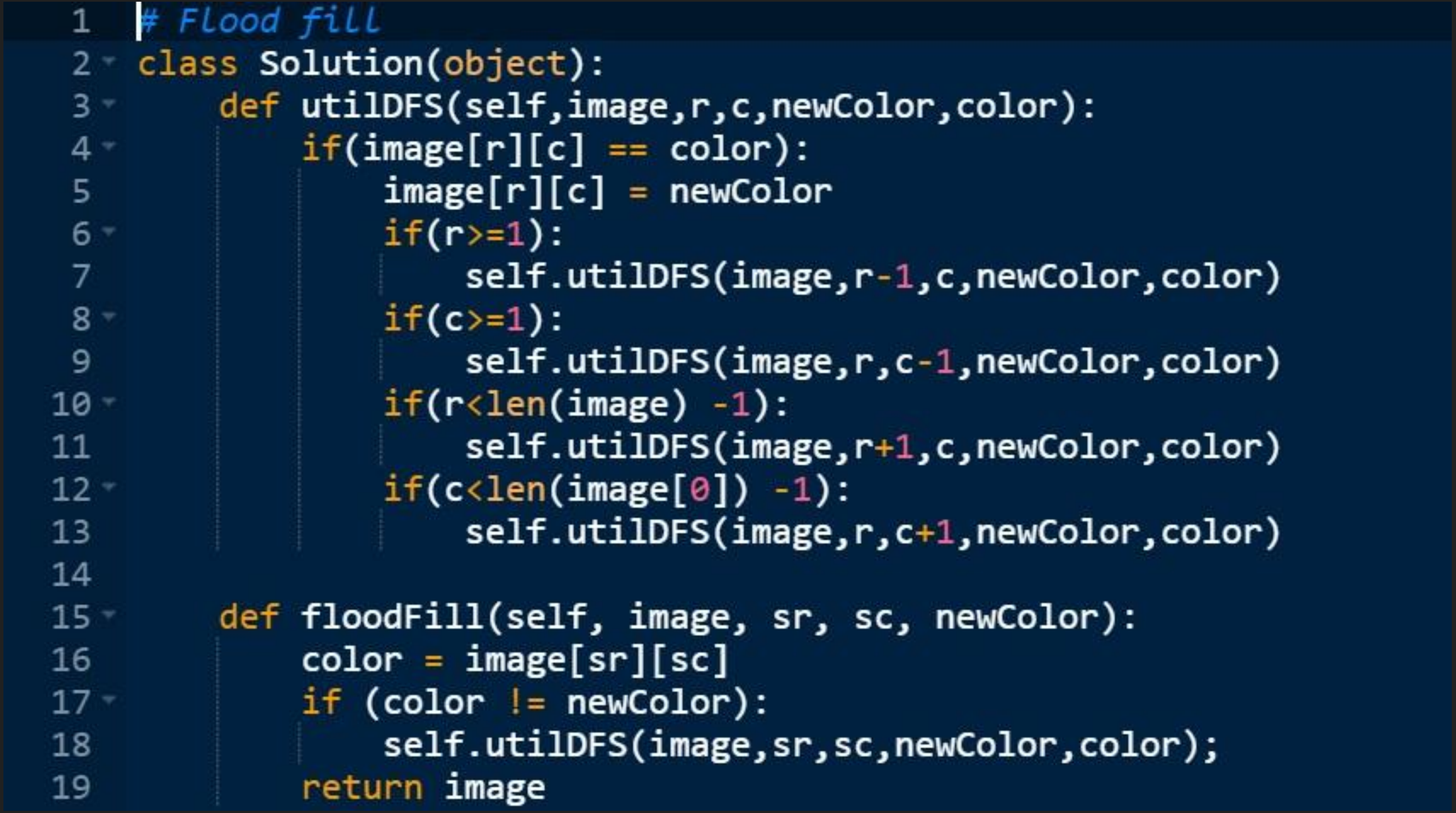
$\Rightarrow$

2	1	1
1	1	0
0	0	0



2	2	1
2	1	0
0	0	0

2	2	2
2	2	0
0	0	0



```python
# Flood fill
class Solution(object):
    def utilDFS(self,image,r,c,newColor,color):
        if(image[r][c] == color):
            image[r][c] = newColor
            if(r>=1):
                self.utilDFS(image,r-1,c,newColor,color)
            if(c>=1):
                self.utilDFS(image,r,c-1,newColor,color)
            if(r<len(image) -1):
                self.utilDFS(image,r+1,c,newColor,color)
            if(c<len(image[0]) -1):
                self.utilDFS(image,r,c+1,newColor,color)

    def floodFill(self, image, sr, sc, newColor):
        color = image[sr][sc]
        if (color != newColor):
            self.utilDFS(image,sr,sc,newColor,color);
        return image
```

