
# CODE WITH COMMUNITY 2

POST CONTEST DISCUSSION

# 1) Next Similar Number

Given a number **A** in a form of string.

You have to find the smallest number that has same set of digits as A and is greater than A.

If A is the greatest possible number with its set of digits, then return -1.

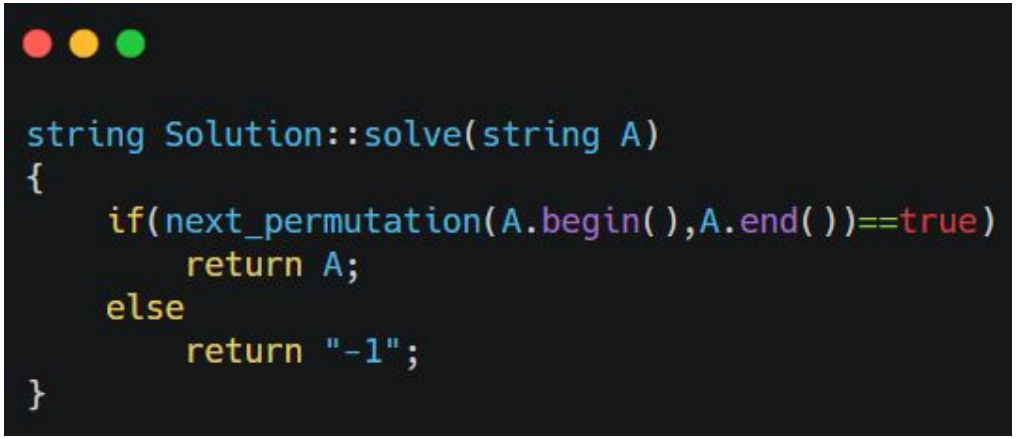
**Problem Constraints**

$1 <= A <= 10^{100000}$

A doesn't contain leading zeroes.

| Input 1: | Output 1: |
| --- | --- |
| A = "218765" → | "251678" |
| **Input 2:** | **Output 2:** |
| A = "4321" → | "-1" |

# APPROACH



```
string Solution::solve(string A)
{
    if(next_permutation(A.begin(),A.end())==true)
        return A;
    else
        return "-1";
}
```

# 2) Rearrange Array

Rearrange a given array so that `Arr[i]` becomes `Arr[Arr[i]]` with `O(1)` extra space.

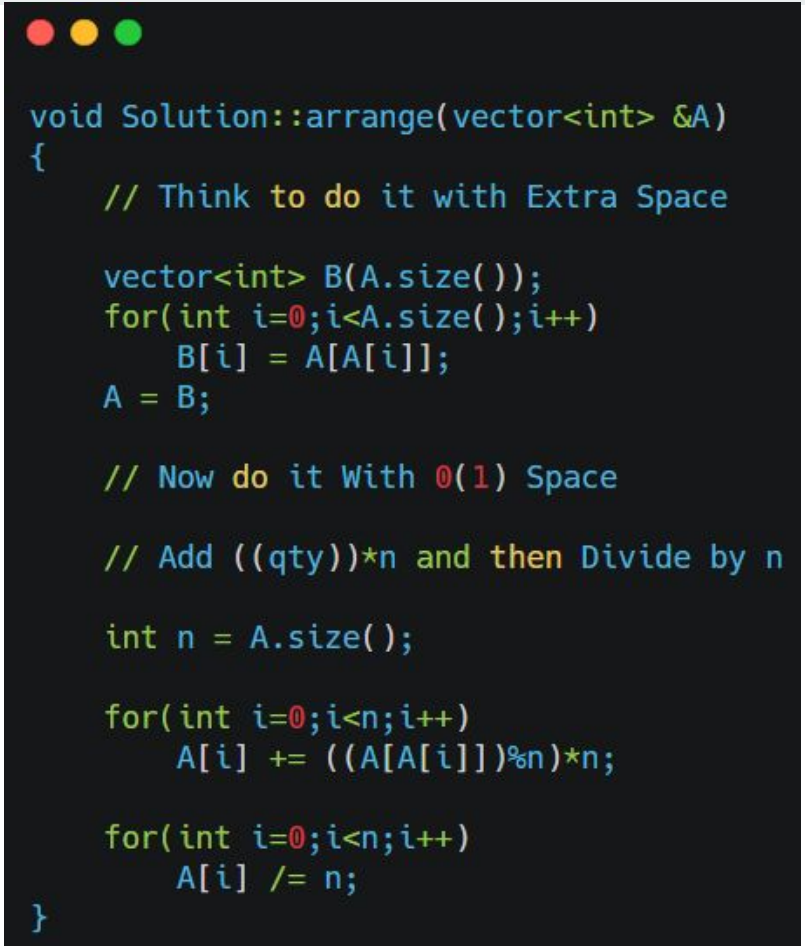
```
Input : [1, 0]
Return : [0, 1]
```

```
Input:

Arr -> [1,3,0,2]
Idx -> [0,1,2,3]

Ans -> [A[1],A[3],A[0],A[2]]
    -> [3,2,1,0]
```

