Mobile Numeric Keypad Problem

4.3

Given the mobile numeric keypad. You can only press buttons that are up, left, right or down to the current button. You are not allowed to press bottom row corner buttons (i.e. * and #).
Given a number N, find out the number of possible numbers of given length.

Examples:
For N=1, number of possible numbers would be 10 (0, 1, 2, 3, …., 9)
For N=2, number of possible numbers would be 36
Possible numbers: 00,08 11,12,14 22,21,23,25 and so on.
If we start with 0, valid numbers will be 00, 08 (count: 2)
If we start with 1, valid numbers will be 11, 12, 14 (count: 3)
If we start with 2, valid numbers will be 22, 21, 23,25 (count: 4)
If we start with 3, valid numbers will be 33, 32, 36 (count: 3)
If we start with 4, valid numbers will be 44,41,45,47 (count: 4)
If we start with 5, valid numbers will be 55,54,52,56,58 (count: 5)
……
……

We need to print the count of possible numbers.

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

N = 1 is trivial case, number of possible numbers would be 10 (0, 1, 2, 3, …., 9)

For N > 1, we need to start from some button, then move to any of the four direction (up, left, right or down) which takes to a valid button (should not go to *, #). Keep doing this until N length number is obtained (depth first traversal).

Recursive Solution:

Mobile Keypad is a rectangular grid of 4X3 (4 rows and 3 columns)

Lets say Count(i, j, N) represents the count of N length numbers starting from position (i, j)

```
If N = 1
  Count(i, j, N) = 10
Else
  Count(i, j, N) = Sum of all Count(r, c, N-1) where (r, c) is new
                   position after valid move of length 1 from current
                   position (i, j)
```

Following is C implementation of above recursive formula.

```
// A Naive Recursive C program to count number of possible numbers
// of given length
#include <stdio.h>

// left, up, right, down move from current location
int row[] = {0, 0, -1, 0, 1};
int col[] = {0, -1, 0, 1, 0};

// Returns count of numbers of length n starting from key position
// (i, j) in a numeric keyboard.
int getCountUtil(char keypad[][3], int i, int j, int n)
{
    if (keypad == NULL || n <= 0)
        return 0;

    // From a given key, only one number is possible of length 1
    if (n == 1)
        return 1;
```

```
    int k=0, move=0, ro=0, co=0, totalCount = 0;

    // move left, up, right, down from current location and if
    // new location is valid, then get number count of length
    // (n-1) from that new position and add in count obtained so far
    for (move=0; move<5; move++)
    {
        ro = i + row[move];
        co = j + col[move];
        if (ro >= 0 && ro <= 3 && co >=0 && co <= 2 &&
           keypad[ro][co] != '*' && keypad[ro][co] != '#')
        {
            totalCount += getCountUtil(keypad, ro, co, n-1);
        }
    }

    return totalCount;
}

// Return count of all possible numbers of length n
// in a given numeric keyboard
int getCount(char keypad[][3], int n)
{
    // Base cases
    if (keypad == NULL || n <= 0)
        return 0;
    if (n == 1)
        return 10;

    int i=0, j=0, totalCount = 0;
    for (i=0; i<4; i++)  // Loop on keypad row
    {
        for (j=0; j<3; j++)   // Loop on keypad column
        {
            // Process for 0 to 9 digits
            if (keypad[i][j] != '*' && keypad[i][j] != '#')
            {
                // Get count when number is starting from key
                // position (i, j) and add in count obtained so far
                totalCount += getCountUtil(keypad, i, j, n);
            }
        }
    }
    return totalCount;
}

// Driver program to test above function
int main(int argc, char *argv[])
{
   char keypad[4][3] = {{'1','2','3'},
                        {'4','5','6'},
                        {'7','8','9'},
                        {'*','0','#'}};
   printf("Count for numbers of length %d: %dn", 1, getCount(keypad, 1));
   printf("Count for numbers of length %d: %dn", 2, getCount(keypad, 2));
   printf("Count for numbers of length %d: %dn", 3, getCount(keypad, 3));
   printf("Count for numbers of length %d: %dn", 4, getCount(keypad, 4));
   printf("Count for numbers of length %d: %dn", 5, getCount(keypad, 5));

   return 0;
}
```

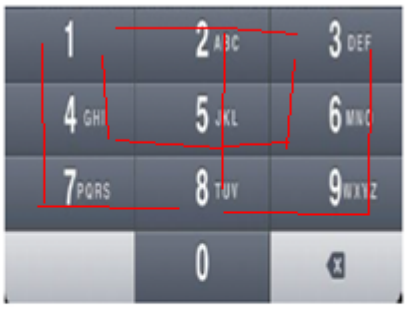
Run on IDE

Output:

```
Count for numbers of length 1: 10
Count for numbers of length 2: 36
Count for numbers of length 3: 138
Count for numbers of length 4: 532
Count for numbers of length 5: 2062
```

Dynamic Programming

There are many repeated traversal on smaller paths (traversal for smaller N) to find all possible longer paths (traversal for bigger N). See following two diagrams for example. In this traversal, for N = 4 from two starting positions (buttons '4' and '8'), we can see there are few repeated traversals for N = 2 (e.g. 4 -> 1, 6 -> 3, 8 -> 9, 8 -> 7 etc).



Few traversals starting for button 8 for N = 4
e.g. 8 -> 7 -> 4 -> 1, 8 -> 9 -> 6 -> 3
8 -> 5 -> 4 -> 1, 8 -> 5 -> 6 -> 3
8 -> 5 -> 2 -> 2, 8 -> 5 -> 2 -> 3

Few traversals starting from button 5 for N = 4
e.g. 5 -> 8 -> 7 -> 4, 5 -> 8 -> 9 -> 6
5 -> 4 -> 1 -> 2, 5 -> 6 -> 3 -> 2
5 -> 2 -> 1 -> 4, 5 -> 2 -> 3 -> 6

Since the problem has both properties: Optimal Substructure and Overlapping Subproblems, it can be efficiently solved using dynamic programming.