TIME COMPLEXITY - O(N), SPACE COMPLEXITY - O(N) [Call stack for DFS]

Question: Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

YOU HAVE 15 MINUTES

Link: https://leetcode.com/problems/number-of-islands/

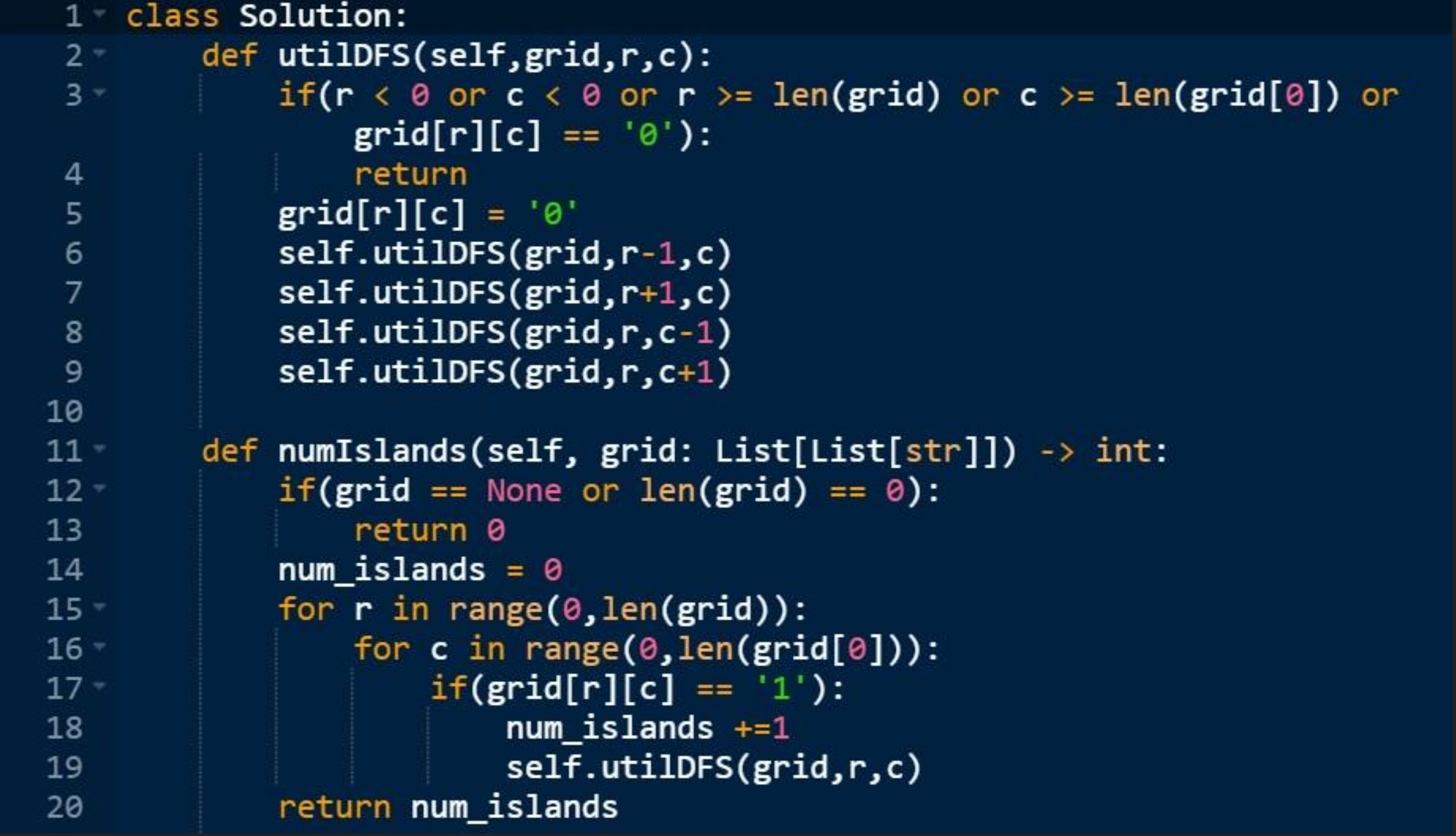
1	1	1	0	0	1
1	1	1	0	0	0
0	0	0	0	0	0
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	1	0	1

1	1	1	0	0	1
1	1	1	0	0	0
0	0	0	0	0	0
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	1	0	1

Use Flood Fill Algorithm to convert all islands to 0 (ocean), whenever encounter 1, update the island count.

