Rushabh Ashok Dharia

Written Assignment 3

1 Binomial Coefficients
Input :- n, k
Output - C[n][k]
C is the array to store the computed Binomial Coefficients.

~~How C[n][k] =~~

Now $C(n,k) = C(n-1, k-1) + C(n-1, k)$

$C(n,0) = C(n,n) = 1$ ← Stopping Condition

```
procedure BinomialCoef(n, k):
    for i from 0 to n
        for j from 0 to k
            if j == 0 or j == i:
                C[i][j] = 1
            else:
                C[i][j] = C[i-1][j-1] + C[i-1][j]
    return C[n][k]
```

The Time Complexity of the above algorithm is O(nk) since inner loop runs 'k' times and outer loop runs 'n' times

2 Algorithm to count the number of ways in which the road can be paved

Input = n = length of road in meters

Output = number of ways in which the road can be paved

roadparing[n] = roadparing[n-2] + roadparing[n-3] + roadparing[n-5]

Stopping Condition → roadparing[0] = 1 & roadparing[1] = 0

[Sanity Check] & if n < 0 then roadparing[n] = 0

```
def paringRoad(n):
    if n<0: return 0
    roadparing[0] = 1
    roadparing[1] = 0
    for i from 2 to n:
        sum := 0
        if i is greater than or equal to 2:
            sum += roadparing[i-2]
        if i is greater than or equal to 3:
            sum += roadparing[i-3]
        if i is greater than or equal to 5
            sum += roadparing[i-5]
        roadparing[i] = sum
    return roadparing[n]
```

3. Number of ways of partitioning a set of n elements

Reference :- en.wikipedia.org/wiki/Bell_triangle

Array B Bell stores the bell triangle.
It is a 2d matrix [lower triangular matrix]
Input = n = number of elements in a set
Output = number of ways they can be partitioned

$$Bell[i][j] = Bell[i-1][j-1] + Bell[i][j-1]$$

```
procedure Partition (n):
    for i from 0 to n
        for j from 0 to i:
            if ((i and j are equal to zero)
                or (i is equal to one and
                    j is equal to zero))
                then Bell[i][j] = 1
// element (0,1) and (0,0) are set to 1
            else if j is equal to 0
                Bell[i][0] = Bell[i-1][i-1]
// j column = 0 then Bell[row][0] = Bell[row-1][row-1]
// , ie, set first element of the row equal to
// the last element of previous row.
```

else:

$Bell[i][j] = Bell[i-1][j-1] - Bell[i][j-1]$

return $Bell[n-1][n-1]$

Note: Stopping condition here is that if the element is for the first element of row 0 and row 1 then set it to 1.

4 Grid pathway problem.

Input = n, m n = number of rows
m = number of cols.
∴ n x m = Size of grid.

Output = Number of shortest paths.

A = array of size n x m

~~$A[n][m] = 1$ if $[(n=1 \& m=0)$ or $(n=0 \& m=1)]$~~

$$A[n][m] = \begin{cases} 1, & \text{if } [(n=1 \& m=0) \text{ or } (n=0 \& m=1)] \\ = \begin{cases} path(n, m-1), & \text{if } n=0 \\ path(n-1, m), & \text{if } m=0 \\ path(n, m-1) + path(n-1, m), & \text{otherwise} \end{cases} \end{cases}$$

Note:- Initially all elements in array A are equal to -1

```
procedure path (n, m):
    if A[n][m] > -1    // if it stores a value
        return A[n][m]
    else if [(n is equal to 1 and m is equal to zero)
        OR n is equal to 0 and m is equal to one]
        return 1
    else if (n is equal to zero)
        A[n][m] = path (n, m-1)
    else if (m is equal to zero)
        A[n][m] = path (n-1, m)
    else
        A[n][m] = path (n, m-1) + path (n-1, m)

    return A[n][m]
```

Explanation

1. Checks if A[n][m] is filled. If yes returns that value
2. Checks if element is A[0][1] or A[1][0] since path from A[0][0] to the elements above is equal to 1, This is the stopping ~~control~~ condition
3. Checks if the element is in the first column if yes check the element below
4. Checks if the element is in the first row. If yes check the element to the left

5 Else returns the addition of path(n, m-1) and path(n-1, m)