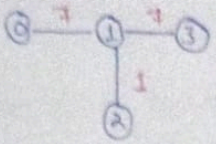
Day_11: DSA

Day-11 : leetcode: 3108 Minimum cost walk in weighted graph

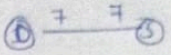
• Union find → Method: Tc → O(E+Q) DFS → O(V+E+Q)



query: [[0,3],[3,4]]

→ It is walk: in this may visit the same edge or vertex more than once

→ Cost = Bitwise AND



7 & 7 => 7

```
class UnionFind:
    def __init__(self, n: int):
        self.id = list(range(n))
        self.rank = [0]*n
        # 2^17 - 1 is the minimum number in the form 2^i - 1 > 10^5
        self.weight = [((1<<17)-1)]*n

    def unionByRank(self, u: int, v: int, w: int) -> None:
        i = self._find(u)
        j = self._find(v)
        newWeight = self.weight[i] & self.weight[j] & w
        self.weight[i] = newWeight
        self.weight[j] = newWeight
        if i == j:
            return
        if self.rank[i] < self.rank[j]:
            self.id[i] = j
        elif self.rank[i] > self.rank[j]:
            self.id[j] = i
        else:
            self.id[j] = i
            self.rank[i] += 1

    def getMinCost(self, u: int, v: int) -> int:
        if u == v:
            return 0
        i = self._find(u)
        j = self._find(v)
        return self.weight[i] if i == j else -1
```

SDE

- Program to create generate Pascal's Triangle

→ This problem has 3 variations.

Variation 1: Given row number and column number. Print the element at position (r,c) in Pas' triangle

Variation 2: Given the row number. Print the n^{th} row of Pascal's Triangle

Variation 3: Given the row number. Print the first n rows of Pascal's Triangle

Eg.

```
      1                  1
    1   1              1   1
  1   2   1          1   2   1
1   3   3   1      1   3   3   1
```

- Variation I: We have formula to find out the element $^{r-1}C_{c-1}$

Code:

```
def nCr(n,r):
    res = 1
    # Calculating nCr:
    for i in range(r):
        res = res * (n-i)
        res = res // (i+1)
    return res

def pascalTriangle(r,c):
    element = nCr(r-1, c-1)
    return element
```

$nCr = \frac{n!}{(r! * n-r)!}$

T.C. O(c)

Variation II:

```
def pascalTriangle(n)
    # printing the row
    for c in range(1, n+1)
        print(nCr(n-1, c-1), end=" ")
    print()
```

T.C. O(N*r)

optimal:

```
def pascalTriangle(n):
    ans = 1
    print(ans, end=" ") # printing 1st Element
    # printing the rest of the part.
```

```
for i in range(1,n):
    ans = ans * (n-i)
    ans = ans // i
    print(ans, end=" ")
print()
```

T.C: O(N)

Current element = prevElement * (rowNumber - colIndex) / colIndex

Naive Solution

Variation: II

```
def nCr(n,r):
    res=1
    # Calculating nCr:
    for i in range(r):
        res = res * (n-i)
        res = res // (i+1)
    return int(res)

def pascalTriangle(n)
    ans=[]
    # Store the entire pascal's triangle
    for row in range(1,n+1):
        tempLst=[]
        for col in range(1,row+1):
            tempLst.append(nCr(row-1,col-1))
        ans.append(tempLst)
    return ans
```

T.C: O(n*n*r) ~ O(n^3)

Optimal Sol:

```
def generateRow(row):
    ans=1
    ansRow=[1]  # inserting the 1st element
    # Calculating rest of rows
    for col in range(1,row):
        ans *= (row-col)
        ans //= col
        ansRow.append(ans)
    return ansRow
```

T.C: O(n^2)

```
def pascalTriangle(n)
    ans=[]
    for row in range(1,n+1):
        ans.append(generateRow(row))
    return ans
```