Following is C program for dynamic programming implementation.

```
// A Dynamic Programming based C program to count number of
// possible numbers of given length
#include <stdio.h>

// Return count of all possible numbers of length n
```

```
// in a given numeric keyboard
int getCount(char keypad[][3], int n)
{
    if(keypad == NULL || n <= 0)
        return 0;
    if(n == 1)
        return 10;

    // left, up, right, down move from current location
    int row[] = {0, 0, -1, 0, 1};
    int col[] = {0, -1, 0, 1, 0};

    // taking n+1 for simplicity - count[i][j] will store
    // number count starting with digit i and length j
    int count[10][n+1];
    int i=0, j=0, k=0, move=0, ro=0, co=0, num = 0;
    int nextNum=0, totalCount = 0;

    // count numbers starting with digit i and of lengths 0 and 1
    for (i=0; i<=9; i++)
    {
        count[i][0] = 0;
        count[i][1] = 1;
    }

    // Bottom up - Get number count of length 2, 3, 4, ... , n
    for (k=2; k<=n; k++)
    {
        for (i=0; i<4; i++)  // Loop on keypad row
        {
            for (j=0; j<3; j++)   // Loop on keypad column
            {
                // Process for 0 to 9 digits
                if (keypad[i][j] != '*' && keypad[i][j] != '#')
                {
                    // Here we are counting the numbers starting with
                    // digit keypad[i][j] and of length k keypad[i][j]
                    // will become 1st digit, and we need to look for
                    // (k-1) more digits
                    num = keypad[i][j] - '0';
                    count[num][k] = 0;

                    // move left, up, right, down from current location
                    // and if new location is valid, then get number
                    // count of length (k-1) from that new digit and
                    // add in count we found so far
                    for (move=0; move<5; move++)
                    {
                        ro = i + row[move];
                        co = j + col[move];
                        if (ro >= 0 && ro <= 3 && co >=0 && co <= 2 &&
                           keypad[ro][co] != '*' && keypad[ro][co] != '#')
                        {
                            nextNum = keypad[ro][co] - '0';
                            count[num][k] += count[nextNum][k-1];
                        }
                    }
                }
            }
        }
    }

    // Get count of all possible numbers of length "n" starting
    // with digit 0, 1, 2, ..., 9
    totalCount = 0;
    for (i=0; i<=9; i++)
        totalCount += count[i][n];
    return totalCount;
}
```

```
// Driver program to test above function
int main(int argc, char *argv[])
{
    char keypad[4][3] = {{'1','2','3'},
                         {'4','5','6'},
                         {'7','8','9'},
                         {'*','0','#'}};
    printf("Count for numbers of length %d: %dn", 1, getCount(keypad, 1));
    printf("Count for numbers of length %d: %dn", 2, getCount(keypad, 2));
    printf("Count for numbers of length %d: %dn", 3, getCount(keypad, 3));
    printf("Count for numbers of length %d: %dn", 4, getCount(keypad, 4));
    printf("Count for numbers of length %d: %dn", 5, getCount(keypad, 5));

    return 0;
}
```

Run on IDE

Output:

```
Count for numbers of length 1: 10
Count for numbers of length 2: 36
Count for numbers of length 3: 138
Count for numbers of length 4: 532
Count for numbers of length 5: 2062
```

A Space Optimized Solution:

The above dynamic programming approach also runs in O(n) time and requires O(n) auxiliary space, as only one for loop runs n times, other for loops runs for constant time. We can see that nth iteration needs data from (n-1)th iteration only, so we need not keep the data from older iterations. We can have a space efficient dynamic programming approach with just two arrays of size 10. Thanks to Nik for suggesting this solution.

```
// A Space Optimized C program to count number of possible numbers
// of given length
#include <stdio.h>

// Return count of all possible numbers of length n
// in a given numeric keyboard
int getCount(char keypad[][3], int n)
{
    if(keypad == NULL || n <= 0)
        return 0;
    if(n == 1)
        return 10;

    // odd[i], even[i] arrays represent count of numbers starting
    // with digit i for any length j
    int odd[10], even[10];
    int i = 0, j = 0, useOdd = 0, totalCount = 0;

    for (i=0; i<=9; i++)
        odd[i] = 1;  // for j = 1

    for (j=2; j<=n; j++) // Bottom Up calculation from j = 2 to n
    {
        useOdd = 1 - useOdd;

        // Here we are explicitly writing lines for each number 0
        // to 9. But it can always be written as DFS on 4X3 grid
        // using row, column array valid moves
```

```
        if(useOdd == 1)
        {
            even[0] = odd[0] + odd[8];
            even[1] = odd[1] + odd[2] + odd[4];
            even[2] = odd[2] + odd[1] + odd[3] + odd[5];
            even[3] = odd[3] + odd[2] + odd[6];
            even[4] = odd[4] + odd[1] + odd[5] + odd[7];
            even[5] = odd[5] + odd[2] + odd[4] + odd[8] + odd[6];
            even[6] = odd[6] + odd[3] + odd[5] + odd[9];
            even[7] = odd[7] + odd[4] + odd[8];
            even[8] = odd[8] + odd[0] + odd[5] + odd[7] + odd[9];
            even[9] = odd[9] + odd[6] + odd[8];
        }
        else
        {
            odd[0] = even[0] + even[8];
            odd[1] = even[1] + even[2] + even[4];
            odd[2] = even[2] + even[1] + even[3] + even[5];
            odd[3] = even[3] + even[2] + even[6];
            odd[4] = even[4] + even[1] + even[5] + even[7];
            odd[5] = even[5] + even[2] + even[4] + even[8] + even[6];
            odd[6] = even[6] + even[3] + even[5] + even[9];
            odd[7] = even[7] + even[4] + even[8];
            odd[8] = even[8] + even[0] + even[5] + even[7] + even[9];
            odd[9] = even[9] + even[6] + even[8];
        }
    }

    // Get count of all possible numbers of length "n" starting
    // with digit 0, 1, 2, ..., 9
    totalCount = 0;
    if(useOdd == 1)
    {
        for (i=0; i<=9; i++)
            totalCount += even[i];
    }
    else
    {
        for (i=0; i<=9; i++)
            totalCount += odd[i];
    }
    return totalCount;
}

// Driver program to test above function
int main()
{
    char keypad[4][3] = {{'1','2','3'},
        {'4','5','6'},
        {'7','8','9'},
        {'*','0','#'}
    };
    printf("Count for numbers of length %d: %dn", 1, getCount(keypad, 1));
    printf("Count for numbers of length %d: %dn", 2, getCount(keypad, 2));
    printf("Count for numbers of length %d: %dn", 3, getCount(keypad, 3));
    printf("Count for numbers of length %d: %dn", 4, getCount(keypad, 4));
    printf("Count for numbers of length %d: %dn", 5, getCount(keypad, 5));

    return 0;
}
```

Run on IDE

Output:

```
Count for numbers of length 1: 10
Count for numbers of length 2: 36
```

```
Count for numbers of length 3: 138
Count for numbers of length 4: 532
Count for numbers of length 5: 2062
```

Asked in: Microsoft,sprinklr

This article is contributed by **Anurag Singh**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

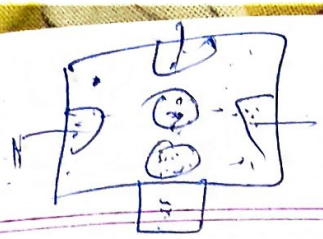
Dynamic Programming | Matrix

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Count of n digit numbers whose sum of digits equals to given sum
Dynamic Programming | Set 15 (Longest Bitonic Subsequence)
Minimum Cost Polygon Triangulation
Minimum Initial Points to Reach Destination
Dynamic Programming | Set 37 (Boolean Parenthesization Problem)
Maximum number of trailing zeros in the product of the subsets of size k
Balanced expressions such that given positions have opening brackets
Check if it is possible to transform one string to another
Maximum difference of zeros and ones in binary string | Set 2 (O(n) time)
Probability of reaching a point with 2 or 3 steps at a time