# APPROACH



```
void Solution::arrange(vector<int> &A)
{
    // Think to do it with Extra Space

    vector<int> B(A.size());
    for(int i=0;i<A.size();i++)
        B[i] = A[A[i]];
    A = B;

    // Now do it With 0(1) Space

    // Add ((qty))*n and then Divide by n

    int n = A.size();

    for(int i=0;i<n;i++)
        A[i] += ((A[A[i]])%n)*n;

    for(int i=0;i<n;i++)
        A[i] /= n;
}
```

# 3) Sum of Pairing Hamming Distance

Given an array of `N` non-negative integers, find the sum of hamming distances of all pairs of integers in the array.

Return the answer modulo `1000000007`

```
Input: arr[] = {1, 2}
Output: 4
All pairs in array are (1, 1), (1, 2)
                       (2, 1), (2, 2)
Sum of bit differences = 0 + 2 +
                         2 + 0
                       = 4
```

```
Input:  arr[] = {1, 3, 5}
Output: 8
All pairs in array are (1, 1), (1, 3), (1, 5)
                       (3, 1), (3, 3) (3, 5),
                       (5, 1), (5, 3), (5, 5)
Sum of bit differences =  0 + 1 + 1 +
                          1 + 0 + 2 +
                          1 + 2 + 0
                       = 8
```

# APPROACH

```
// BRUTE FORCE O(N^2) APPROACH

int ans = 0;
for(int i=0;i<A.size();i++)
{
    for(int j=i+1;j<A.size();j++)
    {
        bitset<32> b(A[j]^A[i]);
        ans += 2*b.count();
        ans %= 1000000007;
    }
}
return ans;
```

```
// OPTIMIZED O(N) APPROACH

ll ans=0;

// Contribution of Each Bit
for(int i=0;i<32;i++)
    ll ith_bit_set = 0;
    for(auto x:A)
    {
        if(x&(1<<i))
            ith_bit_set++;
    }

    ll ith_not_set = A.size() - ith_bit_set;

    // Two Times Contribution [i,j] & [j,i]
    ll k = (ith_bit_set*ith_not_set)%mod;
    ans += k;
    ans %= mod;
    ans += k;
    ans %= mod;

return ans;
```

# Explaination

An **Efficient Solution** can solve this problem in O(n) time using the fact that all numbers are represented using 32 bits (or some fixed number of bits).

We traverse from 0 to 31 and count numbers with i'th bit set. Let this count be 'count'. There would be "n-count" numbers with i'th bit not set.

every pair having one element which has set bit at i'th position and second element having unset bit at i'th position contributes exactly 1 to sum, therefore total permutation count will be count*(n-count)

"count * (n-count) * 2"

# 4) Number of Length N and Value less than K

Given a set of digits (A) in sorted order, find how many numbers of length B are possible whose value is less than number C.

Input: 0 1 5
1
2
Output:
2 (0 and 1 are possible)
Input: 0 1 2 5
2
21
Output: 5 (10, 11, 12, 15, 20 are possible)

# APPROACH

CASE 1: B > digit_in_C ::

There won't be any Number Less than C

CASE 2: B < digit_in_C ::

