GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

Mobile Numeric Keypad Problem

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.

We strongly recommend to minimize the browser and try this yourself first.

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'},
```

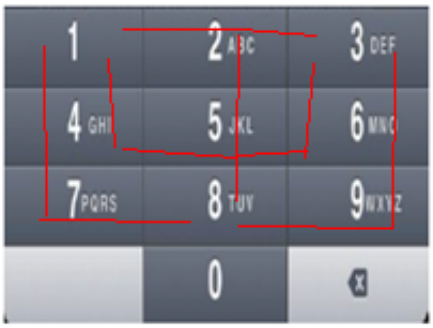
```
                                {'*','0','#'}};
    printf("Count for numbers of length %d: %d\n", 1, getCount(keypad, 1));
    printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));
    printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));
    printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));
    printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));

    return 0;
}
```

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: %d\n", 1, getCount(keypad, 1));
   printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));
   printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));
```

```
    printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));
    printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));

    return 0;
}
```

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: %d\n", 1, getCount(keypad, 1));
    printf("Count for numbers of length %d: %d\n", 2, getCount(keypad, 2));
    printf("Count for numbers of length %d: %d\n", 3, getCount(keypad, 3));
    printf("Count for numbers of length %d: %d\n", 4, getCount(keypad, 4));
    printf("Count for numbers of length %d: %d\n", 5, getCount(keypad, 5));
```

```
    return 0;
}
```

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
```

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.

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming, Matrix

Writing code in comment? Please use **ideone.com** and share the link here.

http://ideone.com/uz0FDI

• Reply • Share ›

Sreeja • 2 months ago

I think there is O(logN) solution here.
If you write a directed graph matrix G[10][10], denoting the moves as edges.
Then sum of all the element in G^N should give the total number.
As G^N by divide and conquer takes O(logN) time complexity and if stack is considered as space O(logN) space.

1 • Reply • Share ›

Sanket Patel ➚ Sreeja • 2 months ago

I could not understand this -
"Then sum of all the element in G^N should give the total number.
As G^N by divide and conquer takes O(logN) time complexity and if stack is considered as space O(logN) space."

Could you please elaborate, or redirect me to some relevant material?

• Reply • Share ›

pavansrinivas • 5 months ago

DP approach in java..Time Complexity O(n)

Space Complexity O(n)
http://ideone.com/eyKzRI

• Reply • Share ›

Preacher • 5 months ago

One more thing Anurag. Both the time and space complexity of your DP is O(2^n) not O(n). In Big O notation, the argument is the size of the input. The input is a number, N, represented in binary. The length of that representation, n, is log(N). Your loop runs N times, which is 2^log(N).

• Reply • Share ›

Anurag Singh ➚ Preacher • 5 months ago

There are two DP approaches discussed above. 1st one is normal DP approach, 2nd one is space efficient approach.
In 1st approach, there are few "for" loops. one loop runs O(n) time, other loops run for constant time. So TC is O(n) only.
Please revert back if you see any issue.

• Reply • Share ›

Preacher ➚ Anurag Singh • 5 months ago

Yes. The second approach has space complexity constant. But both of them have essentially the same approach, and have exponential time. Both of them

have input "N". The number "N" is given in binary form to the program as input. The length of this input is log(N). The order notation is written with function w.r.t the size of the input. If the size of the input is "x"=log(N), then the number of steps in your algorithm is of the order "N", which is O(2^x). That is why, your algorithm is O(2^n) and not O(n).

• Reply • Share ›

Preacher • 5 months ago

Another possible implementation:

http://ideone.com/rBUGnH

Sample output:

1 pushes: 10
2 pushes: 36
3 pushes: 138
4 pushes: 532
5 pushes: 2062
6 pushes: 7990
7 pushes: 30984
8 pushes: 120130
9 pushes: 465832
10 pushes: 1806282

• Reply • Share ›

kanu • 6 months ago

int row[] = {0, 0, -1, 0, 1};
int col[] = {0, -1, 0, 1, 0}; i did not understand how to write this can anybody explain?

• Reply • Share ›

Thanh Quan ➚ kanu • 6 months ago

deltaX, deltaY

• Reply • Share ›

gcc ➚ Thanh Quan • 2 months ago

please elaborate.

• Reply • Share ›

Prabu ➚ Thanh Quan • 4 months ago

Can u explain more clearly

• Reply • Share ›

Thanh Quan ➚ Prabu • 2 months ago

For i in range(5):
col =x + deltaX
row = y + deltaY

So you can access neighbor cells

• Reply • Share ›

Kenneth • 7 months ago

Here is my DP solution:

http://ideone.com/C9cIs7

Tests:
Move count 0 : 0
Move count 1 : 10
Move count 2 : 36
Move count 3 : 138
Move count 4 : 532
Move count 5 : 2062
Move count 6 : 7990
Move count 7 : 30984
Move count 8 : 120130
Move count 9 : 465832
Move count 10 : 1806282

• Reply • Share ›

aa1992 • 7 months ago

if n=3 then how the ans comes to 532?shouldn't it be 558?starting with 0 there are 8 numbers,starting 1 there are 27 numbers ,with 2 there are 64 numbers.
here is my code http://ideone.com/Yhp27a
i don't understand where am i going wrong.please help.