	Options	combinations	Total
4 corners	→ 3	${}^3C_{N-1}$	$4 \times {}^3C_{N-1}$
3 mids	→ 4	${}^4C_{N-1}$	$3 \times {}^4C_{N-1}$
2 middle	→ 5	${}^5C_{N-1}$	$2 \times {}^5C_{N-1}$
1 zero	→ 2	${}^2C_{N-1}$	$1 \times {}^2C_{N-1}$

36

$12 + 12 + 10 + 2$

$4\,{}^3C_1 + {}^4C_1 + 2\,{}^5C_1 + {}^2C_1$

$12 + 16 + 5 + 2$

35

$4\,{}^3C_2 + {}^4C_2 + 2\,{}^5C_2 + {}^2C_2$

$12 + 24 + 10 + 1$

47

$12 + 18 + 20 + 1$

$4\,{}^3C_3 + {}^4C_3 \cdot {}^5P_2 + 2\,{}^5P_2 + {}^2P_2$

$12 + 16 + 20 + 1$

49

$4(3)^{N-1} + 3(4)^{N-1} + 2(5)^{N-1} + 1(2)^{N-1}$

$1 \times 3 \times 3$

$4(3)^2 + 3(4)^2 + 2(5)^2 + 1(2)^2$

$36 + 48 + 50 + 4$

138

$4\,{}^3P_2 + 3\,{}^4P_2 + 2\,{}^5P_2 + {}^2P_2$

$12 + 36 + 120 + 1$

$12 + 36 + 40 + 1$

89

$^3P_2$ $\frac{1 \times 2 \times 1}{1 \times 2 \times 1}$

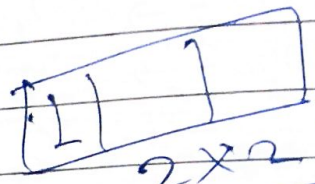
① Corner- $4\ ^3P_2 = 24$

② Mid → $4\ ^4P_2 = 48$

③ Midd → $2\ ^5P_2 = 40$

④ Zero → $1\ ^2P_2 = 2$

114



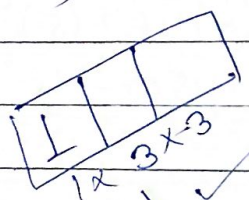
$\frac{3!}{1!} \quad \frac{4 \times 1 \times 2!}{2!} \quad 2!$