1	1	1	0	0	1
1	1	1	0	0	0
0	0	0	0	0	0
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	1	0	1

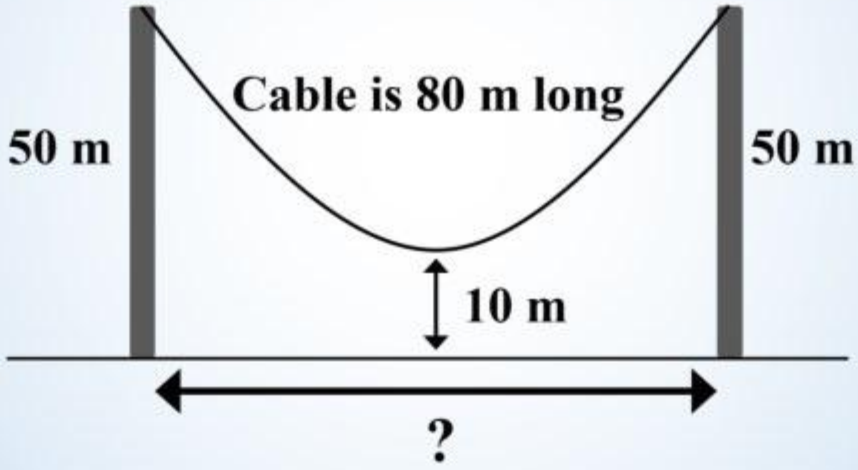
1	1	1	0	0	1
1	1	1	0	0	0
0	0	0	0	0	0
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	1	0	1



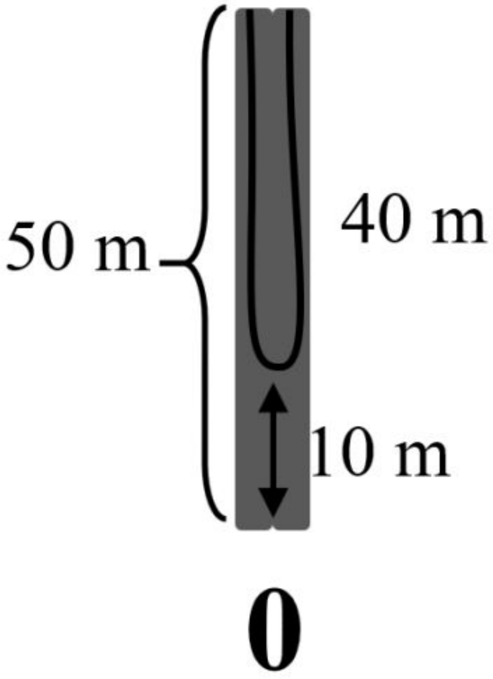
```python
class Solution:
    def utilDFS(self,grid,r,c):
        if(r < 0 or c < 0 or r >= len(grid) or c >= len(grid[0]) or
            grid[r][c] == '0'):
            return
        grid[r][c] = '0'
        self.utilDFS(grid,r-1,c)
        self.utilDFS(grid,r+1,c)
        self.utilDFS(grid,r,c-1)
        self.utilDFS(grid,r,c+1)

    def numIslands(self, grid: List[List[str]]) -> int:
        if(grid == None or len(grid) == 0):
            return 0
        num_islands = 0
        for r in range(0,len(grid)):
            for c in range(0,len(grid[0])):
                if(grid[r][c] == '1'):
                    num_islands +=1
                    self.utilDFS(grid,r,c)
        return num_islands
```

TIME COMPLEXITY - O(M*N) and SPACE COMPLEXITY - O(M*N)

Amazon Interview Question: How Far Apart Are The Poles?



(b) 10 m above ground



50 m
40 m
10 m
0