Zero is Present ↠ [n-1] * n * n * .. b-1 times

n * n * .. b times

CASE 3: B = digit_in_C

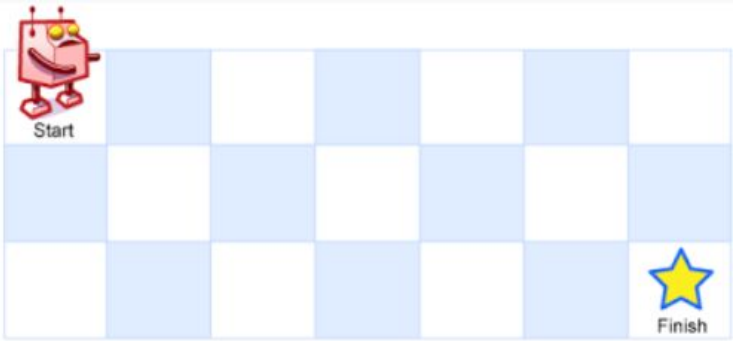
Explained in Code

Explained in More Detail in below Video

https://youtu.be/MT8zeLak_bI

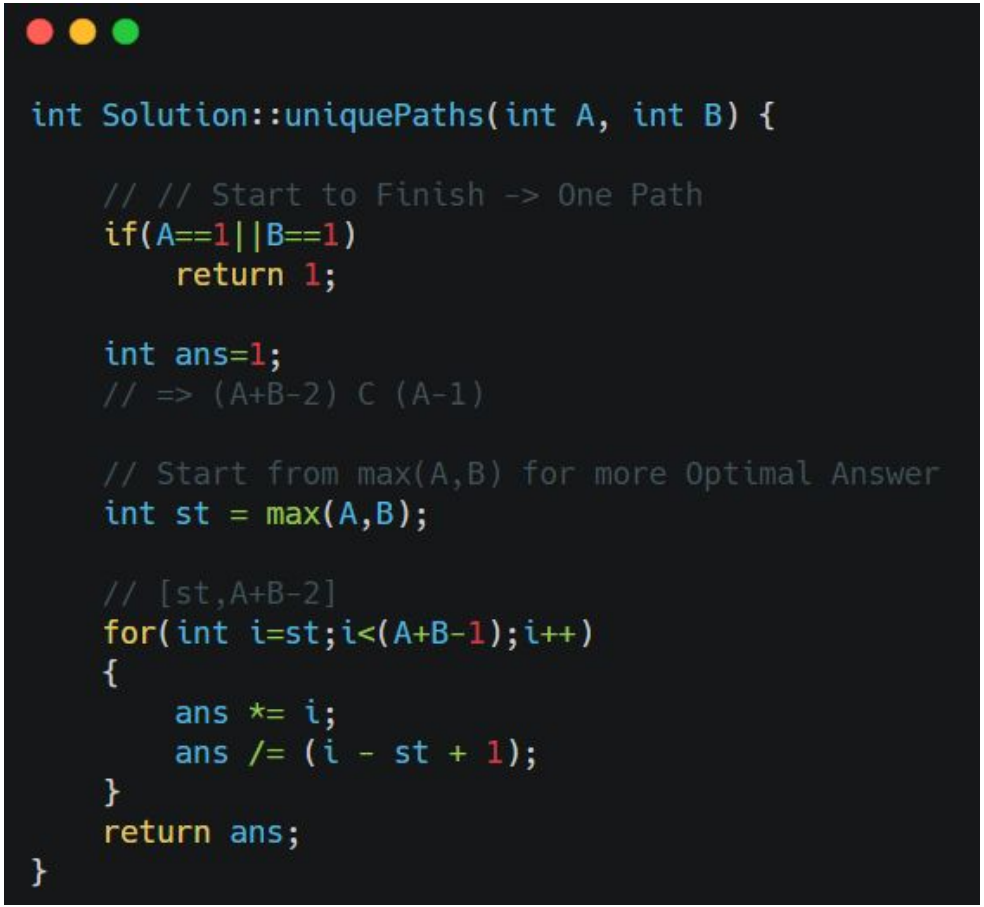
# 5) Grid Unique Path

How many possible unique paths are there?




AXB GRID
-> (A-1) DOWN MOVES & (B-1) RIGHT MOVES
TOTAL MOVES -> (A-1) + (B-1) = A+B-2
OUT OF THOSE MOVES YOU (A-1) DOWN OR (B-1) RIGHT
ANSWER:
(A+B-2)
C
(A-1)

# APPROACH



```
int Solution::uniquePaths(int A, int B) {

    // // Start to Finish -> One Path
    if(A==1||B==1)
        return 1;

    int ans=1;
    // => (A+B-2) C (A-1)

    // Start from max(A,B) for more Optimal Answer
    int st = max(A,B);

    // [st,A+B-2]
    for(int i=st;i<(A+B-1);i++)
    {
        ans *= i;
        ans /= (i - st + 1);
    }
    return ans;
}
```