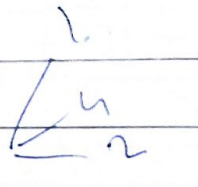
$\frac{5 \times 4 \times 3}{3!}$

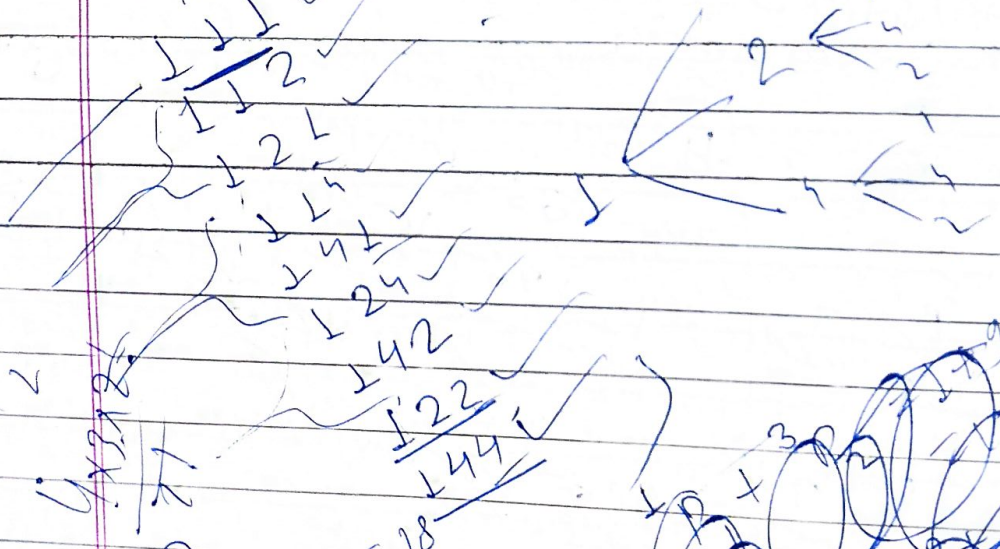
$^3P_3$

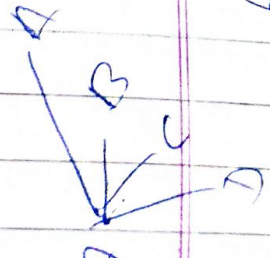
$\frac{3!}{3!}$

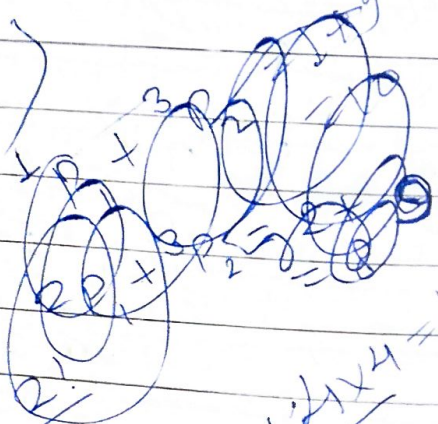
3×2

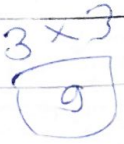


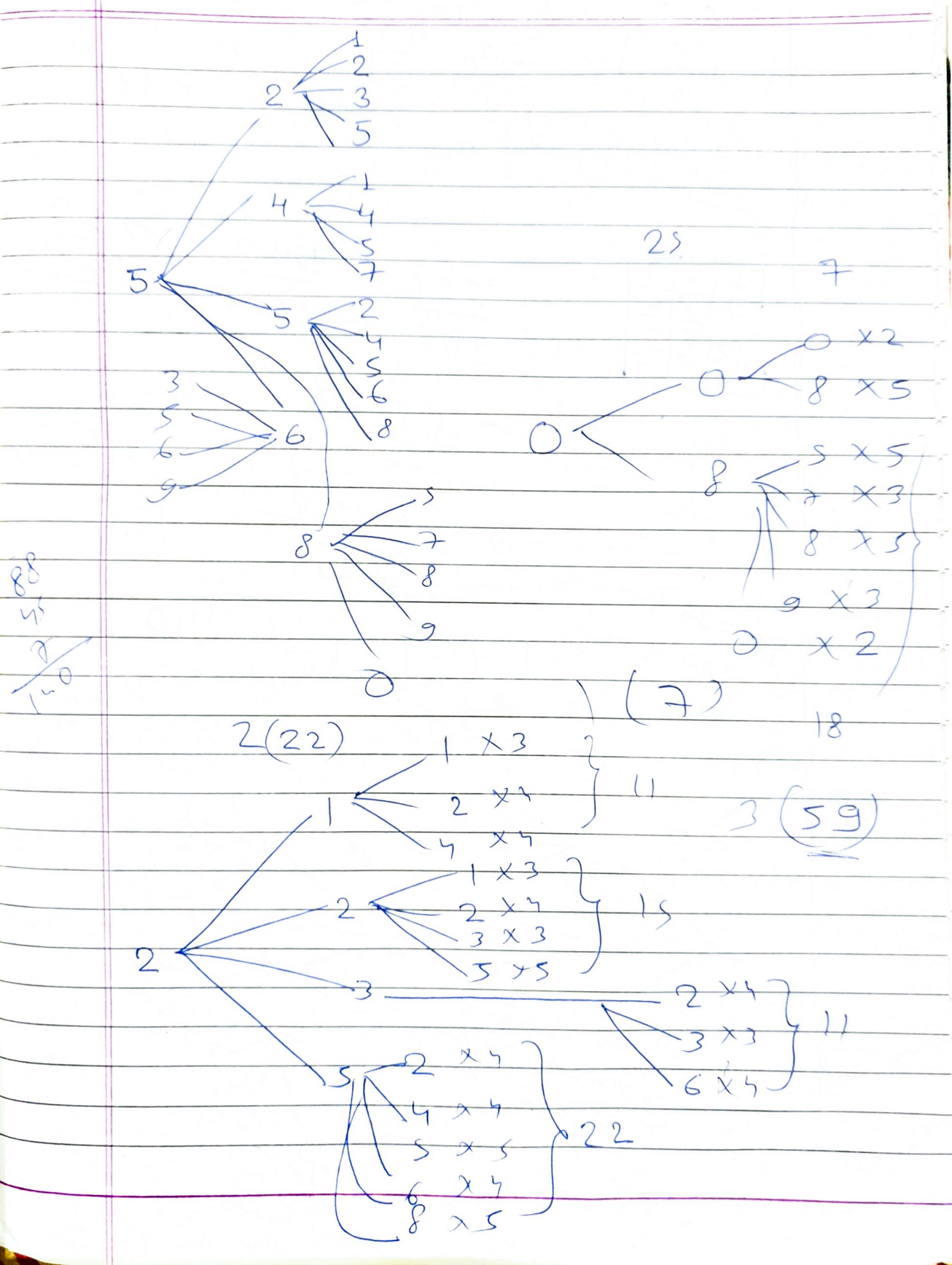


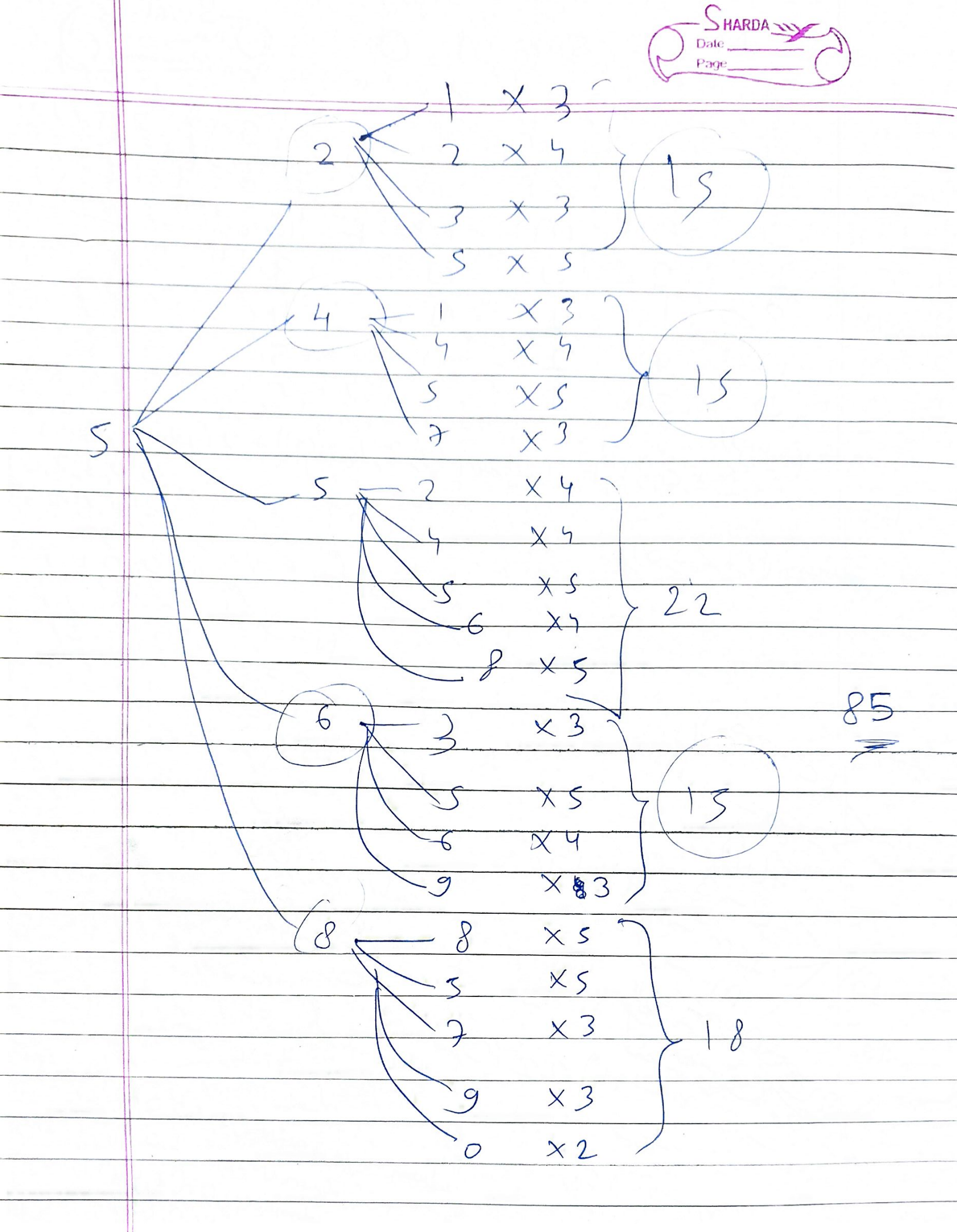









2
1
2
3
5
4
1
4
5
7
5
5
2
4
5
6
8
3
5
6
9
6
8
5
7
8
9
0
88
45
8
1=0
25
7
0
x2
8 x5
8
5 x5
7 x3
8 x5
9 x3
0 x2
(7)
18
2(22)
1
1 x3
2 x4
4 x4
11
3 (59)
2
1 x3
2 x4
3 x3
5 x5
15
3
2 x4
3 x3
6 x4
11
5
2 x4
4 x4
5 x5
6 x4
8 x5
22