2 • Reply • Share ›

Anurag Singh ➚ aa1992 • 7 months ago

532 will come as answer for n = 4 (for n = 3, answer is 138)
Now for possible number count of length n = 3 starting with specific digit,
if number starts with ZERO, then it will be 7 (not 8) (000, 008, 080, 088, 085, 087, 089)
if number starts with ONE, then it will be 11 (not 27)

If we take n=4, then
if number starts with ZERO, then it will be 25
if number starts with ONE, then it will be 41

The logic you have used is giving right answer upto n=3 only, not beyond that.
I can't understand the reasoning behind that. If you can explain that, I can see where the

things are wrong. For now, your logic doesn't look correct to me.

• Reply • Share ›

co8e ➚ aa1992 • 7 months ago

Did you get any resolution? Having the same answer on my end, would be keen to know if this one is correct..

• Reply • Share ›

mohammad • 7 months ago

http://ideone.com/DCQ9Dn

• Reply • Share ›

Abhi • 7 months ago

Just a completely different DP approach...I am new to solving all these stuffs....
http://ideone.com/e.js/4an5kB

• Reply • Share ›

Suman ➚ Abhi • 7 months ago

Some comment in the code will help.

• Reply • Share ›

moiez • 7 months ago

I think this problem can be restated as finiding all K length paths in a graph which has 10 nodes, and where the connectivity between any pair of nodes is decided by the nodes being neighbours of each other, ie, one node can be reached by moving up, down, left or right from the other node. Of course, each node has a self loop too.

1 • Reply • Share ›

Mohammad Moiez Gohar ➚ moiez • 7 months ago

Space Complexity: O(1)
Time Complexity: O(n)

• Reply • Share ›

Suman ➚ Mohammad Moiez Gohar • 7 months ago

Can you please explain what is the algorithm for this time and space complexity

• Reply • Share ›

Mohammad Moiez Gohar ➚ Suman • 7 months ago

1. Initialize a 2-D array with entry at ith row and jth column being 1 if it is possible to move from number i to number j on the numeric keypad by going one step left,right, up or down.

2. Given the length N of the number of digits to be typed on the keypad, the number of possible numbers that can be typed using N digits is just

the number of paths of length N in the graph representing the keypad numbers as nodes and edges defined by the connectivity between the two nodes based on whether it is possible to move from one node to another by going one step left, right, up or down.
And we know that K length path can be computed from the adjacency matrix by multiplying the matrix by itself K times.

Space Complexity: array of 10x10 [0-9].
Time Complexity: no. of matrix multiplications* cost of each multiplication = N*(10*10*10) = O(N)

• Reply • Share ›

Anurag Singh ➚ moiez • 7 months ago

That's right.

• Reply • Share ›

Preacher ➚ Anurag Singh • 5 months ago

No. Because we can press the same button K times. That is equivalent of staying on the same vertex.

• Reply • Share ›

Priyal Rathi • 7 months ago

The space complexity of above dynamic programming solution is O((n+1)*10)

Another approach with constant space complexity:--
while calculating the no. of possible solutions of length n, we only need to have no. of possible solutions of length n-1. So we just need to store no. of possible solutions of length n-1 and n.
Space complexity: O(10*2)= O(20) constant space
Time complexity: O(n*10)
Link : http://ideone.com/3JDb0F

3 • Reply • Share ›

Anurag Singh ➚ Priyal Rathi • 7 months ago

It will work. This is space efficient dynamic programming approach, also discussed here.

• Reply • Share ›

Priyal Rathi ➚ Anurag Singh • 7 months ago

The space complexity of the solution mentioned http://www.geeksforgeeks.org/m... is O((n+1)*10), whereas the space complexity of my code is O(10*2)=O(20) constant space complexity.

• Reply • Share ›

Anurag Singh ➚ Priyal Rathi • 7 months ago

Space Complexity should be O(10*2)=O(20) there too. Using two extra arrays even and odd of length 10. That's all.
And to avoid array copy in every iteration, even and odd arrays are used alternatively.

• Reply • Share ›

Priyal Rathi ➚ Anurag Singh • 7 months ago

Ya I agree, to avoid array copy in every iteration, the 2 arrays even and odd can be used alternatively.
But, The space complexity of dynamic programming code mentioned here is O((n+1)*10) right?? O((n+1)*10) space complexity can be very high for large values of n.
I feel instead of having 2D count array in above code, keeping 2 arrays (even and odd) is better solution with constant space complexity and time complexity of O(n*10).