SHARDA
Date
Page
1 x 3
2
2 x 4
3 x 3
5 x 5
15
4
1 x 3
4 x 4
5 x 5
7 x 3
15
5
5
2 x 4
4 x 4
5 x 5
6 x 4
8 x 5
22
6
3 x 3
5 x 5
6 x 4
9 x 3
15
8
8 x 5
5 x 5
7 x 3
9 x 3
0 x 2
18
85

C SM M O

	C	SM	M	O	E	sum
4 C	1	2	0	0		3
3 SM	2	1	1	0		4
2 M	0	3	1	0		5
1 O	0	0	1	1		2

4(1,4,0)

4(getsum(3,1,2,0,0)) + 3(getsum(3,2,1,1,0)) + 2(getsum(3,0,3,2,0))

4(getsum(2,1,2,0,0))

3,0,0,1,1

2,0,0,1,2

N=3

C=4

SM=3

M=2

O=1

int getsum(N,C,SM,M,O)

{

if(N==2)

return C(3) + SM(4) + M(5) + O(1);

else if(N==1)

return 1;

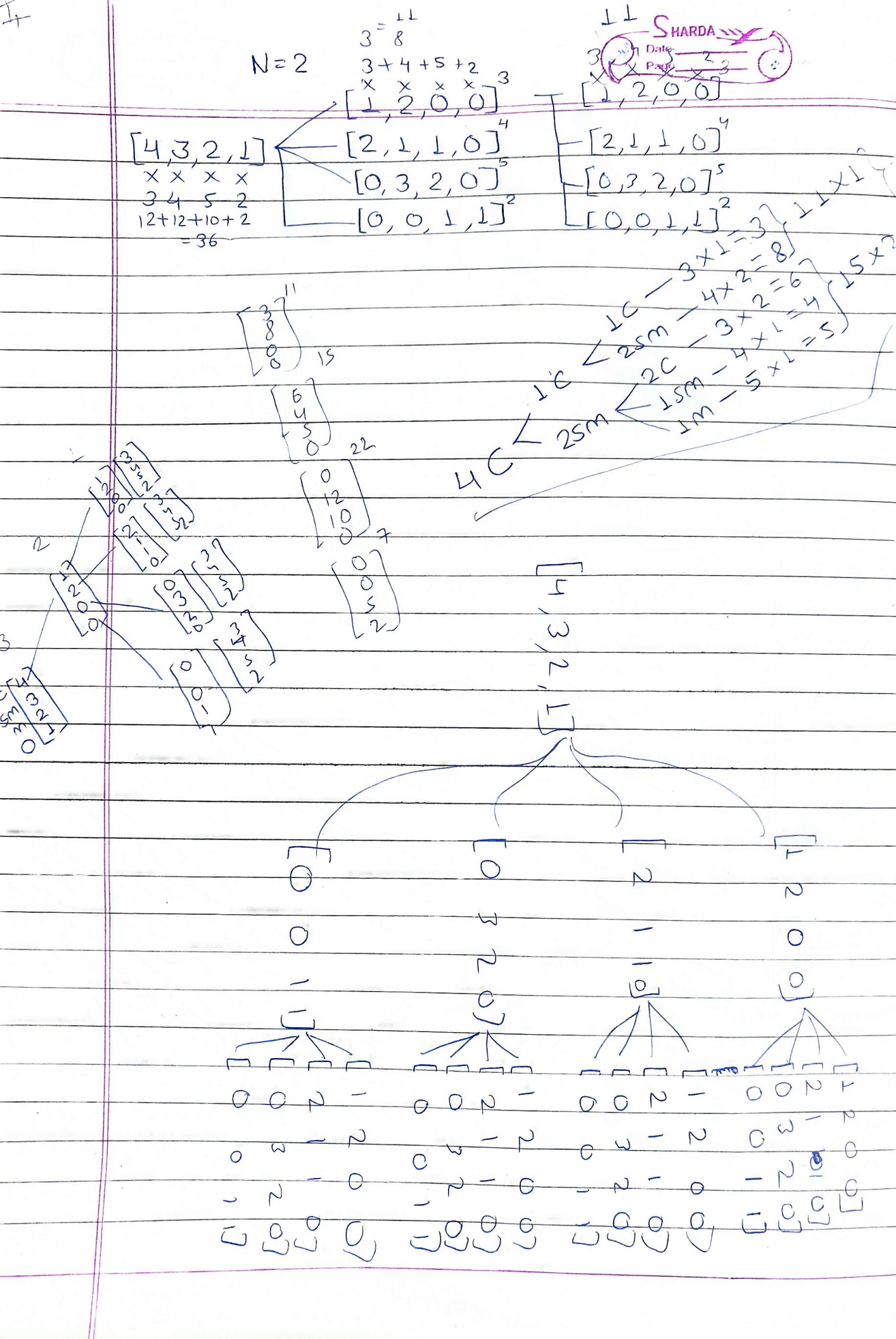
else

return 4(getsum(N-1,1,2,0,0))

+3(getsum(N-1,2,1,1,0))

+2(getsum(N-1,0,3,2,0))

+1(getsum(N-1,0,0,1,1))


N=2
3+4+5+2
[4,3,2,1]
12+12+10+2
=36
[1,2,0,0]
[2,1,1,0]
[0,3,2,0]
[0,0,1,1]
SHARDA
Date
Page

N = 4

P[4] = {3,4,5,2};

m[4] = {4,3,2,1};

C[4][4] = {{1,2,0,0},
{2,1,1,0},
{0,3,2,0},
{0,0,1,1}};

SMARDA
Debe
Pago

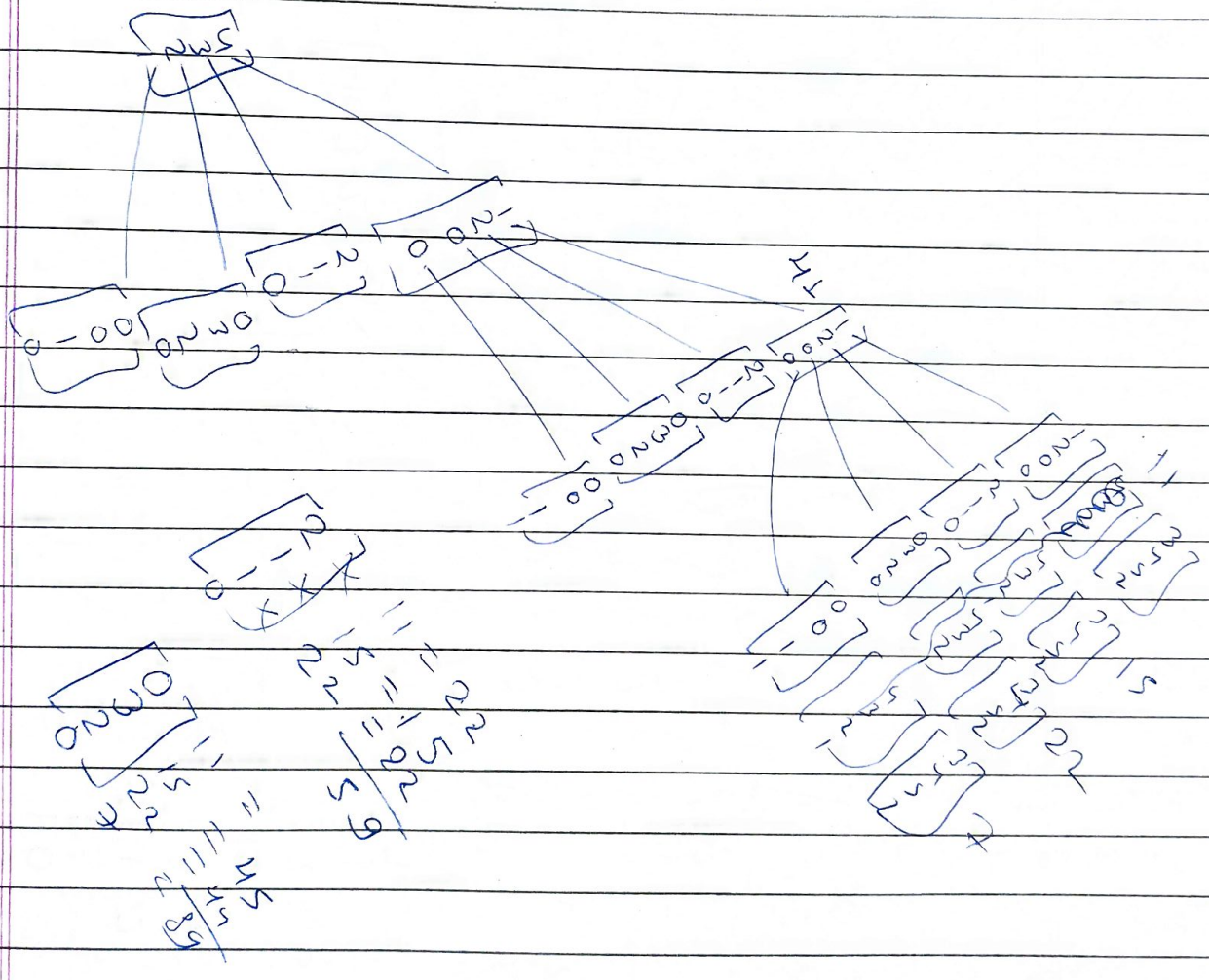
}

while (N! = 1)

F = F *

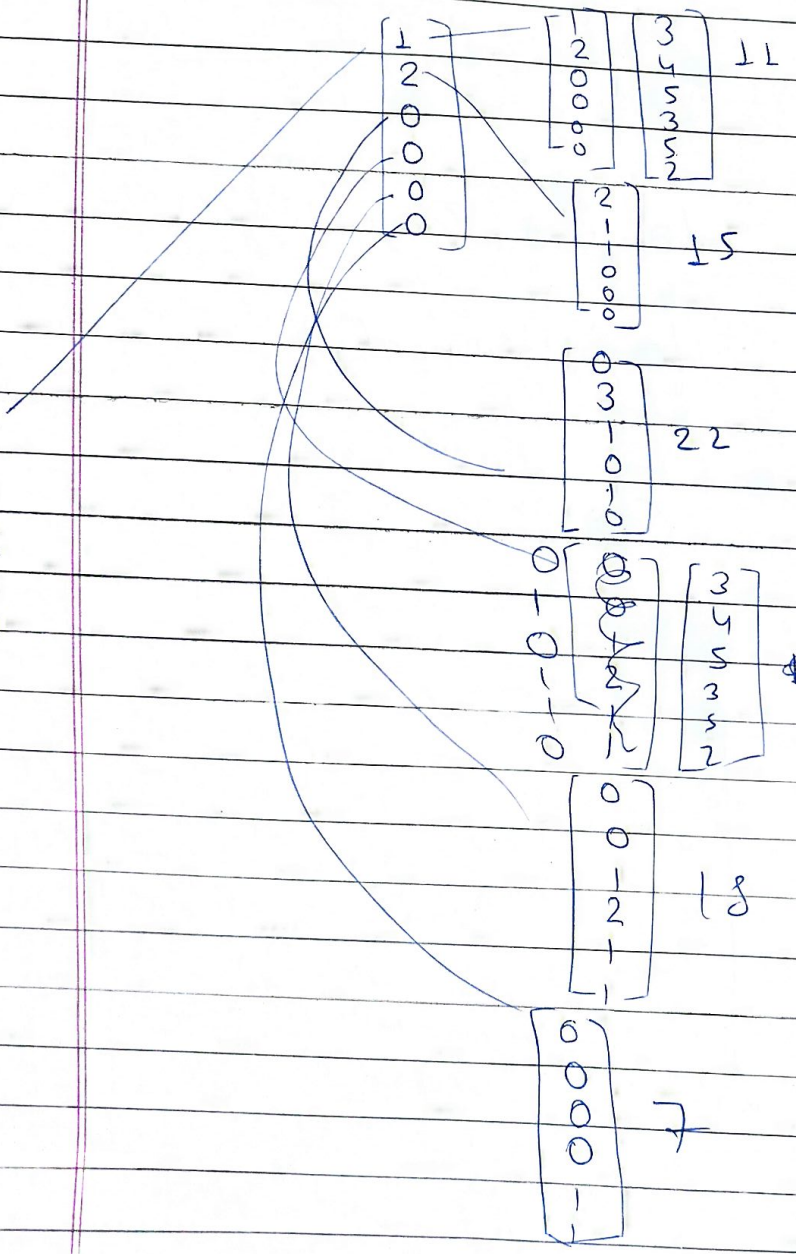
F = [sum(F*F1), sum(F*F2), sum(F*F3), sum(F*F4)]