• Reply • Share ›

Anurag Singh ➚ Priyal Rathi • 7 months ago

That's Right. The space complexity of dynamic programming code posted in article above is O(n) where we used a 2D array. And we can avoid this using space efficient version where we only need two arrays of size 10.

• Reply • Share ›

co8e ➚ Anurag Singh • 7 months ago

Hey anurag, can you confirm if the first answer above by aa1992 is correct?

• Reply • Share ›

Anurag Singh ➚ co8e • 7 months ago

Gave my response above.

• Reply • Share ›

zeal • 7 months ago

Have a look at my approach:

http://ideone.com/QgnOIP

2 • Reply • Share ›

Anurag Singh ➚ zeal • 7 months ago

Looks fine. It's the same dynamic programming approach where DFS is hardcoded.

• Reply • Share ›

Akhil Pratap Singh • 7 months ago

Akhil Pratap Singh • 7 months ago

In this code section, shouldn't count[i][1] be 10 instead of 1:
// 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;
}

• Reply • Share ›

Anurag Singh ➚ Akhil Pratap Singh • 7 months ago

No. count[i][1] doesn't represent count of all numbers of length 1.
It only represent count of numbers of length 1 starting with digit i.

• Reply • Share ›

krishna • 7 months ago

cant we do it like this
http://ideone.com/erzGk7

• Reply • Share ›

Anurag Singh ➚ krishna • 7 months ago

It doesn't give correct answer for numberLength > 2

1 • Reply • Share ›

PC • 7 months ago

The correct answer can be found out by setting row and col arrays as:
int row[] = {0, -1, 0, 1, 0};

int col[] = {-1, 0, 1, 0, 0};
and running the loop 5 times instead of 4

• Reply • Share ›

PC • 7 months ago

The solution doesn't consider the case of selecting the same button twice.
For e.g. Possible numbers for n=2: 00,08 11,12,14 22,21,23,25 and so on.
But the given solution doesn't consider the numbers '00', '11', '22'...etc.

• Reply • Share ›

Nlk • 7 months ago

Cant this be done easily by using a single array and a single for loop??
using the following algorithm:
for i from 0 to 9
arr[i] = 1
for i from 2 to N

newarr[0] = oldarr[0]+oldarr[8];
newarr[1] = oldarr[1] + oldarr[2] + oldarr[4];
newarr[2] = oldarr[2] + oldarr[3] + oldarr[1] + oldarr[5];
so on till 9
oldarr = newarr
newarr = {0}
total = sum all elements in oldarr at the end
This is simple O(N) solution

• Reply • Share ›

Anurag Singh ➚ Nlk • 7 months ago

Thanks. Yes, It can be done.
The given dynamic programming approach also runs in TC: O(n) and SC: O(n) [As only one for loop runs n times, others constant time]
Your approach is a space efficient dynamic programming, as any time, nth iteration needs data from (n-1)th iteration only, so we need not keep the data from older iterations.
We will post this approach soon.
C: http://ideone.com/YUG426

• Reply • Share ›

gdb • 7 months ago

In example it is said that, for N= 2, answer should be 36. But above code is giving 26.
I think above algorithm is not considering numbers formed with repeated digits.
e.g in case of N=2, Following numbers are not considered in the final count:
00, 11,22,33,44,55,66,77,88,99
Kindly correct problem statement or algorithm

• Reply • Share ›

Ankur Dwivedi ➚ gdb • 7 months ago

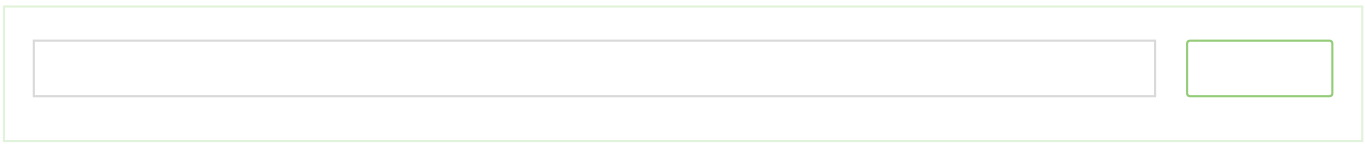
ya i too having the same doubt

• Reply • Share ›

Anurag Singh ➚ Ankur Dwivedi • 7 months ago

Yes. Realised it after the post.
Fix is to add 0 in both row and col array
i.e. We make (0,0) also a valid move from current position.
And in for loop on "move" variable, change "move < 4" to "move < 5"
Fix will be posted soon.

• Reply • Share ›



- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion

- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Follow @GeeksforGeeks

Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 1 hour ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

Because TAN is not a subsequence of RANT. ANT...

Given two strings, find if first string is a subsequence of second · 3 hours ago