41, 59, 89, 29



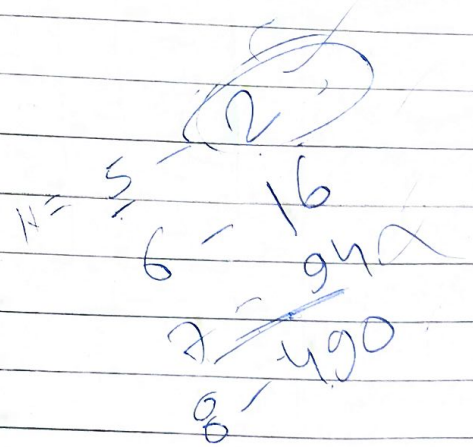
1

	UC	SM	M	LC	E	O	Sum
2 UC	1	2	0	0	0	0	3
3 SM	2	1	1	0	0	0	4
1 M	0	3	1	0	1	0	5
2 LC	0	1	0	1	1	0	3
1 E	0	0	1	2	1	1	5
1 O	0	0	0	0	1	1	2



36 36
138 138
530 532
2046 2062
7896 7990
30494 30984

7990
30984



$6x + y = 16$

$5x + y = 2$

$x = 14$

$y = 16 - 6(14)$

$y = -68$

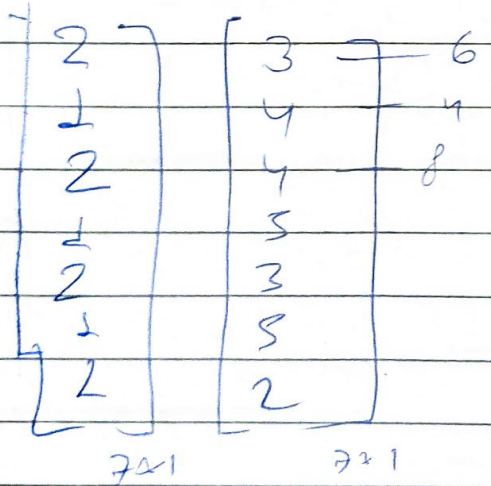
$7(14) - 68$

10 13

for i in Range(10, 20, 3):

0

for i in range(a):



N=1 ⇒ 10

N=2 ⇒

·	·	·
·	·	·
·	·	·
x	·	x

I		F			
~~2~~	UC	3	6	N=2	N=3
3	SM	4	12		
1	M	5	5	36	138
2	LC	3	6		
1	E	5	5		
1	O	2	2		

	UC	SM	M	LC	E	O	sum
UC	1	2	0	0	0	0	3
SM	2	1	1	0	0	0	4
M	0	3	1	0	1	0	5
LC	0	1	0	1	1	0	3
E	0	0	1	2	1	1	5
O	0	0	0	0	1	1	2

I	F			
11	2	1	3	11
15	3	2	4	
22	1	0	5	
12	2	0	3	
18	1	0	5	
7	1	0	2	

4+3+5

5+6+5+2

22+48+22+24

+18+7

= 131

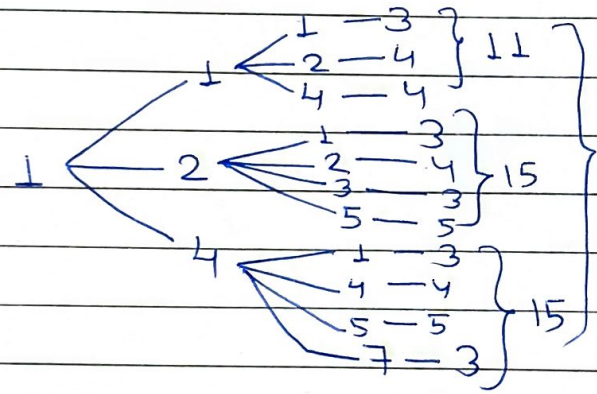
2	3	15
1	4	
1	5	
6	3	
0	5	
0	2	

0	3	
3	4	
1	5	
0	3	
1	5	
0	2	

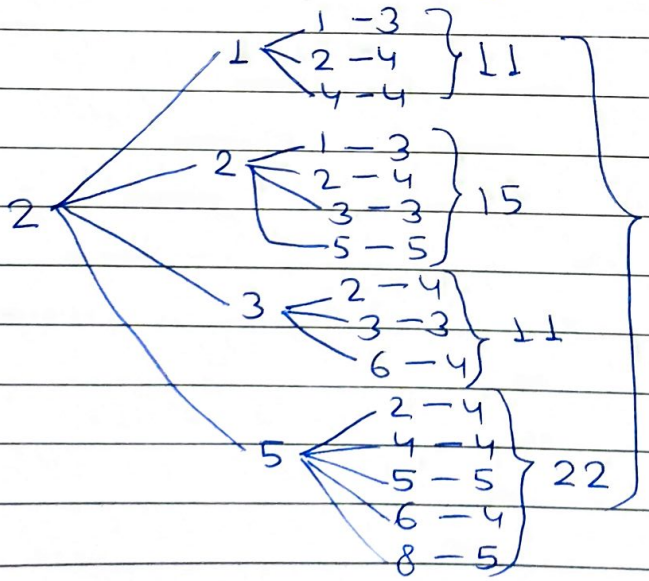
1 UC 41	2 UM 59	3 UC 41
4 60 SM (59)	5 M 85	6 60 SM (59)
7 LC 45	8 E 71	9 LC 45
X	0 0 25	X

1 → 1, 2, 4 = 3
2 → 1, 2, 3, 5 = 4
3 → 2, 3, 6 = 3
4 → 1, 4, 5, 7 = 4
5 → 2, 4, 5, 6, 8 = 5
6 → 3, 5, 6, 9 = 4
7 → 4, 7, 8 = 3
8 → 5, 7, 8, 9, 0 = 5
9 → 6, 8, 9 = 3
0 → 0, 8 = 2

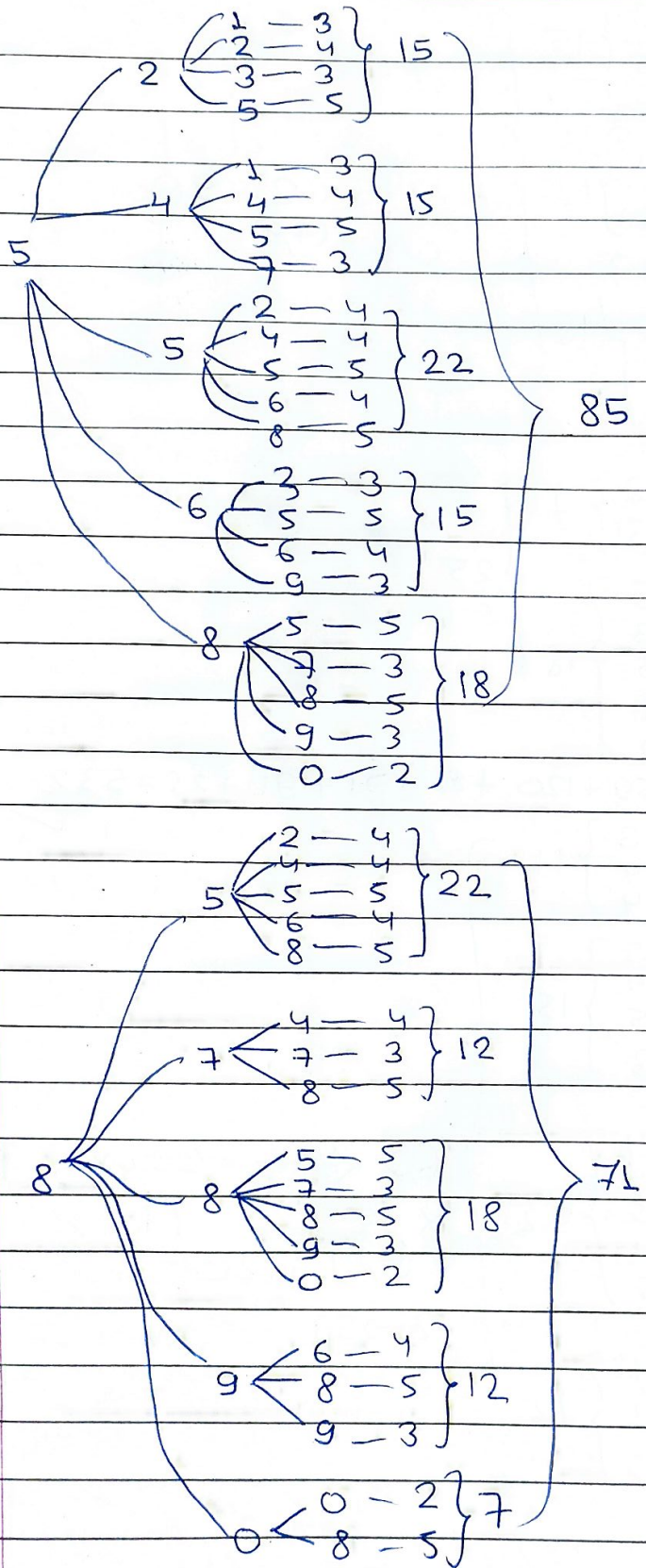
(1) Total = 82 + 177 + 85 + 71 + 90 + 25 = 530 ~~532~~

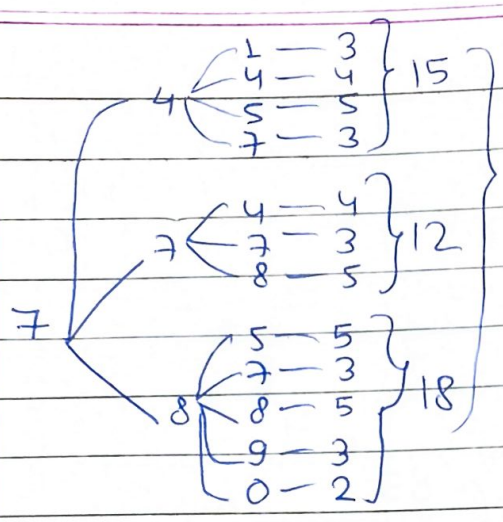


41 × 2 = 82
(1, 3)

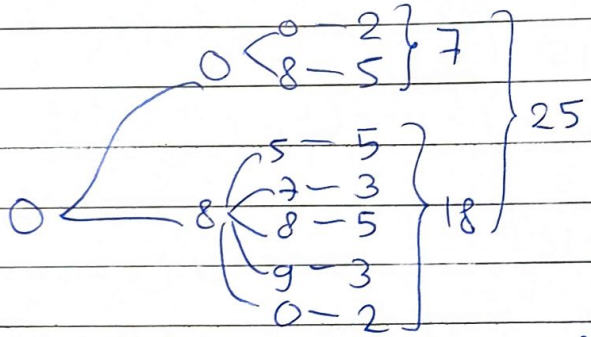


59 × 3 = 177
(2, 4, 6)

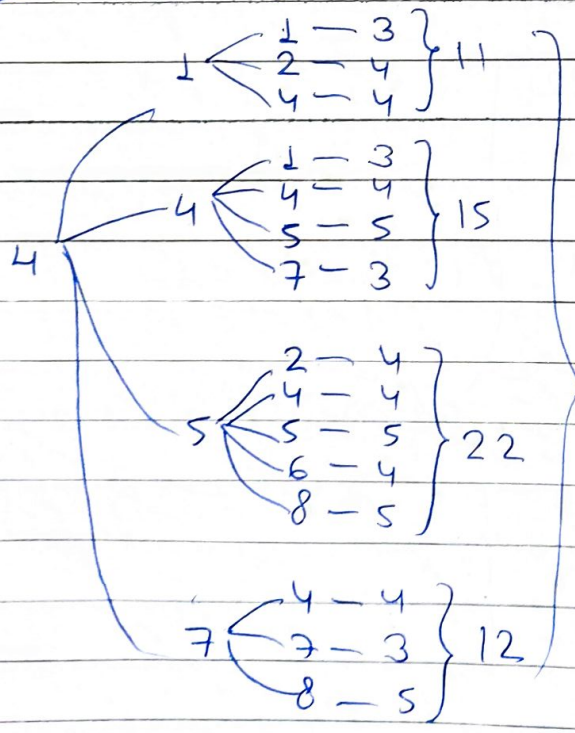




$45 \times 2 = 90$
(7,9)



(2) Total = 82 + 59 + 120 + 85 + 71 + 90 + 25 = 532 ✓



60 ← error of ①
× 2
(4,6) = 120
2 → 59

The New Table:-

	UC	UM	SM	M	LC	E	O	Sum
2 UC	1	1	1	0	0	0	0	3
1 UM	2	1	0	1	0	0	0	4
2 SM	1	0	1	1	1	0	0	4
1 M	0	1	2	0	0	1	0	5
2 LC	0	0	1	0	1	1	0	3
1 E	0	0	0	1	2	1	1	5
1 O	0	0	0	0	0	1	1	2

```
def pro(F,FT):
    return
[F[0]*FT[0],F[1]*FT[1],F[2]*FT[2],F[3]*FT
[3],F[4]*FT[4],F[5]*FT[5],F[6]*FT[6]]
def NOC(N):
    if(N==1):
        return 10
    F=[3,4,4,5,3,5,2]
    I=[2,1,2,1,2,1,1]
    F1=[1,1,1,0,0,0,0]
    F2=[2,1,0,1,0,0,0]
    F3=[1,0,1,1,1,0,0]
    F4=[0,1,2,1,0,1,0]
    F5=[0,0,1,0,1,1,0]
    F6=[0,0,0,1,2,1,1]
    F7=[0,0,0,0,0,1,1]
    for i in range(1,N-1):

F=[sum(pro(F,F1)),sum(pro(F,F2)),sum(pro(
F,F3)),sum(pro(F,F4)),sum(pro(F,F5)),sum(
pro(F,F6)),sum(pro(F,F7))]
    return sum(pro(I,F))
for i in range(1,10):
    print("Count for numbers of length
"+str(i)+": "+str(NOC(i)))
```